Robótica Evolutiva

Plataforma de Simulación Experimental

Autor: Juan Ignacio García

Director: Profesor Javier Blanqué

Universidad Nacional de Luján
Int. Ruta 5 y 7
6700, Luján, Buenos Aires
República Argentina
Año 2009

Robótica Evolutiva
Plataforma de Simulación Experimental

Juan Ignacio García
Universidad Nacional de Luján
Int. Ruta 5 y 7
6700, Luján, Buenos Aires
República Argentina
jignacio@gmail.com

Resumen

El presente trabajo introduce las distintas técnicas y metodologías utilizadas en la Robótica Evolutiva. Se analiza la factibilidad del desarrollo de un entorno de simulación para la realización de estudios experimentales, explorando la posibilidad de evolución de diversas formas de locomoción y estructuras morfológicas en robots simulados. Finalmente se procede a la realización de experimentos inspirados en la literatura de Robótica Evolutiva y al análisis de sus resultados.

Palabras claves: robótica evolutiva, algoritmos evolutivos, algoritmos genéticos, genotipo, fenotipo, programación genética, programación evolutiva, estrategias evolutivas, selección natural, selección proporcional, coevolución, mutación, inversión, entrecruzamiento, subsunción, simulador, aptitud, bugsimevo, breve, robot, hexápodo, locomoción, transferencia.

Agradecimientos

A mi director de Tesis, Javier Blanqué por su dedicación y valiosas sugerencias a lo largo del desarrollo de este proyecto.

A Jon Klein, autor del software de base “Breve”, por sus rápidas respuestas ante consultas y por la retroalimentación de ideas e implementaciones de nuevas características.

A mis padres, amigos y todos quienes me brindaron su colaboración y apoyo.

Índice General

Abreviaturas **xii**

1. Introducción **1**

2. Algoritmos Evolutivos **3**

2.1. Estrategias Evolutivas 6

2.1.1. Estructura de representación 7

2.1.2. Operadores 8

2.1.2.1.Mutación 9

2.1.2.2.Recombinación o Entrecruzamiento 10

2.2. Programación Evolutiva 12

2.2.1. Estructura de Representación (Original) 12

2.2.2. Algoritmo General 14

2.2.3. Operadores 15

2.2.3.1.Mutación 15

2.2.3.2.Selección 16

2.2.4. Variante “meta-PE” 17

2.3. Programación Genética 19

2.3.1. Primitivas 19

2.3.1.1.Terminales 20

2.3.1.2.Funciones 20

2.3.2. Estructura de Representación 22

2.3.3. Operadores Genéticos 25

2.3.3.1.Entrecruzamiento 25

2.3.3.2.Mutación 28

2.3.4. Selección 30

2.3.5. Fitness 30

3. Algoritmos Genéticos **32**

3.1. Genotipo y Fenotipo 33

3.2. Esquemas (Schemata) 33
3.3. Asignación Óptima de Pruebas 34
3.3.1. Tragamonedas de dos brazos (2-armed bandit) 34
3.3.2. Esquemas y la asignación óptima de pruebas 35
3.4. Estructura de Representación 36
3.4.1. Representaciones alternativas 37
3.5. Operadores Genéticos 39
3.5.1. Mutación 39
3.5.2. Entrecruzamiento 40
3.5.3. Inversión 41
3.6. Algoritmos 43
3.6.1. Algoritmo básico 44
3.6.2. Algoritmo básico + elitismo 44
3.7. Reemplazo de la población 45
3.8. Métodos de Selección 46
3.8.1. Selección Proporcional al Fitness 46
3.8.1.1.Selección por Rueda de Ruleta (RWS) 47
3.8.1.2.Muestreo Universal Estocástico (SUS) 50
3.8.2. Selección por Torneos (Tournament Selection) 51
3.8.3. Selección por Rangos (Rank Selection) 52
3.8.4. Selección de Estado Estacionario (Steady-State Selection) 54
3.8.5. Adaptaciones de presión selectiva 55
3.8.5.1.Escalado Sigma (Sigma Scaling) 56
3.8.5.2.Selección Boltzmann (Boltzmann Selection) 56
3.9. Co-evolución 57

4. Robótica Evolutiva 58
4.1. Evolución de Subsunción 60
4.2. Evolución de comportamientos simples 62
4.3. Evolución de comportamientos complejos 64
4.4. Locomoción y biomimética 65
4.4.1. Locomoción terrestre 66
4.4.2. Otros tipos de locomoción 68

4.5. Evolución Encarnada o Personificada (Embodied Evolution) 69
4.6. Transferencia: Simulación vs. Realidad 71
4.6.1. Transferencia de sistemas de control virtuales 71
4.6.2. Transferencia de diseños de cuerpos virtuales 73
4.6.2.1.Prototipado rápido 74
4.6.2.2.Bloques modulares discretos 76

5. BugSimEvo – Introducción 78
5.1. Motivación 79
5.2. Orígenes y desarrollo 79
5.3. Arquitectura general 80
5.4. Características Principales 81

6. Robot Simulado 85
6.1. Cuerpo virtual 85
6.1.1. Partes 85
6.1.2. Movimiento 88
6.2. Genoma 89
6.2.1. Diseño del cuerpo 90
6.2.1.1.Variante con patas del mismo tamaño 91
6.2.1.2.Variante con patas de tamaños independientes 91
6.2.1.3.Variante con cuerpo sólido 93
6.2.1.4.Variante con cuerpo articulado 94
6.2.2. Movimiento del cuerpo 95
6.2.2.1.Variante A: Secciones de patas independientes 97
6.2.2.2.Variante B: Patas independientes 98
6.2.2.3.Variante C: Segmentos independientes 98
6.2.2.4.Variante D: Segmentos del cuerpo sincronizados 99

7. Ambiente 100
7.1. Terreno plano 100
7.2. Terreno plano con obstáculos 101
7.3. Terreno Irregular 102

8. Parámetros e Interfaz de Control **104**

8.1. Configuración 105

8.2. Modo Batch 110

8.3. Carga y simulación de individuos 111

9. Resultados Experimentales **112**

9.1. Configuración Experimental 112

9.2. Simulación Inicial: Parámetros Evolutivos 113

9.2.1. Entrecruzamiento vs. Mutación Uniforme 114

9.2.2. Entrecruzamiento vs. Mutación Gaussiana 116

9.3. Sistemas de Control y cantidad de patas 119

9.4. Obstáculos – Deslizamiento 123

9.5. Obstáculos – Escalado 128

Conclusiones **132**

Futuras Líneas de Investigación **133**

Bibliografía **134**

Anexo A: Entorno de simulación "Breve" **146**

Anexo B: Código Fuente de BugSimEvo **157**

Índice de Figuras

2.1 Máquina de Estado Finito 13
2.2 Máquina de Estado Finito tipo Mealy 13
2.3 Máquinas de Estado Finito tipo Mealy utilizadas c. predictores 14
2.4 Árbol de sintaxis para la expresión 22
2.5 Desarrollo de árbol por el método de árbol completo 23
2.6 Desarrollo de árbol por el método de crecimiento 24
2.7 Entrecruzamiento entre dos individuos 27
2.8 Mutación “subárbol” de un individuo 29
2.9 Mutación puntual de un individuo 30
3.1 Mutación 39
3.2 Entrecruzamiento simple o de un solo punto 40
3.3 Entrecruzamiento multipunto 40
3.4 Inversión 42
3.5 Inversión y entrecruzamiento 43
3.6 Selección por Rueda de Ruleta (RWS) 48
3.7 Muestreo Universal Estocástico (SUS) 51
4.1 Arquitecturas de IA utilizadas en robótica autónoma 61
4.2 Comportamientos emergentes en robótica evolutiva 62
4.3 Robot Khepera 63
4.4 Sony AIBO 68
4.5 Robots obtenidos por medio de evolución en el proyecto GOLEM 75
4.6 Generación capa por capa 75
4.7 Robot generado en base a componentes modulares discretos 77
5.1 Entorno de simulación de robótica evolutiva BugSimEvo 78
5.2 Arquitectura general de BugSimEvo 80
6.1 Robot virtual con a) cuerpo sólido B) cuerpo artitulado 86
6.2 Componentes proximal (p) y distal (d) de las patas 86
6.3 Peso del robot 87
6.4 Cálculo de grosor de las patas en base a su longitud 87
6.5 Ejemplos de variantes de extremidades completas 88

6.6 Componente horizontal del movimiento de una pata 88
6.7 Componente vertical del movimiento de una pata 89
6.8 Componentes (a) horizontal y (b) vertical del movimiento de un segmento del cuerpo 89
6.9 Estructura general del mapeo de genes 90
6.10 Posiciones P_1 y P_2 del genoma y codificación de largo distal y proximal de una pata modelo 91
6.11 Codificación alternada de posiciones de P_1 y P_2 92
6.12 Posiciones C_a y C_l del genoma y codificación de ancho y largo del cuerpo sólido 93
6.13 Posiciones C_a y C_l del genoma y codificación de ancho y largo del segmento de cuerpo articulado 94
6.14 Posición O del genoma y codificación de la frecuencia de la onda sinusoidal 96
6.15 Posiciones F_1 a F_m y codificación de desplazamientos de fase 96
7.1 Terreno plano 100
7.2 Dos variantes de terreno plano con obstáculos 101
7.3 Terreno irregular. Vista en modo grilla 102
7.4 Terreno irregular. Vista en modo normal 103
8.1 Vista general de la sección Configuración de la interfaz 105
8.2 Grupo de parámetros globales de la sección Configuración 106
8.3 Grupo de parámetros relacionados con el cuerpo del robot 106
8.4 Grupo de parámetros relacionados con las patas del robot 107
8.5 Vista general de la sección Batch de la interfaz 110
8.6 Vista general de la sección Individuo de la interfaz 111
9.1 Entrecruzamiento vs. Mutación Uniforme 115
9.2 Entrecruzamiento vs. Mutación Uniforme. Vista 3D 116
9.3 Entrecruzamiento vs. Mutación Gaussiana 118
9.4 Entrecruzamiento vs. Mutación Gaussiana. Vista 3D 118
9.5 Evolución de fitness (distancia recorrida) a lo largo de 20 generaciones para robots de 6 patas utilizando distintas variantes de mapeo de movimientos 121

9.6 Evolución de fitness (distancia recorrida) a lo largo de 20 generaciones para robots de 10 patas utilizando distintas variantes de mapeo de movimientos 122
9.7 Comparación de performance obtenida por individuos con 6 y 10 patas con distintas variantes de movimientos 123
9.8 Ambiente con obstáculos posicionados a alturas progresivamente menores, ubicados entre el robot y su objetivo 124
9.9 Ejemplo de recorridos reales de los mejores individuos de cada generación en una simulación típica completa (vista superior) 126
9.10 Recorridos de los mejores individuos de cada generación – 3D 126
9.11 Evolución del tamaño de los segmentos proximal y distal de las patas con el transcurso de las generaciones 127
9.12 Ambiente con obstáculos posicionados a alturas progresivamente mayores, ubicados entre el robot y su objetivo 128
9.13 Ejemplo de recorridos reales de los mejores individuos de cada generación en una simulación típica completa (vista superior) 130
9.14 Recorridos de los mejores individuos de cada generación – 3D 130
9.15 Evolución del tamaño de los segmentos proximal y distal de las patas con el transcurso de las generaciones 131

Índice de Tablas

3.1 Obtención de gray-code a partir de codificación binaria estándar .. 37
9.1 Resumen de la simulación “Parámetros Evolutivos – Entrecruzamiento vs. Mutación Uniforme” 114
9.2 Resumen de la simulación “Parámetros Evolutivos – Entrecruzamiento vs. Mutación Gaussiana” 117
9.3 Resumen de la simulación “Sistemas de Control y cant. de patas” 119
9.4 Resumen de la simulación “Obst. – Desplazamiento por debajo” . 125
9.5 Resumen de la simulación “Obst. - Escalado” 129

Abreviaturas

AE	Algoritmo Evolutivo
AF	Autómata Finito
AG	Algoritmo Genético
EE	Estrategias Evolutivas /
	Evolución Encarnada (según el contexto)
IA	Inteligencia Artificial
PE	Programación Evolutiva
PG	Programación Genética
MEF	Máquina de Estado Finito
RE	Robótica Evolutiva

Capítulo 1

1 Introducción

La Naturaleza ha servido como inspiración y como modelo para muchos desarrollos científicos. En el campo de las Ciencias de la Computación y en particular en el campo de la Inteligencia Artificial, la adopción de principios y modelos basados en la naturaleza ha dado origen a diversas técnicas de diseño, organización y optimización.

El presente trabajo describe la aplicación de los conceptos naturales de evolución biológica, genética y biomecánica al diseño no supervisado de criaturas virtuales.

Se platea como objetivo que el material sirva como introducción a las diferentes técnicas de Robótica Evolutiva y los Algoritmos Evolutivos utilizados en ésta.

Además, se busca mostrar la aplicación de dichas técnicas y principios evolutivos al desarrollo de una implementación concreta: el desarrollo del entorno de simulación flexible y extensible, que permita la realización de experimentos de Robótica Evolutiva con diversos grados de complejidad.

Se presenta con ese fin al sistema BugSimEvo, un entorno de simulación desarrollado para estudiar en forma experimental la evolución de distintos mecanismos de locomoción en robots y las adaptaciones morfológicas impulsadas por la presión selectiva de diferentes ambientes.

Se busca además evaluar su funcionamiento por medio de la realización de experimentos evolutivos, diseñados en base a experimentos

recogidos de la literatura de la RE y, adicionalmente, para verificar de forma empírica principios biológicos de la locomoción animal.

En el siguiente capítulo se describen las diferentes ramas de Algoritmos Evolutivos, sus orígenes, similitudes y diferencias principales. Se destaca además cómo muchas de las características diferenciadoras han ido desapareciendo con el progreso y desarrollo de variantes de las técnicas originales, que adoptan elementos de diversas subramas.

En el capítulo 3 se hace una descripción detallada de los Algoritmos Genéticos: su funcionamiento, componentes, variantes y bases teóricas.

En el capítulo 4 se analizan ejemplos representativos de Robótica Evolutiva, la aplicación de algoritmos evolutivos al diseño de robots: la generación de esquemas de cuerpos, sus partes y articulaciones, así como también la obtención de comportamientos complejos a partir de objetivos expresados de forma general.

Los capítulos 5 al 8 describen el sistema de simulación de Robótica Evolutiva "BugSimEvo", su origen, características y funcionamiento. BugSimEvo se diseñó como una plataforma flexible para evaluar distintos aspectos de la robótica evolutiva y cómo ciertas consideraciones bio-miméticas pueden influir en la generación de adaptaciones morfológicas, diversidad de comportamientos y en la eficiencia de éstos.

En el capítulo 9 se detallan los experimentos de simulación realizados, sus configuraciones de parámetros, ambiente y función de fitness, objetivos y resultados obtenidos.

Finalmente, en el capítulo 10 se expresan conclusiones generales y en el capítulo 11 se plantean ideas para futuros desarrollos y aplicaciones del sistema BugSimEvo.

Capítulo 2

2 Algoritmos Evolutivos

Introducidos a principios de los ’50 y alcanzando posteriormente una mayor difusión gracias a importantes desarrollos durante los ’60 y ’70 [Fogel1966][Bäck2000], los Algoritmos Evolutivos (AE) han tenido una creciente aplicación a problemas prácticos por medio del modelado de la evolución biológica para obtener soluciones basadas en la adaptación, combinación, especiación y otras características propias del mundo natural.

Aunque existen actualmente un gran número de variantes de AE con características únicas, se pueden mencionar conceptos comunes a todas ellas. Es importante tener en cuenta que si bien los términos utilizados están basados en la biología, su significado dentro de los AE suele ser más acotado, refiriéndose a un modelo simplificado de sus análogos naturales.

En un proceso evolutivo de *selección natural*, una población de individuos compatibles se reproducen y compiten entre si. Los individuos más aptos y que mejor se adaptan al ambiente tienen mayores posibilidades de reproducirse y pasar sus características a la siguiente generación. Este pasaje de características es ocasionalmente imperfecto (mutaciones), lo que genera variabilidad y con el tiempo, aquellas variaciones que benefician la aptitud de los individuos que las poseen terminan esparciéndose por la población de las generaciones futuras. [Dawkins1976]

En un AE, se llama *población* a un conjunto de individuos, *soluciones potenciales* de un problema dado. La *aptitud* se determina por una *función de adaptabilidad o fitness*, la cual evalúa a cada individuo y le asigna un puntaje, el cual generalmente es aproximado. Este puntaje es luego utilizado para un proceso llamado *selección*, en el cual con diferentes estrategias se determina

qué individuos de una generación se reproducirán y pasarán sus características a la siguiente. El proceso de reproducción implica la aplicación de *operadores* como *mutación* y *combinación*, entre otros. [Dumistrescu2000]

A continuación se muestra un esquema general [Dumitrescu2000] del proceso que se sigue en un AE:

1. Crear de forma aleatoria una Población Inicial (P_0) compuesta de n individuos
2. Evaluar y seleccionar individuos de acuerdo a su aptitud.
3. Minetras no se haya llegado a un objetivo predefinido (de iteraciones, de tiempo de simulación, de nivel de adaptación u otro):
    a. Aplicar operadores de recombinación y/o alteración (mutación) generando un nuevo conjunto de individuos
    b. Evaluar y seleccionar individuos de acuerdo a su aptitud, utilizando una estrategia de reemplazo

Dentro de los AE originales se pueden mencionar cuatro ramas principales [Mitchel1998] [Wahde2005]:

- Algoritmos Genéticos (AG), que se caracterizan por la codificación de los parámetros del problema en forma de una cadena de símbolos, generalmente binarios.
- Estrategias Evolutivas (EE), que se diferencian de los AG tradicionales por utilizar valores reales para codificar los parámetros y por poseer un nivel de mutación variable para cada parámetro.
- Programación Evolutiva (PE), que utiliza máquinas de estado finito como método de representación, con variantes actuales basadas en vectores de valores reales.

- Programación Genética (PG), en la cual la representación no consiste en una cadena de símbolos sino en una estructura de árbol (aunque actualmente también hay versiones con representación lineal y de grafos).

A continuación se presenta un resumen de los orígenes, progreso y estado actual de tres ramas de los Algoritmos Evolutivos: EE, PE y PG, mientras que las características de los AG se tratarán por separado en el Capítulo 3. Es de destacar que distintos avances en estas ramas han resultado en la reducción de muchas de las diferencias originales dada la adopción general de características inicialmente consideradas exclusivas de cada rama, como se podrá observar en las siguientes secciones y en el próximo capítulo.

2.1 Estrategias Evolutivas

Las Estrategias Evolutivas (EE) tuvieron sus inicios en la década del 60, en la Universidad Técnica de Berlín, con los trabajos de Bienert, Rechenberg y Schwefel. [Bäck1991]

La primer versión de EE se denominó *EE-(1+1)* [Rechenberg1973][Schwefel1995], término[1] que describe el método de selección utilizado:

- Primero, se genera un individuo consistente en un vector de n elementos (números reales), denominados parámetros o variables objeto, obtenidos de un individuo padre por medio de la aplicación de mutación.
- Luego, se evalúa la aptitud del individuo obtenido y si resulta superior a la de su padre, lo reemplaza y se toma como base para la generación del próximo hijo, convirtiéndose en el nuevo padre. En caso contrario, se descarta y vuelve a aplicarse el proceso con el padre original.

Como se puede observar, en la versión inicial de EE, no existía el concepto de población. En esta versión, la mutación se aplicaba a cada parámetro utilizando una distribución normal, con el mismo desvío estándar para todos.

Una segunda versión, denominada *EE-(μ+1)*, introdujo la idea de utilizar una población [Rechenberg1973]. El proceso consistía en generar una población inicial de tamaño μ (con $\mu > 1$) para luego, de forma iterativa:

- Combinar los individuos para producir un único individuo hijo
- Mutar dicho individuo

[1] En inglés: *(1+1)-ES*, donde ES: Evolution Strategy

- Evaluar su aptitud y reemplazar el peor individuo de la población padre con el nuevo individuo, si la aptitud de este último resultase mayor

Esta versión sirvió de base para el posterior desarrollo de dos nuevas estrategias evolutivas, *EE-(μ+λ)* y *EE-(μ, λ)*. [Schwefel1995]

En el caso de *EE-(μ+λ)*, se generan λ hijos y del conjunto de padres e hijos se seleccionan los μ mejores, que se convertirán en los padres de la siguiente generación. Es interesante destacar que el concepto de reemplazo parcial de los padres también está presente en variantes modernas de AG, en donde es denominado *elitismo*. Por otro lado, para *EE-(μ, λ)* los μ mejores se seleccionan exclusivamente entre los λ hijos (por lo tanto, en este caso necesariamente $\lambda > \mu$), siendo todos los padres reemplazados. Un mecanismo similar es conocido como selección por truncamiento en los AG.

Un cambio importante fue la introducción del concepto de auto adaptación del proceso por medio de la inclusión de los parámetros evolutivos dentro de cada individuo, en la forma de pares de vectores denominados *par objeto-estrategia*.

