UNIVERSIDAD DE BELGRANO

Las tesinas de Belgrano

Facultad de Ingeniería y Tecnología Informática

Optimización evolutiva de trayectorias de plataformas robóticas

Nº 72 **Pablo Luis Pettis**

Tutor: Raimundo O. D'Aquila

Departamento de Investigación
Abril 2003

Resumen

Este trabajo presenta la aplicación de técnicas provenientes de la Computación Evolutiva a la planificación de trayectorias de robots móviles autónomos, dentro de un ambiente conocido. El problema tratado encuadra dentro de la aplicación de técnicas de búsqueda y optimización de trayectorias, orientada a la robótica.

La resolución de este problema requirió la personalización de los operadores genéticos, cruzamiento y la mutación, debido a la naturaleza de longitud variable de los individuos que componen la población. Esto, a su vez, da la posibilidad de visualizar, gráfica y claramente, la evolución de la aptitud de la población a medida que transcurren las generaciones.

De acuerdo a lo experimentado, si el Algoritmo Evolutivo se ejecuta con parámetros adecuados, se obtendrán soluciones satisfactorias, en general serán cuasi-óptimas. Pueden encontrase dificultades ante óptimos locales pronunciados, para lo que se deja planteada una posible solución.

Palabras Clave

- Inteligencia Artificial
- Sistemas Inteligentes
- Robótica
- Computación Evolutiva
- Algoritmos Evolutivos
- Planificación de caminos

Abstract

This work presents the application of techniques coming from Evolutionary Computing to autonomous mobile robot's trajectories, in a well known environment. The problem addressed is focused within trajectories search and optimization applications, aimed to robotics.

The resolution of this problem required the customization of the operators, these are, crossover and mutation, because of the variable nature of the individuals which compose the population. At the same time, this gives the possibility of watching population's aptitude evolution throughout generations, visually and clearly.

According to experimentation, satisfactory solutions, generally almost optimum ones, will be obtained if the Evolutionary Algorithm is executed with appropiate parameters. Difficulties would be found upon marked local optimums, for what a possible solution is left stated.

Keywords

- Artificial Intelligence
- Intelligent Systems
- Robotics
- Evolutionary Computing
- Evolutionary Algorithms
- Path Planification

Índice

1. Introducción

1.1 Planteamiento y objetivos

El planeamiento de trayectorias es probablemente una de las tareas más fundamentales en aplicaciones de robótica. La solución a este problema es muy importante en muchas aplicaciones industriales, como soldadura, pintado y pegado en líneas de montaje. Este problema puede ser tratado tanto mediante técnicas algorítmicas como otras, provenientes de los paradigmas de los Sistemas Inteligentes, en particular, los de la Computación Evolutiva.

Bajo el término de Computación Evolutiva se engloba a una amplio conjunto de técnicas de resolución de problemas complejos basadas en la emulación de los procesos naturales de evolución. El principal aporte de la Computación Evolutiva a la metodología de resolución de problemas consiste en el uso de mecanismos de selección de soluciones potenciales y de construcción de nuevos candidatos por recombinación de características de otros ya presentes, de modo parecido a como ocurre en la evolución de los organismos naturales. Debe quedar bien claro que la Computación Evolutiva no trata tanto de reproducir ciertos fenómenos que suceden en la naturaleza como de aprovechar las ideas genéricas que hay detrás de ellos. Efectivamente, en el momento en que se tienen varios candidatos a solución para un problema surge inmediatamente la necesidad de establecer criterios de calidad y de selección, en función de la calidad, y también la idea de combinar características de buenas soluciones para obtener otras mejores. Dado que fue en el mundo natural donde primeramente se han planteado problemas de ese tipo, no tiene nada de extraño el que al aplicar tales ideas en la resolución de problemas científicos y técnicos se obtengan procedimientos bastante parecidos a los ya encontrados por la naturaleza tras un largo periodo de adaptación.

Precisamente, la Computación Evolutiva surge a finales de los años ’60 cuando John Holland se planteó la posibilidad de incorporar los mecanismos naturales de selección y supervivencia para resolver una gran cantidad de problemas de Inteligencia Artificial que ya habían sido “resueltos” muy eficientemente por la Naturaleza, pero que resultaban decepcionantemente inabordables mediante computadoras; el fruto de sus investigaciones ha sido considerado como el punto de arranque de la Computación Evolutiva. Los intereses de Holland y sus colaboradores fueron en principio académicos: esencialmente trataban de introducir un marco más amplio en el que englobar la Inteligencia Artificial con la intención de resolver ciertos problemas genéricos que constantemente aparecían en dicha disciplina; a la hora de llevarlas a la práctica las ideas de Holland resultaban, en el mejor de los casos, poco eficientes.

Sin embargo. a mediados de los ’80 la aparición de computadoras de altas prestaciones y bajo costo cambia por completo el panorama: la Computación Evolutiva se puede utilizar para resolver con éxito ciertos tipos de problemas de la ingeniería y de las ciencias sociales que anteriormente eran difícilmente resolubles. De ésta época son el libro de Goldberg [1] y el de Davis [5]; en ellos se plantean y resuelven problemas de la vida real en los que hay de por medio apreciables cantidades de dinero. Posiblemente como consecuencia del esfuerzo divulgador de éstos autores y debido también a las nuevas posibilidades que se vislumbraban, el desarrollo de la Computación Evolutiva a partir de entonces fue espectacular. Las implantaciones concretas de los modelos de la Computación Evolutiva se conocen como Algoritmos Evolutivos.

En la actualidad funcionan con éxito aplicaciones industriales de los Algoritmos Evolutivos en campos tan dispares como el diseño de turbinas multietapa, el control de redes de distribución de gas, la planificación de tareas, el cálculo de estrategias de mercado o el cortado de piezas con mínimo desperdicio, por citar algunas. En la última década los Algoritmos Evolutivos específicamente han sido utilizados en muchos campos como lo que tiene que ver con sistemas de identificación, planificación y scheduling, procesamiento de imágenes, reconocimiento de patrones y de voz.

Este trabajo trata la generación de trayectorias para robots móviles autónomos mediante Algoritmos Evolutivos. Estas trayectorias tendrán la particularidad de ser off-line, sobre ejes cartesianos, y libres de obstáculos, los cuales deberán ser esquivados con precisión. La aplicación de esta técnica de la Computación Evolutiva estará acompañada de una interfaz gráfica que permita observar claramente cómo se llega a la solución para cada caso en particular.

Objetivos

Dentro de este contexto se plantea como trabajo final de carrera para Ingeniería Informática, una aplicación práctica relacionada con los Algoritmos Evolutivos. Para ello, el proyecto se basa en los siguientes objetivos:

- Realizar una búsqueda bibliográfica, lo más amplia posible, de toda la información relacionada con este tema seleccionando, que sea más relevante desde un punto de vista práctico.
- Resolver completamente algún problema de "ingeniería real" utilizando las rutinas estudiadas. En este caso consiste en la utilización de la Computación Evolutiva para la búsqueda de una trayectoria adecuada entre dos puntos de una plataforma recorrida por un robot móvil autónomo, mostrando claramente, de modo gráfico, la evolución de la solución generada por el Algoritmo Evolutivo.

Con la intención de cumplir estos objetivos se ha elaborado una aplicación que resuelve el problema anteriormente planteado, mostrando gráficamente la constante evolución de las soluciones que surgen del Algoritmo Evolutivo, además de una investigación que consta de los capítulos que se describen a continuación.

- Estado del arte: Aquí se presenta a la Computación Evolutiva describiendo el lugar que ocupa en el contexto de los problemas de búsqueda y optimización. También se muestra la estructura y componentes básicos de los Algoritmos Evolutivos, así como los métodos y criterios necesarios para implantarlos. Por último se realizará una síntesis de trabajos similares que apuntan a resolver el problema planteado, destacando lo que es diferente en esta propuesta.
- Definición del problema: Planteo preciso del problema que se dispone a resolver, incluyendo todas las consideraciones pertinentes.
- Solución propuesta: Se detallará todo lo que tiene que ver con el método que llevamos adelante para resolver el problema definido.
- Resultados experimentales: En este capítulo se ilustrará lo obtenido mediante cada uno de los métodos aplicados, destacando las virtudes y defectos de cada uno, como así también al más apto para llegar a una solución cercana a la óptima.
- Conclusiones y futuras líneas de investigación.

2. Estado del Arte

2.1 Introducción a la Computación Evolutiva

La Computación Evolutiva es un enfoque alternativo para abordar problemas complejos de *búsqueda* y *aprendizaje* a través de modelos computacionales de procesos evolutivos. Las implantaciones concretas de tales modelos se conocen como Algoritmos Evolutivos. El propósito genérico de los Algoritmos Evolutivos consiste en guiar una búsqueda estocástica haciendo evolucionar a un conjunto de estructuras y seleccionando de modo iterativo las más adecuadas.

La Computación Evolutiva forma parte a su vez de un conjunto de metodologías de resolución de problemas que simulan con mayor o menor exactitud procesos naturales, como las Redes Neuronales. Todas ellas se engloban bajo el término *Computación Natural.* Pese a lo que pudiera parecer, el desarrollo de la Computación Natural no se debió tanto a disquisiciones académicas sino a la necesidad de resolver problemas cada vez más complejos en múltiples campos del conocimiento. Tal como se plantean dichos problemas las técnicas clásicas de búsqueda determinística y analítica no suelen ser de gran utilidad, lo que obliga a desarrollar nuevos paradigmas menos analíticos[1] y más sintéticos[2]. En tal circunstancia. la búsqueda de analogías con la naturaleza es más una necesidad que una preferencia estética.

Debe quedar claro en todo momento que no se persigue una *simulación*[3] de los procesos naturales, sino más bien una *emulación*[4] de dichos procesos. Por eso, un Algoritmo Evolutivo será tanto mejor cuanto mejores resultados proporcione en la resolución del problema planteado, independientemente de su fidelidad a la biología. De hecho, la mayoría de algoritmos que se derivan de este enfoque son exageradamente simplistas desde un punto de vista biológico, pero lo suficientemente complejos como para proporcionar robustos y potentes mecanismos de búsqueda.

2.2 Estructura genérica de los Algoritmos Evolutivos

Como ya quedó dicho, la metodología de la Computación Evolutiva se plasma en los *Algoritmos Evolutivos.* En general, se llama Algoritmo Evolutivo a cualquier procedimiento estocástico de búsqueda basado en el principio de evolución. Dicho principio tiene una doble vertiente: la finalidad última es la "supervivencia del más apto", el modo de conseguirlo es por "adaptación al entorno". De un modo más gráfico, los más aptos tienen más posibilidades de sobrevivir y, como resultado, más oportunidades de transmitir sus características a las generaciones siguientes.

Más concretamente, al ejecutar un Algoritmo Evolutivo, una población de individuos, que representan a un conjunto de candidatos a soluciones de un problema, es sometida a una serie de transformaciones con las que se actualiza la búsqueda y después a un proceso de selección que favorece a los mejores individuos. Cada ciclo de *transformación + selección* constituye una *generación.* Se espera del Algoritmo Evolutivo que, tras cierto número de generaciones (iteraciones) el mejor individuo esté razonablemente próximo a la solución buscada.

Dicho de otra manera, la Computación Evolutiva trata de desarrollar mecanismos estocásticos de búsqueda en paralelo con los que mejorar las técnicas clásicas de búsqueda determinista cuando estas no sean buenas o ni siquiera existan. Para que la mejora sea efectiva, tales mecanismos deben estar "dirigidos", de ahí la necesidad de introducir un procedimiento de selección. En definitiva, para poder emular suficientemente el proceso de evolución el Algoritmo Evolutivo debe disponer de:

1. Una población de posibles soluciones debidamente representadas a través de *individuos.*
2. Un procedimiento de *selección* basado en la *aptitud* de los individuos.
3. Un procedimiento de *transformación,* esto es, de construcción de nuevas soluciones a partir de las disponibles actualmente.

Una vez que estos elementos han sido convenientemente especificados se implanta el Algoritmo Evolutivo siguiendo el esquema general:

```
BeginAlgorithm{EvolutionAlgorithm}
P[0]   =  InitPop();                                  Población inicial
FitP[0]  = EvalPop(P[0]) ;                            Aptitudes iniciales
for(t = 0; t < MaxNumGen; t++)
        {
        Q[t]  = SeleccionarProgenitores(P[t]) ;       Selección de reproductores
        Q[t]  = Transformar(Q[t]);
        Reproducción FitQ[t] = EvalPop(Q[t]) ;
        P[t]  = SelSurv(P[t] ,Q[t] ,FitP[t] ,FitQ[t]);
        FitP[t]  = EvalPop(P[t]) ;                    Se evalúan la nueva población
        If (ChkTermCond(P[t], FitP[t]))               Condición de terminación
        return;
        }
EndAlgorithm
```

Sobre ese esquema general se han desarrollado los siguientes paradigmas fundamentales:

- *Algoritmos Genéticos*: Cada cromosoma está constituido por una secuencia de binits.
- *Programación Evolutiva*: Es una generalización de los Algoritmos Genéticos. Se utilizan otras estructuras de datos que permitan lograr representaciones más eficientes de acuerdo al problema.
- *Programación Genética*: Cada cromosoma es un programa computacional en un lenguaje dado. Está usualmente constituido por secuencias de instrucciones del lenguaje

2.3 La Computación Evolutiva en el contexto de los problemas de búsqueda y optimización.

Antes de entrar con detalle en la descripción de los cuatro paradigmas de la Computación Evolutiva conviene hacer un breve repaso a los conceptos elementales de la teoría general de búsqueda y optimización. Así se introducirá la terminología básica y además se situarán los Algoritmos Evolutivos en el marco de comparación adecuado. Se debe notar que los términos búsqueda y optimización son dos facetas de un mismo concepto: La diferencia estriba en que al decir "búsqueda" se quiere hacer hincapié en el proceso en sí, mientras que al decir "optimización" se destaca más el resultado de tal proceso.

Sin pérdida de generalidad se puede admitir que se resuelve una maximización, pues para minimizar basta con cambiarle el signo a f(x). Para mayor sencillez se supondrá también que la optimización es *mono-objetivo,* esto es, que f(x) devuelve un escalar.

2.3.1 Clasificación general de los procedimientos de optimización

De modo general se pueden establecer cinco criterios de clasificación para los procedimientos de optimización:

1. Basándose en la naturaleza de las soluciones las optimizaciones pueden ser:
- *Numéricas:* Si la solución queda completamente especificada en términos de un conjunto de *m* parámetros o atributos.
- *Combinatorias:* Si para especificar la solución no sólo hay que especificar un conjunto de *m* parámetros, sino también el orden (total o parcial) con que estos se combinan para dar dicha solución. En último término puede ocurrir que sea irrelevante la naturaleza de los atributos y sólo importe su orden. En tal caso se habla de *problemas basados en el orden.*

2. Basándose en el grado de aleatoriedad que se le da al proceso de búsqueda las optimizaciones pueden ser:
- *Deterministas o dirigidas:* El procedimiento de búsqueda es completamente determinista, es decir, en las mismas condiciones de partida proporciona idénticos resultados.
- *Aleatorias o al azar:* El procedimiento de búsqueda es completamente aleatorio. Habitualmente, se delimita una región de búsqueda y se toman puntos al azar dentro de ellas. Después mediante argumentos estadísticos se puede dar una estimación de máxima verosimilitud para el valor del óptimo.
- *Estocásticas u orientadas:* Se combinan en proporción variable la búsqueda determinista con la búsqueda aleatoria. La componente determinista orienta la dirección de búsqueda y la aleatoria se encarga de la búsqueda local.

Las técnicas clásicas de optimización, ya sean analíticas o enumerativas, son todas deterministas; por lo común son las más eficientes, pero son muy específicas y precisan mucho conocimiento adicional sobre la función objetivo así como el uso de hipótesis suplementarias de "buen comportamiento", entre otros inconvenientes. En cuanto a las técnicas aleatorias la situación es opuesta: no requieren ninguna información adicional y se pueden aplicar a cualquier tipo de problemas, pero son poco eficientes. Por estos motivos resulta conveniente buscar un punto intermedio entre la máxima eficiencia de las técnicas deterministas y la máxima eficacia de las aleatorias.

3. Basándose en la dirección preferente de búsqueda las optimizaciones pueden ser:
- *En profundidad ó Explotadoras:* La búsqueda da prioridad a la *explotación* de las soluciones disponibles antes que a la exploración de nuevas soluciones.
- *En anchura* ó *Exploradoras:* La búsqueda da prioridad a la *exploración* de nuevas soluciones antes que a la explotación de las disponibles.

En pocas palabras, la exploración trata de obtener más conocimiento del espacio de búsqueda y la explotación trata de aprovechar el que ya se tiene. Muchos métodos de búsqueda no son explotadores o exploradores puros, sino que combinan ambos procedimientos. De hecho es deseable tener control sobre la relación explotación - exploración. Idealmente, esa relación debería poderse expresar como un cociente entre dos parámetros al que se llama *grado de penetración.*

4. Basándose en el número de candidatos a solución que se mantienen simultáneamente las búsquedas pueden ser:
- *Simples:* Se mantiene un solo candidato a solución que se va actualizando sucesivamente para proporcionar, presumiblemente, soluciones cada vez más exactas del problema.
- *Múltiples:* Se mantienen simultáneamente varios candidatos a solución con los cuales se va acotando cada vez con más precisión la región (o regiones) donde se encuentra el/los óptimo/s. Son las más apropiadas para implantaciones en paralelo.

Aunque son computacionalmente más costosas, las búsquedas múltiples presentan tal cantidad de ventajas sobre las búsquedas simples que se deberían usar siempre que fuera posible, aunque no se disponga de procesadores en paralelo.

5. Basándose en la información disponible sobre la función a optimizar las búsquedas pueden ser:
- *Ciegas:* El proceso a optimizar funciona como una *caja negra* que ante ciertos valores de los parámetros devuelve un valor del objetivo. Es decir, no se dispone de ninguna información explícita sobre la aplicación f(x). A estos métodos de optimización se los denomina como "optimización mediante cajas negras'". La ventaja de estos métodos es que proporcionan algoritmos de búsqueda de propósito general, los cuales son muy fáciles de implantar para un problema específico.
- *Heurísticas:* Se dispone de cierta información explícita sobre el proceso a optimizar, la cual se puede aprovechar para guiar la búsqueda. En la jerga del ramo a dicha información útil para la búsqueda se le llama *conocimiento específico.* Las técnicas heurísticas proporcionan *algoritmos dedicados* de búsqueda, esto es, específicos para un problema concreto y difícilmente adaptables para cualquier otro.