2.1.1 Estructura de representación

Se puede definir un individuo en una Estrategia Evolutiva como:

$$\vec{a} = (\vec{x}, \vec{\sigma}, \vec{\alpha})$$

donde[2]:

$$\vec{x} \in \mathbb{R}^n \qquad \vec{\sigma} \in \mathbb{R}_+^{n_\sigma} \qquad \vec{\alpha} \in [-\pi, \pi]^{n_\alpha}$$

con:

$$n_\sigma \in \{1, ..., n\}$$
$$n_\alpha \in \{0, (2n - n_\sigma)(n_\sigma - 1)/2\}$$

[2] En principio también puede ser $\vec{x} \in \mathbb{N}^n$ e incluso $\vec{x} \in \mathbb{B}^n$ (donde $\mathbb{B} = \{0, 1\}$) [Beyer2001]

Es decir, un individuo *a* está compuesto por un vector *x* de *n* parámetros objeto y por un conjunto de parámetros evolutivos o parámetros estrategia. El vector σ contiene una serie de desvíos estándar, que son utilizados para determinar la mutación a aplicar a los parámetros objeto, mientras que el vector α permite indicar correlaciones entre los parámetros objeto.

Se pueden destacar tres EE principales de acuerdo a la cantidad y tipo de parámetros estrategia:

- Un único desvío estándar, que se aplica por igual a todos los parámetros objeto: $n_\sigma = 1, n_\alpha = 0$
- Un desvío estándar por cada parámetro objeto: $n_\sigma = n, n_\alpha = 0$
- Un desvío estándar por cada parámetro objeto y *n(n-1)/2* ángulos de rotación[3] que definen correlaciones entre parámetros: $n_\sigma = n, n_\alpha = n\,(n-1)\,/2$

Bäck sugiere, ante problemas de topología desconocida, la aplicación de estas tres variantes de forma jerárquica en el orden presentado, es decir, partir simpre de la variante más simple ($n_\sigma = 1, n_\alpha = 0$) y sólo pasar a variantes más complejas de resultar necesario. Para las tres variantes sugiere utilizar una configuración EE-(μ,λ), con μ = 100, λ = 15 y con un σ_i inicial = 3.0 para todo *i*. [Bäck1996]

2.1.2 Operadores

Como Algoritmo Evolutivo, las Estrategias Evolutivas poseen operadores inspirados en el proceso de selección natural. Estos serán descriptos brevemente para los casos clásicos de EE: i) un desvío estándar único aplicable a todos los parámetros objeto y ii) un desvío estándar por cada parámetro objeto, sin parámetros de correlación.

[3] Una explicación detallada de cómo se pueden utilizar ángulos para indicar correlaciones y determinar "*hiperelipsoides de mutación*" no alineados a los ejes de coordenadas puede encontrarse en [Bäck1996].

Los tres operadores empleados por las EE son *selección*, ya analizado previamente en sus distintas variantes[4], *mutación* y *combinación* o *entrecruzamiento*.

2.1.2.1 Mutación

Tanto los parámetros objeto como los parámetros de estrategia están sujetos a mutación.

La mutación del vector de parámetros objeto consiste en la adición de un vector z de valores aleatorios independientes utilizando una distribución normal con media cero y desvío σ_i.

$$\vec{x}_{mut} = \vec{x} + \vec{z} \qquad \vec{z} = \{z_1, ..., z_n\}$$

Para $n_\sigma = 1$ resulta $\sigma_i = \sigma$:

$$z_i \sim N\left(0, \sigma^2\right)$$

Mientras que para el caso $n_\sigma = n$ la mutación de cada parámetro objeto sigue una distribución normal con un desvío independiente:

$$z_i \sim N\left(0, \sigma_i^2\right)$$

La mutación del vector de desvíos o parámetros de estrategia es multiplicativa. La heurística más conocida es la denominada regla de 1/5. Se basa en resultados teóricos obtenidos para EE-(1+1) por Rechemberg [Rechemberg1973] según los cuales, para las distintas clases de problemas analizados, la tasa de progreso ideal depende del porcentaje de mutaciones exitosas (mejoras en el valor de la función objetivo), siendo el porcentaje ideal de 1/5.

Schwefel describe la regla de la siguiente forma [Schwefel1995]:

[4] (1+1), (μ+1), (μ+λ) y (μ,λ)

"Cada n mutaciones, verificar cuantos éxitos se obtuvieron en las últimas 10n mutaciones. Si este número es menor a 2n, multiplicar los largos de los pasos [desvíos] por el factor 0.85; dividirlos por 0.85 si ocurrieron más de 2n éxitos."

Otras variantes se basan en la aplicación, también de forma multiplicativa, de factores que dependen de un valor aleatorio obtenido de una distribución log-normal, particularmente para aquellas variantes donde λ (el número de hijos por generación) es mayor a 1.

2.1.2.2 Recombinación o Entrecruzamiento

La recombinación o entrecruzamiento es utilizada en las variantes de EE donde más de un padre toma parte en la generación de un hijo, siendo este el resultado de una combinación de parámetros objeto y parámetros estrategia de sus padres.

En la *Recombinación Global Discreta*, se selecciona aleatoriamente cada componente x_i de los parámetros objeto para cada posición 1 <= i <= n entre los μ padres, con μ >= 2. Es decir, esto permite una reproducción no solo bisexual sino multisexual [Schwefel1995] o *panmíctica* [Bäck1996]. Este tipo de recombinación es similar al *entrecruzamiento uniforme* de los AG para el caso $\mu = 2$ o cuando sólo se seleccionan (de forma aleatoria en las EE) dos individuos como padres en una población $\mu > 2$ (RGA no panmíctica).

El mismo mecanismo también se aplica para cada σ_i, haciendo que incluso en casos donde inicialmente $\sigma_i = \sigma$ para todo *i*, estos puedan variar de forma independiente con el correr de las generaciones[5].

En la *Recombinación Global Intermedia* [Bäck1996], cada componente se obtiene promediando los componentes en la misma posición en cada padre. Bäck distingue dos variantes:

[5] Esto es particularmente útil cuando se utilizan reglas similares a la regla de 1/5, donde se aplica un único factor para todos los desvíos de un individuo

- RGI: Se seleccionan dos padres al azar S y T y cada x_i se calcula como $\left(x_{S,i} + x_{T,i}\right)/2$

- RGI Panmíctica: Se selecciona un padre S al azar y, para cada componente x_i se selecciona un segundo padre T_i, también de forma aleatoria, resultando luego $x_i = \left(x_{S,i} + x_{T_i,i}\right)/2$

2.2 Programación Evolutiva

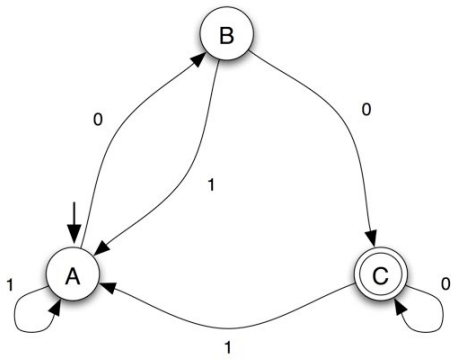
Introducida originalmente como una técnica para la simulación de comportamiento inteligente a mediados de los '60 [Fogel1966] [Dumitrescu2000], la Programación Evolutiva (PE) se basaba en evolución de poblaciones de máquinas de estado finito cuya tarea era predecir un patrón de símbolos, dada una serie de símbolos inicial.

Actualmente la variante con mayor difusión y aplicación presenta muchas similitudes con las Estrategias Evolutivas, tanto en su uso de variables reales como por su utilización de técnicas de auto-adaptación de parámetros evolutivos.

2.2.1 Estructura de Representación (Original)

A diferencia de otros algoritmos evolutivos, la Programación Evolutiva no posee una forma de representación estándar, sino que esta se adapta a cada problema en particular. Otra diferencia importante es que sólo se aplica selección y mutación, sin recombinación. El operador de mutación es también dependiente del problema, siendo necesario definirlo de acuerdo al dominio en que se va a aplicar y la forma de representación seleccionada. [Jacob2001]

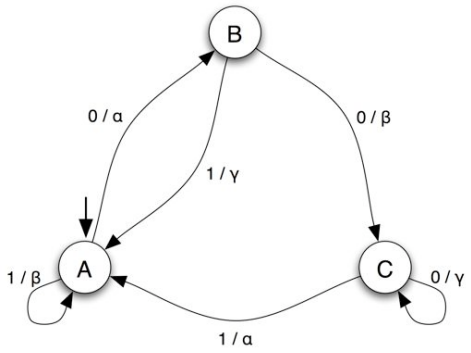
La primera aplicación de PE utilizaba como forma de representación máquinas de estado finito, con el fin de predecir cadenas de símbolos. Una máquina de estado finito (MEF) o Autómata Finito (AF) consiste en una serie de estados, transiciones de estado y señales de entrada y salida.



Estado actual	A	A	B	B	C	C
Símbolo de entrada	0	1	0	1	0	1
Estado siguiente	B	A	C	A	C	A

Figura 2.1: *Máquina de Estado Finito con tres estados: A (estado inicial), B y C (estado terminal). En este caso las señales de entrada (S_E) posibles son {0, 1} mientras que la salida (S_S) es binaria: "si"/"no" o "aceptación"/"rechazo" dependiendo de si la secuencia de señales de entrada finaliza con la MEF en un estado terminal o no.*

La máquina de estado finito utilizada por la PE es de un tipo particular denominado *máquina de estado finito Mealy* [Kolen2001], donde a diferencia de una MEF clásica *(fig. 2.1)* la salida depende no sólo del estado sino del estado y una señal de entrada, funcionando como transductor: para cada transición entre estados se indica con el formato *j / k* la señal de entrada (j) necesaria y la señal de salida (k) producida al pasar al estado destino *(fig. 2.2)*.



Estado actual	A	A	B	B	C	C
Símbolo de entrada	0	1	0	1	0	1
Símbolo de salida	α	β	β	γ	γ	α
Estado siguiente	B	A	C	A	C	A

Figura 2.2: *Máquina de Estado Finito tipo Mealy. Por cada símbolo o señal de entrada que origina una transición se genera una señal de salida. Las señales de entrada y las señales de salida pueden pertenecer a distintos lenguajes, en este caso: $S_E \in \{0,1\}$, $S_S \in \{\alpha,\beta,\gamma\}$*

Cada máquina Mealy es evaluada en función de su habilidad para predecir el "entorno". Dicho entorno consiste en una cadena binaria de *n*

digítos, $m < n$ de los cuales eran presentados como entorno inicial y utilizados por cada máquina como base para la predicción de los siguientes símbolos. La fracción de símbolos correctamente predichos determina el puntaje obtenido. Por ejemplo, para el ambiente cíclico (01101)*, tomando como ambiente inicial 011 y evaluando la performance de predicción en los siguientes ocho símbolos, la máquina *a* de la figura 2.3 tiene un puntaje de 0.375 (3/8) mientras que la máquina *b* tiene un puntaje de 1 (8/8), es decir, 100% correcto (fig. 2.3).

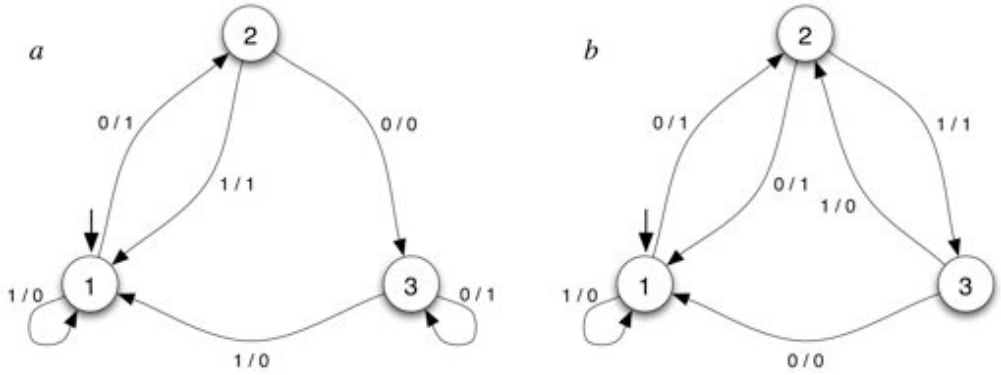


Figura 2.3: *Máquinas de Estado Finito tipo Mealy utilizadas como predictores. Dado un ambiente cíclico (01101)* del cual se toman los primeros tres símbolos como ambiente inicial, en a) la máquina produce los símbolos 01101101 como predicción para las próximas ocho entradas, mientras que en b) la máquina predice 01011010*

2.2.2 Algoritmo General

El algoritmo general es similar a otros algoritmos evolutivos, en particular a las estrategias evolutivas EE-($\mu+\lambda$):

- Se comienza con una población inicial de μ individuos generados de forma aleatoria,
- Se seleccionan λ individuos de forma aleatoria, con $\lambda \leq \mu$[6]

[6] En los primeros experimentos con PE cada individuo generaba un único hijo que luego era sometido a mutación, es decir, $\lambda = \mu$. El torneo para seleccionar los μ componentes de la siguiente generación se realizaba por lo tanto sobre un total de 2μ individuos: μ padres y sus repectivos hijos. [Fogel1999]

- Se aplica un operador de mutación a cada uno de los individuos seleccionados, generando λ nuevos individuos hijos
- El conjunto de padres e hijos constituye una nueva población de μ+λ individuos
- Se aplica un método de torneos para seleccionar un total de μ individuos que pasarán a ser la próxima generación de posibles padres (volviendo luego al segundo paso)

Como se puede observar, el algoritmo es similar al de las estrategias evolutivas del tipo EE-(μ+λ) dado que la selección de los futuros componentes de la población se hace sobre el conjunto de la población actual μ + el conjunto de λ hijos. Sin embargo, hay diferencias importantes tanto en como se generan los λ hijos como así también en la manera de seleccionar los μ componentes de la siguiente generación.

2.1.3 Operadores

La PE se basa en el uso de la mutación y la selección como factores evolutivos, sin utilizar el entrecruzamiento o recombinación.

2.1.3.1 Mutación

Al igual que la estructura de representación, el operador de mutación en la Programación Evolutiva no tiene una forma estándar, sino que debe diseñarse de acuerdo al problema.

Como se describió previamente, en sus orígenes la PE utilizaba como forma de representación máquinas de estado finito. Esto implica operadores de mutación específicos que permitan alterar dichas máquinas de diversas formas.

Se identificaron cinco posibles alteraciones [Fogel1999], para las que se definieron sus correspondientes operadores de mutación:

- Agregar un estado: el operador introduce un nuevo estado en la MEF, siempre que no se supere un límite preestablecido de cantidad de estados
- Eliminar un estado: el operador elimina un estado existente seleccionado al azar, siempre y cuando exista más de uno, así como también todas las transiciones que apuntan a este. Alternativamente, en lugar de eliminar transiciones estas son redireccionadas de forma aleatoria a otros estados
- Modificar el símbolo de salida asociado a una transición de estado: el operador selecciona al azar una transición y altera el símbolo de salida asociado a ésta, seleccionándolo del conjunto de símbolos salida permitidos, también de forma aleatoria.
- Modificar una transición de estado asociada a una entrada: el operador establece aleatoriamente un nuevo estado destino para una transición
- Cambiar el estado inicial: en caso de existir más de un estado en la MEF, el operador selecciona de forma aleatoria y con probabilidad uniforme uno de ellos como estado inicial

La mutación consiste entonces en seleccionar y luego aplicar uno de los cinco operadores de mutación. La selección de un operador se realiza aleatoriamente y, por lo general, con una probabilidad uniforme. [Kennedy2001]

2.1.3.2 Selección

En los primeros experimentos [Fogel1999] la selección de los μ individuos se realizaba evaluando los $\mu+\lambda$ individuos y conservando los mejores μ. En desarrollos posteriores se comenzó a utilizar un método de selección denominado *selección por torneos*.

En la selección por torneos cada individuo $i \in \{1, \ldots, \mu+\lambda\}$ debe competir con $c < \mu+\lambda$ competidores[7], seleccionados aleatoriamente, donde el parámetro c constituye el tamaño del torneo. [Jacob2001] Se procede a comparar el fitness del individuo con el de cada competidor (calculado como proporción de aciertos en la predicción), sumando un punto por cada "victoria". Luego, se ordenan todos los individuos de acuerdo a su puntaje, conservando los primeros μ. Esta metodología implica que ocasionalmente algunos individuos pasarán a formar parte de la próxima generación a pesar de no estar entre los mejores μ si la selección se hiciera sólo en base al fitness, dado que existe un componente aleatorio (selección de contrincantes) en el proceso. El objetivo es conservar una mayor diversidad, con el fin de disminuír la posibilidad de una convergencia prematura en soluciones sub-óptimas (máximos locales).

2.1.4 Variante "meta-PE"

La programación evolutiva comenzó utilizando MEF como forma de representación. Desde entonces, se han desarrollado avances permitiendo la utilización de variables continuas y la posibilidad de auto-adaptación de parámetros. [Kennedy2001]

La aplicación más común de la PE es la optimización de funciones del tipo:

$$f : \mathbb{R}^n \rightarrow \mathbb{R}$$

siendo la representación utilizada en estos casos un vector de valores reales:

[7] Cuanto mayor sea el c elegido más se reducirá el componente estocástico y por lo tanto más se acercará la selección a la obtenida por el método determinista original: en el caso extremo de que cada individuo compita con todos los demás, el orden resultante de contar sus victorias será equivalente al orden por fitness directo.

$$\vec{x} = \langle x_1, ..., x_n \rangle \in \mathbb{R}^n$$

En forma similar a las Estrategias Evolutivas, una variante de la Programación Evolutiva denominada meta-PE agrega un segundo vector de parámetros estrategia a cada individuo para permitir la auto-adaptación de parámetros de mutación [Eiben2003]:

$$\langle x_1, ..., x_n, \sigma_1, ..., \sigma_n \rangle$$

En este caso, la mutación suele tomar la siguiente forma:

$$\sigma_i' = \sigma_i \cdot (1 + \alpha \cdot N(0,1))$$
$$x_i' = x_i + \sigma_i' \cdot N_i(0,1))$$

donde N(0, 1) representa un valor aleatorio obtenido de una distribución normal con media cero y desvío estándar de 1 y $\alpha \approx 0.2$. Otras formas de adaptación de parámetros implican el uso de distribuciones log-normales, de manera similar a las EE. Otra variante que adopta aún más aspectos de las EE es la denominada R-meta-PE, en la cual se utiliza una matriz de covarianzas para permitir no sólo la adaptación de los niveles de mutación sino también la dirección de dicha mutación (ángulos de rotación), definiendo hiper-elipsoides de mutación [Eiben2003][Bäck1993].

2.3 Programación Genética

Introducida a principios de los 90 [Koza1990][Koza1992], la Programación Genética (PG) comenzó como una extensión de los Algoritmos Genéticos, bajo la denominación *Algoritmo Genético Jerárquico* [Koza1989].

La PG utiliza una *población de programas*, la cual es sujeta a transformaciones estocásticas (aplicación de operadores genéticos) y selección a través de sucesivas generaciones.

El algoritmo general consiste en los siguientes pasos [Poli2008]:

- Generar una población inicial de programas de forma aleatoria utilizando las primitivas disponibles
- Ejecutar cada programa y determinar su fitness
- Seleccionar uno o dos programas de la población con una probabilidad basada en el fitness
- Crear nuevos programas aplicando con probabilidades pre-especificadas operadores genéticos a el o los programas seleccionados en el paso anterior
- Repetir a partir del segundo paso mientras no se haya llegado a una solución aceptable o a alguna condición de parada determinada (por lo general, un número máximo de generaciones)
- Finalmente, retornar como solución el programa con mayor fitness obtenido

2.3.1 Primitivas

Los componentes utilizados por la PG se denominan primitivas y se dividen en dos conjuntos: Conjunto de Terminales y Conjunto de Funciones.

2.3.1.1 Terminales

Las terminales incluyen *entradas*, *constantes* y *funciones sin argumentos*.

Las *entradas* constituyen las variables de entrada del programa y sus valores provendrán del entorno de evaluación.

Las *constantes* pueden ser predefinidas, generadas de forma aleatoria al inicio del proceso evolutivo o creadas por mutación. Las constantes generadas de forma aleatoria al inicio del proceso se suelen indicar incluyendo una terminal denominada *constante aleatoria efímera (ephemereal random constant)* representada con el símbolo ℜ en el conjunto de terminales. Cada vez que la terminal ℜ es seleccionada en la construcción de un individuo al inicio del proceso (es decir, durante la generación de la población inicial) se genera un valor aleatorio, el cual permanecerá constante en las subsiguientes generaciones. [Banzhaf1998]

Las *funciones sin argumentos* pueden ser incluídas si ademas de retornar un valor provocan *efectos secundarios* al ser ejecutadas o si el valor que retornan puede variar cada vez que se invocan. [Poli2008]

2.3.1.2 Funciones

El Conjunto de Funciones puede abarcar funciones booleanas, aritméticas, trascendentales, etc., así como también ser específico al dominio del problema.

Un requisito importante a tener en cuenta en la selección de las funciones a incluir en el conjunto de funciones es el concepto de *clausura*: las funciones deben retornar un valor para cualquier entrada posible, sin producir errores o excepciones que obliguen a detener el programa. Esto implica contemplar casos particulares reemplazando ciertas funciones por versiones

protegidas, donde cualquier posible excepción es contemplada y un valor aceptable predefinido es devuelto como resultado. Por ejemplo, la operación de división puede reemplazarse por una *división protegida*, la cual contempla la división por cero retornando un valor predeterminado.

Otro aspecto importante en la clausura es la necesidad de garantizar que los tipos de datos sean consistentes, es decir, las entradas y salidas de todas las funciones deben ser del mismo tipo o, alternativamente, debe contemplarse una trasformación automática de tipos. Por ejemplo, si una función necesita un valor booleano como entrada pero recibe un valor entero de otra función, se puede utilizar una regla que convierta dicho valor a *falso* si el entero es igual a cero y *verdadero* en cualquier otro caso. Una tercera alternativa, más compleja, es adapatar los operadores genéticos para que al aplicar transformaciones tengan en cuenta los tipos de datos involucrados, evitando producir alteraciones que ocasionen una combinación de funciones y datos de tipo incorrecto.

El segundo requisito a tener en cuenta para la selección de funciones (y de primitivas en general) es la *suficiencia*: El conjunto de funciones y terminales deben ser suficientes para poder expresar la solución como una combinación de estos. Esto puede ser difícil de determinar y dependerá del dominio del problema. En aquellos casos donde el conjunto de primitivas resulte insuficiente, sólo exisitirá la posibilidad de evolucionar una solución aproximada. Si el conjunto de funciones es mayor al mínimo necesario, el efecto sobre la velocidad de obtención de una buena solución es variable, mientras que si el conjunto de terminales es mayor al necesario el efecto suele ser negativo. [Koza1992]

2.3.2 Estructura de Representación

La estructura de representación más utilizada en la PG son los árboles de sintaxis *(fig. 2.4)*, los cuales permiten una organización jerárquica de sus componentes.

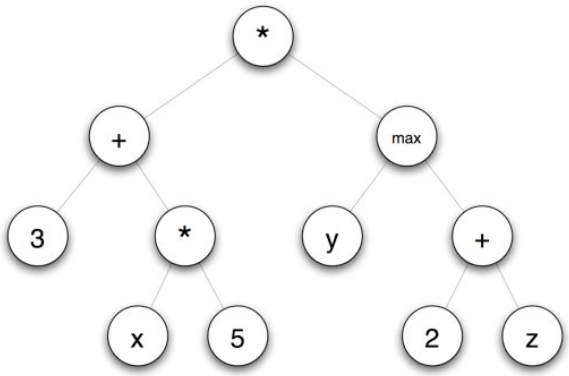


Figura 2.4: *Árbol de sintaxis para la expresión (3+x*5)*max(y,2+z)*

Las estructuras suelen expresarse con notación polaca o prefija. En particular, Koza utilizó el lenguaje LISP para sus experimentos iniciales, donde se utiliza el formato (función parámetro_1 … parámetro_n). La estructura de la figura 2.4 puede entonces expresarse como:

((+ 3 (* x 5)) (max y (+ 2 z)))*

Los nodos internos del árbol corresponden al conjunto de funciones, mientras que las hojas o nodos terminales corresponden al conjunto de terminales[8] (entradas, constantes y funciones sin argumentos). En este caso:

$F = \{max, *, +\}$ $\quad$ $T = \{x, y, z, 2, 3, 5\}$

[8] El conjunto de terminales se denominó así con el fin de reflejar su posición utilizando la terminología de árboles. Si bien actualmente también existen en la PG otros tipos de representaciones (lineal, grafos), se sigue utilizando la misma denominación.

Como se describió en la sección 2.3, el primer paso del algortimo evolutivo utilizado en la PG consiste en generar una población inicial compuesta de individuos (árboles) generados de forma aleatoria. Existen varias maneras de generar cada individuo. Dos métodos básicos son los denominados *árbol completo* y *crecimiento*. Ambos toman como parámetro un valor de profundidad máxima que determina cual será el máximo nivel posible para el árbol generado, considerándose que el nivel se determina contando la cantidad de segmentos necesarios para alcanzar otro nodo a partir nodo raíz (nivel cero).

En el método del árbol completo, se procede de forma recursiva incorporando al árbol nodos seleccionados del conjunto de funciones, conectando nuevos nodos a cada función, uno por cada argumento, mientras el nivel sea menor al nivel máximo permitido. Luego, se conectan nodos tomados del conjunto de terminales, nuevamente respetando la cantidad de argumentos esperado por cada función. *(fig 2.5)*

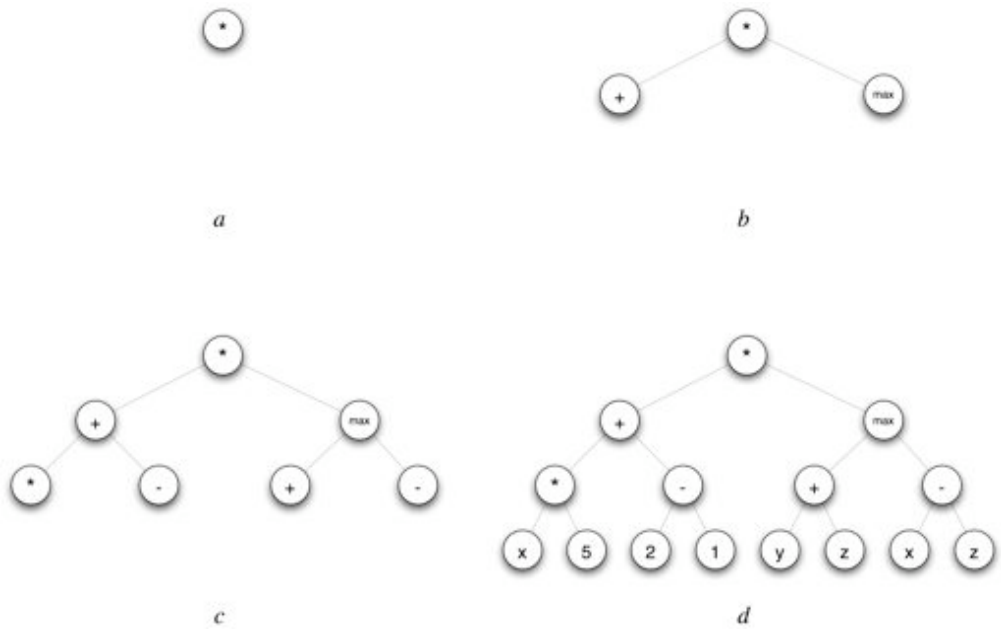


Figura 2.5: *Desarrollo de árbol por el método de árbol completo para un nivel máximo de 3. En los niveles 0, 1 y 2 (subfiguras a, b, c) se agregan nodos tomados del conjunto de funciones F = {max, *, +}, mientras que en el último nivel (subfigura d) se toman del conjunto de terminales T = {x, y, z, 1, 2, 5}.*

En el método de crecimiento el proceso es similar, con la diferencia de que los nodos se seleccionan del conjunto de todas las primitivas (funciones y terminales) mientras el nivel sea menor al nivel máximo permitido. Luego, se conectan nodos tomados exclusivamente del conjunto de terminales. A diferencia del método del árbol completo este método no garantiza que la profundidad final sea igual al nivel máximo, sólo garantiza que será menor o igual. *(fig 2.6)*

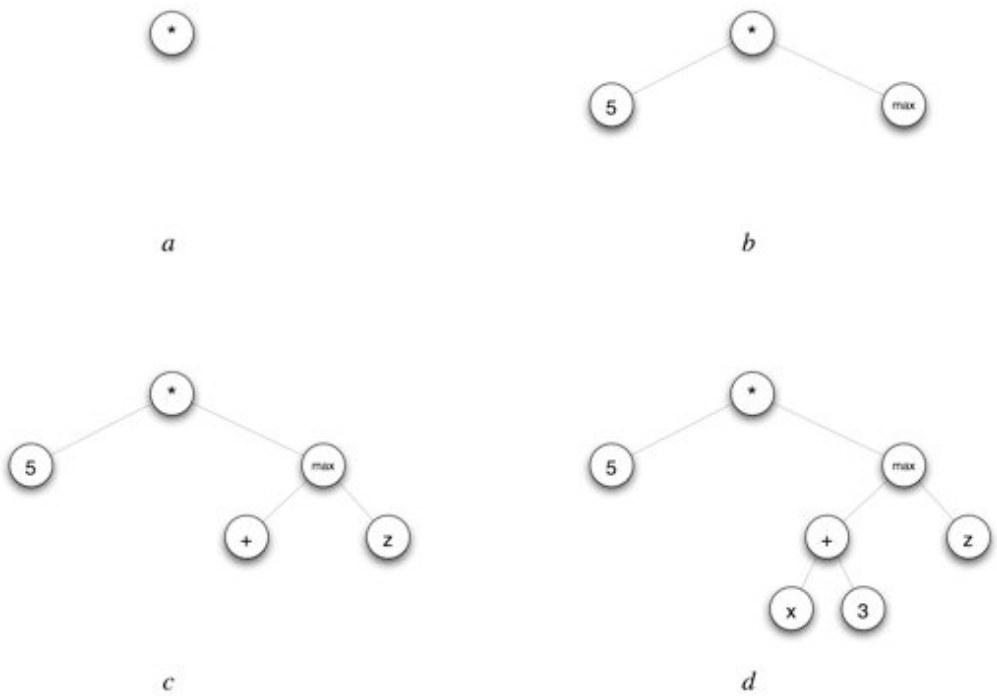


Figura 2.6: *Desarrollo de árbol por el método de crecimiento. En cada etapa se agregan nodos tomados del conjunto completo de las primitivas P = {max, *, +, x, z, 3, 5}, conectando (b, c) nodos en cada nivel con los nodos de función en el nivel anterior y se continúa así hasta llegar al nivel máximo (subfigura d) donde los nodos a conectar se toman exclusivamente del conjunto de terminales T = {x, z, 3, 5}.*

Para obtener una mayor variedad de formas y tamaños de árboles Koza [Koza1992] propuso un método de inicialización denominado *ramped half-and-half*. En éste método, la mitad de los individuos se generan con el método de árbol completo y la otra mitad con el método de crecimiento. En todos los casos se usa un rango de profundidades máximas en lugar de un único valor global.

Una alternativa a la inicialización aleatoria descripta es utilizar un proceso de "sembrado" de soluciones aproximadas o parciales conocidas en la población inicial. En este caso, es conveniente que la población inicial este compuesta por *variantes* de dichas soluciones aproximadas y no sólo por una mezcla de soluciones generadas manualmente y soluciones generadas en forma aleatoria. La razón es que de no procederse así la diferencia en fitness entre ambos grupos originará rápidamente la desaparición, con el correr de las generaciones, de las soluciones aleatorias iniciales, reduciendo la diversidad de los individuos y dificultando una exploración adecuada del espacio de solución.

Además de la representación por árboles, existen variantes de PG lineales y basadas en grafos. Las variantes lineales utilizan registros, permitiendo el acceso global a valores, a diferencia de lo que ocurre en la representación por grafos, donde los valores sólo existen de manera local y están disponibles sólo para la función a la que están conectados. En el caso de la PG basada en grafos, el transporte de información entre nodos se hace a través de una pila, existiendo además un espacio de memoria indexada de acceso global. Mas información sobre éstas y otras variantes puede encontrarse en [Banzhaf1998][Poli2008][Brameier2004].

2.3.3 Operadores Genéticos

La programación genética se originó a partir de los algoritmos genéticos. Sin embargo, en sus inicios la PG no utilizaba el operador de mutación sino que se limitaba exclusivamente al operador de entrecruzamiento.

2.3.3.1 Entrecruzamiento

El intercambio de componentes entre individuos (programas) en la PG, cuando se utiliza la representación de árbol, se realiza a nivel de subramas

completas (subrutinas). La versión más utilizada genera un nuevo individuo a partir de dos padres:

- Dados dos individuos, a y b, se seleccionan puntos de corte (nodos) en forma aleatoria[9], uno en cada individuo, pa y pb
- Se realiza una copia del individuo a reemplazando la subrama a partir del nodo seleccionado como p_a por la subrama a partir de p_b del individuo *b*, generando el individuo hijo *a'*. *(fig. 2.7)*

Otra alternativa, aunque no muy utilizada en la PG, es generar dos hijos en lugar de uno:

- Dados dos individuos, a y b, se seleccionan puntos de corte (nodos) en forma aleatoria, uno en cada individuo, pa y pb
- Se realiza una copia del individuo *a* reemplazando la subrama a partir del nodo seleccionado como p_a por la subrama a partir de p_b del individuo *b*, generando el individuo hijo *a'*. Se procede de manera similar con el individuo *b*, generando un hijo *b'* en el cual la subrama a partir de p_b es reemplazada por la subrama a partir de p_a del individuo *a*. *(fig. 2.7)*

[9] Dada la distribución general de nodos en un árbol típico la selección completamente aleatoria tiende a seleccionar subramas relativamente pequeñas (incluso terminales) por lo que una opción comúnmente usada es darle mayor probabilidad a la selección de funciones.

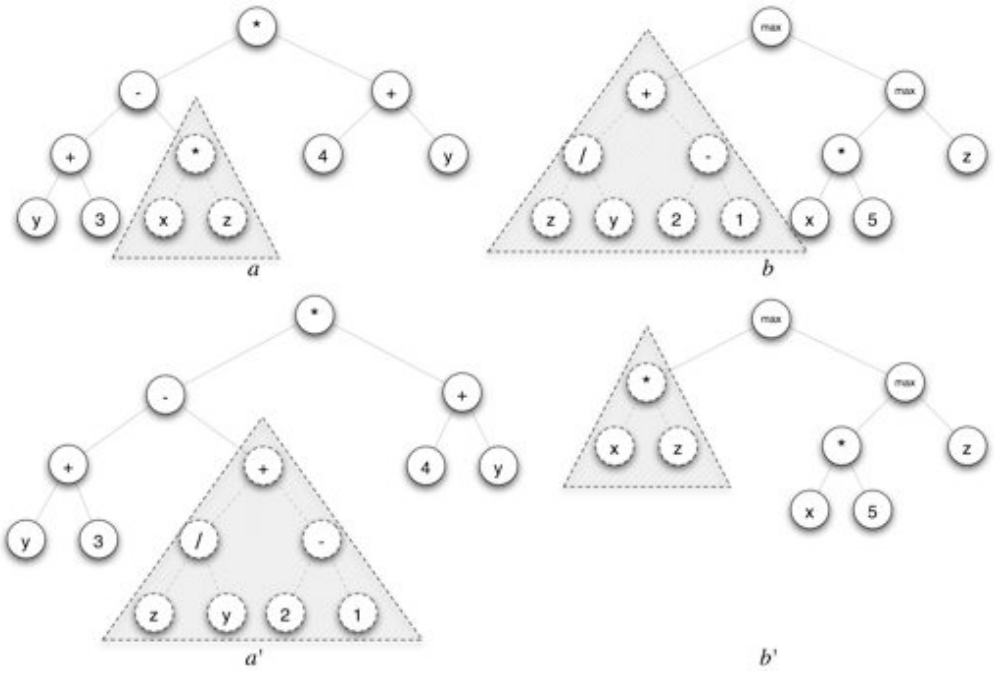


Figura 2.7: *Entrecruzamiento entre dos individuos, a y b. Se selecciona aleatoriamente un nodo en cada árbol como puntos de corte y luego dichos nodos y toda la subrama que depende de cada uno son intercambiados, generando un nuevo individuo a' y, adicionalmente en el caso del crossover de dos hijos, b'.*

A diferencia de los que sucede en los AG *(capítulo 3)*, en la PG el operador de mutación no mantiene el contexto de la estructura dado que los puntos de cortes se seleccionan aleatoriamente en ambos individuos. Esto implica que si bien la PG tiende a una convergencia funcional en sus individuos, estructuralmente esta convergencia no se produce. En algunos casos es deseable también la convergencia estructural dado que la solución es muy específica (por ejemplo, en árboles que representan estructuras de reglas de decisión IF-THEN) y el entrecruzamiento aleatorio suele tener efectos altamente negativos. Koza [Koza1994] planteó una alternativa para estos casos denominada *ADF (automatically defined functions / funciones definidas automáticamente)* en la cual subpartes de la solución se consideran funciones del árbol completo y evolucionan por separado. Esto facilita la reutilización de estructuras y genera individuos (programas) de menor tamaño. ADF implica, sin embargo, la especificación de parámetros adicionales y

subconjuntos de funciones y terminales a utilizar en el árbol principal y en los subárboles.

2.3.3.2 Mutación

Como se mencionó en el punto anterior, a diferencia de los AG la PG tiene una diferencia crucial en la selección de puntos de corte. Esto implica que el operador no sólo permite la combinación de bloques sino que genera nuevas estructuras, incluso cuando el entrecruzamiento se da entre dos individuos idénticos. Debido a esto, Koza sostiene que el entrecruzamiento es el único operador esencial de la PG y prácticamente no utilizó el operador de mutación en sus diversos experimentos. Su objetivo era demostrar que la selección proporcional junto con el operador de entrecruzamiento puede, por sus características distintivas en la PG, sostener el proceso evolutivo sin causar una progresiva pérdida de diversidad en la población que cree convergencias prematuras.

En la actualidad se suelen utilizar los operadores entrecruzamiento y mutación en forma probabilística, con una probabilidad del 90% de aplicar el operador de entrecruzamiento y de 1 a 10% de aplicar el operador de mutación[10]. El uso de estas proporciones se basan en gran medida en los resultados experimentales originales obtenidos por Koza [Koza1992]. Experimentos posteriores realizados por otros investigadores con distintas proporciones de operadores indican que si bien el entrecruzamiento es más efectivo que la mutación, esto sólo se da en condiciones determinadas (particularmente en poblaciones grandes) e incluso en esos casos, la diferencia es muy leve [Luke1998].

[10] En aquellos casos donde se definen probabildades de entrectruzamiento y de mutación tal que $p_e + p_m < 1$, se utiliza un operador de reproducción (copia sin cambios de un individuo) cuando no resultan seleccionados ni el operador de entrecruzamiento ni el de mutación.

Hay dos tipos de mutación principales: *mutación de subárbol* y *mutación puntual*, siendo la mutación de subárbol la más comumente utilizada.

Se puede considerar a la mutación de subárbol como un entrecruzamiento especial. En lugar de seleccionar puntos de corte en dos individuos, sólo se selecciona un punto de corte en un individuo y el subárbol es reemplazado no por un subárbol de otro individuo sino por un nuevo subárbol generado aleatoriamente. *(fig. 2.8)*

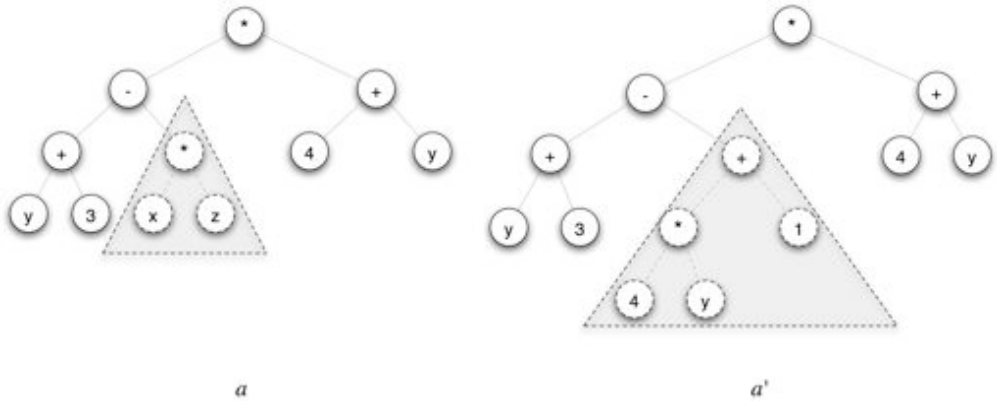


Figura 2.8: *Mutación "subárbol" de un individuo. Dado un individuo (a) se selecciona aleatoriamente uno de sus nodos, reemplazandolo junto con toda la subrama que depende de éste por un nuevo árbol generado por el método de crecimiento (u otro método) obteniéndose un nuevo individuo (a').*

La mutación puntual, por otro lado, consiste en la alteración no a nivel de subárboles sino a nivel de nodos (primitivas). La mutación consiste en el reemplazo de una primitiva por otra de la misma aridad (cantidad de parámetros, cero en el caso de una terminal). A diferencia de la mutación subárbol, la mutación puntual se aplica en forma probabilística a todos los nodos de un individuo. *(fig. 2.9)*

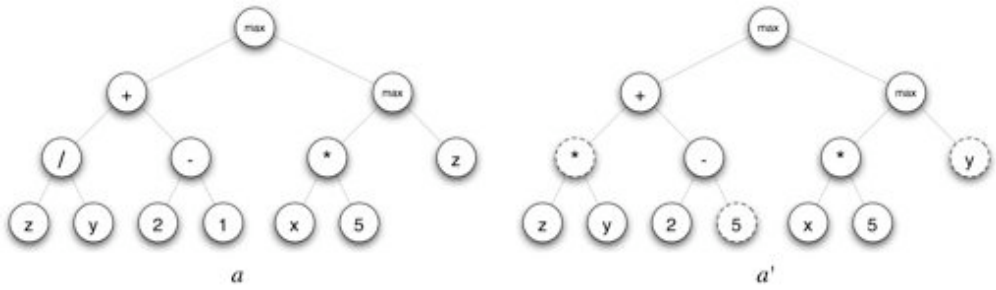


Figura 2.9: *Mutación puntual de un individuo. Dado un individuo (a) se recorren todos los nodos y por cada uno se decide con una probabilidid predeterminada la aplicación o no de una mutación puntual. Esta consiste en reemplazar la primitiva del nodo por otra con la misma aridad (cantidad de parámetros). En el caso de las terminales, estas se consideran de aridad cero. Al finalizar el proceso se obteniene un nuevo individuo (a'). En este ejemplo la mutación alteró tres nodos: una función y dos terminales.*

2.3.4 Selección

Existen distintas formas de seleccionar los individuos a los cuales se les aplicará operadores genéticos para generar nuevos individuos. La selección proporcional basada en el fitness y la selección por torneos descriptas en secciones anteriores son las de mayor difusión en la PG.