Tradicionalmente se ha tratado de añadir la máxima cantidad de conocimiento específico en los problemas de optimización, dado que es lo que más eficacia da a la búsqueda. Sin embargo, en problemas reales de mediana complejidad resulta muy difícil, cuando no imposible, encontrar tal información y la que se encuentra no suele ser de muy buena calidad. A efectos prácticos, el conocimiento específico sólo sirve verdaderamente cuando es de buena calidad, y aun así se debe tener cuidado porque acentúa la tendencia a estancar la búsqueda en óptimos locales.

La cuestión de si añadir conocimiento específico a un problema y de cuánto y cómo añadirlo es fundamental en la teoría moderna de optimización.

2.3.2 Complejidad de los problemas de optimización

Atendiendo a su complejidad, se dice de un problema de optimización que es *polinómicamente programable* cuando existen procedimientos de resolución que dan la respuesta tras realizar un número de operaciones que es, a lo sumo, una función polinómica del tamaño del problema. Este es un requisito mínimo de complejidad, es decir, si un problema no es polinómicamente programable todos los métodos de resolución serán inútiles para tamaños crecientes del problema ya que ningún computador sería capaz de afrontarlos en un tiempo razonable.

Por desgracia, gran número de los problemas de optimización que surgen en la práctica no son polinómicamente programables y, lo que es peor, existen buenas razones teóricas para pensar que ésta es una característica intrínseca de tales problemas, esto es, es imposible encontrar un procedimiento polinómico de búsqueda que los resuelva.

Dado que tales problemas se deben resolver de un modo u otro caben las siguientes alternativas:

1. Modificar el problema de tal manera que admita un algoritmo de resolución aproximada, esto es, que encuentre buenas soluciones pero no necesariamente la mejor. A tales soluciones se les llama *cuasi óptimos.*
2. Desarrollar algoritmos específicos que encuentren la solución en algunos casos, pero no necesariamente en todos, con la esperanza (más o menos fundada) de que los problemas que surgen en la práctica entren en el primer grupo.
3. Rebajar las restricciones del problema para que admita un algoritmo de resolución en tiempo polinómico y obtener varias soluciones. Se elegirá la que más se ajuste a las restricciones originales.
4. Investigar el intervalo de algoritmos “cuasi-polinómicos”. Los algoritmos de resolución en tiempo no exponencial dan algunas esperanzas de resolver problemas moderadamente grandes.

Los nuevos métodos de optimización deben poder contemplar la utilización de estas u otras alternativas para proporcionar soluciones razonables a problemas de este tipo. Concretamente, los Algoritmos Evolutivos suelen hacer uso de la primera alternativa, buscan cuasi-óptimos.

2.3.3 Fortaleza y robustez. El conflicto entre generalidad y eficiencia

Tradicionalmente, la característica global más importante de un método de búsqueda es su *fortaleza***.** Esta se mide en términos de eficiencia: un método es tanto más fuerte cuanto más eficientemente lleve a cabo su tarea.

Ahora bien, un postulado fundamental de la teoría de optimización afirma que la eficiencia y la generalidad son objetivos contrapuestos, esto es, cuanto más fuerte sea un método menor cantidad de problemas será capaz de resolver y viceversa. Es precisamente a esto a lo que se refiere la bibliografía al hablar del *conflicto entre generalidad y eficiencia*.

Los métodos heurísticos y/o deterministas son los que más fortaleza poseen. Tradicionalmente han sido los más usados porque es preferible tener poca generalidad antes que poca eficiencia. Sin embargo, el incremento de la cantidad y complejidad de los problemas a resolver llevó a mediados de la década de los ’60 formular la siguiente cuestión:

> “Admitiendo que para los procedimientos tradicionales de búsqueda la relación generalidad vs. eficiencia es inevitablemente constante. ¿Existen otros procedimientos de búsqueda para los que dicho producto sea mayor?”

Esto lleva a cuestionar el papel fundamental que se otorga a la fortaleza para evaluar la calidad de un método de búsqueda y a introducir una nueva característica de los métodos de búsqueda: la *robustez.* Un método es tanto más robusto cuanto mayor sea la relación generalidad x eficiencia. Desde esta perspectiva, se trata de determinar si existen métodos de búsqueda más robustos que los tradicionales.

Como consecuencia del gran número de investigaciones iniciadas en aquella época se han encontrado efectivamente nuevos procedimientos de búsqueda más robustos que las tradicionales. Para ello ha sido

necesario relativizar las exigencias de fortaleza y aprovechar las buenas cualidades en términos de robustez que proporcionan el no determinismo y la *opacidad,* esto es la no utilización de conocimiento específico.

2.4 Técnicas clásicas de búsqueda *y* optimización

Se hará un recorrido somero sobre las técnicas de optimización más comunes, no tanto para describirlas sino para enumerar sus limitaciones en términos de robustez.

Técnicas analíticas de búsqueda y optimización

De manera general, las técnicas clásicas de búsqueda y optimización se pueden clasificar como analíticas o como enumerativas. Las *técnicas analíticas* están especialmente indicadas para problemas numéricos con las siguientes propiedades:

- El espacio de búsqueda ha de estar claramente definido y ser continuo y regular para que en todo momento tenga sentido el concepto de ”vecindad”.
- La función objetivo no debe ser ruidosa (a cada entrada le ha de corresponder siempre la misma salida) y ha de ser lo suficientemente “lisa” como para que en todo momento se pueda definir una “dirección preferente de búsqueda”.

Las técnicas clásicas hallan el óptimo resolviendo ecuaciones de modo directo o indirecto. La *resolución directa* sólo es abordable cuando f(x) se expresa como una composición analítica de funciones elementales y aún en tal caso puede ocurrir que no sea el mejor método. Los *métodos indirectos de resolución* buscan el óptimo trepando iterativamente la función objetivo en la dirección de máxima pendiente. Procediendo de este modo se garantiza encontrar un óptimo local o *subóptimo* (un punto rodeado de puntos peores que él), aunque no necesariamente el óptimo global.

Este procedimiento de *escalada* da lugar a algoritmos de resolución muy sencillos, siempre y cuando se disponga de una buena técnica de determinación de la dirección de máxima pendiente. Habitualmente, tal dirección se determina a través de modelos locales aproximados de primer o segundo orden auxiliados por procedimientos diversos que evitan el estancamiento de la búsqueda en mesetas, en grietas o en puntos singulares.

Todas estas técnicas analíticas tienen dos serias desventajas, que van en detrimento de su robustez. Primeramente. todas admiten de un modo u otro que aunque no se pueda dar una forma cerrada a la función objetivo, tiene sentido el concepto de ‘pendiente’; ello restringe grandemente su campo de aplicación. A decir verdad, se han desarrollado técnicas analíticas de búsqueda que no hacen uso de dicho concepto, pero aun así siguen requiriendo que en todo momento esté completamente definida una *dirección preferente de búsqueda.* En definitiva, por una causa o por otra se precisa una gran cantidad de conocimiento específico. En segundo lugar, las técnicas analíticas son esencialmente locales, esto es, sólo pueden hallar subóptimos y, lo que es peor, en el caso de que haya varios (a esto se le llama *multimodalidad*) el acabar en uno u otro depende del punto inicial, siendo imposible determinar a priori a qué subóptimo llevará un punto inicial dado.

Esos dos resultados se resumen en que. por un lado, las técnicas analíticas de optimización son muy eficientes pero muy específicas, esto es, muy fuertes. Por otro lado, las desventajas de la alta especificidad no se logran compensar con la alta eficiencia, lo que las hace poco robustas.

Técnicas enumerativas de búsqueda y optimización

El campo natural de aplicación de las *técnicas enumerativas* de optimización es el de los problemas combinatorios, aunque no de modo exclusivo. El método del simples, sin ir más lejos, es una técnica enumerativa para resolver problemas de optimización numérica y la Programación Dinámica de Bellman sirve tanto para problemas numéricos como combinatorios.

No existen métodos generales para resolver problemas de optimización combinatoria. ni siquiera existe un problema típico. Debido en gran parte a esto, todas las técnicas enumerativas de búsqueda *y* optimización requieren gran cantidad de conocimiento específico.

Las técnicas de búsqueda en grafos constituyen las herramientas más comunes para la búsqueda enumerativa. Procedimientos como el algoritmo *A** o el de *ramificación y acotación* dan resultados bastante aceptables al resolver problemas combinatorios de mediano tamaño. Sin embargo, la mayoría de las técnicas clásicas para resolver problemas combinatorios tienen complejidad exponencial (esto es, el número de operaciones a realizar aumenta exponencialmente con el tamaño del problema), por lo que resultan ser inútiles en la resolución de problemas grandes.

En definitiva, bien sea por su tendencia al crecimiento exponencial o por su alta solicitación de conocimiento específico, las técnicas enumerativas de optimización resultan ser poco robustas.

La conclusión final que se extrae de este apartado es que las técnicas clásicas de optimización, bien por demasiado específicas bien por demasiado ineficientes, no tienen gran robustez. Precisamente, la Computación Evolutiva trata de incrementar la robustez de los métodos de búsqueda añadiéndoles eficacia manteniendo su generalidad y viceversa.

2.5 Algoritmos Genéticos

De entre los paradigmas principales de la Computación Evolutiva, los Algoritmos Genéticos son los de mayor popularidad. Las razones de esta preeminencia son tanto teóricas como prácticas, siendo las más importantes:

- Los Algoritmos Genéticos reúnen de modo natural, todas las ideas fundamentales de la Computación Evolutiva.
- Son muy flexibles, es decir, pueden adoptar con facilidad nuevas ideas, generales o específicas, que surjan en el campo de la Computación Evolutiva. Además, se pueden hibridar fácilmente con otros paradigmas y enfoques, aunque no tengan ninguna relación con la Computación Evolutiva.
- De entre todos los paradigmas de la Computación Evolutiva son los que menos conocimiento específico necesitan para su funcionamiento, y en consecuencia, los más versátiles. Pero es que además pueden incorporar conocimiento específico con poco esfuerzo adicional.
- Son fácilmente implantables en computadores de capacidades medias, proporcionando resultados francamente aceptables, en cuanto a precisión y recursos empleados, para una gran cantidad de problemas difícilmente resolubles por otros métodos.
- Posiblemente como consecuencia de los anteriores argumentos los Algoritmos Genéticos son el paradigma más usado de entre los de la Computación Evolutiva y, junto con las Redes Neuronales, los más usados de toda la Computación Natural. Esto es importante en la medida en que pone a disposición del usuario gran cantidad de ensayos empíricos previos que proporcionan operadores, parámetros e implantaciones específicas para una amplia gama de problemas.

Debido a todo esto, todo progreso en el campo de la Computación Evolutiva se plantea teniendo en mente este paradigma. La consecuencia final es que es imprescindible realizar un estudio amplio y pormenorizado de tales métodos si se quiere tener un conocimiento cabal de la Computación Evolutiva.

Una definición general de los Algoritmos Genéticos podría ser como sigue:

Los Algoritmos Genéticos son métodos estocásticos de búsqueda ciega de soluciones cuasi-óptimas. En ellos se mantiene una población que representa a un conjunto de posibles soluciones la cual es sometida a ciertas transformaciones con las que se trata de obtener nuevos candidatos y a un proceso de selección sesgado en favor de los mejores candidatos.

Sin entrar de momento en pormenores, se debe señalar que la principal innovación de los Algoritmos Genéticos en el dominio de los métodos de búsqueda es la adición de un mecanismo de *selección* de soluciones. Recuérdese que la selección tiene dos vertientes: a corto plazo los mejores tienen más posibilidades de sobrevivir y a largo plazo los mejores tienen más posibilidades de tener descendencia. A causa de esto. el mecanismo de selección se desdobla en dos: uno elige los elementos que se van a trasformar posteriormente este es el operador de *selección* a secas; el otro operador elige los elementos que van a sobrevivir, por lo que se le llama reemplazo.

En cuanto a las restantes características, cabe destacar que los Algoritmos Genéticos son métodos de búsqueda:

- **Ciega,** es decir, no disponen de ningún conocimiento específico del problema. de manera que la búsqueda se basa exclusivamente en los valores de la función objetivo.
- **Codificada**, puesto que no trabajan directamente sobre el dominio del problema, sino sobre representaciones de sus elementos.
- **Múltiple**, esto es, buscan simultáneamente entre un conjunto de candidatos.
- **Estocástica** referida tanto a las fases de selección como a las de transformación. Ello proporciona control sobre el factor de penetración de la búsqueda.

Todas las características enumeradas se introducen deliberadamente para proporcionar más robustez a la búsqueda. esto es, para darle más eficiencia sin perder la generalidad y viceversa.

Goldberg [1] justifica esta afinación del siguiente modo:

Los Algoritmos Genéticos manejan variables de decisión o de control representadas como cadenas con el fin de explotar similitudes entre cadenas de altas prestaciones. Otros métodos tratan habitualmente con las funciones y sus variables de control directamente. Dado que los Algoritmos Genéticos operan en el nivel de códigos, son difíciles de engañar aun cuando la función sea difícil para los enfoques tradicionales.

Los Algoritmos Genéticos trabajan con una población; muchos otros métodos trabajan con un único punto. De este modo, los Algoritmos Genéticos encuentran seguridad en la cantidad. Al mantener una población de puntos bien adaptados, se reduce la probabilidad de alcanzar un falso óptimo.

Los Algoritmos Genéticos consiguen gran parte de su amplitud ignorando la información que no sea la del objetivo. Otros métodos se basan fuertemente en tal información, y en problemas donde la información necesaria no está disponible o es difícil de conseguir, estos otros métodos fallan. Los Algoritmos Genéticos son generales porque explotan la información disponible en cualquier problema de búsqueda. Los Algoritmos Genéticos procesan similitudes en el código subyacente junto con información proveniente de la ordenación de las estructuras de acuerdo con sus capacidades de supervivencia en el entorno actual. Al explotar una información tan fácilmente disponible, los Algoritmos Genéticos se pueden aplicar en prácticamente cualquier problema.

Las reglas de transición de los Algoritmos Genéticos son estocásticas: otros muchos métodos tienen reglas de transición deterministas. Hay una diferencia. no obstante, entre los operadores estocásticos de los Algoritmos Genéticos y otros métodos que no son más que paseos aleatorios. Los Algoritmos Genéticos usan el azar para guiar una búsqueda fuertemente explotadora. Esto puede parecer inusual, usar del azar para conseguir resultados concretos (los mejores puntos), pero hay gran cantidad de precedentes en la naturaleza.

Sin embargo, la característica esencial de los Algoritmos Genéticos no se observa directamente: es su capacidad de intercambio estructurado de información en paralelo, o abreviadamente el *paralelismo implícito.* De modo intuitivo, esta característica se refiere a lo siguiente: los Algoritmos Genéticos procesan externamente cadenas de códigos, sin embargo, lo que se está procesando internamente son similitudes entre cadenas y de manera tal que al procesar cada una de las cadenas de la población se están procesando a la vez todos los patrones de similitud que contienen, que son muchos más. Es precisamente por esta propiedad que los Algoritmos Genéticos son mucho más eficaces que otros métodos de búsqueda ciega. y por tanto más robustos con independencia de otras consideraciones.

2.5.1 Diferencias entre los Algoritmos Genéticos y los métodos tradicionales.

Para sobrepasar a los métodos tradicionales en el búsqueda de robustez, los Algoritmos Genéticos deben diferir de estos en algunos puntos fundamentales. Los Algoritmos Genéticos difieren de otros procedimientos de búsqueda y optimización en los siguientes aspectos:

1. Los Algoritmos Genéticos trabajan con una codificación de los parámetros, no con los parámetros en sí.
2. Los Algoritmos Genéticos realizan búsquedas sobre un conjunto de puntos, no sobre un único punto.
3. Los Algoritmos Genéticos toman información a través de una función objetivo, no a través de información auxiliar.
4. Los Algoritmos Genéticos usan reglas de transición probabilísticas, no determinísticas.
5. Los operadores utilizados no dependen del problema.

Los Algoritmos Genéticos requieren que la codificación de los parámetros posea una longitud finita basada en un alfabeto también finito.

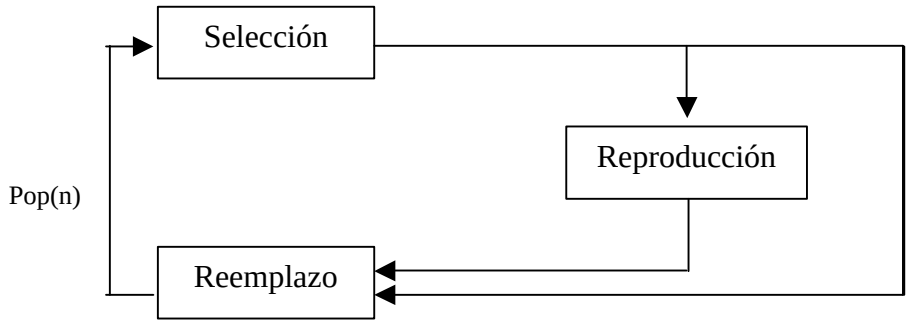
En muchos métodos de optimización nos movemos de un punto a otro cautelosamente utilizando reglas de transición para determinar el próximo. Este método punto-a-punto es peligroso porque lleva a encontrar falsos picos en espacios de búsqueda multimodal (de varios picos). A diferencia de esto, los Algoritmos Genéticos trabajan con una numerosa cantidad de puntos en simultáneo (individuos), escalando varios picos en paralelo; por lo tanto la probabilidad de encontrar un pico falso se reduce.

2.5.2 Estructura y componentes básicos

Como ya quedó dicho en el capítulo anterior, al ejecutar un Algoritmo Evolutivo una población de individuos, que representan a un conjunto de candidatos a soluciones de un problema, es sometida a una serie de transformaciones con las que se actualiza la búsqueda *y* después a un proceso de selección que favorece a los mejores individuos. Cada ciclo de selección y búsqueda constituye una generación, y se espera del algoritmo evolutivo que, al cabo de un número razonable de generaciones, el mejor individuo represente a un candidato lo suficientemente próximo a la solución buscada. Los Algoritmos Genéticos, como ejemplo fundamental de Algoritmos Evolutivos, siguen dicho esquema básico con ciertas peculiaridades propias de la implantación que se verán seguidamente.