2.3.5 Fitness

Para evaluar el fitness de un individuo (programa) este es ejecutado para una cierta cantidad de casos de prueba, los cuales proveen los valores para las terminales de entrada. La salida del programa es comparada con el valor esperado en cada caso de prueba, obteniendose la suma de los errores absolutos[11]. El *fitness estandarizado* para *n* casos de prueba se calcula como:

$$\frac{1}{1+\sum_{i=1}^{n}|y_i - y_i'|}$$

[11] También puede utilizarse la suma de errores cuadrados: $\sum_{i=1}^{n}(y_i - y_i')^2$

Una vez calculado el fitness estandarizado de cada individuo se calcula el *fitness normalizado*, de forma que la suma del total de los individuos sea 1 y que por lo tanto el fitness normalizado represente la probabilidad de ser seleccionado que le corresponde a cada individuo.

Capítulo 3

3 Algoritmos Genéticos

Los algoritmos genéticos (AG) son métodos de optimización estocásticos altamente paralelizables y, como subrama de los AE, están inspirados en la evolución por medio de selección natural.

Un AG utiliza una *población de cromosomas*. Estas codifican soluciones a un problema dado en forma de una cadena de símbolos, generalmente binarios. La composición de dicha cadena determina el *genotipo* de un individuo. Esta cadena es evaluada por una *función objetivo* particular al dominio del problema obteniendo así un puntaje, llamado *nivel de adaptación* o *fitness*. Nuevas generaciones reemplazan a la población actual con individuos generados a partir de esta por medio del uso de *procesos de selección* y *operadores genéticos*, como mutación, entrecruzamiento e inversión.

Los AG presentan el atractivo de ser fácilmente implementables y aplicables a problemas de clases muy diversas. Son particularmente usados en aplicaciones donde una solución analítica es impracticable, ya sea por su complejidad o por falta de una teoría que se pueda utilizar como base para el análisis, pero en las cuales es relativamente simple evaluar la calidad de una solución dada.

Como ejemplos de áreas de aplicación se pueden mencionar, entre otros:

- Bioquímica: Predicción de estructuras de moléculas complejas [Kaiser1997]
- Economía y finanzas [Mehta1999][Bauer1994]

- Diseño industrial y hardware evolutivo [Thompson1998] [Castellani2003] [Lohn2005]
- Evolución de redes neuronales [Montana1989]
- Optimización de parámetros de comportamiento de oponentes autónomos en juegos [van Waveren2001]
- Asignación y programación de tareas
- Robótica: evolución de controladores y estructuras físicas (*Capítulo 4)*

3.1 Genotipo y Fenotipo

Se denomina *genoma* al total de información genética de un organismo, mientras que el conjunto particular de genes de instancia específica (un individuo) se denomina *genotipo*.

El *fenotipo* es la expresión observable del genotipo de un individuo. La relación entre genes y características observables no es necesariamente uno a una: un gen puede determinar una característica observable, varias *(pleitropia)*, o actuar en combinación con otros genes para determinar una única característica *(poligenia)*. Si un gen no determina ninguna característica observable, se dice que es un gen *no expresado*. Aquellos genes cuyos efectos fenotípicos no se ven afectados por otros se consideran *ortogonales* entre si. [Dumitrescu2000]

3.2 Esquemas (Schemata)

En su trabajo inicial Holland describía, como una característica crucial del funcionamiento de los AG, la forma altamente paralela en la que buscan y combinan *bloques de construcción* útiles para alcanzar la solución de un problema dado.

Para explicar este comportamiento formalmente, Holland definió el concepto de *esquema*. Un esquema es un conjunto de cadenas de símbolos, binarios según la definición original, que se pueden representar con una cadena expresada también con ceros y unos, pero con el agregado de un tercer símbolo: un comodín, que se indica con un asterisco. Por ejemplo, para representar el conjunto de cadenas de 5 elementos que empiezan con 11 y terminan con 0 se puede utilizar el esquema:

E = 11**0

Cadenas como *11110* y *11010* se denominan *instancias* del esquema *E*. El *orden* de un esquema está determinado por el número de bits fijos (bits definidos), en este caso *E* es de orden 3. Otra propiedad importante de un esquema es denominada *longitud de definición*, distancia entre los bits definidos más externos: *ld = 4* para el esquema *E* del ejemplo.

Se considera que los esquemas útiles como bloques de construcción para el progreso de un AG son aquellos con una longitud de definición corta y cuyas instancias muestran una performance mayor al promedio de la población.

3.3 Asignación Óptima de Pruebas

Un problema común en optimización es encontrar el balance entre *exploración* y *explotación*: por un lado, es necesario explorar nuevas soluciones, por el otro, es necesario concentrarse en las mejores soluciones encontradas hasta el momento y mejorarlas. Esto se conoce como *asignación óptima de pruebas*, y es una de las características centrales del funcionamiento y generalidad de los AG.

3.3.1 Tragamonedas de dos brazos (2-armed bandit)

Un modelo clásico del problema de asignación óptima de pruebas es el del *tragamonedas de dos brazos*.

Dado un tragamonedas con dos brazos en el cual cada brazo otorga un premio con una media y varianzas fijas pero desconocidas, el objetivo es obtener la mayor ganancia dado un número limitado de pruebas. Este tipo de objetivo se conoce como performance *en línea*, es decir, el proceso se debe optimizar de forma continua durante su ejecución, a diferencia de otros problemas clásicos de optimización.

En su análisis de este problema, Holland [Holland1975] muestra que el mejor curso de acción es asignar pruebas al brazo con mayor media observada hasta el momento con una probabilidad cada vez mayor, incrementando esta en forma exponencial con respecto a la probabilidad de seleccionar el otro brazo.

3.3.2 Esquemas y la asignación óptima de pruebas

En la sección 3.1 donde se introdujo el concepto de esquema, se indicaba que se consideran útiles a aquellos esquemas cuyas instancias muestran una performance mayor al promedio. Sin embargo, esto implica una estimación, ya que no todas las instancias posibles de un esquema dado van a estar presentes necesariamente en la población y la performance de un esquema puede definirse como el promedio de la performance de todas sus posibles instancias. En particular, dados dos esquemas E y E' y una población finita, puede resultar que el cálculo de los promedios de las instancias presentes de E y de E' indique que E tiene una media mayor a E', pero que en realidad esto se deba simplemente a un error de muestreo por no contar con suficientes instancias.

Por un lado, si un esquema tiene una buena performace observada tiene sentido generar más instancias de este con el objetivo de obtener una mejor solución. Por el otro, el error de muestreo puede hacer que se concentre la generación de instancias en el esquema incorrecto y para evitar esto también se deberían generar instancias de otros esquemas con el fin de estimar mejor su verdadera performance. Nuevamente se presenta el problema de determinar un adecuado balance entre explotación y exploración.

Tomando como base la sección anterior sobre asignación óptima de pruebas y en particular el ejemplo del tragamonedas, la mejor política consistiría en asignar pruebas (es decir, generar instancias) con una proporción mayor a aquellos esquemas cuya performance observada sea mejor, incrementando esta proporción de forma exponencial.

Esta política es aplicada por los AG por medio de un *paralelismo intrínseco*, procesando esquemas indirectamente a través de la selección de instancias y la utilización de operadores genéticos como mutación y entrecruzamiento [Holland1975][Goldberg1989].

3.4 Estructura de Representación

Los algoritmos genéticos generalmente utilizan una representación lineal basada en codificación binaria, dado que esta codificación es la mejor desde el punto de vista de la teoría de esquemas (sección 3.2) propuesta originalmente por Holland [Holland1975]. La justificación, según lo expresa Holland, es que una codificación binaria procesa implícitamente más esquemas que una representación en una base mayor.

Cuando se utiliza esta codificación para representar parámetros enteros, puede presentarse un problema denominado *pared de Hamming (Hamming wall)*. En la codificación binaria estándar ciertos valores enteros consecutivos pueden ser muy diferentes desde el punto de vista de la secuencia de unos y ceros que los representan. Por ejemplo:

22 = 10110
23 = 10111
24 = 11000

La cantidad de posiciones en que una cadena difiere con otra se denomina *distancia Hamming (d_h)*. Una mutación de un solo bit sería suficiente para pasar del valor representado 22 al 23 ($d_h = 1$), pero para pasar de 23 a 24 se necesitaría alterar demasiados bits simultáneamente ($d_h = 4$), es decir, existe una dificultad artificial impuesta por la codificación. Para evitar este tipo de problemas se suele utilizar una codificación denominada *gray-coding* en la cual valores consecutivos difieren siempre en un solo bit ($d_h = 1$) en su representación binaria. En [Caruana1988], [Whitley1997], [Whitley1999] y [Greiner2005] se pueden encontrar comparaciones empíricas y teóricas entre la representación binaria estándar y la representación binaria con gray-coding.

Una forma de calcular un gray-code para un valor entero expresado con codificación binaria estándar consiste en desplazar sus bits una posición a la derecha, aplicando luego el operador XOR (operador "o exclusivo") entre ésta y la secuencia original de bits. Para los ejemplos planteados originalmente resulta:

	Valor representado	***22***	***23***	***24***
a	*Codificación estándar*	***10110***	***10111***	***11000***
b	*a >> 1*	*01011*	*01011*	*01100*
c	*Gray-code (a XOR b)*	***11101***	***11100***	***10100***

Tabla 3.1: *Ejemplo de obtención de gray-code a partir de codificación binaria estándar*

3.4.1 Representaciones alternativas

Si bien la codificación binaria es la más utilizada, otras formas de representación pueden ser necesarias dependiendo del tipo de problema. Por

ejemplo, en problemas combinatorios la representación binaria puede resultar demasiado compleja (dados *N* elementos, se necesitan $log_2(N!)$ símbolos para poder representar cualquier solución), mientras que una representación con un alfabeto mayor resulta más natural y compacta. Además, la relación entre bits en posiciones cercanas puede no existir, debido a la falta de relación directa entre el espacio de solución del genotipo y del fenotipo, haciendo que esta representación no sea fácilmente explotable por un AG [Tate1993].

La teoría de esquemas de Holland indicaría que codificaciones no binarias deberían tener una menor performance, pero en el caso de los problemas combinatorios es común el uso alfabetos con mayor cantidad de símbolos, sin que genere esto controversias dado que se consideran una categoría especial de problemas.

Por otro lado, diversos autores sostienen que la representación binaria puede no ser la mejor en muchos casos generales. Por ejemplo, [Davis1991] sugiere utilizar siempre la representación que resulte más natural para el problema, presentando varios casos de aplicaciones híbridas en entornos industriales, con resultados empíricos positivos. En [Antonisse1989] se describe una forma alternativa de definir los esquemas según la cual una representación con alfabetos grandes presenta mejores características que una representación binaria. En [Janikow1991] se realiza una comparación entre representación binaria y representación por medio de números reales, sugiriendo operadores genéticos alternativos. En [Wright1991] también se analiza la representación en base a reales, definiéndose operadores de mutación y entrecruzamiento sobre cromosomas compuestas de valores reales. Se define a su vez una versión del concepto de esquemas pero aplicada a valores reales, mostrando que la teoría de esquemas de Holland puede aplicarse y, por medio de resultados experimentales, se muestra que dicha representación y conjunto de operadores resulta superior a la codificación y operadores clásicos en la mayoría de los casos evaluados. En [Goldberg1990] se realiza un análisis de AG que utilizan números reales o alfabetos de gran cardinalidad, mostrando como se puede aplicar la teoría de esquemas a estos y

la posibilidad de que existan bloqueos al progreso de un AG de este tipo bajo ciertas condiciones. En [Radcliffe1992], por otro lado, se argumenta que la teoría de esquemas no es adecuada para explicar la performance de un AG, presentándose varias objeciones a la interpretación tradicional de esquemas y a la capacidad de estos para representar "bloques de construcción" significativos. En [Altemberg1995] se hacen objeciones similares al significado que se le suele atribuír a los esquemas y se presentan formas alternativas de evaluar la performance de un AG.

3.5 Operadores Genéticos

Inspirados en la naturaleza, los algoritmos genéticos utilizan *operadores genéticos* para seleccionar, modificar y combinar cromosomas de una población, las cuales representan soluciones a un problema dado.

3.5.1 Mutación

Una mutación consiste en la alteración del valor en uno o más componentes de un cromosoma. El nivel de mutación se determina definiendo una probabilidad de mutación p_m que indica la probabilidad de que dada una posición, su valor sea alterado. En el caso de cadenas de bits, esta alteración consiste en el valor complementario (0 si el valor original era 1 y viceversa), mientras que en cadenas de valores reales suele seleccionarse un nuevo valor de forma aleatoria.

Este operador tiene una función exploratoria, brindando variabilidad necesaria para el correcto funcionamiento de los demás operadores y reduciendo las chances de que el AG se enfoque exclusivamente en máximos locales.

Figura 3.1: *Mutación. En este ejemplo, las posiciones 3 y 5 fueron afectadas por la mutación, modificando sus valores.*

Efecto sobre los esquemas

La probabilidad de supervivencia p_s de un esquema E en un individuo sujeto a mutación en sus componentes con probabilidad p_m se define como:

$$p_s(E) = (1 - p_m)^{o(E)}$$

donde *o(e)* indica el *orden* del esquema *E*.

3.5.2 Entrecruzamiento

Este operador permite generar individuos nuevos (hijos) a partir de la combinación de dos individuos de la generación actual (padres). Consiste de dos pasos: primero, se selecciona aleatoriamente un punto de corte dentro del genoma. Luego, se intercambian las cadenas a partir de dicho punto de corte. Esto es lo que se conoce como *entrecruzamiento simple* o *de un solo punto*.

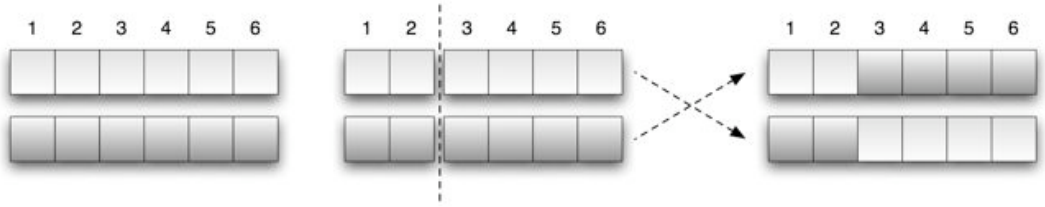


Figura 3.2: *Entrecruzamiento simple o de un solo punto*

También es posible realizar un *entrecruzamiento multipunto*, que opera de la misma manera que el entrecruzamiento simple con la diferencia de que se seleccionan aleatoriamente varios puntos de corte:

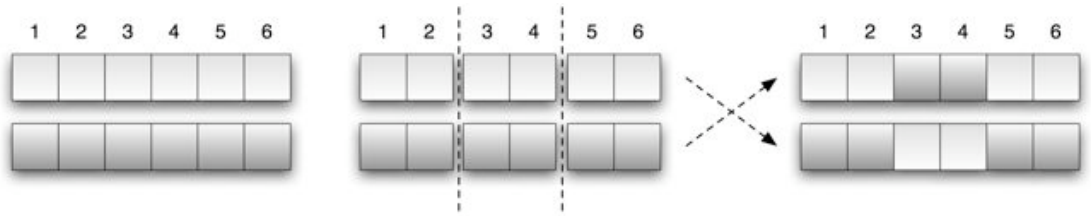


Figura 3.3: *Entrecruzamiento multipunto*

El entrecruzamiento es un operador fundamental de los AG, teniendo una función de explotación de soluciones parciales, combinándolas y favoreciendo bloques de construcción pequeños para conseguir progresivamente mejores soluciones.

Efecto sobre los esquemas

La probabilidad mínima de supervivencia p_s de un esquema E en un individuo sujeto a entrecruzamiento simple con otro individuo con probabilidad p_e se define como:

$$p_s(E) \geq 1 - p_e\left(\frac{ld(E)}{l}\right)$$

donde l indica la la longitud de las cadenas en la población y *ld(E)* la *longitud de definición* del esquema *E*.

3.5.3 Inversión

Como se mencionó en secciones anteriores, el entrecruzamiento posibilita la combinación de características útiles, *bloques de construcción* formados por bits cercanos, que favorecen al individuo que los posee. Sin embargo, la probabilidad de que 2 bits se mantengan juntos luego de un entrecruzamiento depende directamente de la distancia entre ellos. Debido a esto, es importante para el buen funcionamiento de un AG que al seleccionar el método de codificación se sitúen en posiciones cercanas aquellos parámetros que se estima tienen una relación importante. Claramente esto no siempre es factible o simple de determinar, por lo que Holland incluyó en su

propuesta original [Holland1975] un operador adicional a sus operadores genéticos básicos, el operador *inversión*, también inspirado en la biología.

En la naturaleza, la función que cumplen los genes es hasta cierto punto independiente de su posición. Ocasionalmente, durante la división celular, puede ocurrir que un segmento de ADN se desprenda y vuelva a conectarse pero con los extremos invertidos, sin que esto afecte gravemente su funcionamiento.

El operador genético propuesto por Holland causa esa misma transformación en las cadenas de bits de un AG, manteniendo por medio de diferentes estrategias el significado de cada bit individual. *(fig. 3.4)*

La idea detrás de este operador es reducir el posible impacto negativo sobre la performance del AG si no se conoce de antemano que parámetros están relacionados y accidentalmente son colocados en posiciones alejadas, lo que dificultará su pasaje como bloque a la siguiente generación durante un entrecruzamiento.

El operador de inversión no modifica el fitness de un individuo, pero dado que modifica la posición de los bits, altera la forma en que el entrecruzamiento formará bloques significativos y hace la selección inicial de la ubicación de los parámetros codificados menos relevante. *(fig. 3.5)*

A pesar de estas características, este operador no es muy utilizado, dado que no existen evaluaciones teóricas detalladas sobre su efecto en el desempeño de los AGs. Por otro lado, resultados experimentales iniciales [Golberg1989] no mostraron un efecto importante que justifique la complejidad adicional involucrada en la implementación y el costo en CPU y memoria necesario para la aplicación de este operador, mientras que resultados positivos posteriores sólo mostraron ser aplicables para dominios muy reducidos.

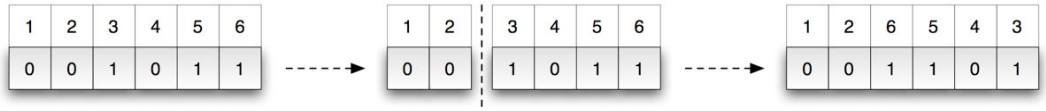


Figura 3.4: *Durante la inversión, se selecciona una sección de la cadena y se invierte su orden, en este caso los bits afectados son la sección desde la tercer posición hasta la sexta. Para conservar el significado de cada bit individual, la inversión hace necesario, además de la cadena de bits, almacenar y procesar una cadena adicional de posiciones asociada a esta (sección superior de la figura): cada bit posee ahora una posición lógica específica (e independiente de su posición física) que será luego utilizada en el proceso de decodificación.*

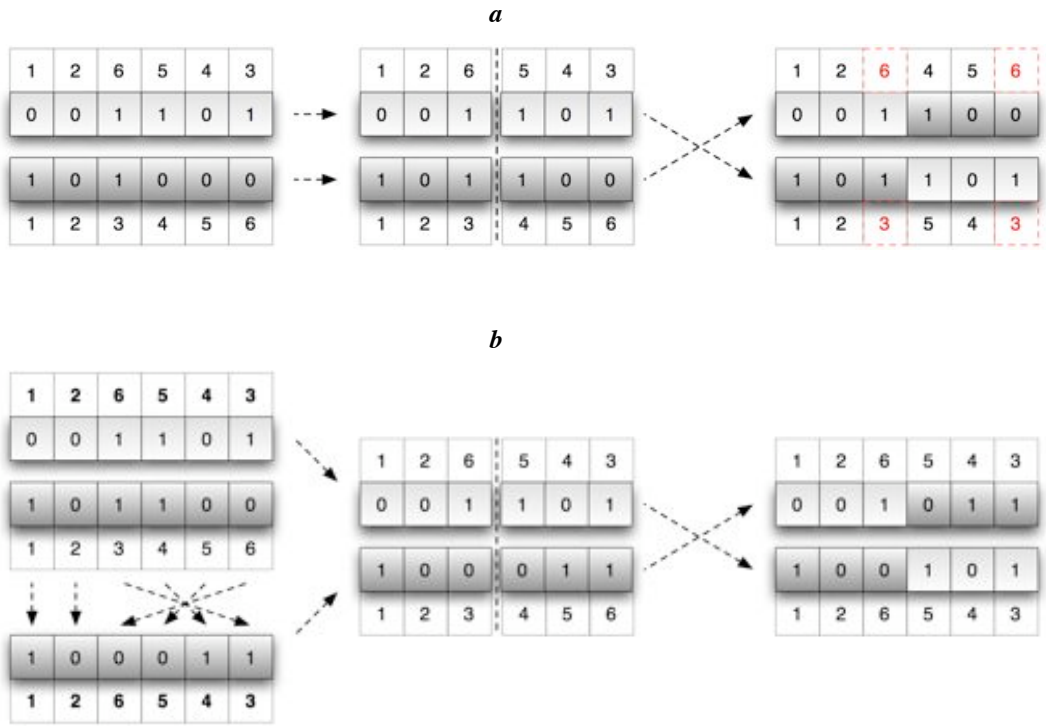


Figura 3.5: *El objetivo del operador de inversión es facilitar la combinación de bits que originalmente se hallaban en posiciones alejadas, facilitando la formación de nuevos bloques de construcción útiles que no dependan de el orden y ubicación inicial de los parámetros y que se mantengan intactos durante el entrecruzamiento. El uso del operador de inversión introduce una complejidad adicional al proceso: dado que durante el entrecruzamiento el corte se realiza a nivel de posiciones físicas, la aplicación directa de este operador podría generar cromosomas incompletos (a). Una solución posible es introducir un paso extra al proceso de entrecruzamiento: aplicar al segundo padre el mismo orden indicado por la cadena de posiciones lógicas del primero, para luego continuar con el proceso de forma normal (b). Opcionalmente, y como último paso, se puede restaurar uno de los hijos al orden del segundo padre, de manera similar al reordenamiento efectuado al comienzo del proceso.*

3.6 Algoritmos

Existen diferentes variantes de AGs, pero todas se basan en los mismos principios. A continuación se muestran el algoritmo básico general y una variante de este comúnmente utilizada.