De modo gráfico, la estructura genérica del bucle básico de un Algoritmo Genético se sintetiza en el siguiente diagrama:

1. Análisis: Separación y distinción de los elementos de un todo.
2. Síntesis: Composición de un todo por la reunión de sus partes.
3. Simular: Representar alguna cosa, fingiendo lo que no es.
4. Emular: Imitar las acciones de otro procurando igualarle y aun excederle.



Como puede verse, una población *Pop,* que consta de *n* miembros se somete a un proceso de *selección* para constituir una población intermedia *AuxPop* de *n criadores.* De dicha población intermedia se extrae un grupo reducido de individuos llamados *progenitores* que son los que efectivamente se van a reproducir. Sirviéndose de los *operadores genéticos,* los progenitores son sometidos a ciertas transformaciones de *alteración* y *recombinación* en la fase de *reproducción* en virtud de las cuales se generan S nuevos individuos que constituyen la *Descendencia.* Para formar la nueva población *Pop[t + 1]* se deben seleccionar *n* supervivientes de entre los *n + s* de la población auxiliar y la descendencia, eso se hace en la fase de *reemplazo.* Como ya se comentó, la selección se hace en dos etapas con la idea de emular las dos vertientes del Principio de Selección natural: selección de criadores o 'selección' a secas y selección de supervivientes para la próxima generación o 'reemplazo'.

Los objetos que se someten a evolución en un Algoritmo Genético son cadenas binarias y sobre las que se codifican los elementos del espacio de búsqueda; ambos reciben el nombre de *individuo, codificado* el uno y *sin codificar* el otro. También son usuales las denominaciones "ejemplar", "muestra" o "punto".

Cada individuo x consta de *m posiciones* que son ocupadas por otros tantos atributos o *genes* los cuales se codifican en $L = L_1 + .. + L_m$ bits. Naturalmente, si un atributo puede tomar más de dos valores se deberá representar mediante varios bits; entonces se deberá codificar mediante $Lj = [\text{Log}_2 a_j]$ bits. En este punto conviene distinguir entre la estructura de los individuos y el contenido de cada uno: la estructura se deriva de la *representación,* es igual para todos y se expresa así «los 4 primeros bits representan al primer gen, los 8 siguientes al segundo, etc...», el contenido se deriva de la *codificación y* es simplemente el entero binario con que se codifica cierto individuo.

En la jerga al uso a la estructura de un individuo se le dice *genotipo* y a su contenido *fenotipo.* A la hora de programar un Algoritmo Genético, el genotipo se implanta a través de una clase y los fenotipos a través de objetos o «instanciaciones» de dicha clase.

Todo el procedimiento de búsqueda está guiado exclusivamente por una *función de aptitud* u(x), la cual se obtiene directamente a partir de la *función de evaluación f(x)* del correspondiente problema. La función de evaluación no tiene por qué estar expresada de forma cerrada como una función objetivo clásica; basta con que proporcione un "índice de idoneidad" para cada uno de los candidatos a solución que se le presenten. De todos modos, es conveniente que el procedimiento de obtención de dicho índice se pueda implantar con facilidad en un computador, dado que la evolución de todo el algoritmo va a depender de él. Comúnmente, a los valores proporcionados por la función de evaluación se les dice *evaluaciones* o bien *aptitudes brutas* mientras que a los valores proporcionados por la función de aptitud se les dice *aptitudes* a secas, y también, para distinguirlas de las aptitudes brutas, *aptitudes netas.*

2.5.3 Métodos y criterios necesarios para implantar un Algoritmo Genético

Para implantar un Algoritmo Genético es necesario definir de modo inequívoco los siguientes métodos y criterios.

- ***Criterio de codificación:*** Dado que los Algoritmo Genéticos operan exclusivamente con cadenas binarias *(representación)* se debe especificar el procedimiento *(codificación)* con el que se hace corresponder cada punto del dominio del problema con una cadena, o con la terminología anterior, el mecanismo de paso del genotipo a los fenotipos.
- ***Criterio de tratamiento de los individuos no factibles:*** No siempre es posible establecer una correspondencia punto por punto entre el dominio de un problema y el conjunto de las cadenas binarias de tamaño *t* usadas para resolverlo, o sea, el espacio de búsqueda. Como consecuencia, no todas las cadenas codifican elemento válidos del espacio de búsqueda y se deben habilitar procedimientos útiles para distinguirlas.

- ***Criterio de inicialización:*** Se refiere a cómo se debe construir la población inicial con la que se arranca el bucle básico del Algoritmo Genético.
- ***Criterio de parada:*** Se deben concretar las condiciones con las que se considera que el Algoritmo Genético ha dado con una solución aceptable o, en su defecto, ha fracasado en la búsqueda y no tiene sentido continuar.
- ***Funciones de evaluación y aptitud:*** Se debe concretar la función de evaluación más apropiada para el problema, así como la función de aptitud que utilizará el Algoritmo Genético para resolverlo.
- ***Operadores genéticos:*** Se llaman así a los operadores con los que se lleva a cabo la reproducción. Todo Algoritmo Genético hace uso de al menos dos operadores genéticos, el cruce y la mutación; no obstante ellos no son los únicos posibles y además admiten variaciones.
- ***Criterios de selección:*** La selección debe dirigir el proceso de búsqueda en favor de los miembros más aptos, pero existen muchas maneras de llevar esto a cabo.
- ***Criterios de reemplazo:*** Los criterios con que se seleccionan los criadores no necesariamente han de ser los mismos con que se seleccionan los supervivientes, de ahí la necesidad de especificarlos por separado.
- ***Parámetros de funcionamiento:*** Un Algoritmo Genético necesita que se le proporcionen ciertos parámetros de funcionamiento, tales como el tamaño de la población, las probabilidades de aplicación de los operadores genéticos, la precisión de la codificación, las tolerancias de la convergencia, etc.

La elección de unas u otras no sólo depende del tipo de problema a resolver, sino también de que no se violen ciertos requisitos imprescindibles para el buen funcionamiento del Algoritmo Genético.

2.5.4 Mecanismos de muestreo de poblaciones

Un mecanismo auxiliar pero fundamental para los Algoritmos Genéticos es el de *muestreo de poblaciones*, esto es, la elección según ciertos criterios de un subconjunto de *k* individuos de una población especificada. Los mecanismos de muestreo son muy variados, distinguiéndose tres grupos fundamentales según el grado de intervención del azar en el proceso:

- ***Muestreo directo:*** Se toma un subconjunto de individuos de la población siguiendo un criterio fijo, del estilo de «los *k* mejores», "los *k* peores», «a dedo», etc.
- ***Muestreo aleatorio simple o equiprobable:*** Se asignan a todos los elementos de la población base las mismas probabilidades de formar parte de la muestra y se constituye ésta mediante ensayos de Bernuolli simples.
- ***Muestreos estocásticos:*** Se asignan probabilidades de selección o *puntuaciones* a los elementos de la población base en función (directa o indirecta) de su aptitud.

Existen muchos mecanismos de muestreo estocástico según para lo que se apliquen. En concreto, al implantar Algoritmos Genéticos se usan fundamentalmente cuatro tipos de muestreo estocástico, que se describirán seguidamente. Para ilustrarlos, se considerará el problema de muestrear estocásticamente dos elementos de la población $P = \{x_1,x_2,x_3,x_4\}$ siendo las puntuaciones asociadas $p_1 = 0.1$, $p_2 = 0.3$, $p_3 = 0.1$ *y* $p_4 = 0.5$ respectivamente.

- ***Por sorteo:*** Se consideran las puntuaciones estrictamente como probabilidades de elección para formar la muestra, y se constituye ésta realizando *k* ensayos de una variable aleatoria con dicha distribución de probabilidades. Dado que habitualmente sólo se dispone de un generador de números aleatorios simples (uniformemente distribuidos entre 0 y 1), para simular un ensayo de la distribución $\{p_{1:}... ,p_n\}$ se hace lo siguiente:

1. Se calculan las *puntuaciones acumuladas* así:
$q_o := 0$
$q_i := P_1 + \bullet\bullet\bullet + P_i$ ($" _i = 1,..., n$)
2. Se genera un número aleatorio simple $r \neg Rand[0,1)$
3. Se elige el individuo x_i que verifique $q_{i-1} < r < qi$

Para muestrear por sorteo *k* individuos, se modifican trivialmente los dos últimos pasos así:
2. Se generan *k* números aleatorios simples $r_j \neg Rand [0,1)$ ($" _j = 1,..., k$).
3. Para cada $j = 1,..., k$ se elige el individuo x_i que verifique $q_{i-1} < r < qi$

Para el ejemplo propuesto las puntuaciones acumuladas son:
$q_1 = 0.1$ $q_2 = 0.4$ $q_3 = 0.5$ $q_4 = 1.0$
respectivamente, y los $k = 2$ números aleatorios simples se han hallado como

$r_1 = 0.02433$ y $r_2 = 0.51772$

De esta manera, el primer elemento de la muestra resulta ser x_i dado que $q_0 < r_1 < q_1$ el segundo será x_4 puesto que $q_3 < r_1 < q_4$.

Nótese que existe la posibilidad de que algunos individuos puedan ser elegidos varias veces en una muestra (los que tengan mayor puntuación o un golpe de buena suerte con los r_j) y que otros individuos pueden no ser elegidos nunca (los que tengan menor puntuación o mala suerte).

Debido a sus buenas propiedades teóricas, este es el muestreo que se utiliza por defecto con los Algoritmo Genéticos. No obstante. por diversos motivos sus prestaciones en la práctica no están siempre a la altura de las expectativas teóricas.

- ***Por restos:*** A cada individuo x_i se le asignan directamente ë$p_i * k$û puestos en la muestra. Seguidamente los individuos se reparten los puestos vacantes en función de sus puntuaciones. El reparto se suele hacer por sorteo y entonces se le dice "muestreo estocástico por restos". Para el ejemplo propuesto, a x_4 le corresponde un puesto en la muestra; el individuo que ocupe el otro puesto se elige por sorteo, se genera un número aleatorio simple $r = 0.39874$ y se compara con las puntuaciones acumuladas q_1, en este caso el puesto vacante le corresponde a x_2 dado que $qi < r < q_2$.

Existe una variante en la que el reparto de las vacantes se hace por muestreo directo; nótese que en esta variante no hay ninguna intervención del azar, o sea, no es un método de muestreo estocástico sino directo, por este motivo se le llama "muestreo determinista por restos", para distinguirlo del descrito anteriormente.

- ***Universal o por ruleta:*** Es análogo al muestreo por sorteo sólo que ahora se genera un único número aleatorio simple *r* y con él se asignan todas las muestras de modo parecido a como se haría al girar una ruleta.

Para simular una tirada de ruleta se definen a partir de $r \neg \text{Rand}[0,1)$ los suficientes números:

$r_1 := (r + j - 1) / k$ ($«_j = 1,..., k$)

Con eso ya se determina la muestra comparando los r_j con las puntuaciones acumuladas a la manera habitual.

Para el ejemplo propuesto, si el número aleatorio es $r = 0.39874$ entonces se tiene:

$r_1 = 0.19937$ $r_2 = 0.69937$

lo que proporciona la muestra $\{x_2, x_4\}$.

Como puede verse, la implantación del muestreo estocástico universal es muy sencilla y rápida, y en la práctica proporciona unas prestaciones análogas a las del muestreo por sorteo; por este motivo se suele usar en sustitución de éste.

- ***Por torneos:*** Cada elemento de la muestra se toma eligiendo el mejor de los individuos de un conjunto de z elementos tomados al azar de la población base: esto se repite k veces hasta completar la muestra. El parámetro z suele ser un entero pequeño comparado con el tamaño de la población base, normalmente 2 ó 3. Nótese que en este caso no es necesario hacer la asignación de puntuaciones.

Todos los mecanismos de muestreo vistos admiten algunas variantes no necesariamente excluyentes; las más empleadas al trabajar con Algoritmos Genéticos son estas tres:

1. ***muestreo diferenciado:*** Cada elemento de la población base se puede tomar para formar la muestra a lo sumo una vez.
2. ***muestreo conservador:*** Todos los elementos de la población base tienen alguna oportunidad de ser elegidos. También se le conoce como "muestreo duro".
3. ***muestreo excluyente:*** Se excluyen a priori algunos individuos del proceso de muestreo. También se le llama "muestreo extintivo".

De esta manera se habla, por ejemplo, de que el proceso de selección de criadores de cierto Algoritmo Genético se ha implantado a través de un "muestreo estocástico por torneos de tamaño 2 en la variedad conservadora".

2.5.5 Procedimientos básicos de un Algoritmo Genético

Una vez vistos los principales mecanismos de muestreo, se pueden describir los tres procedimientos básicos de que consta un Algoritmo Genético, a saber: selección (de criadores). reproducción (o transformación) y reemplazo (o selección de supervivientes). Nótese que para implantar un Algoritmo Genético se requiere un mínimo de dos parámetros: el tamaño de la población *n* y el tamaño de la descendencia *s.*

- **El proceso de selección:** Consiste en muestrear, a partir de la población inicial. los *n* elementos de la población de criadores. El criterio concreto de muestreo depende del problema y del buen juicio del programador. Los más usados en la práctica son los muestreos por sorteo, por ruleta y por torneos, en sus variedades conservadoras. Los muestreos deterministas se usan muy poco, entre otros motivos porque van contra la filosofía del método.
- **El proceso de reproducción:** Aplicando los operadores de transformación (cruce, mutación, inversión,...) sobre ciertos miembros de la población de criadores se obtiene una descendencia de *s* nuevos miembros. La determinación exacta de qué miembros se van a reproducir (los progenitores) se suele realizar mediante ensayos de Bernuolli; se admite a veces la variedad excluyente. No es necesario (de hecho, no es habitual) dar por adelantado el valor de s, normalmente sólo se da su valor esperado de modo indirecto a través de las probabilidades de aplicación de los operadores genéticos. Cuanto más grande sea el valor (real o esperado) de *s* más variará la población de una generación a otra. Salvo indicación en contra. lo común es trabajar con valores de *s* no mayores del 60 % de *n.*

Más adelante se verán los diversos operadores genéticos que se usan con los Algoritmo Genéticos. Ahora bien. existen dos grupos de operadores que nunca faltan en un algoritmo genético: el cruce y la mutación.

Los operadores de cruce son el arquetipo de operadores de *recombinación:* actúan sobre parejas de individuos y normalmente originan otro par de individuos que combinan características de los progenitores. Dado que en los Algoritmos Genéticos los individuos están representados a través de cadenas binarias, el cruce se lleva a cabo por intercambio de segmentos.

Cruce

Los operadores de mutación, por su parte, son el arquetipo de operadores de *alteración* dado que actúan sobre individuos solos, realizando una pequeña modificación alguno de sus genes o en el conjunto.

El motivo de hacer esta separación es el siguiente: entendiendo que la búsqueda propiamente dicha se lleva a cabo en la fase de reproducción, resulta conveniente diferenciar los operadores de búsqueda en profundidad de los de búsqueda en anchura; los primeros se encargarán de explotar las mejores características de que disponga la población actual, los otros se encargarán de explorar nuevos dominios en busca de mejores soluciones. Desde esta perspectiva, los operadores de cruce son los que principalmente se encargan de la búsqueda en profundidad y los de mutación de la búsqueda en anchura. Así, dando mayor importancia a unos o a otros se puede ajustar el tipo de búsqueda a las necesidades del problema. Sin necesidad de entrar de momento en mayores precisiones es evidente que la división tiene un valor práctico para diferenciar dos modos complementarios de implantar la reproducción: el cruce se encarga de realizar un intercambio estructurado de información, la mutación proporciona una garantía de accesibilidad para todos los puntos del espacio de búsqueda.

Existe una variante del proceso de reproducción que recibe el nombre de *reproducción operacional;* en ella cada miembro de la descendencia es producido por un solo operador genético y además éstos operadores sólo pueden actuar sobre los individuos, no sobre las poblaciones o sobre los genes.

El proceso de reemplazo: A partir de los *n* miembros de la población de criadores y de los *s* miembros de la población de descendientes se debe obtener una nueva población de *n* miembros. Para hacerlo existen varios criterios:

1. *Reemplazo inmediato (o al vuelo):* Los *s* descendientes sustituyen a sus respectivos progenitores.
2. *Reemplazo con factor de llenado:* Los *s* descendientes sustituyen a aquellos miembros de la población de criadores que más se les parezcan.
3. *Reemplazo por inserción:* Según el tamaño relativo de la descendencia respecto de la población se distinguen dos casos,

1. s <= n: Se muestrean para ser eliminados *s* miembros de la población de criadores (según cierto criterio; normalmente, los peores). Esos miembros serán sustituidos por los descendientes.

2. s > n: Se muestrean *n* miembros de la población de descendientes y se constituye con ellos la nueva población. Nótese que de este modo cualquier individuo sólo puede vivir a lo sumo una generación.

4. Reemplazo por inclusión (o de tipo 'más'): Se juntan los *s* descendientes con los *n* progenitores en una sola población, y en ella se muestrean *n* miembros (normalmente, los mejores).

Naturalmente. los dos primeros criterios sólo se pueden usar cuando *s* < *n.*

Sin hacer de esto una cuestión de principios, una buena elección de criterios para un algoritmo genético incorpora la selección mediante muestreo estocástico universal en la variedad conservadora; el reemplazo suele ser inmediato aunque es preferible el reemplazo por inserción con el mismo tipo de muestreo.

Considerando todo lo discutido en esta sección. se muestra a continuación un pseudocódigo con el que implantar un algoritmo genético:

```
BeginAlgorithm{GeneticAlgorithm}
P[0]   = InitPop();
FitP[0]  = EvalPop(P[0]);
for(t=0;t< MaxNumGen;  t++)
        {
        Q[t]  = SelectBreedersFrom(P[t]);          Población de criadores
        Q[t]  = Crossover(Q[t]);
        Q[t]   = Mutate(Q[t]) ;
        FitQ[t]  = EvalPop(Q[t]) ;
        P[t]  = SelSurviv(P[t], Q[t], FitP[t], FitQ[t]);
        FitP[t]  = EvalPop(P[t]);
        if( ChkTermCondP[t] , FitP[t]) )
                return;
        }
EndAlgorithm
```

2.5.6 El Algoritmo Genético Básico (SGA). Implantación y ejemplos.

En esta sección se estudiará una implantación elemental pero completa del algoritmo descrito anteriormente y se resolverá con ella un ejemplo práctico de optimización paramétrica multimodal. Se trata, por un lado, de aclarar y concretar todo lo discutido en la sección anterior referido a los métodos, y por el otro lado facilitar la comprensión del estudio teórico de la siguiente sección.

Dicha implantación fue propuesta por Goldberg [1], y es la que, con cambios menores, se usa habitualmente para ilustrar el funcionamiento de los Algoritmos Genéticos.

El SGA *(simple genetic algorithm)* es una algoritmo genético que incorpora los siguientes métodos y criterios:

1. *Criterio de codificación:* Específico de cada problema. Debe hacer corresponder a cada punto del dominio del problema un elemento del espacio de búsqueda. el cual necesariamente consta de enteros binarios sin signo, esto es, cadenas binarias simples.
2. *Criterio de tratamiento de los individuos no factibles:* No hay. Se considera que la codificación se hace de tal manera que todas las cadenas posibles representan a individuos factibles.
3. *Criterio de inicialización:* La población inicial consta de cadenas binarias elegidas al azar.
4. *Funciones de evaluación y aptitud:* La función de aptitud coincide con la de evaluación. Preferiblemente la función de evaluación viene dada a través de una función objetivo.
5. *Operadores genéticos:* Cruce monopunto y mutación bit a bit sobre los individuos codificados.
6. *Criterio de selección:* Por sorteo.
7. *Criterio de reemplazo:* Inmediato.
8. *Criterio de parada:* Fijando el número máximo de iteraciones.
9. *Parámetros de funcionamiento:* Discrecionales. Una elección típica es:

PopSize = 30 *MaxIter* = 50 p_{xov} = 60% p_{mut} = 3.33%

De todas esas especificaciones las únicas que precisan explicación son las referidas a los operadores genéticos. El *cruce monopunto* genera dos descendientes a partir de dos progenitores cortándolos en una posición elegida al azar e intercambiando los respectivos segmentos. Es decir. dados dos progenitores codificados como v = $(b_1 ... b_t)$ y w = $(c_1 ... c_t)$ con b_j. c_j e 0,1 (“j = 1, ...,k) se genera un número aleatorio r ¬ RandInt[1, k - 1] y se procede así:

$(b_1,...,b_{pos-1} \mid b_{pos},...,b_t)$ $(b_1,...,b_{pos-1} \mid c_{pos},...,c_t)$

$(c_1,...,c_{pos-1} \mid c_{pos},...,c_t)$ $(c_1,...,c_{pos-1} \mid b_{pos},...,b_t)$

Cada aplicación de la *mutación bit a bit* conmuta un bit de entre todos los *(n* x ?) de la población.

Como ya se comentó, la determinación exacta de qué individuos se van a cruzar y qué bits se van mutar se lleva a cabo mediante ensayos de Bernuolli. Para el cruce el proceso es así:

1. Para cada individuo se genera un número aleatorio $r_i \neg Rand[0,1)$.
2. Se comparan los r_i con la probabilidad de cruce, p_{vxov}, que es un parámetro del método.
3. Son seleccionados para cruzarse todos los individuos v_i para los que $r_i < p_{xov}$.

Para la mutación se realiza el mismo proceso con cada uno de los n x k bits de la población y usando la probabilidad de mutación p_{mut}

El emparejamiento de los individuos a cruzar se hace al azar; habitualmente, las etapas anteriores al cruce han introducido el suficiente desorden en la población como para que un emparejamiento sucesivo se pueda considerar aleatorio. Todavía queda un pequeño problema que resolver para implantar completamente el cruce: ¿Qué hacer cuando el número de individuos a cruzar sea impar? En tal caso, el último individuo quedará desemparejado y lo que se hace es bien eliminarlo o bien emparejarlo con cualquier otro elegido al azar.

Un Algoritmo Genético Básico [3]

La mecánica de un Algoritmo Genético Básico es sorprendentemente simple, involucra nada más complejo que copiar individuos[1] y alternar individuos parcialmente. La simplicidad de operación poder de efecto son las dos principales atracciones de los Algoritmos Genéticos.

Supongamos que queremos maximizar la función $f(x) = x^2$ en el intervalo [0,31], y tenemos una población de cuatro individuos, cuyos bits fueron elegidos al azar:

01101
11000
01000
10011

Un Algoritmo Genético Básico que cosecha buenos resultados, de los que se espera mejores poblaciones sucesivas, está compuesto por tres operadores:

1. Reproducción
2. Cruce
3. Mutación

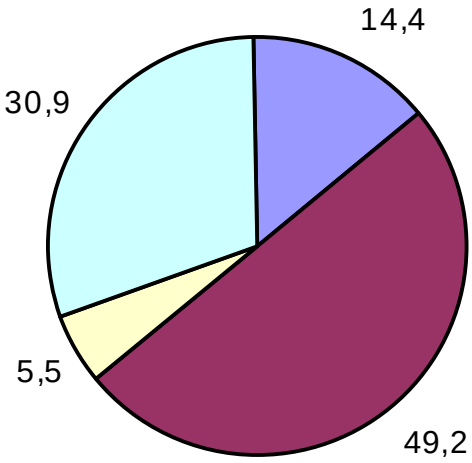
La reproducción es el proceso en el que cada individuo es copiado de acuerdo al valor de su función objetivo[2], la cual es una medida de beneficio, utilidad, aptitud o bondad que pretendemos maximizar. Copiar individuos de acuerdo a sus valores de aptitud[3] significa que los individuos con mayor valor tendrán mayor probabilidad de contribuir en la descendencia de la siguiente generación. En poblaciones naturales la aptitud está determinada por la habilidad de la criatura para sobrevivir a depredadores, pestes y otros obstáculos hacia la adultez y posterior reproducción.

El operador reproducción puede ser implementado en forma algorítmica de varias maneras. Tal vez la más sencilla sea a través de la "ruleta pesada", que fue explicada anteriormente.

Número	String	Fitness	% del Total
1	01101	169	14.4
2	11000	576	49.2
3	01000	64	5.5
4	10011	361	30.9
Total		**1170**	**100.0**

El porcentaje de aptitud de cada individuo de la población está representado en la siguiente figura:

1. Cadena de caracteres.
2. Fitness function.
3. Fitness value.



Para reproducir simplemente hacemos girar la ruleta cuatro veces. Cada vez que un individuo es seleccionado para reproducción, una réplica exacta de éste se realiza. Luego se lo ingresa en un lote al que se le aplicarán otras operaciones genéticas.

Luego de la reproducción, el cruce[4] simple puede proceder en dos pasos. Primero, los miembros de los individuos nuevos reproducidos son ubicados en pareja aleatoriamente. Segundo, cada par experimenta el cruce de la siguiente manera: se elige una posición *k* en un rango entre *"1"* y la *"longitud del* individuo *– 1"*. Dos nuevos individuos se crean alternando todos los caracteres entre las posiciones *k + 1* hasta el final del individuos inclusive.

Por ejemplo consideremos los individuos A_1 y A_2 de nuestra población inicial:

A_1 = 0 1 1 0 | 1
A_2 = 1 1 0 0 | 0

Supongamos que escogemos un número aleatorio entre 1 y 4, obteniendo *k = 4* (como indica el separador '|'). El cruce resultante genera los dos nuevos individuos:

A_1' = 0 1 1 0 0
A_2' = 1 1 0 0 1

La mecánica de la reproducción y el cruce son sorprendentemente simples, involucran la generación automática de números, copias de individuos, e intercambio parcial de individuos. Sin embargo, la combinación de la reproducción y el estructurado, aunque aleatorizado, intercambio de información del cruce le da a los Algoritmos Genéticos mucho de su poder. A partir de esto hay mucha confusión acerca del rol de la mutación en la genética (tanto natural como artificial), pero más allá de la confusión encontramos que la mutación juega un rol secundario en los Algoritmos Genéticos. La mutación es necesaria porque, aunque la reproducción y el cruce efectivamente buscan y recombinan conceptos, ocasionalmente pueden tornarse repetitivos y perder material genético útil potencialmente. En sistemas genéticos artificiales, el operador mutación protege sobre semejante pérdida irrecuperable. En un algoritmo genético simple, la mutación es la alteración ocasional (con baja probabilidad) de un valor del individuo. Mientras sea usado con moderación junto con la reproducción y el cruce, es un seguro contra la pérdida prematura de criterios importantes.

El operador mutación juega un rol secundario en los Algoritmos Genéticos, la frecuencia de mutación para obtener buenos resultados, encontrada en estudios empíricos realizados, está e2n el orden de una mutación por cada mil bits. Las tasas de mutación son también pequeñas en las poblaciones naturales, llevándonos a concluir que la mutación es apropiadamente considerada como un mecanismo secundario de la adaptación de los Algoritmos Genéticos.

Continuando con nuestro ejemplo, utilizamos individuos de cinco bits para obtener número de nuestro intervalo entre 0 (00000) y 31 (11111). Con la función objetivo ya definida al igual que la codificación, ya estamos en condiciones de simular una generación de un algoritmo genético con reproducción, cruce y mutación. Ya habíamos obtenido una población inicial de cuatro individuos al azar.

La generación del algoritmo genético comienza con la reproducción. Elegimos la descendencia haciendo girar la ruleta pesada cuatro veces.

4. Crossover

String Nro.	Población Inicial (generada aleatoriamente)	Valor (en decimal)	$f(x) = x^2$	Probabilidad de elección ***(fi/Sf)***	Cantidad de copias esperadas ***(fi/f)***	Cantidad de copias (reproducción)
1	01101	13	169	0.14	0.58	1
2	11000	24	576	0.49	1.97	2
3	01000	8	64	0.06	0.22	0
4	10011	19	361	0.31	1.23	1
Sum			*1170*	*1.00*	*4.00*	*4.0*
Promedio			*293*	*0.25*	*1.00*	*1.0*
Máx			*576*	*0.49*	*1.97*	*2.0*

La simulación de este proceso resulta en los individuos 1 y 4 recibiendo una copia cada uno, y el 2 recibiendo dos copias, el 3 no recibe copias. Comparando la cantidad de copias obtenidas con las esperadas nos damos cuenta de que en este caso los mejores obtuvieron más copias, los que tienen una aptitud promedio quedaron igual y los peores no obtuvieron copias.

Ahora, ya con un lote de individuos se deben formar parejas y proceder con el cruce, esto se realiza en los dos siguientes pasos:

1. Se juntan individuos en parejas aleatoriamente.
2. Las parejas se cruzan, seleccionando aleatoriamente el sitio a partir del cual cruzarlas.

Aleatoriamente que se juntaron los individuos 1 y 2, y 3 y 4. Para la primera pareja se cruzan a partir de la posición 4 y para la segunda en la posición 2. Esto se refleja en la siguiente tabla. El último operador, la mutación, se realiza bit por bit. Asumimos una probabilidad de mutación de 0.001. Con veinte bits esperaríamos 20 * 0.001 = 0.02 bits que sufrirán mutación en una generación determinada. La simulación de este proceso indica que a ningún bit le afecta la mutación para esta probabilidad.

Luego de la reproducción, cruce y mutación, la nueva población está lista para ser evaluada. Para hacer esto, simplemente decodificamos los nuevos individuos creados por el algoritmo genético simple y calcular la función aptitud para cada valor decodificado.

Nótese en la tabla que el máximo y el promedio han mejorado en esta nueva población. El promedio de aptitud mejoró de 293 a 439 en una generación. La aptitud máxima subió de 576 a 729 durante ese mismo período. Así comenzamos a ver que estas mejoras no son un golpe de suerte. El mejor individuo de la población inicial recibe dos copias por su gran aptitud. Cuando este combina aleatoriamente con el siguiente individuo en aptitud, uno de los individuos resultantes demuestra tener realmente una elevada aptitud. De esta manera vemos cómo los Algoritmos Genéticos generan un búsqueda robusta.

Lote luego de reproducción	**Pareja**	**Posición para crossover**	**Nueva Población**	**Valor Decimal**	**$f(x) = x^2$**
01101	2	4	01100	12	144
11000	1	4	11001	25	625
11000	4	2	11011	27	729
10011	3	2	10000	16	256
Sum					***1754***
Promedio					***439***
Máx					***729***

2.6 Dificultades en el uso de los Algoritmos Genéticos. Planteamiento y resolución.

A la hora de llevar a la práctica el modelo de los Algoritmos Genéticos para resolver problemas no triviales de optimización se plantean varias dificultades de orden teórico y práctico. Todas ellas surgen como consecuencia de cuatro limitaciones intrínsecas de dicho modelo. Por eso, para corregir o mitigar sus efectos conviene estudiar antes con cierto detenimiento las implicaciones generales de tales limitaciones. Sólo entonces se podrá acometer el diseño de procedimientos eficaces de resolución.

2.6.1 Limitaciones de los Algoritmos Genéticos e inconvenientes asociados

De modo general, las cuatro limitaciones principales del Algoritmo Genético Básico (SGA) son:

1. *La representación:* El espacio de búsqueda sólo puede ser representado mediante cadenas binarias.
2. *La irrestricción:* El mecanismo del SGA no considera las posibles de restricciones o ligaduras que pudieran imponerse a la búsqueda.
3. *La opacidad:* El SGA es un algoritmo de búsqueda ciega, esto es, sólo está guiado por la aptitud de los individuos, y no incorpora ningún otro conocimiento específico del problema en cuestión.
4. *La finitud:* A efectos prácticos, el SGA sólo puede trabajar con poblaciones finitas no muy grandes.

Los inconvenientes que ocasionan las dos primeras limitaciones no se salen de lo previsible: por contra. la opacidad y la finitud son causa original de otros muchos

inconvenientes que habrá que atacar por separado. De manera esquemática, pude decirse que todos esos inconvenientes tienen su base en dos hechos:

1. Fundamentalmente, la opacidad es causa inevitable de *debilidad.*
2. Por otra parte, la opacidad acentúa fuertemente los *problemas de diversidad* derivados de la finitud de la población.

Seguidamente se estudiarán más detenidamente sus consecuencias.

2.6.2 El problema de la debilidad de los Algoritmos Genéticos

El inconveniente más visible de la debilidad es la *ineficiencia:* los métodos débiles de resolución de problemas presentan las ventajas de ser muy poco específicos, poco exigentes y notablemente eficaces; sin embargo. todos ellos son ineficientes en mayor o menor grado, y especialmente si se les compara con métodos heurísticos. Los Algoritmos Genéticos sólo se pueden considerar eficientes en comparación con otros métodos estocásticos de búsqueda ciega, pero se puede garantizar que si se encuentra un método heurístico para resolver un problema este siempre lo hará mucho más eficientemente que un Algoritmos Genético.

Pero eso no es todo: otro de los inconvenientes prácticos de la debilidad es la tendencia al *extravío* de la búsqueda. El fenómeno es así: el SGA realiza la búsqueda de los mejores puntos utilizando únicamente la aptitud de los individuos para recombinar internamente los bloques constructivos; en ocasiones ocurre que la información proporcionada (la aptitud) resulta insuficiente para orientar correctamente al algoritmo en su búsqueda del óptimo. A esto se le llama *desorientación*[5]*.* A nivel interno esto se concreta en que no se verifica la hipótesis de los bloques constructivos, es decir, ciertas combinaciones válidas de buenos bloques constructivos originan individuos de baja aptitud. En el peor de los casos puede ocurrir que este fenómeno tenga tanta fuerza como para impedir que el Algoritmos Genético converja al óptimo global, desviándolo finalmente hacia un óptimo local. Esto es el extravío propiamente dicho. Afortunadamente, aunque un Algoritmos Genético se desoriente no necesariamente se va a extraviar; para eso es necesario partir de una mala población inicial o plantear un problema especialmente diseñado para que se extravíe.

En resumen. para que funcione correctamente el mecanismo básico de los Algoritmos Genéticos es necesario proporcionarle una mínima cantidad (y calidad) de información. Si no se le proporciona ese mínimo el mecanismo no funciona con propiedad, y el Algoritmo Genético evolucionará incorrectamente.

Como es de suponer, el factor que más contribuye a la existencia de desorientación es la forma de la función de aptitud. Típicamente, la desorientación es frecuente en problemas en los que los puntos óptimos están aislados y rodeados de puntos de muy baja aptitud. Sin embargo, conviene señalar que la presencia de desorientación suele estar fuertemente estimulada por una mala codificación que "oculte" la ya de por sí escasa información disponible. A nivel interno esto se explica por la mayor dificultad para constituir los bloques constructivos. De modo recíproco, una buena codificación reduce la probabilidad de extravío.

Un caso especialmente desfavorable ocurre cuando hay una fuerte interacción entre dos o más atributos (genes), de tal forma que la contribución a la aptitud de un individuo que realiza cierto gen depende grandemente de los valores que tomen otros. A este fenómeno se le dice *epistasis o acoplamiento y* su presencia garantiza la desorientación dado que en esas condiciones resulta muy difícil constituir buenos bloques constructivos.

Más adelante se verán algunas maneras de atenuar los efectos de los inconvenientes asociados a la debilidad. De todos modos, hay ocasiones en que estos remedios resultan insuficientes y no cabe más solución que eliminar la debilidad incorporando conocimiento específico al Algoritmos Genético.

2.6.3 El problema de la diversidad en los Algoritmos Genéticos.

A efectos prácticos. es fundamental que todo Algoritmos Genético verifique lo siguiente:

5. En la bibliografía se le hacer referencia como "deception".

Requisito de variedad en los Algoritmos Genéticos: Es esencial para el buen funcionamiento de un Algoritmos Genético tener controlada en todo momento la diversidad de la población. Se entiende la diversidad en sentido general como "diversidad de individuos" y en particular como "diversidad de aptitudes".

La necesidad de que haya diversidad de individuos es fácil de entender: con poca variedad de individuos hay poca variedad de esquemas, a causa de ello el operador de cruce pierde casi por completo la capacidad de intercambio de información útil entre individuos, y en definitiva, la búsqueda se estanca. La necesidad de tener controlada la diversidad de aptitudes radica en la imposibilidad práctica de trabajar con una población infinita, como enseguida se verá.

Es fácil darse cuenta de que no es conveniente tener poca diversidad de aptitudes ya que en tal caso todos los individuos tendrían más o menos las mismas posibilidades de sobrevivir, la selección reproduciría la situación anterior y todo el peso de la búsqueda recaería en los operadores genéticos, lo que a la larga sería poco más que una búsqueda aleatoria. El resultado final es que la búsqueda se estanca y, como luego se verá, la situación puede empeorar si la población es finita. En definitiva. para que la selección sea efectiva, la población debe contener en todo momento una cierta variedad de aptitudes.