3.6.1 Algoritmo básico

El algoritmo genético básico consta de los siguientes pasos:

1. Crear una Población Inicial (P_0) compuesta de n individuos generados de forma aleatoria
2. Calcular el nivel de adaptación de cada individuo usando la función objetivo
3. Generar una nueva población (P_{i+1}), generando los n individuos de la siguiente manera:
    a. Seleccionar dos individuos de P_i utilizando un método de selección que elija con mayor probabilidad a individuos con mayor fitness.
    b. Con una probabilidad p_e *(probabilidad de entrecruzamiento)* aplicar el operador de Entrecruzamiento. En caso de que el operador no resulte aplicado, los hijos resultantes serán una copia idéntica de sus padres.
    c. Aplicar el operador de Mutación a cada hijo con una probabilidad p_m *(probabilidad de mutación)* de afectar a cada componente de la cadena que codifica su genoma.
4. Reemplazar la población actual por la obtenida en el paso 3
5. Repetir a partir del paso 2 hasta llegar a un límite predefinido (de iteraciones, de tiempo, de nivel de adaptación u otro)

3.6.2 Algoritmo básico + elitismo

Esta es una variante del algoritmo básico, diseñada para evitar que factores aleatorios eliminen accidentalmente las mejores combinaciones de características. Estos factores aleatorios se refieren principalmente a la selección, pero también es particularmente útil en los casos en que la función de fitness sea aproximada. Para esto se introduce un factor llamado *elitismo* [De Jong1975], que consiste en conservar sin cambios los primeros m

individuos, ordenados por fitness descendiente, de una generación a la próxima:

1. Crear una Población Inicial (P_0) compuesta de n individuos generados de forma aleatoria
2. Calcular el nivel de adaptación de cada individuo
3. Generar una nueva población (P_{i+1}), copiando los primeros m individuos con mejor fitness de la población P_i
4. Completar la población (P_{i+1}), generando los restantes n - m individuos de la siguiente manera:
    a. Seleccionar dos individuos de P_i utilizando un método de selección que elija con mayor probabilidad a individuos con mayor fitness.
    b. Con una probabilidad p_e *(probabilidad de entrecruzamiento)* aplicar el operador de Entrecruzamiento. En caso de que el operador no resulte aplicado, los hijos resultantes serán una copia idéntica de sus padres.
    c. Aplicar el operador de Mutación a cada hijo con una probabilidad p_m *(probabilidad de mutación)* de afectar a cada componente de la cadena que codifica su genoma.
5. Reemplazar la población actual por la obtenida en los pasos 3 y 4
6. Repetir a partir del paso 2 hasta llegar a un límite predefinido (de iteraciones, de tiempo, de nivel de adaptación u otro)

3.7 Reemplazo de población

En el algoritmo estándar de AG se asume un *reemplazo generacional*, es decir, cada nueva generación reemplaza completamente a la anterior.

Sin embargo, si bien es la forma más utilizada no es la única: se pueden distinguir estrategias de reemplazo de población de acuerdo a la

fracción *G (salto generacional)* de la población que es reemplazada con cada generación. En particular, en el reemplazo generacional, resulta:

$$G = \frac{n}{n} = 1$$

dado que todos los individuos son reemplazados. En otros casos, donde sólo se reemplazan algunos individuos, se debe determinar una estrategia de reemplazo.

3.8 Métodos de Selección

Existen varias formas de efectuar la selección de los individuos, de forma que los genomas de aquellos con mayor fitness tengan mayor probabilidad de pasar a la generación siguiente. Es necesario, sin embargo, que esto se realice sin eliminar completamente la posibilidad de que otros genomas también estén presentes, dado que la variedad es un requisito para evitar llegar a soluciones sub-óptimas (máximos locales). Entre ellas dos de los métodos más utilizados son el método de *selección proporcional al fitness* y el de *selección por torneos*. Métodos alternativos incluyen la *selección por rangos*, *selección de estado estacionario* y *selección por truncamiento*.

3.8.1 Selección Proporcional al Fitness

Este método, propuesto por Holland [Holland1975], asigna una probabilidad de selección a cada individuo en forma directamente proporcional a su fitness. Dado un proceso de selección de λ individuos en una población de tamaño μ, la cantidad de λ_i de veces que se espera será seleccionado un individuo *i* es [Dréo2006]:

$$\lambda_i = \frac{\lambda}{\sum_{j=1}^{\mu} f_j} f_i$$

Aplicado a la teoría de esquemas, se puede decir que un esquema E tiene un número de instancias esperadas $\lambda(E)$ luego de la selección proporcional de:

$$\lambda(E) = \sum_{x \in E} \frac{f_x}{\bar{f}}$$

donde x ∈ E indica que x es una instancia del esquema E en la población actual i y $\bar{f}$ indica el fitness promedio de la población. Combinando esto con las probabilidades de supervivencia de un esquema a los operadores de mutación y entrecruzamiento descriptas en la sección 3.5 se obtiene:

$$\lambda(E) \geq \sum_{x \in E} \frac{f_x}{\bar{f}} \left[1 - p_e \left(\frac{ld(E)}{l} \right) \right] \left[(1 - p_m)^{o(E)} \right]$$

Este es el denominado *Teorema de Esquemas* de Holland, el cual describe el crecimiento de esquemas de una generación a la próxima.

Las dos implementaciones principales de la selección proporcional al fitness son *RWS (roulette wheel selection – selección por rueda de ruleta)* y *SUS (stochastic universal sampling – muestreo universal estocástico)*.

3.8.1.1 Selección por Rueda de Ruleta (RWS - Roulette Wheel Selection)

En este método de selección, dada una población de μ individuos[1] se le asigna a cada uno un segmento de una rueda de ruleta imaginaria con ángulo de apertura que dependerá de su nivel de adaptación. De esta forma, al girar la ruleta la probabilidad de seleccionar un individuo será directamente proporcional a su adaptación (f_i).

[1] En todos los ejemplos donde no se indique λ se asumirá λ = μ para los AG, es decir, que el proceso seleccionará siempre una cantidad de individuos igual al tamaño de la población.

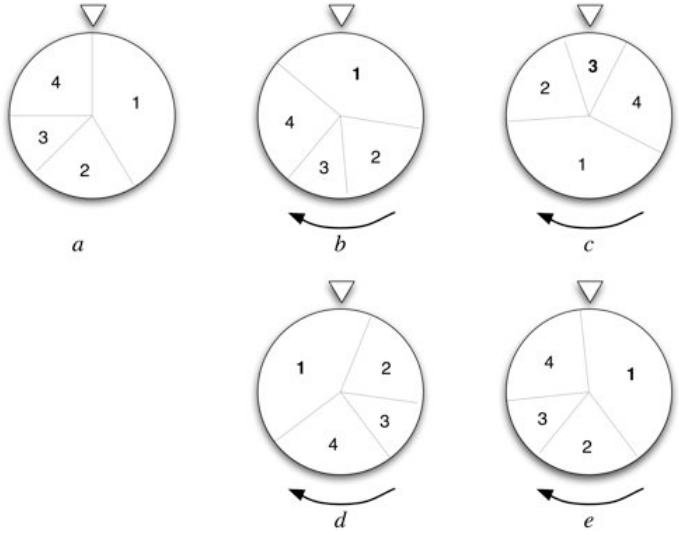


Figura 3.6: *Selección por Rueda de Ruleta (RWS). En (a) se asignan diferentes sectores de la ruleta a cada individuo con un área proporcional a su fitness. Luego (b-e) se realizan tantos giros de la ruleta como individuos se desee seleccionar.*

Una forma simple de implementar esta selección es generar un número aleatorio r entre 0 y 1 y seleccionar el menor valor j que satisface la desigualdad [Wahde2005]:

$$\frac{\sum_{i=1}^{j} f_i}{\sum_{i=1}^{\mu} f_i} > r$$

Una forma equivalente[2], utilizada en BugSimEvo, es generar un valor aleatorio r:

$$0 \le r < \sum_{i=1}^{\mu} f_i$$

y seleccionar el menor valor de j que satisface la desigualdad:

$$\sum_{i=1}^{j} f_i > r$$

[2] Así como en este caso se define $0 \le r < \sum_{i=1}^{n} f_i$ en el caso anterior se toma $0 \le r < 1$

La versión descripta es la versión clásica y más comúnmente utilizada, denominada *SSR (stochastic sampling with replacement – muestreo estocástico con reemplazo[3])*. Otra variante es la denominada *SSPR (stochastic sampling with partial replacement – muestreo estocástico con reemplazo parcial)*.

En SSPR, cada vez que se gira la ruleta y se selecciona un individuo, se reduce el tamaño del segmento que lo representa en 1.0 (se asume una ruleta de tamaño normalizado tal que el total de sus segmentos sume inicialmente λ), estableciéndose en cero si el resultado es negativo. Esto garantiza un limite máximo a la cantidad de veces que puede ser seleccionado un individuo, pero no un valor mínimo.

Otra alternativa es la denominada *RSSR (remainder stochastic sampling with replacement – muestreo estocástico de restos con reemplazo)*. En esta se utiliza una primera etapa determinista, en la cual cual se seleccionan tantas instancias de cada individuo como la parte entera de su tamaño de segmento (λ_i). Luego se procede a generar una ruleta con las partes fraccionales utilizando selección con reemplazo (es decir, se aplica SSR), para completar el total requerido de individuos a seleccionar. En *RSSWR (remainder stochastic sampling without replacement – muestreo estocástico de restos sin reemplazo)* se procede de la misma manera, sólo que en la ruleta de partes fraccionales se procede a eliminar (establecer el tamaño de su segmento a cero) cada individuo que resulte seleccionado en cada giro.

3.8.1.1.1 Complejidad algorítmica

Como indica Goldberg en [Golberg1991] para la mayoría de los problemas interesantes en AG el tiempo necesario para evaluar las funciones

[3] Es importante destacar que *reemplazo* en este contexto se refiere exclusivamente al proceso de selección. El reemplazo en el contexto de la población o *reemplazo generacional* se describe en la sección 3.7

de fitness suele ser mucho mayor al tiempo necesario para la iteración de la aplicación de operadores genéticos. Por lo tanto, si bien para cada método de selección se indicará su complejidad algorítmica, su impacto en la decisión de usar un método en particular es menor y sólo debería ser tenido en cuenta luego de considerar otros factores de comparación como velocidad de convergencia, presión selectiva, etc.

SSR realiza *n* giros de la ruleta y si se utiliza una búsqueda lineal, en promedio recorre $O(n)$ pasos para ubicar cada individuo, llevando la complejidad total a $O(n^2)$. Si se utiliza una búsqueda binaria, la complejidad total se reduce a $O(n \log n)$.

RSSR posee la misma complejidad que SSR, mientras que RSSWR tiene una complejidad de $O(n)$. [Goldberg1991]

3.8.1.2 Muestreo Universal Estocástico (SUS - Stochastic Universal Sampling)

El método RWS tiene el inconveniente de que si bien cada individuo tiene un probabilidad proporcional a su fitness de ser elegido, no hay ningún tipo de garantías de que esto suceda. Debido al azar involucrado en cada giro, un individuo con bajo fitness podría ser seleccionado repetidamente mientras otro con un mayor fitness podría ser seleccionado en menor medida o directamente nunca (dependiendo de que variante específica de RWS se utilice). Para reducir estos efectos, denominados *genetic drift*, se desarrolló el muestreo universal estocástico. Se comienza de forma similar: dada una población de *n* individuos se le asigna a cada uno un segmento de una rueda de ruleta imaginaria con ángulo de apertura que dependerá de su nivel de adaptación. Luego, en lugar de girar la ruleta n veces se procede a ubicar a espacios regulares *n puntos de selección*. Finalmente se procede a girar la ruleta una única vez. Este método garantiza que cada individuo será seleccionado un mínimo de $\lfloor \lambda_i \rfloor$ veces y un máximo de $\lceil \lambda_i \rceil$.

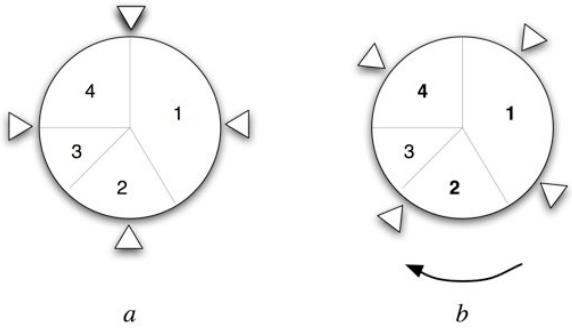


Figura 3.7: *Muestreo Universal Estocástico (SUS). En (a) se asignan diferentes sectores de la ruleta a cada individuo con un área proporcional a su fitness y se ubican a espacios regulares tantos "puntos de selección" como individuos se desee seleccionar. Luego (b) se realiza un único giro de la ruleta y se seleccionan los individuos bajo cada punto de selección (una instancia por punto).*

3.8.1.2.1 Complejidad algorítmica

SUS tiene una complejidad de *O(n)* dado que sólo requiere de un giro de la ruleta una vez generada.

3.8.2 Selección por Torneos (Tournament Selection)

Este método consiste en seleccionar de forma aleatoria *k* individuos de la población y luego compararlos, seleccionando el mejor. El proceso se repite hasta haber seleccionado la cantidad de individuos necesarios para completar la siguiente generación.

Un torneo en el que se comparan grupos de *k* individuos se denomina *torneo de tamaño k*. A mayor tamaño, mayor será la presión selectiva de una selección por torneos, obteniéndose una convergencia más rápida [Miller1995].

La versión simple de este método es el torneo de tamaño 2 o *torneo binario*, donde se seleccionan al azar dos individuos de la población y se elige el de mayor nivel de adaptación. Esta es la versión más utilizada en los AG

cuando se utiliza selección por torneos, comúnmente incluyendo un factor probabilístico en la selección: en lugar de seleccionar siempre el mejor de los dos individuos, se hace con una probabilidad fija $0.5 < p \leq 1$.

La selección por torneos comparte con la selección por rangos (sección 3.6.5) el mantenimiento de la presión selectiva a lo largo de las generaciones.

Una ventaja importante del método de selección por torneos es que no depende de información global de la población, característica que lo hace fácilmente paralelizable.

3.1.1.1 Complejidad algorítmica

La selección por torneos tiene una complejidad de $O(n)$ dado que para cada torneo se selecciona en forma aleatoria una cantidad constante de individuos, se comparan, lo que también se puede realizar en un tiempo constante, y se repite dicho proceso n veces hasta completar una generación.

3.1.3 Selección por Rangos (Rank Selection)

En la selección por rangos [Baker1985] se comienza con una etapa de ordenamiento, donde los individuos de la población se ordenan en forma creciente de acuerdo a su fitness. A continuación, se realiza una asignación de cantidad de selecciones esperadas para cada uno utilizando una función de asignación no decreciente. Finalmente, en base a las cantidades de copias esperadas se aplica una selección proporcional.

Baker propuso una función lineal de asignación de cantidad de copias esperadas, donde se establecen dos parámetros: cantidad de selecciones

esperadas para el mejor individuo (parámetro Max, con el requerimiento Max ≥ 0) y cantidad de selecciones esperadas para el peor individuo (parámetro Min). Para un individuo i en una población de μ individuos, la cantidad de selecciones esperadas λ_i dado su rango r_i en dicha población se define como:

$$\lambda_i = Min + (Max - Min)\frac{r_i - 1}{\mu - 1}$$

Teniendo en cuenta que para mantener una población de tamaño constante se necesita:

$$\sum_{i=1}^{\mu} \lambda_i = \mu$$

Esto determina las siguientes restricciones adicionales:

$$1 \leq Max \leq 2 \quad y \quad Min = 2 - Max.$$

En base a resultados obtenidos en distintos casos de prueba, Baker sugería utilizar Max = 1.1 (y por lo tanto, Min = 0.9).

En la selección proporcional existe generalmente una mayor presión selectiva al comienzo y esta se va reduciendo con el correr de las generaciones, conforme la población se hace más homogénea, el fitness medio aumenta y se reduce la diferencia entre los mejores y peores individuos. En el método de selección por rangos, por otro lado, la presión selectiva es constante durante todas las generaciones. Esto puede constituir una desventaja dado que en muchos casos reducirá la convergencia requiriendo un mayor número de generaciones para alcanzar una solución aceptable. Por otro lado [Whitley1989], la presión selectiva constante puede reducir las chances de una convergencia prematura, debido a individuos con fitness inusualmente alto con respecto al promedio de la población, y de estancamiento por falta de presión selectiva conforme se reduce la diferencia entre las mejores y peores soluciones.

3.1.1.1 Complejidad algorítmica

La selección por rangos tiene una complejidad de *O(n log n)* dado que consta de dos etapas consecutivas: ordenamiento, de complejidad *O(n log n)* y selección proporcional, también *O(n log n)*.

3.1.4 Selección de Estado Estacionario (Steady-State Selection)

Como se describió en la sección 3.6, el AG clásico utiliza reemplazo generacional, es decir, cada generación reemplaza completamente la anterior. Cuando se utiliza elitismo, se reemplaza toda la población con excepción de una pequeña fracción (los mejores *m* individuos). En la selección de estado estacionario, por el contrario, sólo se reemplaza una pequeña fracción de la población, manteniendo todos los individuos con excepción de un pequeño grupo compuesto por los individuos menos adaptados.

Una de las implementaciones más conocidas de selección estacionaria es el denominado algoritmo GENITOR [Whitley1989], en el cual en cada ciclo sólo se reemplaza un único individuo (el de menor fitness). Para seleccionar los padres que por medio de entrecruzamiento y mutación generarán el individuo hijo que complete la población, GENITOR utiliza una selección por rangos.

Análisis comparativos de GENITOR y métodos de selección generacionales puede encontrarse en [Whitley1989] y [Goldberg1991]. En [Syswerda1991], [De Jong1993], [Hancock1997] y [Sarma1997] se presentan comparaciones entre selección generacional y distintas variantes de métodos de selección de estado estacionario. Como observación general se puede decir que la selección de estado estacionario es equivalente a la selección generacional si la selección de los individuos a eliminar se hace de forma aleatoria. Por otro lado, si se selecciona siempre los *m* peores individuos para

ser reemplazados (método más usual), la velocidad de convergencia se ve incrementada, pero a costa de una gran sensibilidad al ruido en los valores de fitness.

3.1.4.1 Complejidad algorítmica

La selección por estado estacionario comienza realizando un ordenamiento de los individuos de acuerdo a su fitness. Este ordenamiento sólo se realiza una vez en la primer generación, en generaciones sucesivas los nuevos individuos son simplemente insertados en la posición correcta, lo que tiene una complejidad de $O(\log n)$ si se utiliza una búsqueda binaria. La selección de cada individuo también se puede realizar en $O(\log n)$. Ambos pasos deben repetirse $m \leq n$ veces para completar la siguiente generación, resultando en un complejidad de $O(n \log n)$.

3.1.5 Adaptaciones de presión selectiva

Como se describió en la sección 3.7.5, la selección proporcional basada en fitness puede presentar problemas de convergencia prematura debido a que al inicio pueden existir individuos con un fitness excesivamente alto comparado con el resto de la población y que por lo tanto rápidamente se dispersan por esta reduciendo la exploración de soluciones alternativas. Además, con el correr de las generaciones se puede presentar un problema de estancamiento dado que al converger todas los individuos a niveles de fitness similares se reduce la presión selectiva.

Además de la selección por rangos, existen otras alternativas basadas en hacer un uso indirecto del fitness, como el método de *escalado sigma* y la *selección Boltzmann*.

3.1.1.1 Escalado Sigma (Sigma Scaling)

En este método la cantidad de selecciones esperadas de cada individuo no es una función directa de su fitness con respecto al resto de los individuos sino que depende también de la media y el desvío estándar de la población [Tanese1989]:

$$\lambda_i = \begin{cases} 1 + \dfrac{f_i - \bar{f}_i}{2\sigma} & si \quad \sigma \neq 0 \\ 1 & si \quad \sigma = 0 \end{cases}$$

donde f_i representa el fitness promedio de la población y σ el desvío estándar. Adicionalmente, se puede limitar la cantidad máxima de selecciones esperadas a 1.5 (es decir, un desvío por encima de la media de la población), mientras que la cantidad mínima esperada se suele limitar a 0.1.

Dado que al comenzar el AG el desvío estándar tiende a ser alto, es poco probable que existan individuos con fitness varios desvíos por encima de la media, mientras que al avanzar el proceso el desvío estándar de la población se reduce y la diferencia de fitness entre cada individuo y la media cobra mayor importancia, manteniendo por lo tanto la presión selectiva.

3.1.1.2 Selección Boltzmann (Boltzmann Selection)

El escalado sigma sirve para mantener una presión relativamente constante a lo largo de las generaciones. La selección Boltzmann, por otro lado, intenta disminuír la presión selectiva inicialmente, favoreciendo una mejor exploración del espacio de soluciones, para luego ir progresivamente incrementando ésta conforme avanzan las generaciones, favoreciendo la explotación de las soluciones más prometedoras.

Este método, en forma similar al templado simulado *(simulated annealing)*, emplea un parámetro de "temperatura", la cual varía de forma preestablecida. Se comienza con una alta temperatura, lo que causa una baja

presión selectiva inicial, y progresivamente se va reduciendo, lo que incrementa la selectividad.

3.9 Co-evolución

En la sección anterior se analizaron distintas formas de selección y como afectan estas la presión selectiva durante el proceso evolutivo. En ciertos casos, sin embargo, un AG puede llegar a un punto de estancamiento en un máximo local. Una posible causa la constituyen funciones objetivo aproximadas y/o casos de prueba limitados, que hacen que el AG aproveche debilidades en estos generando una sobreadaptación.

En [Hillis1990] se describe este problema en experimentos relacionados con redes de ordenamiento[4]. El objetivo era obtener una red que utilizara la menor cantidad de comparaciones posibles. Luego de resultados poco satisfactorios, causados principalmente por la sobreadaptación de las soluciones a los casos de prueba, se probaron diferentes estrategias centradas en la variariación de dichos casos de prueba. La que resultó más exitosa se basó en el concepto de *co-evolución*, permitiendo la evolución en paralelo de las redes y de los casos de prueba, en un modelo que Hillis denominó de *huésped-parásito*. De esta forma se evitaba la sobreadaptación de las redes y se reducían las posibilidades de estancamiento en máximos locales, dado que estos se convertían en el foco de casos de prueba que evolucionaban para encontrar fallas.

En [Angeline1993] y [Rosin1993] pueden encontrarse detalles adicionales sobre diferentes tipos de co-evolución.

[4] Una red de ordenamiento consiste en un una serie de comparaciones e intercambios de datos en un orden fijo y cuya aplicación a una cadena de entradas resulta en su ordenamiento.

Capítulo 4

4 Robótica Evolutiva

El diseño de robots autónomos y sus sistemas de control tiene que enfrentar el hecho de que tanto los sensores de un robot (medidores de luz, sonido, presión, etc.) como sus efectores (motores, ruedas, pistones, etc.) tienen una precisión limitada y siempre existe la posibilidad de fallas totales o parciales de cualquier componente. Debido a estos factores y al necesariamente finito poder de procesamiento, robots diseñados de forma manual por medio del uso de técnicas tradicionales, presentan serias dificultades al enfrentarse al mundo real.

En ambientes no controlados, la cantidad de variables involucradas hace que la tarea de diseñar manualmente un sistema de control adaptable y robusto sea extremadamente difícil. Lo mismo se puede afirmar acerca del diseño del cuerpo de un robot, dado que las exigencias sobre este dependerán en gran medida de la forma en la que el robot use sus efectores (fuerzas, velocidades, sincronización) y dependa de sus sensores.

La robótica evolutiva (RE) tiene como objetivo la generación de sistemas de control de robots (o más generalmente, cerebros), el diseño de sus cuerpos, o ambos, a través de la utilización de algoritmos evolutivos (AE). [Wahde2005]

La RE tiene ventajas importantes y se basa en aprovechar el hecho de que los AE son particularmente aptos para ofrecer soluciones a problemas en los cuales es relativamente fácil determinar qué se debe hacer pero muy complejo determinar cómo hacerlo.

Una limitación de este enfoque es que el uso de AE implica la evaluación de muchas instancias y esto se traduce por lo general en la necesidad de realizar el proceso por medio de simulaciones. Tanto en el diseño de cerebros como de cuerpos, la simulación deberá presentar un mundo con la suficiente semejanza al ambiente real en el que funcionará el robot. Además, es extremadamente importante que la simulación presente un adecuado nivel ruido. El objetivo es evitar que el robot se sobre-especialice y dependa de medidas y un nivel de control sobre sus partes móviles que no corresponderán luego a ninguna posible implementación física.

Los pasos para el desarrollo de un robot por medio de la RE se basan en la aplicación de los AE, con características específicas en lo que se refiere a la evaluación de cada individuo. A continuación se muestra un esquema general:

1. Crear una Población Inicial (P_0) compuesta de n individuos
2. Para cada individuo:
   a. Generar una instancia
   b. Evaluar dicha instancia por medio de una simulación, obteniendo una medida de performance. Dicha simulación incluirá normalmente un ambiente simulado con un motor de física en dos o tres dimensiones, dependiendo del tipo de robot y comportamiento a simular.
3. Si no se llegó a un objetivo predefinido (de iteraciones, de tiempo de simulación, de nivel de adaptación u otro):
   a. Generar una nueva población P_{i+1} reemplazando a la actual por medio de la aplicación de técnicas de AE
   b. Repetir a partir del paso 2.
4. Si se llegó al límite predefinido, validar los resultados obtenidos por medio de la implementación física de la o las mejor(es) instancia(s).

Aunque por lo general se suelen utilizar simulaciones para la RE, en algunos casos puede utilizarse directamente robots físicos, en particular, cuando el AE es utilizado únicamente para evolucionar un cerebro o sistema de control de un robot. En este caso, un mismo robot puede ser utilizado para evaluar sucesivamente distintos sistemas de control, los cuales son generados en una computadora y luego copiados a éste. De todas formas, incluso en estos casos suele ser común que la primera serie de generaciones se realice de forma simulada, para luego continuar la aplicación del AE de forma física [Miglino1996]. Para esta segunda fase, suele utilizarse una población reducida y un número pequeño de iteraciones.

A continuación se presentan diferentes aspectos a tener en cuenta y diferentes enfoques para la aplicación de principios de Robótica Evolutiva, junto con ejemplos ilustrativos.

4.1 Evolución de Subsunción

En la metodología clásica de la Inteligencia Artificial (IA) y en particular de la robótica autónoma, se divide el problema en una serie de pasos ordenados: percepción, modelado, planificación y acción, los cuales se repiten de forma cíclica. Distintas tareas son realizadas dependiendo de la etapa: pre-procesamiento de datos de sensores (determinación de ángulos y distancias, identificación de bordes en datos visuales, etc), generación de representación interna en base al procesamiento e integración de datos sensores preprocesados, planificación de caminos, planificación de movimientos de manipulación, ejecución de acciones por medio de la traducción de planes a activaciones precisas y específicas de efectores.

Esta metodología mostró funcionar eficazmente en dominios limitados, particularmente en simulaciones y ambientes controlados o semi-estáticos. Se hicieron desarrollos importantes en planificación de recorridos,

procesamiento de imágenes, planificación de movimientos, procesamientos de conceptos y muchas otras áreas.

A pesar de ello, las implementaciones en el mundo real de robots autónomos en ambientes dinámicos siguió mostrando serias limitaciones, indicando problemas de escalabilidad y adaptación en la integración de las diferentes etapas.

A mediados de los 80, una nueva metodología comenzó a desarrollarse para tratar de superar estos problemas, basada en la descomposición de tareas complejas no ya en el ciclo percepción-modelado-planificación-ejecución sino en conjuntos de comportamientos simples, cada uno de ellos con acceso a los sensores y efectores. Esta metodología, denominada *arquitectura de subsunción* [Brooks1986] [Brooks1991], está especialmente adaptada a la interacción con ambientes dinámicos y por lo tanto es particularmente aplicable al diseño de sistemas de control de robots autónomos que deben enfrentar ambientes no controlados.

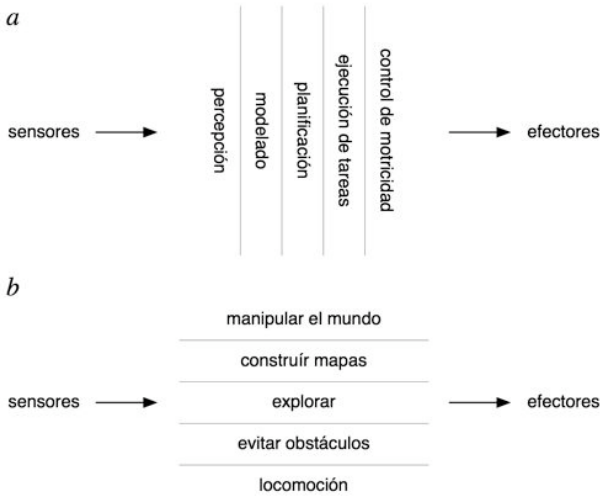


Figura 4.1: *Distintas arquitecturas de IA utilizadas en robótica autónoma (a) arquitectura tradicional, donde el proceso se divide en módulos de procesamiento que se ejecutan en un orden específico (b) arquitectura de subsunción, donde el procesamiento se divide en términos de módulos de comportamientos, pudiendo cualquiera de ellos tomar datos de sensores y controlar efectores. [Brooks1986]*

En la arquitectura de subsunción, es necesario identificar y diseñar los distintos comportamientos que se consideren necesarios, los cuales variarán de acuerdo a la tarea a realizar y al ambiente en el que operará el robot *(fig. 4.1)*. El proceso generalmente consiste en diseñar primero comportamientos básicos y luego ir integrando capas que combinan, extienden y coordinan estos comportamientos para obtener comportamientos complejos.

La RE puede utilizarse para obtener de forma automática, a través de la evolución, comportamientos inteligentes sin necesidad de delinear éstos de antemano ni de implementarlos de forma manual, simplemente diseñando una función de fitness adecuada y un ambiente que permita su correcta aplicación. *(fig. 4.2)*.

En [Koza1993], [Calabretta1998] y [Auerbach2009a] pueden encontrarse ejemplos de experimentos específicamente diseñados para favorecer la evolución de conjuntos de comportamientos a través de procesos evolutivos. En [Bongard2008] y [Auerbach2009b] se presentan análisis donde se muestra cómo un incremento gradual en la complejidad del ambiente favorece de forma natural la evolución de comportamientos modulares especializados.

Figura 4.2: *La robótica evolutiva comparte aspectos de la robótica basada en comportamientos, pero hace innecesaria la especificación de los comportamientos y su coordinación, pudiendo estos emerger automáticamente a través de procesos evolutivos [Nolfi2000]*

4.2 Evolución de comportamientos simples

Los comportamientos simples son la base sobre la que se construyen comportamientos más complejos en la RE y han sido objeto de varios experimentos para mostrar la factibilidad de su evolución no supervisada.

Por ejemplo, en [Mondada1995] se describe la realización de experimentos de un proceso evolutivo directamente sobre robots reales *(fig. 4.3)*, provistos de dos ruedas/motores y ocho sensores de proximidad infrarrojos. El objetivo inicial era obtener comportamientos simples: el robot debía poder desplazarse hacia un objetivo y evitar obstáculos. El proceso evolutivo se basaba en el uso de un AG con escalado de fitness, utilizando cromosomas con valores de punto flotante para codificar pesos en una red neuronal de estructura fija.

En [Sprinkhuzien-Kuyper2000] se evolucionó un comportamiento denominado "box-pushing" (empujar cajas) en un ambiente bidimensional simulado, que luego fue probado en un robot real. El proceso consistió en evolucionar los pesos de conexiones de una red neuronal simple, alimentada por sensores de proximidad (implementados como sensores infrarrojos en el robot real) y cuya salida determinaba la activación de dos motores.

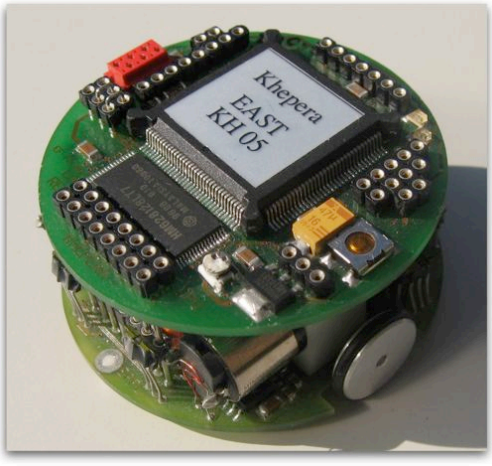


Figura 4.3: *Robot Khepera (primera generación). Utilizado en gran cantidad de experimentos en robótica evolutiva, Khepera posee ocho sensores infrarrojos de proximidad, dos motores que controlan su desplazamiento, un procesador a 16MHz, 256KB de RAM y 512 KB de EEPROM. (Foto: Stéphane Magnenat)*

4.3 Evolución de comportamientos complejos

La RE ha sido también utilizada para la obtención de comportamientos complejos, utilizando estructuras de representación genética adecuadas y funciones de fitness adaptadas a tareas complejas y/o que requieren varias etapas para su realización.

En [Calabretta1998], por ejemplo, se estudió el efecto que tiene la estructura de representación en la modularidad de comportamientos obtenibles por medio de procesos evolutivos. Se implementó un AG con un operador adicional de duplicación genética, permitiendo cromosomas de longitud variable. El objetivo planteado era la realización de una tarea compleja consistente en:

- Explorar el ambiente, evitando chocar con las paredes
- Reconocer el objeto buscado y posicionarse de tal forma de poder tomarlo
- Tomar el objeto
- Moverse hacia una pared llevando el objeto y evitando otros objetos en su camino
- Posicionarse con respecto a la pared de forma de poder soltar el objeto fuera del terreno delimitado por ésta
- Soltar el objeto

El proceso se repite hasta que no queden más objetos en el terreno delimitado por las paredes.

Se realizaron diez simulaciones, con cien individuos cada una y mil generaciones, utilizando AG con un proceso selectivo por truncamiento, en el cual se seleccionó el 20% superior de los individuos de cada generación para servir como padres de la siguiente. Se compararon tres alternativas: tradicional, en la cual no se hace ninguna modularización explícita, modularizado manual, en la cual para cada posible control de efectores se

codifican siempre dos alternativas, sólo una de las cuales puede estar activa en un momento dado, y modularización dinámica, en la cual el AG es modificado por medio del agregado de un operador de duplicación genética, que permite reproducir los efectos de la modularización manual pero sin especificar de antemano que sistemas serán afectados.

Los resultados indicaron que este último mecanismo facilitó la modularización de comportamientos observables, mientras que en el caso de la modularización prefijada esta división de comportamientos no era tan clara, ya que tendía a darse desde el punto de vista del robot y no del observador.

Más recientemente, en [Capi2005] se utilizó un AG basado en números reales para codificar la estructura y pesos de conexiones de distintas variantes de redes neuronales. El objetivo era evolucionar en un entrono simulado un robot con un repertorio de comportamientos complejos. En el entorno se situaron un "nido", "agua" y "alimento" virtuales, y el comportamiento complejo deseado era la visita secuencial y cíclica *alimento-nido-agua-nido*. Para ello, se definió una función de fitness que asignaba un puntaje positivo por cada nodo visitado en el orden correcto y negativo por cada nodo incorrecto. El robot virtual contaba con sensores táctiles y un sensor visual con un campo de visión limitado y que le permitían identificar los distintos objetivos y su ángulo con respecto a su dirección actual, al cual se le agregaba un 5% de ruido simulado. El experimento mostró que es posible la evolución de comportamientos que implican la realización de diferentes tareas en un orden secuencial específico. La implementación de los controladores obtenidos por medio de simulación en robots reales mostró resultados similares, aunque con un fitness menor debido a diferencias entre los sensores y efectores simulados y los reales.

4.4 Locomoción y biomimética

Muchos de los experimentos en RE utilizan robots simples provistos de ruedas, sensores y en algunos casos efectores de manipulación (para tomar

objetos). En ellos, el acto de desplazamiento se reduce a una correcta aplicación de potencia a los motores conectados a cada rueda para avanzar, retroceder, girar o mantener la posición. Este tipo de robots son aptos para entornos artificiales, donde la superficie de desplazamiento es plana y mayormente regular.

La RE sin embargo también puede utilizarse para evolucionar sistemas de control de robots con medios de locomoción más complejos y adaptables a entornos naturales. Para ello, el sistema de control deberá poder sincronizar distintos tipos de efectores en secuencias, duraciones y/o ritmos específicos con el fin de lograr una locomoción efectiva.

4.4.1 Locomoción terrestre

De la misma forma que la RE se inspira en el proceso natural de la evolución, encuentra también inspiración en la biología para el diseño de formas de locomoción.

La inspiración no se reduce sólo a la utilización de patas en lugar de ruedas, sino que incluye también aspectos como simetrías bilaterales, sistemas de sincronización, balance de masa, retroalimentación y sensores, entre otros muchos aspectos. La RE puede descubrir muchos de estos principios naturales de forma automática, pero de todas formas la aplicación de estos principios es importante ya que permite ahorrar esfuerzos para la obtención de sistemas de control y diseños de robots eficientes.

Como ejemplos de estudios de estos principios y su aplicación a la robótica (*biomimética*) se pueden mencionar por ejemplo las experiencias descriptas en [Espenschied1993]. En dicho trabajo se estudiaron diferentes sistemas de sincronización para un robot hexápodo basados en señales de activación e inhibición entre las diferentes patas, inspirados en estudios de los mecanismos de locomoción de los "insectos palo" *(Phasmatodea)*. En [Ferrel1995] se comparan controladores inspirados en insectos, incluyendo un

sistema basado en reflejos y otro basado en un generador de patrones central. En [Espenschied1996] se describe la implementación en un robot de principios biológicos adicionales, como reflejos, flexibilidad mecánica y control distribuido. En [Cham2002] se evalúan técnicas de construcción de robots basadas en estudios biomecánicos de la cucaracha, emulando propiedades visco-elásticas que permiten una locomoción robusta.

Como experiencias en la utilización de RE para la obtención de comportamientos de locomoción en robots de cuatro patas y en robots hexápodos se pueden destacar varios ejemplos.

En [Bongard2000] se evaluó por medio de simulaciones evolutivas la importancia de la simetría bilateral para la locomoción eficiente. Utilizando un entorno simulado con un motor de física realista, llegaron a la conclusión de que existe una correlación directa entre el nivel de simetría y la eficiencia de la locomoción.

En [Hornby2000a] se utilizó un AE para la generación de patrones de locomoción en los robots AIBO. El AE utilizado fue un AG de estado estacionario basado en valores reales, con selección por torneos, entrecruzamiento y mutación. La definición completa de un patrón de locomoción en AIBO requiere de 61 parámetros reales, que por medio de simplificaciones (estableciendo algunos valores fijos y utilizando los mismos valores para ciertos grupos en base a consideraciones morfológicas) se redujeron a un vector de 20 valores reales. Los resultados de la aplicación del proceso evolutivo, que se realizó íntegramente sobre robots físicos, fueron muy buenos, obteniéndose una velocidad de avance de 900cm/min, mientras que el mejor patrón de locomoción generado de forma manual alcanzaba una velocidad de 660cm/min. El patrón generado de forma evolutiva es el que se utilizaría luego en la versión comercial de AIBO, en la cual se integraban además otros comportamientos obtenidos por medio de técnicas de RE, como la capacidad de seguir una pelota [Hornby2000b].

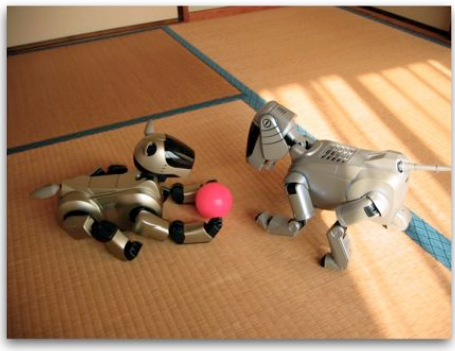


Figura 4.4: *Sony AIBO. El sistema de locomoción de AIBO y comportamientos como la búsqueda de una pelota fuero generados mediante técnicas de robótica evolutiva (Foto: 汲平 - Kyuu Taira)*

Otros ejemplos se pueden encontrar en [Busch2002], donde se utilizó PG, para evolucionar robots de dos, tres, cuatro y seis patas, [Hapke2003], donde por medio de un AE se optimizaron valores de activación de reglas de lógica difusa para el control motriz de robots virtuales, [Zykov2004] que aplicó un AG para obtener patrones rítmicos de locomoción sobre un robot físico y [Wolff2002] donde se optimizó la locomoción de un robot bípedo humanoide utilizando EE (estrategias evolutivas), también sobre un robot físico.

4.4.2 Otros tipos de locomoción

La mayoría de las experiencias en RE se refieren a robots autónomos con ruedas o patas, es decir a la locomoción terrestre, pero las técnicas son potencialmente aplicables a otros medios.