Por otra parte. cuando la población es finita —es decir, siempre— tampoco se puede tener gran disparidad de aptitudes, pues ello suele afectar muy negativamente a la diversidad de la población, como seguidamente se explica.

2.6.4 Convergencia prematura por problemas con la diversidad.

Sea un Algoritmo Genético básico con población finita *y* admítase por simplicidad que la función de aptitud coincide con la función de evaluación. En algún momento puede ocurrir que un individuo o un grupo de ellos obtengan una aptitud notablemente superior a los demás. Esto es especialmente probable en las fases tempranas de la evolución, en las cuales de entre una población de individuos mediocres suele surgir un buen candidato por aplicación de los operadores genéticos. En tal circunstancia existe el riesgo de que se produzca una *evolución en avalancha:* al incrementar los individuos más aptos su presencia en la población, por ser ésta finita la diversidad disminuye, ello hace que en la siguiente generación se favorezca aún más a los individuos más aptos hasta que éstos acaban dominando por completo la población. A esto se le conoce como el fenómeno de los *superindividuos.* Habitualmente ocurre que tales superindividuos sólo son los *más* aptos en cierto momento, pero no los más aptos absolutos —téngase en cuenta que la falta de diversidad estanca la búsqueda— lo que en último término provoca una *convergencia prematura* del Algoritmo Genético, habitualmente hacia un subóptimo. La única circunstancia en que este fenómeno es tolerable e incluso deseable es en las fases tardías de la evolución, cuando el Algoritmo Genético ha "localizado" correctamente la solución óptima, pero nunca antes.

La posibilidad de convergencia en avalancha es un fenómeno consustancial a los Algoritmos Genéticos con población finita. Tanto es así que dicho fenómeno puede ocurrir incluso cuando no haya diferencias apreciables en las aptitudes de la población. Ya se comentó que en tal circunstancia la búsqueda se estanca, dado que la selección no concede mayor preeminencia a uno u otro individuo. Sin embargo, cuando la población es finita y especialmente cuando no es muy grande, puede ocurrir el siguiente fenómeno, conocido como *deriva genética:* En una población finita los procesos de muestreo son siempre imperfectos. pudiendo favorecer más de lo que les corresponde a individuos ocasionalmente más aptos; debido a la finitud y como consecuencia de esos errores estocásticos en los muestreos, un individuo puede acabar "expulsando" de la población a los restantes, haciendo que el Algoritmo Genético converja prematuramente hacia tal "individuo afortunado".

Dado que en una población finita son inevitables los ruidos estocásticos introducidos por el muestro, el peligro de deriva genética está siempre presente; en la práctica sólo es importante cuando las aptitudes son todas parecidas (poca diversidad de aptitudes) o con pequeños tamaños de población.

En resumen, es imprescindible tener control sobre la diversidad de aptitudes de la población (finita) para evitar que se produzca una convergencia prematura (avalancha) ya sea por mucha diversidad (superindividuos) o incluso por demasiado poca (deriva genética).

2.7 Resolución de inconvenientes, mejoras y alternativas

Se van a describir algunas modificaciones del modelo básico de Algoritmos Genéticos. Se mantendrá punto por punto la estructura fundamental, introduciendo simplemente variantes y parametrizaciones en las configuraciones por defecto de los nueve métodos y procedimientos indicados anteriormente.

2.7.1 El problema de la representación y la codificación

Con la idea de hacer la búsqueda más eficiente, se representan los puntos del espacio de búsqueda mediante cadenas binarias pues esta es la representación que proporciona la mayor variedad de esquemas

para una longitud dada. Esto plantea graves inconvenientes que se resumen en lo siguiente: la representación binaria no resulta *útil* para la mayoría de problemas de búsqueda en el sentido de que no siempre es posible encontrar una codificación que dé significado a cada una de las posiciones de que consta el genotipo; es más, aunque se encuentre, eso no garantiza que represente con fidelidad el dominio del problema. Ambas cuestiones se estudiarán por separado.

Comenzando por la segunda cuestión —la representación con fidelidad del dominio del problema—, es conveniente introducir la siguiente definición: Se dice que una representación es perfecta desde el punto de vista de los objetos representados cuando cumple estas cinco propiedades:

1. *Completitud:* Debe poder representar todos los objetos de la clase indicada.
2. *Coherencia:* Únicamente debe representar objetos de esa clase.
3. *Uniformidad:* Todos los objetos deben estar representados por la misma cantidad de codificaciones, no debe haber objetos 'sobre representados' ni 'infra representados'.
4. *Sencillez:* Debe ser fácil de aplicar el mecanismo de codificación individuo tanto en sentido directo como inverso.
5. *Localidad:* Pequeños cambios en los individuos se han de corresponder con pequeños cambios en los objetos.

En la práctica suele ser imposible encontrar una *representación perfecta* del dominio del problema a través de cadenas binarias, lo cual introduce una diferencia más o menos grande entre lo que se está buscando y lo que se quiere buscar.

La otra cuestión, la de la utilidad de la representación, no es trivial: es muy importante que las *Len* posiciones de que consta el fenotipo tengan algún significado concreto; de este modo, si se ordenan apropiadamente dichas posiciones, los bloques constructivos serán cortos *y* compactos, lo que dará eficacia a la búsqueda genética. De hecho, es tan necesario que las regiones de gran aptitud se representen mediante esquemas cortos *y* compactos que en ocasiones se sacrifica el requisito de representación binaria si ello conlleva mayor naturalidad. Nótese que ésta es la única vía por la que se puede introducir conocimiento específico en un Algoritmo Genético. En la práctica, la ordenación *más* común consiste en agrupar las posiciones por atributos (genes) y concatenar los grupos a la manera que se indicó anteriormente.

Toda esta discusión se sintetiza en la siguiente recomendación, se aconseja siempre realizar la elección de la representación basándose en estos dos principios:

1. *Principio de los bloques constructivos con significado:* Las zonas del espacio de búsqueda relevantes para el problema deben poderse representar mediante esquemas cortos y de bajo orden (bloques constructivos).
2. *Principio del alfabeto de símbolos mínimo:* Se debe elegir el alfabeto de símbolos más pequeño (preferentemente binario) que permita una expresión natural del problema.

2.7.2 Elección de la población inicial

En términos generales, la población inicial debe ser lo más variada posible. Es necesario que la distribución de aptitudes sea uniforme para evitar la convergencia prematura, y también es necesario que haya variedad de esquemas. En la práctica, a falta de más información se elige la población inicial al azar. Una vez más, el conocimiento específico puede ayudar para elegir una población inicial factible y/o próxima al óptimo.

2.7.3 Convergencia en los Algoritmos Genéticos y criterios de terminación.

Hasta el momento se ha aceptado por defecto como criterio determinación de los Algoritmos Genéticos el del máximo número de iteraciones. Se admite que, dada la hipótesis de la construcción por bloques y una adecuada población inicial, los mejores individuos de la población se acercarán cada vez más al óptimo. Es decir, se asume implícitamente que al llegar a cierto valor de "t" el algoritmo ha convergido a efectos prácticos. Esta manera de determinar la convergencia es excesivamente arbitraria en muchos casos, lo que lleva a plantear otros criterios de terminación.

Tradicionalmente se distinguen dos tipos de criterios de terminación:

- *Criterios de terminación enfocados al costo:* Abreviadamente, criterios de 'tipo MAX'. Son los que limitan a priori el número máximo de iteraciones *(MaxIter)* o el máximo número de evaluaciones *(MaxNrEval,* habitualmente usados cuando la función de evaluación es complicada). Son los más sencillos, pero no siempre tienen significado, ya que normalmente no hay ninguna razón para dar a dichos números máximos uno u otro valor. En tal circunstancia o bien ocurrirá que el Algoritmo Genético "se quede corto", esto es, no esté al terminar lo suficientemente cerca del óptimo, o bien consumirá recursos computacionales inútilmente estando ya lo suficientemente cerca del óptimo.
- *Criterios de terminación enfocados a la calidad:* También se les conoce como *criterios incrementales de terminación* o, abreviadamente, criterios de "tipo TOL". Se hace parar al algoritmo una vez que se hayan

superado ciertos requisitos de convergencia. independientemente de la iteración en que se encuentre. Dependiendo del criterio práctico que se utilice para evaluar la convergencia se subdividen en dos tipos:

- *Enfocados a la estructura:* Se evalúa la convergencia de la población verificando la cantidad de genes que han convergido. Se considera que un gen del genotipo ha convergido cuando cierto porcentaje de la población (alrededor del 90%) tiene los alelos iguales en dicho gen (o parecidos. si la representación no es binaria). La búsqueda termina cuando el número de genes que han convergido excede cierto porcentaje (alrededor del 95%) del total de genes de que consta el genotipo.
- *Enfocados al contenido:* Miden el progreso hecho por el algoritmo en cierto número de iteraciones; si es menor que cierta tolerancia *TOL,* que es un parámetro del método, la búsqueda ha terminado. Habitualmente se mide el "progreso hecho por el algoritmo" como el incremento relativo de la aptitud total de la población; también se puede usar la aptitud máxima o la mínima, según sea el problema.

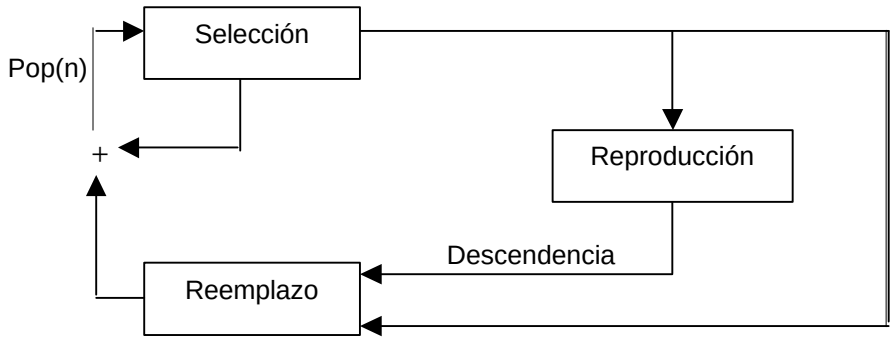
La elección de unos valores u otros para las tolerancias es delicada: no sólo depende del problema, sino que también influye la suerte que se haya tenido en la ejecución del Algoritmo Genético (recuérdese que son métodos estocásticos). Además, una pequeña reducción de las tolerancias permitidas puede conllevar un tiempo de ejecución mucho más grande. Habitualmente, se usa un criterio de terminación híbrido que consta de una tolerancia y un mecanismo de seguridad del tipo MAX para cuando se alargue demasiado la búsqueda.

Elitismo

El método más utilizado para mejorar la convergencia de los Algoritmos Genéticos es el *elitismo.* Consiste básicamente en realizar la etapa de selección en dos partes:

1. Se muestrea una elite de *r* miembros de entre los mejores de la población inicial y se incorporan directamente a la población final, sin pasar por la población intermedia.
2. La población auxiliar de criadores se muestrea de entre los *n - r* restantes miembros de la población inicial.

Comúnmente el tamaño de la elite r es bastante pequeño (1 ó 2 para n = 50), y el tipo muestreo es bien directo o bien por sorteo, ambos en la variedad diversa. La siguiente figura muestra cómo se modifica el bucle básico del funcionamiento de un Algoritmo Genético al introducir ese mecanismo y el código a continuación muestra la implantación genérica de un algoritmo de ese tipo.




```
BeginAlgorithm{ElitistGeneticAlgoritm}
P[0]  =  InitPopO ;
FitP[0]  = EvalPop(P[0]);
for( t = 0;  t < MaxNumGen;  t++){
        Q[t]  = SelectBreedersFrom(P[t]);
        R[t]  = SelectEliteFrom(P[t]);
        Q[t]  = Crossover(Q[t]);
        Q[t]  = Mutate(Q[t]);
        FitQ[t]  = EvalPop(Q[t]);
        P[t]  = SelSurv(P[t] ,  Q[t],  FitP[t] ,  FitQCt]);
        P [t]  = AddElite (P [t] , R [t] );
        FitP[t]  =  EvalPop(P[t]);
        if( Chi:TermCond(P[t] ,  FitP[t])  )
                return;
}
EndAlgorithm
```

Bajo ciertas condiciones muy generales. la introducción del elitismo garantiza la convergencia teórica al óptimo global; en la práctica, mejora la velocidad de convergencia de los Algoritmos Genéticos cuando la función de evaluación es unimodal —esto es, no existen subóptimos—, sin embargo la velocidad de convergencia empeora con funciones fuertemente multimodales.

Con tamaños de población pequeños se consiguen efectos similares a los del elitismo introduciendo *reinicializaciones* periódicas en los Algoritmos Genéticos: cada vez que el Algoritmo Genético converge se salvan los mejores individuos, se reinicializan los demás y se vuelve a comenzar. La reinicialización tiene efectos beneficiosos sobre las prestaciones del método debido a que introduce diversidad, requisito especialmente crítico en los Algoritmos Genéticos con poblaciones pequeñas.

2.8 Parametrización de los Algoritmos Genéticos

Para que un algoritmo genético pueda funcionar se deben dar valores apropiados a un conjunto de parámetros de entre los que destacan el tamaño de la población *(PopSize),* la longitud de los individuos *(Len),* las probabilidades de mutación y cruce *(p_{mut}* y *P_{xox})*, el máximo número de iteraciones *(MaxIter),* la precisión *(TOL) y* otros específicos de cada método como, por ejemplo. los coeficientes de la función de penalización o de la función de asignación de aptitudes.

No existen criterios definitivos para dar valores a esos parámetros. por lo que esta tarea depende en gran medida del buen criterio del programador. En principio; los Algoritmos Genéticos están reputados como técnicas de búsqueda *paramétricamente robustas,* es decir, funcionan —mejor o peor, pero funcionan— en un amplio rango de valores de sus parámetros. Sin embargo. las diferencias de eficiencia en la práctica son lo suficientemente espectaculares como para solicitar una mayor atención en dicha elección.

Se ha realizado una gran cantidad de pruebas [2] sobre una amplia variedad de problemas de búsqueda *y* optimización con la intención de determinar rangos de buen funcionamiento para los parámetros del SGA. Aunque los resultados,—medidos en términos de prestaciones— son dependientes en mayor o menor medida del problema en particular, se pueden extraer las siguientes conclusiones generales:

- El tamaño de la población *PopSize* varía habitualmente entre 50 y 100 individuos; valores menores suelen plantear graves problemas de convergencia prematura y valores mayores requieren un gran esfuerzo computacional sin obtener mejoras apreciables.
- El tamaño de los individuos *Len* suele venir dado como una consecuencia de los criterios de representación y codificación del problema. El único consejo que cabe dar es el de no alargar innecesariamente dicha cantidad salvo si se tiene la garantía de que al introducir redundancias se van a obtener prestaciones muy superiores.
- Las mejores prestaciones se obtienen en general con tasas de cruce que varían entre el 20% y el 60% y tasas de mutación entre el 0,1% Y el 5%, aunque no es raro ver otros valores. En lo que se refiere a otros operadores específicos, no existen criterios generales; tan sólo que los operadores de alteración (ej., la inversión, y en general todos los unitarios) se usan con mucha menor frecuencia que los de recombinación.
- El máximo número de iteraciones depende de la precisión especificada y varía mucho de un problema a otro; sirvan como valores orientativos *MaxIter* ~ 50 para problemas de evaluación compleja y *MaxIter* ~ 1000 para problemas de evaluación sencilla. En cuanto a la tolerancia de error se hace notar que está fuertemente condicionada por la representación elegida y por la "rugosidad" de la función de evaluación. Para problemas 'reales' de ingeniería se usan valores entre 10^{-2} y 10^{-5}.

2.9 Trabajos realizados en relación al tema.

En este apartado haremos un breve análisis de trabajos que tratan problemas dentro del área definida para este trabajo, destacando las diferencias y semejanzas con lo presentado aquí. Para el desarrollo de este tema se encontraron soluciones desarrolladas aplicando Algoritmos Genéticos y en un caso Redes Neuronales combinadas con Algoritmos Genéticos.

Genetic Algoritms for Adaptive Planning of Path and Trajectory of a Mobile Robot in 2D Terrains. [8]

En este paper encontramos probablemente lo más parecido a lo aquí planteado en cuanto al tema propuesto pero no tan así en cuanto a la solución propuesta. En él introduce este tema con el fin de planificar el movimiento de un robot submarino autónomo, utilizando una función de aptitud muy similar a la planteada en este trabajo. En ella se tiene en cuenta como función de aptitud el "peso" del individuo en cuestión, esto se refiere al esfuerzo de ir de un punto a otro, dado en este caso por la distancia u obstáculos parciales entre puntos.

Existen dos diferencias principales con el trabajo propuesto, en primer lugar se plantea una planificación de movimiento on-line, adaptativa, o sea que pueden realizarse modificaciones en el contexto mientras el

algoritmo evoluciona buscando la solución, desviando lo que el algoritmo estaba buscando inicialmente. Por supuesto que esto incluye la planificación de movimiento off-line, en donde no hay cambios en lo inicialmente planteado. En el caso de este trabajo eso no fue contemplado totalmente ya que se propone una serie de obstáculos fijos. De todos modos es parcial, ya que sólo podremos realizar cambios "on-line" respecto de los obstáculos parciales, estos son, el fuego y el huracán, pero no respecto de los casilleros bloqueados por completo.

La otra diferencia radica en la codificación del problema. Ellos plantean poblaciones representadas por individuos de longitud fija, donde éste está subdividido en tramos que en sus primeros genes indican la dirección y el resto la longitud a recorrer en esa dirección. En esta obra los individuos tendrán longitud variable, lo que nos obliga a personalizar los operadores genéticos, crossover y mutación, al caso particular. Uno de los principales motivos por el que se utilizan individuos de longitud variable en esta aplicación es la posibilidad que da para notar la evolución de ellos gráficamente, así se podrá observar cómo van creciendo en longitud buscando el mejor camino hacia la meta.

Cabe destacar nuevamente que al utilizar individuos de longitud variable se está obligado a personalizar los operadores genéticos. En el trabajo referido por el paper mencionado, se utiliza el método de cruzamiento y mutación estándar, esto es, la mutación aplicada a algún bit elegido al azar y el cruzamiento entre dos cadenas a partir de un punto determinado intercambiando sus partes derechas. El Algoritmo Genético fue implementado por ellos a partir de una herramienta estándar "GA Tookit", la cual ellos desarrollaron para el diseño y prototipos de Algoritmos Genéticos en la web. En una siguiente sección se explicará cómo se realizó la mutación y el cruzamiento en este trabajo.