En [Sims1994a][Sims1994b], por ejemplo, se simuló la evolución de sistemas de control y morfologías de robots de diversos tipos, incluyendo robots similares a gusanos que podían desplazarse por medio de contorsiones y robots nadadores.

En [Augustsson2002] se utilizó un AE de estado estacionario con selección por torneos para evolucionar un comportamiento de aleteo en una

implementación robótica de alas fijadas a un cilindro. Se evolucionaron distintos tipos de aleteos, ubicando el cilindro en posición vertical para la evolución de capacidad de elevación y en posición horizontal para la evolución de la capacidad de avance.

4.5 Evolución Encarnada o Personificada (Embodied Evolution)

La RE puede realizarse utilizando exclusivamente robots, sin un componente de simulación, pero evaluar cada instancia de sistemas de control sobre un robot físico puede requerir tiempos prohibitivos para poblaciones grandes y/o una cantidad significativa de generaciones.

La alternativa de utilizar una simulación, si bien reduce en gran medida los tiempos de evaluación puede presentar sus propios inconvenientes. En particular, si no se captura con suficiente precisión los detalles del ambiente y del robot (sensores, efectores, estructura, etc.) el sistema de control obtenido no resultará apropiado al transferirse al robot real. Por el otro lado, una simulación demasiado exacta puede requerir demasiados recursos computacionales, lo que puede implicar también un tiempo de simulación considerable.

En base a estas y similares consideraciones, Watson, Ficici y Pollack [Watson1999][Ficici2000] desarrollaron una metodología alternativa para la aplicación de principios de la RE, la cual denominaron *Embodied Evolution (EE - Evolución Encarnada o Personificada).*

En la EE, la evolución se realiza en una *población* de robots reales, donde la evolución, selección y reproducción se realiza de forma distribuida. Este es un proceso autónomo y asincrónico basado en la interacción de los robots.

Si bien la EE al realizarse completamente en robots reales elimina el problema de transferencia de una simulación a la realidad, así como el problema de simular un ambiente con suficiente detalle y precisión, no escapa al problema de transferencia del ambiente de prueba al ambiente de aplicación, en caso de ser estos diferentes. Para ello se deberán tomar las mismas precauciones que en otras técnicas de RE, asegurándose de exponer a los robots a un ambiente lo mas cercano posible al real. [Watson2002]

La implementación del proceso de EE requiere que cada paso del algoritmo general de la RE sea realizado de forma descentralizada, autónoma y asincrónica. Esto implica que cada robot debe ser capaz de evaluar su propio fitness de forma independiente pero aún así comparable con el de cualquier otro individuo, en cualquier momento.

La reproducción también debe realizarse de forma descentralizada, lo que requiere de un mecanismo de selección/reproducción especial denominado PGTA (Probabilistic Gene Transfer Algorithm – Algoritmo de Transferencia de Genes Probabilístico). Bajo este mecanismo cada robot realiza transmisiones a corto rango y de forma probabilística cada cierto tiempo, en relación a su nivel fitness. Cada transmisión contiene una versión mutada de un gen seleccionado aleatoriamente de su genoma. Si la transmisión es recibida por otro robot, este podrá, también en forma probabilística y en relación a su propio fitness, aceptar dicho gen y sobrescribir uno propio.

Watson, Ficici y Pollack señalan las siguientes condiciones bajo las cuales la EE puede ser recomendable por sobre otras metodologías de RE [Watson2002]:

- La simulación para el dominio es imposible, no está disponible o no resulta suficientemente exacta

- Un algoritmo adaptativo centralizado y coordinado de forma global no puede implementarse o no está disponible, o la coordinación entre pruebas en paralelo es difícil
- Existe interés en evolucionar comportamientos colectivos y/o interactivos
- Existe interés en estudiar la interacción entre comportamientos orientados a tareas y comportamientos orientados a la reproducción
- Los agentes deben aprender directamente en el entorno de aplicación
- Se desea realizar un proceso de transferencia de simulación de agentes a robots físicos (en este caso se tratará de una RE híbrida, donde la última etapa es realizada por medio de EE)

4.6 Transferencia: Simulación vs. Realidad

Como se mencionó en secciones anteriores, en la RE se suelen utilizar simulaciones que luego deben ser transferidas a robots reales. La realización de RE directamente sobre robots reales es posible, pero costosa en términos de tiempo de ejecución y otros recursos.

En muchos casos se utilizan diseños de robots físicos estándar, como el robot Khepera (descripto en la sección 4.2) o similares, siendo el sistema de control el objeto del proceso evolutivo y de una posterior transferencia. En otros, tanto el diseño del cuerpo como su sistema de control son evolucionados de forma conjunta, lo que supone desafíos aún mayores al momento de transferir dichos aspectos a un robot físico.

4.6.1 Transferencia de sistemas de control virtuales

Una simulación solo puede modelar de forma aproximada el mundo real y por lo tanto al transferir un controlador obtenido en un entorno virtual al mundo real surgirán diferencias. La metodología tradicional es tratar de hacer

la simulación lo más realista posible, cubriendo cualquier aspecto que pueda afectar el comportamiento del robot.

En [Jakobi1997] se plantea una metodología alternativa para tratar de garantizar que la transferencia de simulación a realidad sea exitosa. Primero se deben identificar los *aspectos base* de la simulación, que son aquellos en los cuales se considera que el sistema de control dependerá para sus distintos comportamientos. Si se pueden simular adecuadamente estos factores y además se puede garantizar que el sistema dependerá sólo de ellos, entonces se podrá realizar luego una transferencia exitosa. De existir aspectos de la simulación que no se consideran aspectos base pero de los que de todas formas el sistema de control resulta dependiente (denominados *aspectos de implementación*), la transferencia probablemente no tendrá buenos resultados.

El objetivo entonces es lograr que el sistema de control dependa *exclusivamente* de los aspectos base. La forma de lograr esto es evitando que exista una dependencia de cualquier aspecto de implementación, haciendo que estos no sean confiables por medio de variaciones aleatorias en cada prueba. De esta manera, el controlador sólo podrá adaptarse a los aspectos base, que serán los únicos que se mantendrán relativamente estables. Esto es así porque incluso los aspectos base deben tener un mínimo nivel de variabilidad, dado que normalmente es imposible hacer una simulación 100% exacta de ellos.

La variabilidad en ambos casos se introduce en forma de ruido, el nivel del cual se debe variar entre cada prueba, de lo contrario el sistema podría adaptarse a la presencia de cierto nivel de ruido y llegar a depender de éste para su correcto funcionamiento.

Un concepto importante de la metodología propuesta en [Jakobi1997] es el de *simulaciones mínimas*. Una transferencia exitosa no necesita que los aspectos base sean lo más exactos posible ni que se reduzca al mínimo la cantidad de aspectos de implementación. Es suficiente con que se aplique el correcto nivel de ruido en cada componente para que el sistema evolucione

adaptándose exclusivamente a aquellos aspectos en los que estamos interesados y no pueda adaptarse, y por ende depender, de aspectos cuya implementación en la realidad difiera de la simulación. Teniendo esto en cuenta, pueden reducirse al mínimo los aspectos base simulados y enmascarar la falta de precisión en éstos con ruido, haciendo más eficiente la simulación. Esto garantizará una transferencia exitosa, pero obviamente seguirá siendo necesario que los aspectos base considerados sean suficientes para permitir la evolución exitosa del comportamiento deseado.

4.6.2 Transferencia de diseños de cuerpos virtuales

Los casos de transferencia más comunes se basan en la aplicación de un sistema de control obtenido en una simulación por medio de técnicas evolutivas a un cuerpo físico prefijado. De la misma forma, cuando se realizan procesos de RE directamente sobre robots las distintas soluciones se prueban en forma consecutiva sobre el mismo robot o sobre una serie de robots con especificaciones similares.

Sin embargo, el proceso evolutivo no tiene por que estar limitado sólo a la obtención de sistemas de control, sino que puede evolucionar en forma paralela el diseño del cuerpo del robot.

En [Paul2001] se analizó la performance obtenida en la evolución de locomoción bípeda en un ambiente virtual. El robot simulado poseía dos piernas, cada una con tres motores, tres sensores angulares y un sensor de contacto. Se hicieron corridas con poblaciones de 300 individuos durante 300 generaciones, evolucionando conjuntamete sus sistemas de control y la distribución de masa en la morfología. Se utilizó un algoritmo genético con genoma basado en codificación de reales y longitud de genoma fija, utilizando como método de selección un torneo de tamaño 3 entre individuos seleccionados de la mitad superior de cada generación (los 150 individuos con mejor fitness, el cual se definió como la distancia a la que se desplazó cada

individuo luego de un tiempo predeterminado). Se experimentó con tres niveles de alteración máximos permitidos en la distribución de masa en la estructura morfológica de cada individuo: cambios microscópicos, cambios de nivel medio y cambios macroscópicos. Los resultados mostraron que a mayor nivel cambios permitidos en la distribución de masa mayores eran las posibilidades de obtener una locomoción bípeda estable. Esto indica que la libertad de hacer modificaciones estructurales concurrentemente con el controlador puede resultar en mejoras significativas, resultado de una co-evolución de ambos aspectos.

Se plantea por lo tanto el problema de cómo reflejar luego en un robot real no sólo el sistema de control sino también los resultados morfológicos obtenidos en la simulación.

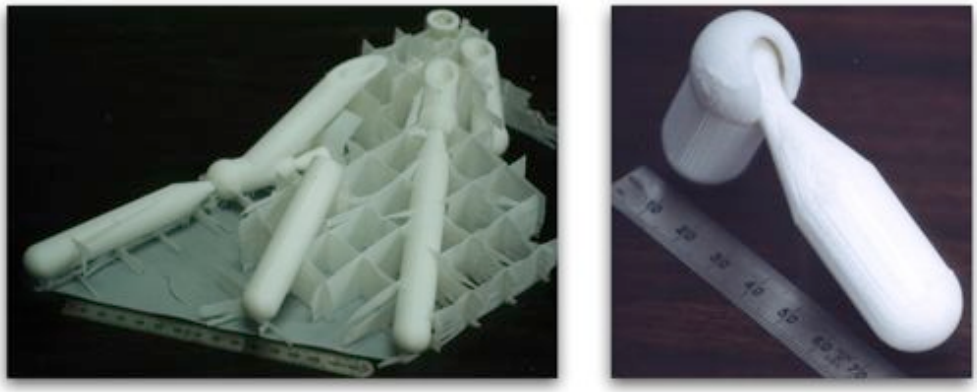
4.6.2.1 Prototipado rápido

En [Lipson2000] se plantea la utilización de una simulación física basada en un "universo limitado", en el cual el robot virtual puede evolucionar su comportamiento y morfología utilizando bloques de construcción básicos: barras, efectores (estructura) y neuronas artificiales (control). Las barras pueden conectarse entre si por medio de articulaciones esféricas libres y, por medio de combinaciones de barras simular también conexiones con desplazamiento circular, lineal o en un plano. Las neuronas se pueden conectar entre si de forma libre, utilizando sinapsis, o con barras, en cuyo caso su salida se traduce en una alteración en la longitud de la barra por medio de un actuador lineal. En el experimento reportado no se utilizaron sensores y la función de fitness se definió como la máxima distancia a la que el robot logre desplazar su centro de masa del punto de origen luego de un tiempo predeterminado.

a *b*

Figura 4.5: *Uno de los robots obtenidos por medio de evolución en el proyecto GOLEM: a) versión virtual, b) robot real generado por técnicas de prototipado rápido a partir de su análogo virtual. (Lipson & Pollack, Brandeis University)*

La generación del robot físico basado en la estructura morfológica y de control obtenida por la simulación se realiza por medio del prototipado rápido, en el cual la estructura se genera por capas depositando material con una impresora/plotter 3D, permitiendo la construcción automática de estructuras complejas. *(figs. 4.5 y 4.6)*



a *b*

Figura 4.6: *El robot es generado capa por capa (a) con todas sus partes, incluyendo también (b) secciones articuladas. (Lipson & Pollack, Brandeis University)*

Los resultados de los experimentos iniciales del proyecto GOLEM en [Lipson2000] mostraron la factibilidad del proceso automático basado en

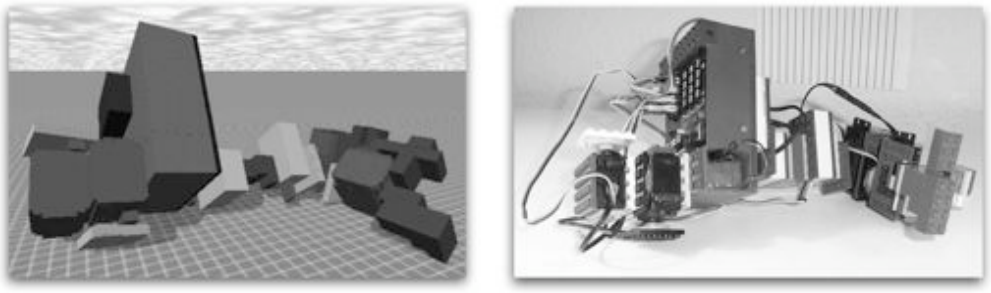
bloques de construcción y prototipado rápido, pero al mismo tiempo resaltaron ciertas deficiencias en el motor de simulación virtual utilizado. En particular, el ambiente de simulación virtual sólo soportaba movimiento en el cual no se tenían en cuenta efectos dinámicos (momento), lo que facilitaba la transferencia posterior al robot real, pero limitaba al mismo tiempo los comportamientos que podían obtenerse. Además, en comparaciones entre las versiones virtuales y físicas de un robot, en algunos casos el fitness obtenido (distancia recorrida) no era proporcional al obtenido en la simulación, debido a que el modelo de fricción utilizado por el simulador no era suficientemente realista.

Como se puede observar en las figuras 4.5b y 4.6, si bien el cuerpo del robot es generado en una sola pieza y con articulaciones funcionales, es necesario instalar luego efectores, una fuente de control (PIC en el que se carga la lógica de control del individuo virtual) y una fuente de energía, conectando todo esto con una cableado eléctrico. Con el objetivo de simplificar aún más el proceso de prototipado rápido, en [Malone2004] se presentan distintos avances relacionados con la generación de forma integrada y automatizada no sólo de estructuras sino también de circuitos, efectores, sensores, cableado eléctrico e incluso baterías.

4.6.2.2 Bloques modulares discretos

Una alternativa a la metodología descripta es la utilización de bloques de construcción modulares discretos y estándar, simplificando de esa forma la construcción de prototipos reales. Ese es el enfoque adoptado en [Macinnes2004], donde se utilizan bloques LEGO y motores a los que se le adhieren piezas en lugares específicos para permitir su integración con otros bloques. *(fig. 4.7)*

En el experimento, cada individuo cuenta con un numero variable de piezas LEGO, un número variable de motores con sensores de ángulo de rotación integrados y un bloque controlador, conteniendo un microprocesador.



a *b*

Figura 4.7: *Robot generado en base a componentes modulares discretos estándar (a) versión virtual en el entorno de simulación (b) versión real. (Macinnes & Di Paolo, University of Sussex)*

El genotipo de cada individuo consiste en una estructura de árbol en el cual cada nodo representa un bloque LEGO, un motor o el controlador, y donde cada conexión entre nodos representa una conexión física entre componentes en el robot. Cada nodo posee además atributos como dimensiones, densidad, posición de la conexión con otro bloque, etc. Existe, por lo tanto, un mapeo directo entre el genotipo y el fenotipo del robot. El proceso evolutivo utiliza la selección por rangos y mutación estructural (agregar / quitar nodos) y de parámetros en cada nodo. No se utiliza entrecruzamiento. La función de fitness, al igual que en muchos casos de RE, se define como la distancia recorrida en un determinado período.

Una consideración adicional en el experimento fue el uso de ruido en la simulación para evitar una sobreadaptación y obtener individuos más robustos, facilitando luego la transferencia a un robot real que necesariamente presentaría alteraciones con respecto a su versión virtual. El ruido consistió en este caso en alterar levemente y de forma aleatoria las dimensiones y densidades de los bloques antes de cada simulación.

Capítulo 5

5 BugSimEvo – Introducción

El presente Capítulo y los capítulos subsiguientes detallan los pasos seguidos durante el desarrollo de un sistema de simulación de robots con patas denominado “BugSimEvo”, su motivación y las características del mismo. Posteriormente, en el capítulo 9, se describen experimentos realizados y sus resultados.

A través de la aplicación de algoritmos genéticos y principios de robótica evolutiva y bio-mimética, BugSimEvo permite simular la evolución de generaciones de robots, con el fin de obtener individuos mejor adaptados, tanto en su morfología como en su comportamiento, partiendo de individuos generados en forma aleatoria.

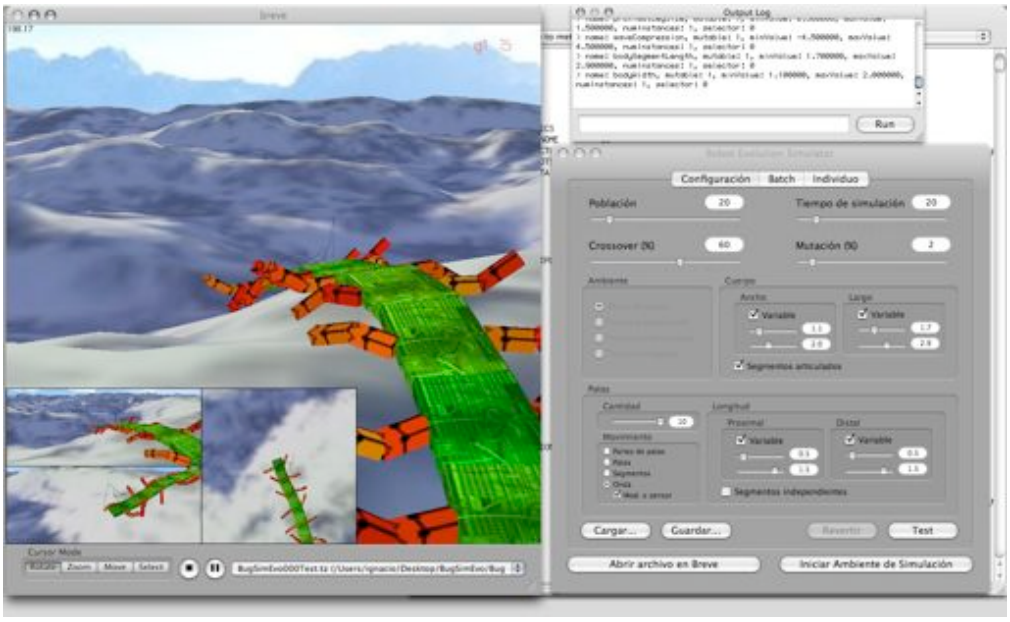


Figura 5.1: *Entorno de simulación de robótica evolutiva BugSimEvo*

5.1 Motivación

El objetivo buscado en el desarrollo del sistema BugSimEvo es investigar la factibilidad de construir un entorno flexible con las características adecuadas para la realización de experimentos de Robótica Evolutiva, tanto en forma manual e interactiva como en forma automatizada y por lotes.

Se busca además evaluar su funcionamiento por medio de la realización de experimentos evolutivos, diseñados en base a experimentos recogidos de la literatura de la RE y, adicionalmente, para verificar de forma empírica principios biológicos de la locomoción animal. En particular, se analizará la capacidad de locomoción obtenida en base a distintos parámetros de configuración: mutación, entrecruzamiento, cantidad de patas, ambiente en el que se realiza la evolución (plano, con obstáculos, terreno irregular). Se analizará también el efecto del ambiente sobre la morfología del robot, estudiando que tipo de adaptaciones evolucionan en entornos diferentes.

5.2 Orígenes y desarrollo

El sistema BugSimEvo se construyó sobre la base del ambiente integrado de simulación Breve. Breve es un IDE *(Integrated Development Environment)* multiplataforma y de código abierto, especialmente diseñado para la programación de simulaciones 3D. Brinda, entre otras funcionalidades importantes, la posibilidad de usar un motor de física 3D integrado (ODE – *Open Dynamics Engine*), facilidades de manipulación de archivos y acceso directo a comandos del sistema operativo. En el Anexo I puede encontrarse un resumen de sus características principales.

BugSimEvo es una versión avanzada del sistema BugSim desarrollado previamente por el autor del presente trabajo, el cual estaba inspirado en una pequeña demo provista por Breve denominada “SuperWalker”, que permitía

simular de forma muy limitada la evolución de un robot de cuatro patas, utilizando una población de cuatro individuos y selección por torneo. Dicha demo presentaba problemas de diseño en su algoritmo evolutivo que causaban que dependiera en forma casi exclusiva de mutaciones y tuviera un progreso extremadamente lento, necesitando horas o incluso días para evolucionar una forma de locomoción estable.

BugSimEvo se diseñó como una plataforma flexible para evaluar distintos aspectos de la robótica evolutiva y cómo ciertas consideraciones bio-miméticas *(capítulo 4)* influyen en la generación de adaptaciones morfológicas, diversidad de comportamientos y en la eficiencia de éstos.

5.3 Arquitectura general

En la figura 5.2 se puede observar la arquitectura general del sistema y sus diferentes módulos, los cuales serán descriptos brevemente a continuación y en mayor detalle en capítulos posteriores.