A Genetic Algorithm for Robust Motion Planning. [6]

Ahora, analizando el trabajo realizado en la Universidad de Alicante nos encontramos con la búsqueda de un Algoritmo Genético para la planificación de movimiento robusta, también dirigidas a robots móviles. El término "robusto" se definió anteriormente. En esa obra se tienen en cuenta conceptos como la velocidad lineal y angular, como así también la noción de ruido, que en esta presentación no nos atañe, pero sí nos encontramos con que se utilizan individuos de longitud variable, con la consecuente personalización de los operadores genéticos, como se mencionó anteriormente.

La técnica utilizada en el cruzamiento presenta una fuerte similitud con la aplicación planteada en este trabajo. Al intercambiar la parte derecha de los individuos a partir de una posición determinada al azar se podrá conseguir un individuo mejor, incluso con acceso al destino, como así también un individuo que no sirve. La mutación tiene en cuenta la velocidad, cosa que no forma parte de los conceptos de este trabajo.

La característica variable de los individuos utilizados lleva a otra semejanza con el trabajo de esta obra. Esto se refiere a la representación de la función aptitud. Se tiene en cuenta la distancia Euclideana entre la posición final del robot y el destino, además de tener en cuenta el retraso generado por obstáculos.

Hybridisation of Neural Networks and Genetic Algorithms for On-line Trajectory Planning. [7]

Como lo indica el título, se realiza una hibridización entre Algoritmos Genéticos y Redes Neuronales para un sistema de control de trayectoria. Las Redes Neuronales son utilizadas como asistentes con el fin de minimizar error de rastreo. Por el contrario, los Algoritmos Genéticos son usados para la resolución de la planificación.

En este caso también se tienen en cuenta variables como tiempo, velocidad angular, momento de torsión, las cuales son tenidas en cuenta en la función de aptitud. Todo esto no tiene similitud con el trabajo aquí presentado. A diferencia con el trabajo planteado es que no se utilizan operadores genéticos personalizados, aunque sí se utiliza el elitismo a la hora de elegir la descendencia.

An Evolutionary Approach to Robot Structure and Trajectory Optimization. [9]

Este trabajo propone para obtener una forma adecuada para el brazo de un robot y también la trayectoria de su movimiento. La manipulación óptima será la que minimiza tanto la longitud de la trayectoria como las ondulación a lo largo del movimiento, sin colisiones con obstáculos en el espacio de trabajo.

Para la selección de progenitores y nueva población se utiliza el método de torneos [1], y los operadores genéticos no son personalizados a diferencia de lo planteado en este trabajo. Otras diferencias consisten en las variables que se tienen en cuenta, la función de aptitud está adaptada para este caso muy particular. Una semejanza es la elección de la población inicial que es al azar.

3. Definición del problema

El objetivo es encontrar una trayectoria adecuada para que un robot móvil y autónomo se traslade dentro de una plataforma, desde un punto a otro, conociendo el mapa y los obstáculos, totales y parciales presentes en él, desde antes de arrancar su marcha.

En el problema planteado se tiene una plataforma de desplazamiento en dos dimensiones, representada en una grilla de 21 x 24 casilleros, a través de la cual el robot debe desplazarse desde el extremo superior izquierdo (casillero 0) hasta el inferior derecho (casillero 527). Cada uno de los casilleros que componen la grilla puede estar libre para el tránsito, parcial o completamente bloqueado. Aquellos casilleros parcialmente bloqueados presentan obstáculos representados por fuego o huracanes, lo cual dificultará el paso del robot por ese lugar.

El robot tiene la capacidad de moverse en cuatro direcciones, hacia arriba, abajo, derecha o izquierda.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
0	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
1	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*	*43*	*44*	*45*	*46*	*47*
2	*48*	*49*	*50*	*51*	*52*	*53*	*54*	*55*	*56*	*57*	*58*	*59*	*60*	*61*	*62*	*63*	*64*	*65*	*66*	*67*	*68*	*69*	*70*	*71*
3	*72*	*73*	*74*	*75*	*76*	*77*	*78*	*79*	*80*	*81*	*82*	*83*	*84*	*85*	*86*	*87*	*88*	*89*	*90*	*91*	*92*	*93*	*94*	*95*
4	*96*	*97*	*98*	*99*	*100*	*101*	*102*	*103*	*104*	*105*	*106*	*107*	*108*	*109*	*110*	*111*	*112*	*113*	*114*	*115*	*116*	*117*	*118*	*119*
5	*120*	*121*	*122*	*123*	*124*	*125*	*126*	*127*	*128*	*129*	*130*	*131*	*132*	*133*	*134*	*135*	*136*	*137*	*138*	*139*	*140*	*141*	*142*	*143*
6	*144*	*145*	*146*	*147*	*148*	*149*	*150*	*151*	*152*	*153*	*154*	*155*	*156*	*157*	*158*	*159*	*160*	*161*	*162*	*163*	*164*	*165*	*166*	*167*
7	*168*	*169*	*170*	*171*	*172*	*173*	*174*	*175*	*176*	*177*	*178*	*179*	*180*	*181*	*182*	*183*	*184*	*185*	*186*	*187*	*188*	*189*	*190*	*191*
8	*192*	*193*	*194*	*195*	*196*	*197*	*198*	*199*	*200*	*201*	*202*	*203*	*204*	*205*	*206*	*207*	*208*	*209*	*210*	*211*	*212*	*213*	*214*	*215*
9	*216*	*217*	*218*	*219*	*220*	*221*	*222*	*223*	*224*	*225*	*226*	*227*	*228*	*229*	*230*	*231*	*232*	*233*	*234*	*235*	*236*	*237*	*238*	*239*
10	*240*	*241*	*242*	*243*	*244*	*245*	*246*	*247*	*248*	*249*	*250*	*251*	*252*	*253*	*254*	*255*	*256*	*257*	*258*	*259*	*260*	*261*	*262*	*263*
11	*264*	*265*	*266*	*267*	*268*	*269*	*270*	*271*	*272*	*273*	*274*	*275*	*276*	*277*	*278*	*279*	*280*	*281*	*282*	*283*	*284*	*285*	*286*	*287*
12	*288*	*289*	*290*	*291*	*292*	*293*	*294*	*295*	*296*	*297*	*298*	*299*	*300*	*301*	*302*	*303*	*304*	*305*	*306*	*307*	*308*	*309*	*310*	*311*
13	*312*	*313*	*314*	*315*	*316*	*317*	*318*	*319*	*320*	*321*	*322*	*323*	*324*	*325*	*326*	*327*	*328*	*329*	*330*	*331*	*332*	*333*	*334*	*335*
14	*336*	*337*	*338*	*339*	*340*	*341*	*342*	*343*	*344*	*345*	*346*	*347*	*348*	*349*	*350*	*351*	*352*	*353*	*354*	*355*	*356*	*357*	*358*	*359*
15	*360*	*361*	*362*	*363*	*364*	*365*	*366*	*367*	*368*	*369*	*370*	*371*	*372*	*373*	*374*	*375*	*376*	*377*	*378*	*379*	*380*	*381*	*382*	*383*
16	*384*	*385*	*386*	*387*	*388*	*389*	*390*	*391*	*392*	*393*	*394*	*395*	*396*	*397*	*398*	*399*	*400*	*401*	*402*	*403*	*404*	*405*	*406*	*407*
17	*408*	*409*	*410*	*411*	*412*	*413*	*414*	*415*	*416*	*417*	*418*	*419*	*420*	*421*	*422*	*423*	*424*	*425*	*426*	*427*	*428*	*429*	*430*	*431*
18	*432*	*433*	*434*	*435*	*436*	*437*	*438*	*439*	*440*	*441*	*442*	*443*	*444*	*445*	*446*	*447*	*448*	*449*	*450*	*451*	*452*	*453*	*454*	*455*
19	*456*	*457*	*458*	*459*	*460*	*461*	*462*	*463*	*464*	*465*	*466*	*467*	*468*	*469*	*470*	*471*	*472*	*473*	*474*	*475*	*476*	*477*	*478*	*479*
20	*480*	*481*	*482*	*483*	*484*	*485*	*486*	*487*	*488*	*489*	*490*	*491*	*492*	*493*	*494*	*495*	*496*	*497*	*498*	*499*	*500*	*501*	*502*	*503*
21	*504*	*505*	*506*	*507*	*508*	*509*	*510*	*511*	*512*	*513*	*514*	*515*	*516*	*517*	*518*	*519*	*520*	*521*	*522*	*523*	*524*	*525*	*526*	*527*

El robot, antes de comenzar a marchar, tiene información del entorno en donde se encuentra, esto se refiere al mapa y a los obstáculos presentes.

El mapa podrá ser inicializado con tres grados de dificultad:

- Alta: Cada casillero tendrá un 33% de probabilidad de estar bloqueado.
- Media: Cada casillero tendrá un 25% de probabilidad de estar bloqueado.
- Baja: Cada casillero tendrá un 20% de probabilidad de estar bloqueado.

Además se le podrá configurar manualmente la presencia, o no, de los obstáculos parciales; estos son el huracán y el fuego. Ambos dificultarán la marcha del robot si éste pasa por allí.

4. Solución Propuesta

De acuerdo al problema planteado anteriormente se propone aplicar Algoritmos Evolutivos en la búsqueda y optimización de una solución aceptable para éste.

4.1 Representación

Cada individuo de la población está representado por genes, los cuales tomarán el valor de una de las cuatro letras los puntos cardinales, que indican en qué dirección se mueve el robot.

N: Norte **S**: Sur **E**: Este **O**: Oeste

Estos individuos poseerán longitud variable. A medida que transcurra el paso de las generaciones estos individuos crecerán en longitud acercándose al destino, o tendiendo a desaparecer si no evolucionan los suficiente o quedando varados en un óptimo local, siendo superados en aptitud por otros individuos.

Cabe destacar que se introdujeron dos restricciones para mejorar la performance de la evolución de los individuos:

1. Un individuo no puede pasar por un lugar por el que ya haya pasado anteriormente. Si esto ocurre, el individuo termina en el casillero anterior al que estaría por pasar por segunda vez.
2. Si el individuo se dirige desde un casillero hacia otro, en el siguiente movimiento se le prohibirá volver al casillero desde donde partió.

4.2 Definición de la Función Aptitud

Para comenzar se debe definir una función de aptitud que sirva para resolver el problema. Ésta deberá indicar, para cada individuo que el algoritmo genere, en qué medida es útil para llegar a una solución. Para esto se necesitará evaluar, a la vez, tres factores:

1. Cuán cerca de la meta se encuentra.
2. Cuánto recorrió para llegar a ese estado.
3. Si atravesó algún obstáculo parcial (fuego o huracán).

Como primer paso se toma la grilla y a cada casillero se le asigna la distancia Euclideana[6] desde éste hasta la meta, mediante la siguiente fórmula:

$$dist = \sqrt{(x - x_{meta})^2 + (y - y_{meta})^2}$$

Realizando los cálculos queda definida la siguiente grilla en la que en cada casillero se muestra la distancia desde éste hasta la meta.

6. Distancia entre dos puntos.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	2	2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
3	3	3	3	4	5	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
4	4	4	4	5	5	6	7	8	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
5	5	5	5	5	6	7	7	8	9	10	11	12	13	13	14	15	16	17	18	19	20	21	22	23
6	6	6	6	6	7	7	8	9	10	10	11	12	13	14	15	16	17	18	18	19	20	21	22	23
7	7	7	7	7	8	8	9	9	10	11	12	13	13	14	15	16	17	18	19	20	21	22	23	24
8	8	8	8	8	8	9	10	10	11	12	12	13	14	15	16	17	17	18	19	20	21	22	23	24
9	9	9	9	9	9	10	10	11	12	12	13	14	15	15	16	17	18	19	20	21	21	22	23	24
10	10	10	10	10	10	11	11	12	12	13	14	14	15	16	17	18	18	19	20	21	22	23	24	25
11	11	11	11	11	11	12	12	13	13	14	14	15	16	17	17	18	19	20	21	21	22	23	24	25
12	12	12	12	12	12	13	13	13	14	15	15	16	16	17	18	19	20	20	21	22	23	24	25	25
13	13	13	13	13	13	13	14	14	15	15	16	17	17	18	19	19	20	21	22	23	23	24	25	26
14	14	14	14	14	14	14	15	15	16	16	17	17	18	19	19	20	21	22	22	23	24	25	26	26
15	15	15	15	15	15	15	16	16	17	17	18	18	19	19	20	21	21	22	23	24	25	25	26	27
16	16	16	16	16	16	16	17	17	17	18	18	19	20	20	21	21	22	23	24	24	25	26	27	28
17	17	17	17	17	17	17	18	18	18	19	19	20	20	21	22	22	23	24	24	25	26	27	27	28
18	18	18	18	18	18	18	18	19	19	20	20	21	21	22	22	23	24	24	25	26	26	27	28	29
19	19	19	19	19	19	19	19	20	20	21	21	21	22	23	23	24	24	25	26	26	27	28	29	29
20	20	20	20	20	20	20	20	21	21	21	22	22	23	23	24	25	25	26	26	27	28	29	29	30
21	21	21	21	21	21	21	21	22	22	22	23	23	24	24	25	25	26	27	27	28	29	29	30	31

Distancia Euclideana desde cada casillero hasta la meta.

Como vemos en el gráfico la diferencia de distancia va variando muy gradualmente a medida que nos acercamos a la meta. Necesitaríamos que los valores, a medida que nos acercamos al destino final varíen más abruptamente, con el fin de marcar una diferencia mayor a medida que nos acercamos. Por lo tanto multiplicaremos por 2 a cada uno de los valores anteriores.

Con el fin de que quede directamente el valor de la aptitud al procesar la distancia, también se multiplica por (-1) el valor de la distancia y se le suma la máxima distancia posible, con el fin de que no quede ningún valor negativo.

Lo expresado anteriormente se resume en la siguiente fórmula:

$$distancia = \sqrt{(x - x_{meta})^2 + (y - y_{meta})^2} * (-2) + 63$$

Realizando nuevamente los cálculos para cada uno de los casilleros de la grilla, nos quedan determinados los siguientes resultados:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	1	3	4	5	7	8	9	11	12	13	14	15	16	17	18	19	19	20	20	21	21	21	21	21
1	3	4	5	7	8	10	11	12	13	15	16	17	18	19	20	20	21	22	22	23	23	23	23	23
2	4	5	7	8	10	11	13	14	15	16	17	19	20	21	21	22	23	24	24	25	25	25	25	25
3	5	7	8	10	11	13	14	15	17	18	19	20	21	22	23	24	25	26	26	27	27	27	27	27
4	6	8	9	11	13	14	15	17	18	19	21	22	23	24	25	26	27	27	28	29	29	29	29	29
5	7	9	11	12	14	15	17	18	20	21	22	23	25	26	27	28	29	29	30	31	31	31	31	31
6	9	10	12	13	15	17	18	20	21	22	24	25	26	27	29	29	30	31	32	32	33	33	33	33
7	10	11	13	15	16	18	19	21	22	24	25	27	28	29	30	31	32	33	34	34	35	35	35	35
8	11	12	14	16	17	19	21	22	24	25	27	28	29	31	32	33	34	35	36	36	37	37	37	37
9	12	13	15	17	19	20	22	23	25	27	28	30	31	32	33	35	36	37	37	38	39	39	39	39
10	13	14	16	18	20	21	23	25	26	28	29	31	32	34	35	36	37	38	39	40	41	41	41	41
11	13	15	17	19	21	22	24	26	27	29	31	32	34	35	37	38	39	40	41	42	43	43	43	43
12	14	16	18	20	21	23	25	27	29	30	32	33	35	37	38	39	41	42	43	44	45	45	45	45
13	15	17	19	20	22	24	26	28	29	31	33	35	36	38	39	41	42	43	45	46	46	47	47	47
14	15	17	19	21	23	25	27	29	30	32	34	36	37	39	41	42	44	45	46	47	48	49	49	49
15	16	18	20	22	24	26	27	29	31	33	35	37	38	40	42	43	45	47	48	49	50	51	51	51
16	16	18	20	22	24	26	28	30	32	34	36	37	39	41	43	45	46	48	49	51	52	53	53	53
17	17	19	21	23	25	27	29	31	32	34	36	38	40	42	44	46	47	49	51	52	53	55	55	55
18	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	46	48	50	52	53	55	56	57	57
19	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51	53	55	56	58	59	59
20	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51	53	55	57	59	61	61
21	17	19	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51	53	55	57	59	61	63

Distancias desde cada casillero hasta la meta (adaptado).

Hasta aquí se tiene resuelto el primer problema de la función evaluación, la distancia que le restaría a cada individuo para llegar a la meta.

Ahora se analizará el segundo factor: cuánto se recorrió para llegar a ese estado. Para esto se genera un cociente entre la distancia calculada anteriormente y la cantidad de casilleros recorrido por ese individuo multiplicado por 4, este último número fue considerado adecuado a consecuencia de pruebas que se realizaron. Por último, este cociente deberá restárselo a la distancia obtenida mediante la fórmula anterior, teniendo que sumarle 31 unidades más debido a que ahora es más probable que el resultado final quede negativo. La fórmula actual sería la siguente:

$$distancia = \sqrt{(x - x_{meta})^2 + (y - y_{meta})^2} * 2$$

*aptitud = distancia – distancia / casilleros_recorridos * 4 +94*

Hasta aquí se tienen resueltos los dos primeros factores a resolver para la función aptitud, queda el último punto que resulta ser el más sencillo, lo que ocurre cuando atraviesa un obstáculo parcial, esto es el fuego o el huracán. Cada vez que el individuo pase por uno de estos dos obstáculos se le restarán 2 unidades a la aptitud calculada anteriormente, por supuesto que este valor es arbitrario e intenta indicar un retraso y dificultad en el camino del robot hacia la meta. Por lo tanto:

Si (casillero = HayFuego) o (casillero=HayHuracán)

aptitud = aptitud – 2;

Con esto queda resuelta la función de aptitud, ahora queda revisar cómo se administrará el algoritmo y otros detalles de éste.

4.3 Parametrización

Ante todo se dará la posibilidad de elegir los parámetros para ejecutar el algoritmo:

- Población: Entre 10 y 100 individuos.
- Progenitores: Entre 2 y 60 individuos, nunca superando a la cantidad de individuos de la población. Recordemos que se sugiere que la cantidad de progenitores ronde el 60% de la cantidad de individuos de la población.

· Probabilidad de Mutación
· Probabilidad de Cruzamiento

4.4 Población inicial y procesos de selección

La población inicial será creada de manera totalmente aleatoria, esto provocará una diversidad de individuos deseada desde el comienzo del proceso. La cantidad de individuos creados será la cantidad especificada en los parámetros recién descriptos. En todos los casos en que se creen individuos estos se irán generando aleatoriamente de acuerdo a la dirección que van tomando (**N**orte, **S**ur, **E**ste, **O**este) hasta que se topen con un casillero bloqueado o el límite del mapa.

El proceso de selección se realizará mediante el "método de la ruleta pesada", explicado en el capítulo 2 de esta obra. De esta manera se elegirán tantos progenitores como fueron especificados en los parámetros.

Luego se realizará el proceso de reproducción, haciendo uso de los dos operadores usados típicamente en los Algoritmos Genéticos: el cruzamiento y la mutación, cada uno con la probabilidad respectiva de ocurrencia definida inicialmente en los parámetros. Una vez que los operadores hayan actuado se procederá a determinar la nueva población, esto se realizará mediante "Reemplazo por inclusión", explicado anteriormente. De esta manera se tomará la población actual más los progenitores, luego de aplicarles los operadores, y se elegirán de todos esos individuos sólo la cantidad que se había definido como el total de población. Este muestreo se realizará por los siguientes métodos configurables al principio del proceso:

· Método de la ruleta pesada.
· Método de la ruleta pesada con elitismo. En este caso se conserva siempre el mejor de todos los individuos, y se realiza el método de la ruleta sobre el resto.

En la siguiente sección se analizará el resultado experimental de uno y otro método aplicado en el proceso de muestreo luego de la reproducción.

4.5 Operadores genéticos

Dado que el problema se aplicará mediante individuos de longitud variable, la aplicación de los operadores requiere cierto nivel de personalización como se verá en detalle:

Cruzamiento

De acuerdo a la probabilidad de ocurrencia definida inicialmente, se realizará el cruzamiento entre cada pareja de progenitores o no. El cruzamiento consistirá en intercambiar los genes entre un individuo y otro, a partir de una posición que se determinará aleatoriamente. Una vez que esté realizado este intercambio se verificará, a partir de la posición de intercambio, que cada individuo sea válido, es decir que no choque con un casillero bloqueado o se vaya afuera del mapa. Cuando esto ocurra, el individuo se trunca a partir de esa posición, inclusive.

Ejemplo.
Teniendo en cuenta que el individuo comienza desde el extremo "noroeste" del mapa.
Individuo 1: ESSSWSEEEEEEE
Individuo 2: SSSSSSSS

Cruzamiento a partir de la posición 3:
Individuo 1': ESS|SSSSS
Individuo 2': SSS|SWSEEEEEEE

El individuo 1' es válido ya que permanece dentro del mapa en toda su extensión. El 2' no es válido en su totalidad ya que llega un momento en el que se va del mapa por el oeste. Por lo tanto, este se truncaría a partir del gen que lo hace inválido y quedaría con el siguiente formato:

Individuo 2': SSS|S

Mutación

También, de acuerdo a la probabilidad de ocurrencia definida inicialmente, se realizará la mutación de cada individuo, o no, en un gen determinado. El individuo se explora gen por gen desde una punta a la otra y, de acuerdo a la probabilidad, se mutará el gen, o no.

Una vez que se decida la mutación en un determinado gen, se irá regenerando el individuo hasta que se tope con una posición no válida (casillero bloqueado o fuera del mapa).

Ejemplo.
Teniendo en cuenta que el individuo comienza desde el extremo “noroeste” del mapa.

Individuo 1: ESSSWSEEEEEEE
Mutación en el gen 2.
Individuo 1’: ES|EEN
Suponiendo que el siguiente gen era “N” (Norte), el individuo ya se iría del mapa, por lo tanto el individuo termina allí.
Existen dos variedades de mutación, una convencional comenzando a analizar la probabilidad de mutación del individuo a partir de su primer gen desde el comienzo (de izquierda a derecha), o en reversa, desde el fin del individuo hacia el principio (de derecha a izquierda). El resultado experimental de esta diferencia también será analizada en la próxima sección.

4.6 Criterio de parada.
Para esta aplicación no se tomarán en cuenta criterios de parada, ya que la simulación se mostrará paso a paso en pantalla y se tendrá la posibilidad de finalizarlo cuando uno lo desee.

5. Resultados Experimentales

Se realizaron sucesivas pruebas con la aplicación desarrollada, evaluando el rendimiento de las distintas alternativas. En esta sección se comentará lo analizado.
En la aplicación se tiene como parámetros: el número constante de individuos pertenecientes a la población, la cantidad de progenitores por generación, y la probabilidad de mutación y cruzamiento. Por otro lado, respecto de los métodos de selección se implementó el método de la ruleta pesada, con y sin elitismo. Dentro de los operadores tenemos el cruzamiento y la mutación, ambos personalizados de acuerdo a la característica variable de los individuos en este problema. Cabe destacar que la mutación tiene dos posibilidades, comenzar a evaluar la posibilidad de mutación gen por gen desde el principio del individuo hasta el final o desde el final del individuo hasta el principio, esto es, la mutación reversa.

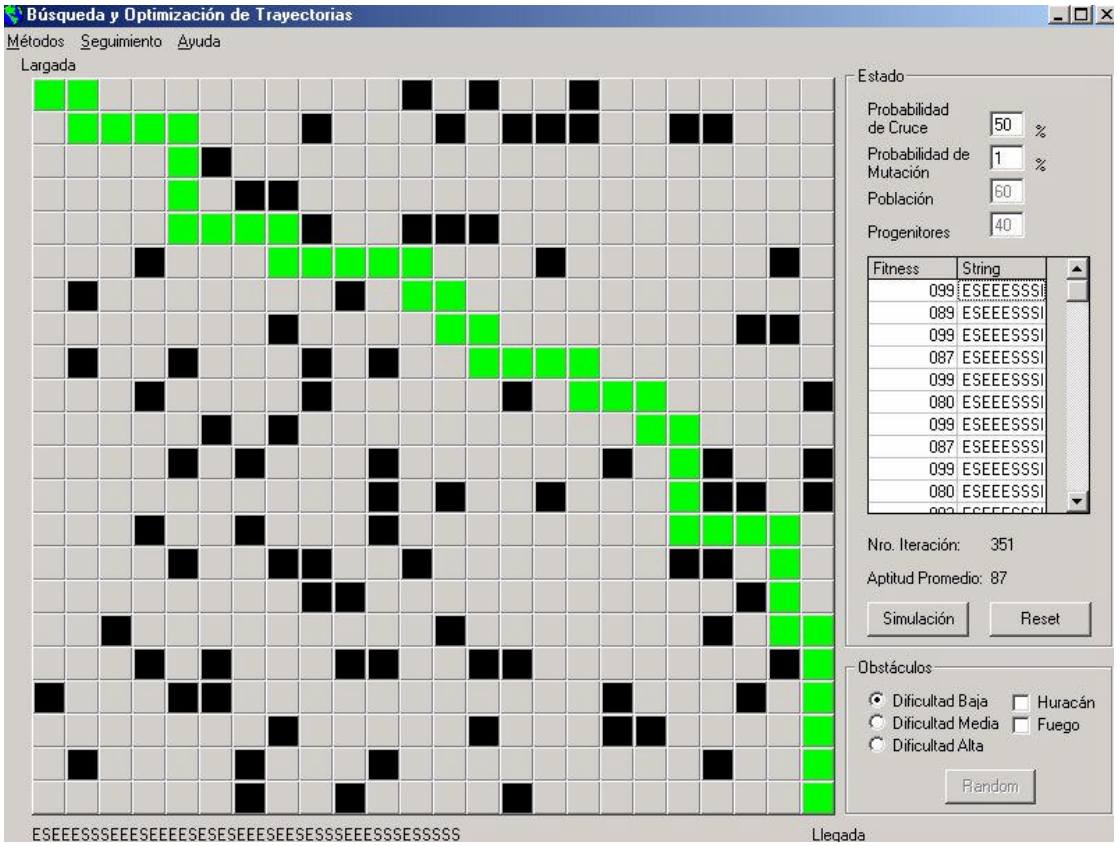


Figura 1 - Desarrollo con parámetros adecuados y pocos obstáculos.

Dentro del contexto de la aplicación se observó que el método de la ruleta pesada convencional no satisface la resolución de el problema planteado, la evolución de la población es lentísima, generalmente al extremo de ser ineficiente, debido a que la diferencia en la aptitud entre individuos no es tan grande como en otros problemas, y esto lleva a que los individuo más aptos no tengan una prioridad los suficientemente alta como para tener más posibilidad real que los demás a la hora de reproducirse. Gráficamente se ve que los mejores individuos de la población cambian el rumbo desde el origen constantemente y no evolucionan como se espera en este planteo.

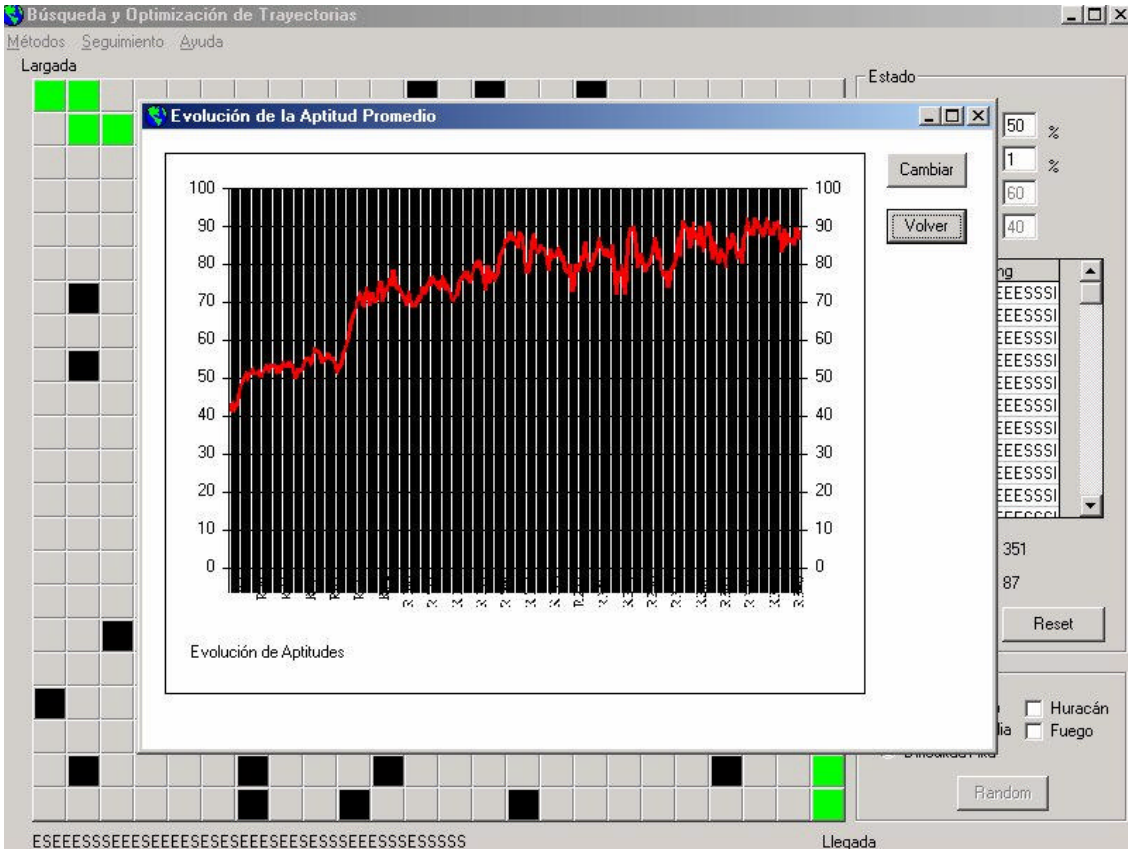


Figura 1 - Evolución de la aptitud con parámetros adecuados y pocos obstáculos.

Para contrarrestar lo expresado anteriormente se introdujo el elitismo en la ruleta, esto es el asegura-miento del mejor individuo de la población anterior para la reproducción. Con esto también tenemos certeza de que en la próxima población tendremos presente al mejor individuo de la población o, probablemente, un individuo mejor que éste si es que fue transformado por alguno de los operadores genéticos. Haciendo uso de este método sí se observa claramente la evolución de la población a través de la aplicación, siempre y cuando se configure adecuadamente al resto de los parámetros.

Por cada iteración existe la posibilidad de observar en la aplicación los procesos que se están generando internamente; esto son, la población inicial y sus progenitores, las operaciones genéticas en detalle, y por último la nueva población. Esto ayuda a analizar el desenvolvimiento de la operación y se muestra en la forma que se ilustra a continuación a modo de ejemplo.

```
Posic.  Fit. String
———  —— ———
00      078  SESSSEESSEEEEESEESEEEEEEEESSS
01      075  SESSSEESSEEEEESEESEEEEEEEES
02      078  SESSSEESSEEEEESEESEEEEEEEESSS
03      078  SESSSEESSEEEEESEESEEEEEEEESSS
04      078  SESSSEESSEEEEESEESEEEEEEEESSS
05      065  SESSSEESSEEEEESEEEENEEE
06      078  SESSSEESSEEEEESEESEEEEEEEESSS
07      078  SESSSEESSEEEEESEESEEEEEEEESSS
08      075  SESSSEESSEEEEESEESEEEEEEEES
09      078  SESSSEESSEEEEESEESEEEEEEEESSS
10      075  SESSSEESSEEEEESEESEEEEEEEES
11      075  SESSSEESSEEEEESEESEEEEEEEES
12      078  SESSSEESSEEEEESEESEEEEEEEESSS
13      078  SESSSEESSEEEEESEESEEEEEEEESSS
14      078  SESSSEESSEEEEESEESEEEEEEEESSS
15      078  SESSSEESSEEEEESEESEEEEEEEESSS
16      076  SESSSEESSEEEEESEESEEEEEEEESS
17      075  SESSSEESSEEEEESEESEEEEEEEES
18      078  SESSSEESSEEEEESEESEEEEEEEESSS
19      078  SESSSEESSEEEEESEESEEEEEEEESSS
20P(15) 078  SESSSEESSEEEEESEESEEEEEEEESSS
21P(12) 078  SESSSEESSEEEEESEESEEEEEEEESSS
22P(01) 075  SESSSEESSEEEEESEESEEEEEEEES
23P(01) 075  SESSSEESSEEEEESEESEEEEEEEES
24P(07) 078  SESSSEESSEEEEESEESEEEEEEEESSS
25P(10) 075  SESSSEESSEEEEESEESEEEEEEEES
26P(13) 078  SESSSEESSEEEEESEESEEEEEEEESSS
27P(02) 078  SESSSEESSEEEEESEESEEEEEEEESSS
28P(17) 075  SESSSEESSEEEEESEESEEEEEEEES
29P(09) 078  SESSSEESSEEEEESEESEEEEEEEESSS
30P(10) 075  SESSSEESSEEEEESEESEEEEEEEES
31P(04) 078  SESSSEESSEEEEESEESEEEEEEEESSS
32P(14) 078  SESSSEESSEEEEESEESEEEEEEEESSS
33P(11) 075  SESSSEESSEEEEESEESEEEEEEEES
Se realiza Crossover entre strings 24 y 25 a partir de posicion 19
24CR    075  SESSSEESSEEEEESEESEEEEEEEES
25CR    078  SESSSEESSEEEEESEESEEEEEEEESSS
Se realiza Crossover entre strings 30 y 31 a partir de posicion 12
30CR    078  SESSSEESSEEEEESEESEEEEEEEESSS
31CR    075  SESSSEESSEEEEESEESEEEEEEEES
Se realiza Crossover entre strings 32 y 33 a partir de posicion 8
32CR    075  SESSSEESSEEEEESEESEEEEEEEES
33CR    078  SESSSEESSEEEEESEESEEEEEEEESSS
Se realiza Mutacion en string 26 a partir de posicion 6.
26MT    037  SESSSENENWSWW
Se realiza Mutacion en string 30 a partir de posicion 18.
30MT    066  SESSSEESSEEEEESEESENNWSSEEE
Se realiza Mutacion en string 33 a partir de posicion 16.
33MT    060  SESSSEESSEEEEESE

Nueva Poblacion:
Posic.  Fit. String
———  —— ———
00      078 SESSSEESSEEEEESEESEEEEEEEESSS
01      075 SESSSEESSEEEEESEESEEEEEEEES
02      078 SESSSEESSEEEEESEESEEEEEEEESSS
03      078 SESSSEESSEEEEESEESEEEEEEEESSS
04      075 SESSSEESSEEEEESEESEEEEEEEES
05      078 SESSSEESSEEEEESEESEEEEEEEESSS
06      078 SESSSEESSEEEEESEESEEEEEEEESSS
07      075 SESSSEESSEEEEESEESEEEEEEEES
08      078 SESSSEESSEEEEESEESEEEEEEEESSS
```

```
09      075 SESSSEESSEEEEESEESEEEEEEEES
10      075 SESSSEESSEEEEESEESEEEEEEEES
11      059 SESSSEESSEEEEESEESENNW
12      059 SESSSEESSEEEEESEESENNW
13      075 SESSSEESSEEEEESEESEEEEEEEES
14      075 SESSSEESSEEEEESEESEEEEEEEES
15      078 SESSSEESSEEEEESEESEEEEEEEESSS
16      078 SESSSEESSEEEEESEESEEEEEEEESSS
17      075 SESSSEESSEEEEESEESEEEEEEEES
18      078 SESSSEESSEEEEESEESEEEEEEEESSS
19      075 SESSSEESSEEEEESEESEEEEEEEES
```

Respecto del número de individuos de la población y la cantidad de progenitores en cada generación, es conveniente, y se comprobó así, que se mantenga la relación sugerida en el Capítulo 2, esto es que la cantidad de progenitores debe aproximarse al 60% de la cantidad total de individuos de la población. Por otra parte se llegó a la conclusión de que para este caso, como mínimo se deben tener 20 individuos de población constante, convenientemente 60, que es lo máximo que soporta la aplicación. Si la población es pequeña, la solución no será de tanta calidad como si se hubiera utilizado una población mayor, además habrá mayor velocidad en la evolución.

Pasando a la evaluación de los operadores concluimos que el cruzamiento debe tener una probabilidad de ocurrencia de entre 20 y 50%, para obtener buenos resultados. De esta manera estaríamos sacando el mejor provecho del intercambio de información de los individuos presentes. Se recuerda que la probabilidad de cruzamiento se aplica por pareja de progenitores.

En la *figura 1* se muestra la resolución de un problema en el que se plantea una dificultad baja en cuanto a los obstáculos, lo que obliga al algoritmos a realizar un mayor análisis entre los caminos “despejados”, con una probabilidad de cruzamiento del 50%, mutación del 1%, y una población de 60 individuos con 40 progenitores. Se notará que se encontró una solución, que aunque no es la óptima, es muy buena. Y en la siguiente figura la evolución de la aptitud muestra que la evolución de ésta fue escalonada y con muchos altibajos, lo que denota un gran proceso de análisis.

Por último se probó la mutación y la mutación reversa, que fueron explicadas anteriormente. Los resultados de ambas son satisfactorios. En un principio se pensó en que la mutación fuera analizada en un sentido o en otro debido a que se suponía que la probabilidad de mutación debía ser muy alta, rondando el 30%, pero la experiencia demostró que con probabilidades bajas de mutación, como la que sugiere la bibliografía se logran mejores resultados, ya que se permite un mejor proceso de análisis con los individuos que se tiene en un momento determinado, sin entrar a analizar en profundidad precipitadamente, como sí ocurrió en la *figura 2*. Allí se utilizó una probabilidad de mutación del 40% y puede verse que en sólo en 57 iteraciones se llegó a una solución, pero que tiene demasiados tramos que están lejos de ser óptimos. Además en la siguiente figura a esta se muestra la evolución de la aptitud, y a diferencia de lo que se verá más adelante, ésta crece más rápida y linealmente.

Por lo tanto con niveles bajos de probabilidad de mutación, casi no hay diferencia si se comienza a analizar desde un extremo u otro.

Cabe destacar que de la experiencia se notó que conviene que el porcentaje de probabilidad de ocurrencia en algunos casos sea superior que en otros. Esto ocurre cuando tenemos demasiados obstáculos, allí se necesitará mayor mutación, de lo contrario el progreso será más lento, perjudicando la performance del algoritmo. De todos modos la diferencia encontrada no es tan importante.

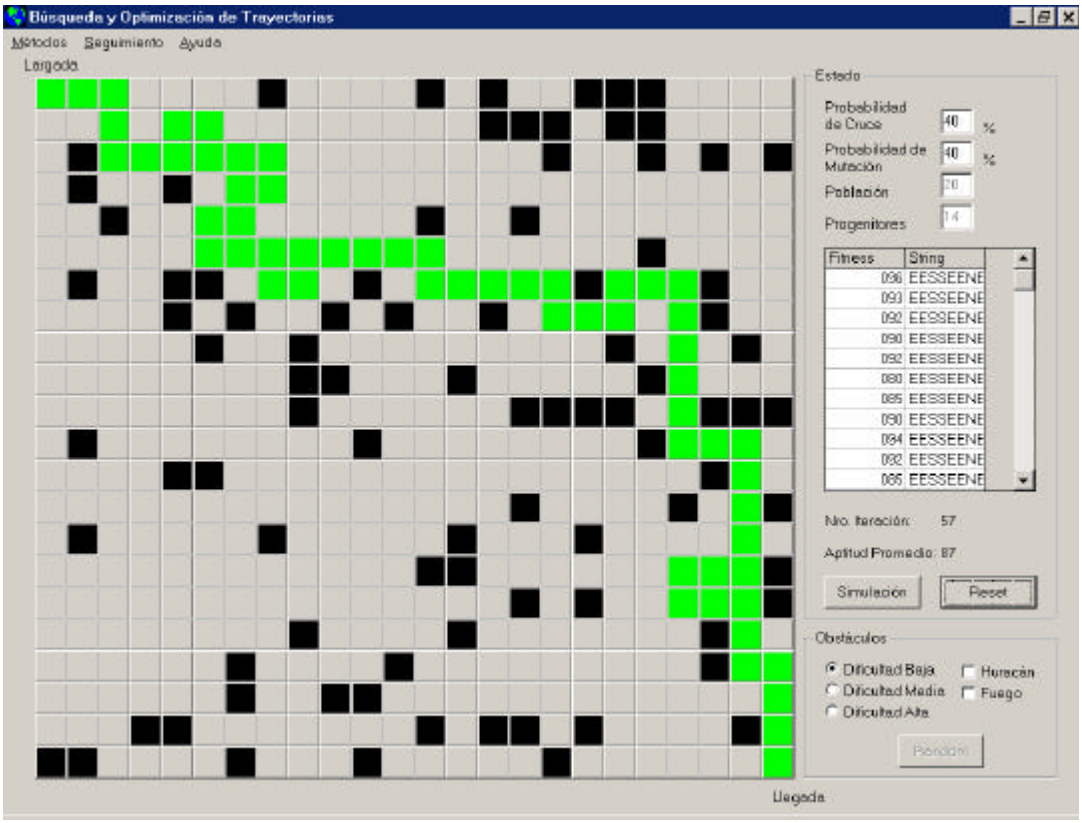


Figura 2 - Desarrollo con alta probabilidad de mutación y baja cantidad de población.

Para llegar a una solución se requieren habitualmente alrededor de 500 iteraciones. De todos modos esto depende del problema en particular. Al estar el mapa más despejado de obstáculos más rápido se llegará a una solución. Esta solución también es muy buena, salvo un defecto en la mitad del individuo, en el que podría ser más óptimo, pero de todos modos es aceptable. Al igual que en el primer ejemplo también se observa que la evolución de la aptitud también es escalonada y crece lentamente.

Se encontró una dificultad en la resolución del problema respecto de los óptimos locales, en muchos casos el algoritmo lo supera, a veces después de numerosas iteraciones, como en el ejemplo anterior. Según lo visto anteriormente esto diferencia a los Algoritmos Genéticos respecto de otros métodos de solución. pero hay casos, en el que los óptimos locales son muy pronunciados y los individuos quedan atrapados allí.

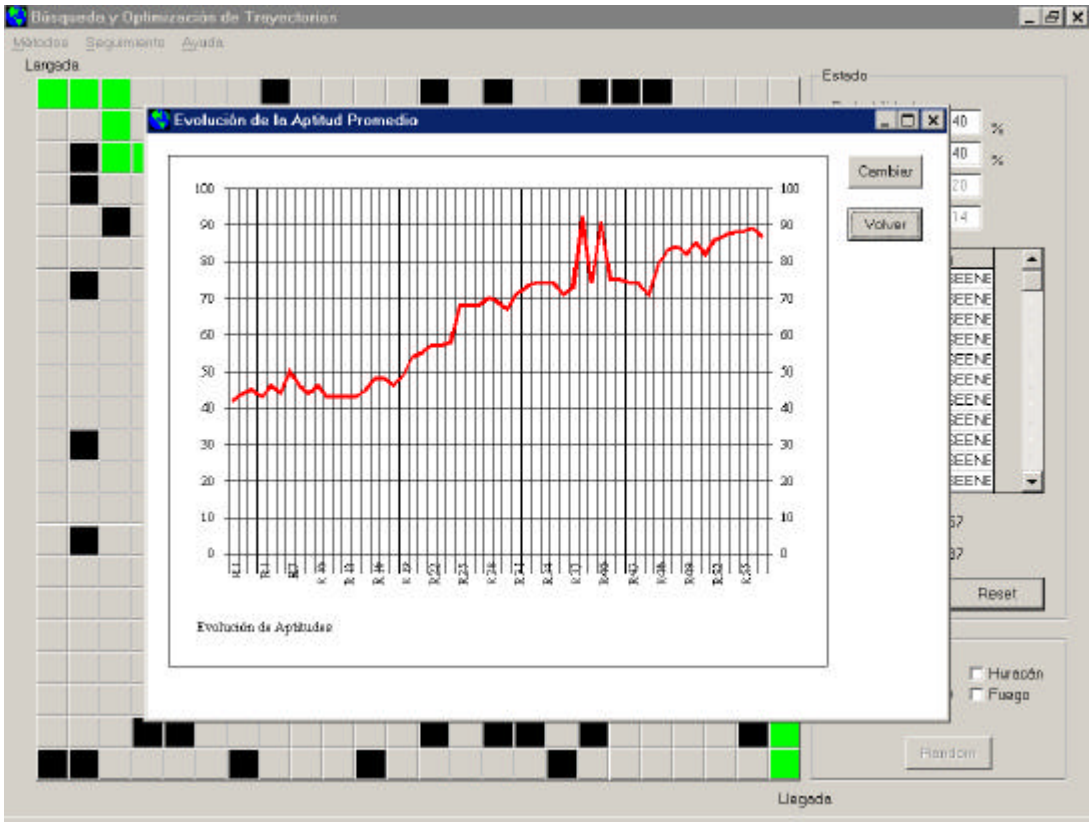


Figura 2 - Evolución de la aptitud con alta probabilidad de mutación y baja cantidad de población.

Para finalizar esta sección se muestra en la *figura 3* un proceso en el que hay una dificultad alta en cuanto a los obstáculos. De todos modos el algoritmo no queda encerrado en óptimos locales, la solución definitiva es muy buena, podría ser mejor todavía, para eso habría que utilizar una probabilidad de mutación menor a la empleada de 5%, o realizar más ajustes a la función de aptitud para poner todavía más énfasis en la ventaja de los individuos de menor longitud.

En la evolución de la aptitud vemos que debido a que se utilizó una probabilidad de mutación de 5%, mayor que en los ejemplos anteriores, la aptitud tuvo más altibajos que en aquellos ejemplos.

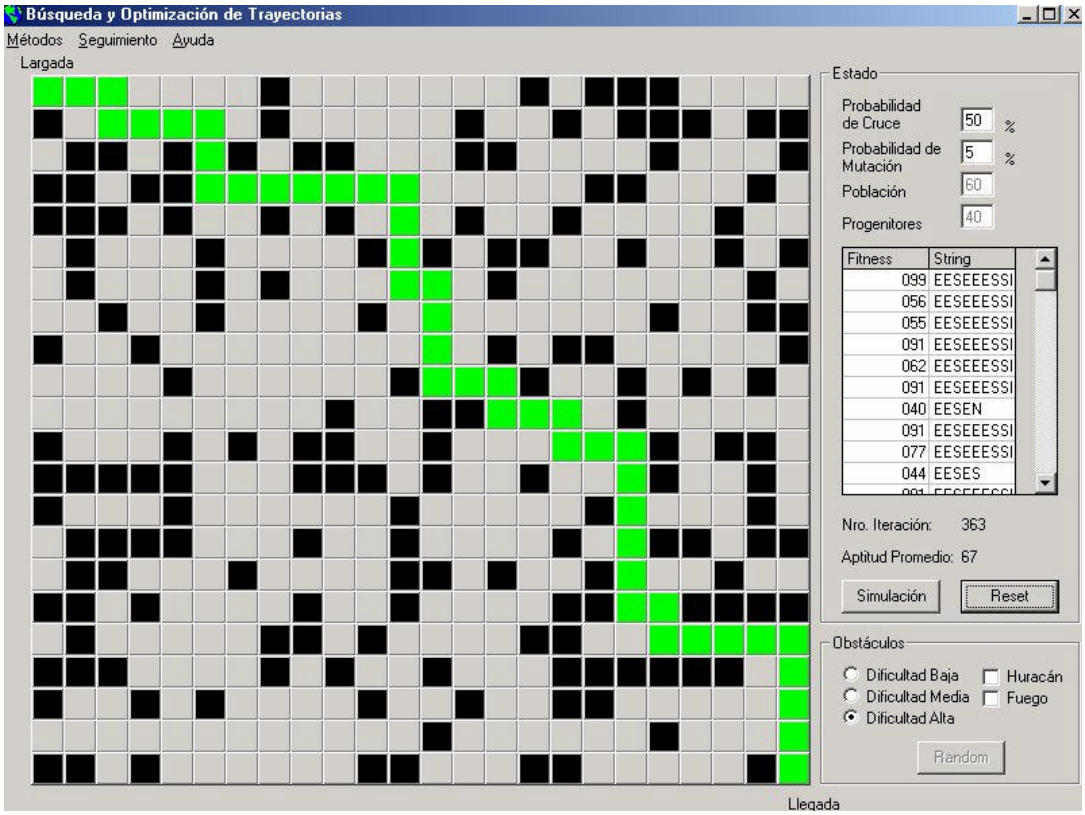


Figura 3 - Desarrollo con parámetros adecuados y muchos obstáculos.

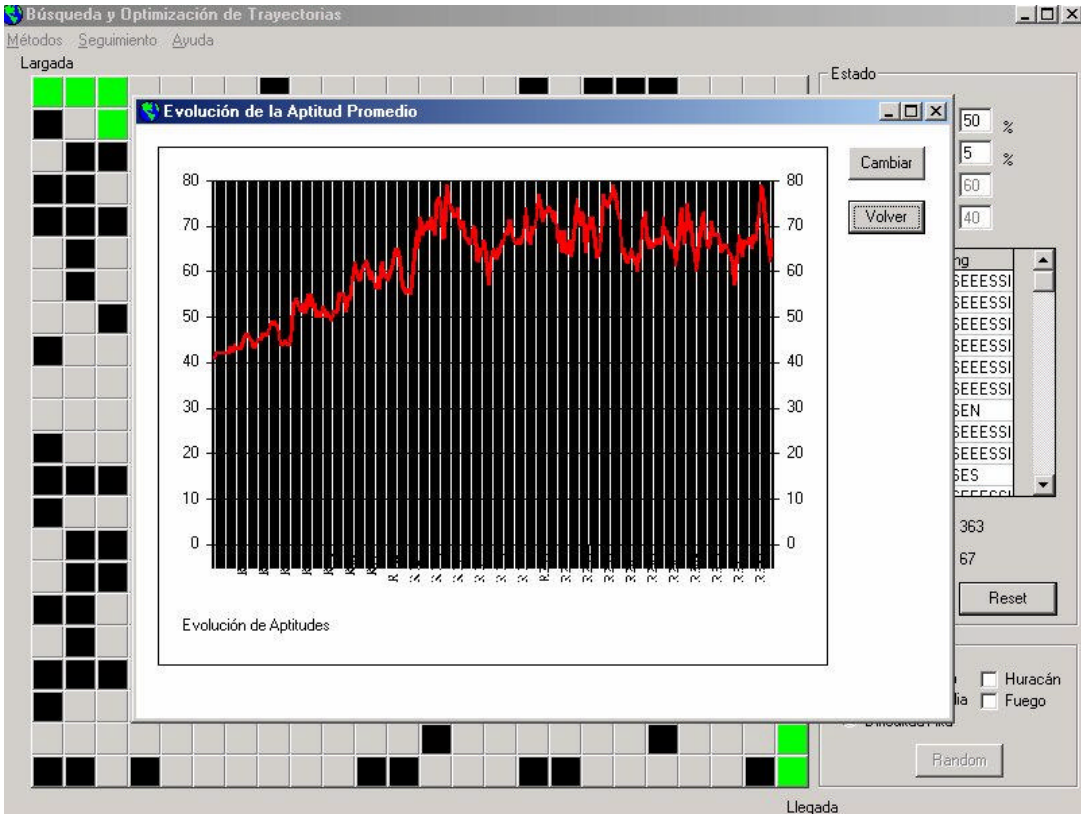


Figura 3 - Evolución de la aptitud con muchos obstáculos.

6. Conclusiones y futuras líneas de investigación.

Los capítulos precedentes presentan el resultado del estudio de la Computación Evolutiva enfocado hacia el desarrollo de una aplicación de búsqueda y optimización de trayectorias orientada a la robótica, más específicamente a la planificación de trayectorias de robots móviles autónomos en dos dimensiones.

La robustez de los Algoritmos Evolutivos se hace notar por ser métodos que no requieren información específica para funcionar, es decir, son procedimientos de búsqueda extremadamente generales, además de ser métodos de aproximación sucesiva.

En el caso de lo aplicado en este trabajo, se necesitó personalizar los operadores genéticos, estos son, el cruzamiento y la mutación, debido a la naturaleza de longitud variable de los individuos que componen la población. Los resultados obtenidos son satisfactorios y pueden ser aplicados en casos concretos de robótica.

Se plantea como investigación futura una funcionalidad que permita al algoritmo salir de estancamientos ante óptimos locales pronunciados. Esto podría llevarse adelante asignando “edades” a los individuos, es decir, haciendo desaparecer a aquellos individuos repetidos, a partir de una determinada cantidad de veces en las últimas generaciones.

Referencias Bibliográficas

[1] David E. Goldberg. Genetic Algorithms in Search, Optimization & Machine Learning. Addison-Wesley Co., Inc, Reading, MA, 1989.

[2] Zbignew Michalewicz. Genetic Algorithms + Data Structures = Evolution Programs. Second, Extended Edition. Springer-Verlag, 1994.

[3] Melanie Mitchell. An Introduction to Genetic Algorithms. MIT Press, 1997.

[4] David B. Fogel. Evolutionary Computation: Towards a New Philosophy of Machine Intelligence. Second Edition. IEEE Press, 2001.

[5] Lawrence Davis. Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York, 1991.

[6] A Genetic Algorithm for Robust Motion Planning. Departamento de Ciencia de la Computación e Inteligencia Artificial. Universidad de Alicante, España, 1998.
http://rvg.dccia.ua.es/publications/gallardo98b-IEAAIE.pdf (en inglés)
http://www.dccia.ua.es/~domingo/articulos/gallardo97-SCETA.pdf (en castellano)

[7] Hybridisation of Neural Networks and Genetic Algorithms for On-line Trajectory Planning. N. Chairyaratana y A. Zalzara. Heriot-Watt Univeristy, UK, 2000.
http://rcis.kmitnb.ac.th/publication/Control2000%20-%20On-line%20Trajectory%20Planning.pdf

[8] Genetic Algoritms for Adaptive Planning of Path and Trajectory of a Mobile Robot in 2D Terrains. Kazuo Sugihara y John Smith, 1999.
http://search.ieice.or.jp/1999/pdf/e82-d_1_309.pdf

[9] An Evolutionary Approach to Robot Structure and Trajectory Optimization. E. Pires, J. A. Tenreiro Machado, P. B. de Moura Oliveira. Universidade de Tras-os-montes E Alto Douro, Portugal,2001.
http://www.utad.pt/~epires/PO38.pdf