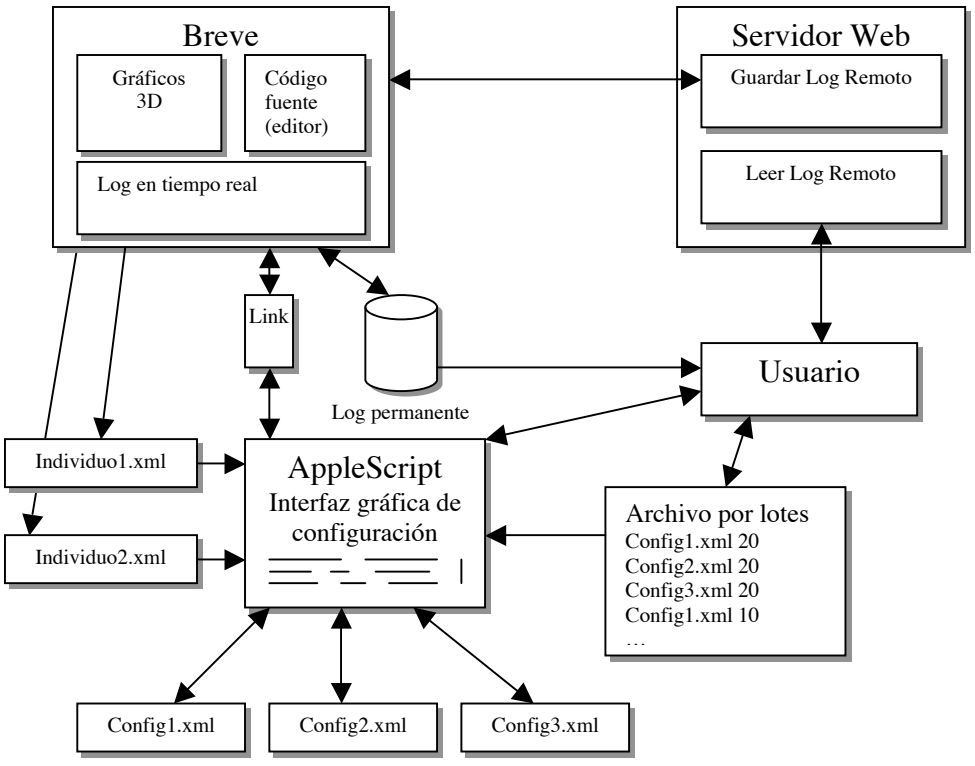


Figura 5.2: *Arquitectura general de BugSimEvo*

5.4 Características Principales

- BugSimEvo implementa un **algoritmo genético** que utiliza:
    - **Selección:** Selección proporcional al fitness RWS y Elitismo (opcional, con nivel configurable).
    - **Entrecruzamiento:** Entrecruzamiento de punto único, con probabilidad configurable.
    - **Mutación:** Mutación Uniforme o Rectangular y Mutación Gaussiana o Normal, seleccionable y con probabilidad configurable.
- **Individuos con genomas serializables en formato XML**, lo que permite archivar y recuperar individuos específicos interesantes.
- **Configuraciones generales serializables en formato XML**, que pueden ser archivadas y recuperadas en cualquier momento.
- **Archivos de procesos por lotes** en formato txt, que permiten indicar una secuencia de configuraciones, junto con la cantidad de generaciones que se desea simular bajo cada una. El sistema puede cargar automáticamente las configuraciones y ejecutarlas en forma secuencial.
- **Log en tiempo real en pantalla (opcional) que indica:**
    - Individuos seleccionados como padres de cada nueva generación
    - Mutaciones producidas en los hijos (indicando que parámetro en particular fue afectado)
    - Estadísticas globales de cada generación:
        - Mejor Fitness obtenido por un individuo en la última generación simulada hasta el momento
        - Fitness Promedio obtenido por un individuo en la última generación simulada hasta el momento
        - Varianza del Fitness de la población
        - Configuración de movimientos, medidas del cuerpo y de cada segmento de cada pata para el mejor individuo de la última generación simulada hasta el momento

        - Configuración de movimientos, medidas del cuerpo y de cada segmento de cada pata promedio de la población
        - Varianza de dichos valores en la población
- **Log permanente (configurable) en el disco rígido.** El sistema automáticamente creará, de ser necesario, una carpeta para guardar logs permanentes de simulaciones pertenecientes a procesos por lotes. Se creará automáticamente una carpeta cuyo nombre indicará el nombre del archivo por lotes junto con la fecha/hora de comienzo de ejecución. Dentro de esta carpeta, y para cada archivo de configuración XML indicado en el archivo por lotes, se crea una carpeta con un número secuencial seguido del nombre de la configuración. En el interior de ésta, se almacenará:
    - **Archivo CSV con estadísticas globales:** Contiene un listado de varias características de la población, ordenados por generación:
        - Mejor Fitness
        - Fitness Promedio
        - Varianza del Fitness
        - Configuración de movimientos, medidas del cuerpo y de cada segmento de cada pata para el mejor individuo de cada generación
        - Configuración de movimientos, medidas del cuerpo y de cada segmento de cada pata promedio de la población para cada generación
        - Varianza de dichos valores en la población para cada generación
    - **Además, opcionalmente por cada generación:**
        - **Genoma completo del mejor individuo de la generación:** En formato XML
        - **Recorrido 3D del robot:** En formato CSV

        - **Imagen 3D del robot:** En su posición final luego de terminada su ejecución y en etapas intermedias, combinado en una única imagen compuesta

- **Log remoto (opcional) en un servidor Web:** Permite enviar los datos de fitness por cada configuración y generación simulada a un servidor web, donde es almacenada en una base de datos, que puede ser consultada en cualquier momento desde cualquier lugar a través de Internet.
- **Aplicación de configuración:** Realizada en AppleScript, permite configurar gráficamente los siguiente parámetros:
    - % de Entrecruzamiento: Probabilidad de entrecruzamiento.
    - % de Mutación: Probabilidad de mutación.
    - Población: Cantidad de individuos por generación.
    - Tiempo de Simulación: Cuanto tiempo virtual se debe simular la acción del robot antes de evaluar el fitness alcanzado por éste.
    - Ancho del cuerpo: Fijo o definiendo un rango con un valor máximo y mínimo.
    - Largo del cuerpo: Fijo o definiendo un rango con un valor máximo y mínimo (nota: el largo se refiere al largo de un segmento, es decir, a la distancia entre dos pares de patas)
    - Cantidad de patas: Entre 2 y 10 (si se utiliza el parámetro interno de multiplicación, la cantidad de patas máxima es ilimitada)
    - Longitud proximal de las patas: Fija o definiendo un rango con un valor máximo y mínimo (proximal: parte superior de la pata)
    - Longitud distal de las patas: Fija o definiendo un rango con un valor máximo y mínimo (distal: parte inferior de la pata)
    - Permitir que los pares de patas puedan ser de medidas diferentes o forzar todas las patas a las mismas medidas.
    - Sistema de sincronización de movimientos
    - Función de fitness: distancia al origen o hacia un objetivo

- Guardar o recuperar una configuración completa, con todos los parámetros ya mencionados, a / de un archivo XML.
- Iniciar una simulación de prueba con los parámetros actuales
- Iniciar un conjunto de simulaciones cargando un archivo por lotes

La versión final del sistema BugSimEvo consiste en más de 3700 líneas[1] de código "Steve" (el lenguaje empleado por el entorno Breve), más de 700 líneas de código AppleScript para el manejo de la interfaz, y alrededor de 200 líneas de código PHP para el manejo del log remoto en un servidor web.

En el capítulo 6 se describen las características del robot simulado, en el 7 los distintos ambientes de simulación y en el capítulo 8 los parámetros generales de la simulación y la interfaz de configuración. Finalmente, en el capítulo 9 se describen diversos experimentos de robótica evolutiva realizados sobre el entorno BugSimEvo y los resultados obtenidos.

[1] Como referencia, la versión anterior del sistema, BugSim, contenía alrededor de 1900 líneas de código, mientras que la demo SuperWaker que inspiró ambos sistemas consiste en alrededor de 400 líneas de código.

Capítulo 6

6 Robot Simulado

El robot posee un cuerpo virtual *(fig. 6.1)* cuyas características están determinadas por un genoma propio del individuo. Dicho genoma determina no sólo el tamaño y proporción de las distintas partes sino también la velocidad, frecuencia y sincronización de los movimientos.

6.1 Cuerpo virtual

BugSimEvo permite la evolución no sólo de sistemas de control sino también de parámetros morfológicos, los cuales se detallan a continuación.

6.1.1 Partes

El robot posee dos componentes principales, el tronco y las patas. El tronco puede ser sólido *(fig. 6.1a)* o con múltiples segmentos articulados *(fig. 6.1b)*. Las patas poseen cada una dos partes *(fig. 6.2)*: un segmento proximal, que se conecta con el tronco, y un segmento distal, el cual entra en contacto directo con el terreno.

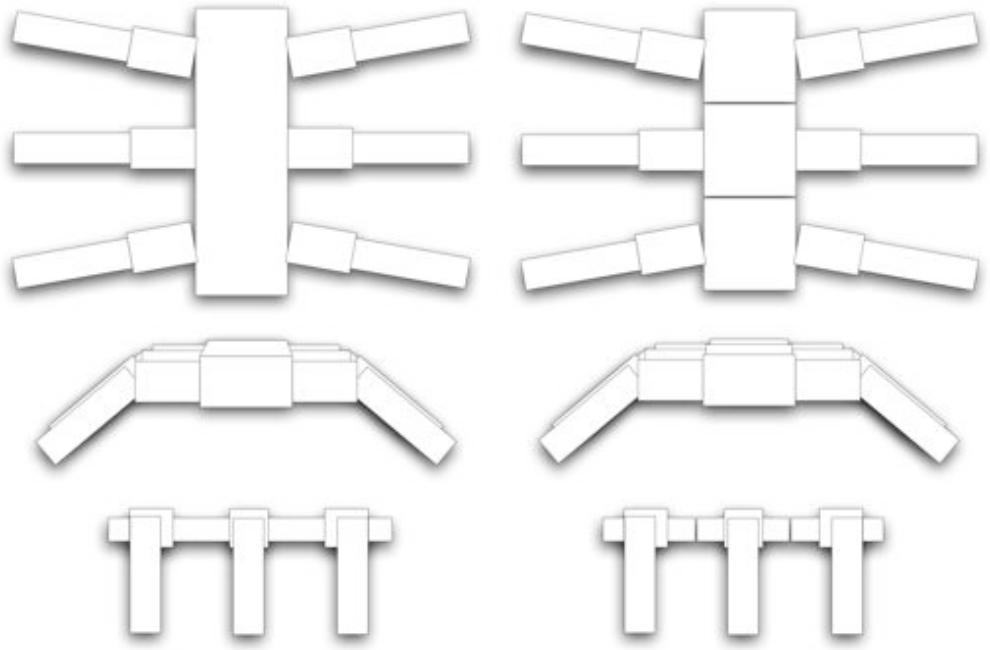


Figura 6.1: *Robot virtual con a) cuerpo sólido b) cuerpo articulado*

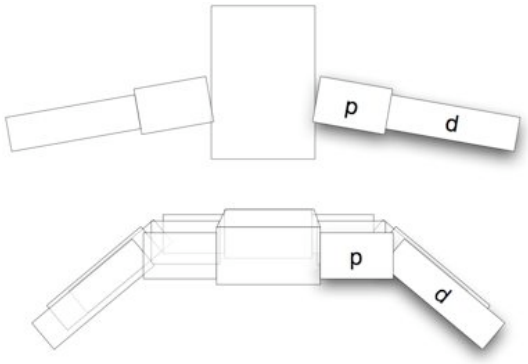


Figura 6.2: *componentes proximal (p) y distal (d) de las patas*

Tanto el ancho como el largo del cuerpo son variables y están determinados de forma independiente por el genoma de cada individuo. Cada componente posee una densidad fija lo que hace que a mayores dimensiones aumente el peso del robot y por lo tanto la fuerza requerida para que este se desplace. *(fig. 6.3)*

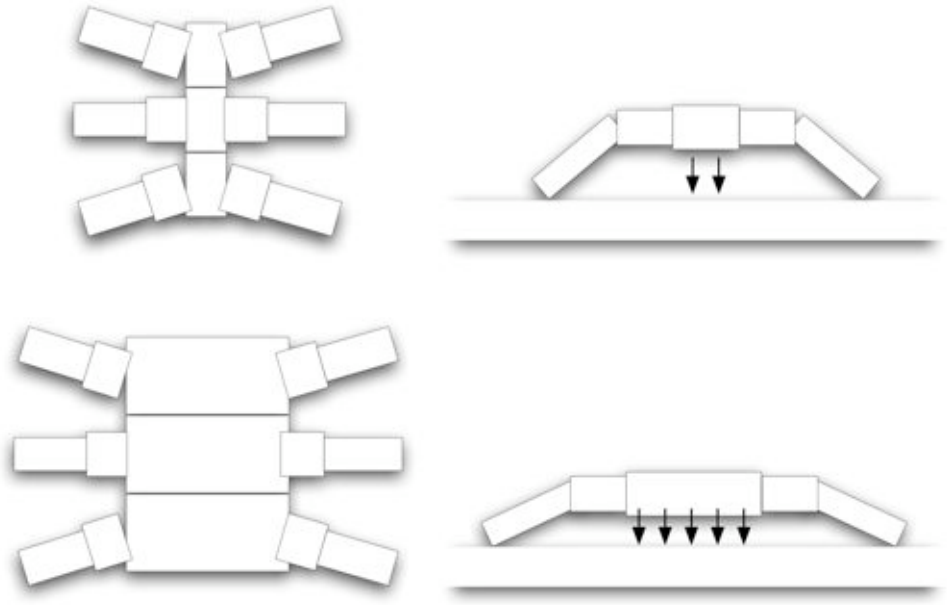


Figura 6.3: *El peso del robot se incrementa al aumentar el ancho y/o largo de los segmentos del cuerpo, incrementando las fuerzas a las que están sometidas las articulaciones.*

Cada segmento de cada pata posee un largo, el cual está determinado por el genoma, mientras que su grosor se calcula de forma automática para que el peso se mantenga constante. *(fig. 6.4 y 6.5)*

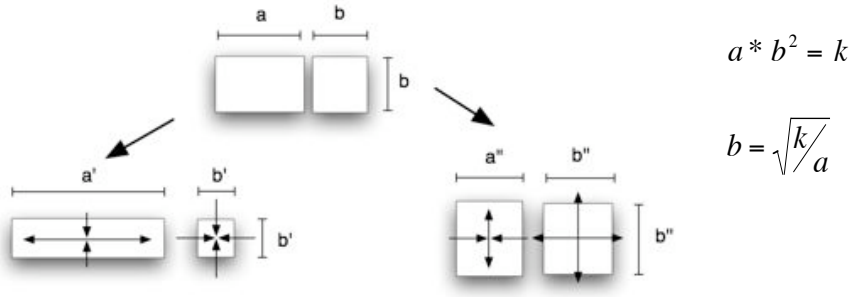


Figura 6.4: *Al modificar la longitud (a) de una sección de una extremidad, su grosor (b, ancho/alto) se recalcula automáticamente para conservar el volumen (k) y por lo tanto el peso de la misma. Si la longitud se incrementa (a'), el grosor disminuye (b') y viceversa (a'' y b'')*

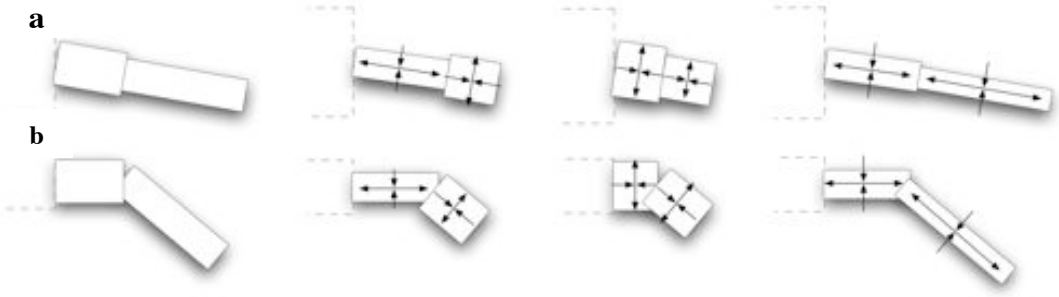


Figura 6.5: *Ejemplos de variantes de extremidades completas (a) vista superior (b) vista frontal*

6.1.2 Movimiento

El robot controla su desplazamiento por medio del movimiento de sus patas. Los parámetros codificados en el genoma afectan la velocidad, frecuencia y sincronización de los movimientos. Cada pata posee dos grados de libertad, un movimiento horizontal a nivel del segmento proximal de la pata y un movimiento vertical a nivel del segmento distal. *(fig. 6.6 y 6.7)*

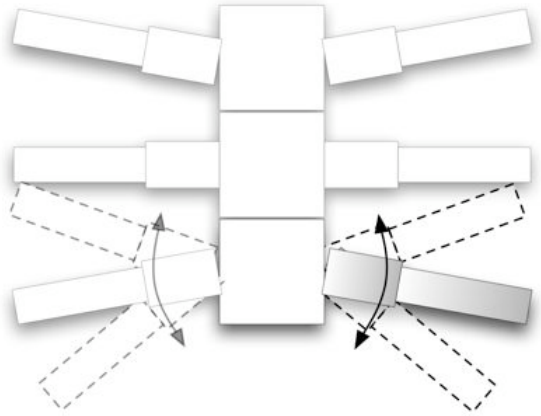


Figura 6.6: *Componente horizontal del movimiento de una pata. Este se produce a nivel de la articulación entre el segmento proximal de la pata y el cuerpo.*

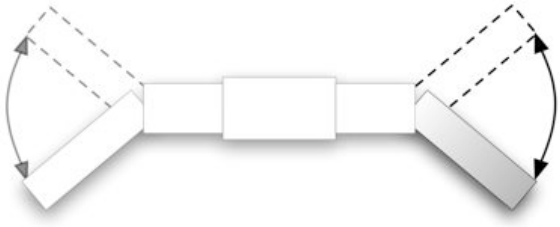
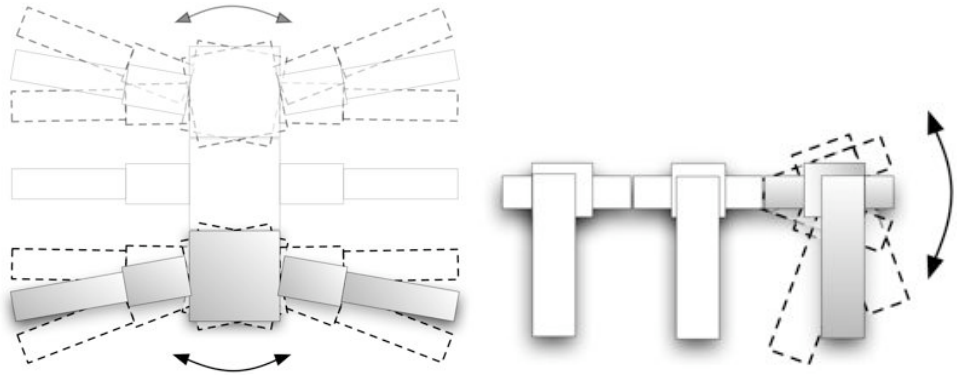


Figura 6.7: *Componente vertical del movimiento de una pata. Este se produce a nivel de la articulación entre el segmento distal y el segmento proximal de la pata.*

Además del movimiento de las patas, una de las variantes del robot posee un cuerpo articulado. En este caso, las articulaciones no se encuentran bajo control directo de los parámetros codificados en el genoma. *(fig. 6.8)*



a **b**

Figura 6.8: *Componentes (a) horizontal y (b) vertical del movimiento de un segmento del cuerpo. Este se produce a nivel de la articulación entre segmentos y no es controlado por parámetros codificados en el genoma. La articulación es de tipo resorte, brindando flexibilidad y amortiguación al robot.*

6.2 Genoma

Los parámetros que determinan las características físicas y de comportamiento del robot se encuentran codificados y organizados en el genoma de acuerdo a la variante de cuerpo (sólido o articulado) y el controlador de movimiento a utilizar.

La estructura general consiste en un grupo de genes que determinan el tamaño del cuerpo y las patas y otro grupo que determina el movimiento de cada pata. *(fig. 6.9)*

Figura 6.9: *Estructura general del mapeo de genes, común a todas las variantes de diseño de cuerpo y de control de movimientos.*

A continuación se describen las distintas variantes con sus codificaciones específicas correspondientes.

6.2.1 Diseño del cuerpo

El robot posee dos grandes componentes estructurales: el tronco y las patas *(fig. 6.1)*. Las dimensiones de ambos están codificadas en el genoma, en dos grupos de genes.

Existen dos variantes de mapeo asociadas al diseño de las patas. Uno de los mapeos permite la variación de tamaño entre pares de patas sucesivos, mientras que en otro todas las patas poseen las mismas dimensiones. En los dos casos, las patas poseen simetría bilateral.

Existen además dos variantes de diseño de cuerpo para el robot: cuerpo sólido y cuerpo articulado. Asociadas a cada una hay un diseño de genoma con un mapeo específico.

6.2.1.1 Variante con patas del mismo tamaño

En esta variante, todas las patas son del mismo tamaño, por lo que sólo es necesario codificar las longitudes proximal y distal de una pata modelo en el genoma (el grosor se calcula automáticamente como se mostró en las figuras 4 y 5). Estos dos valores serán utilizados para el diseño de todas las extremidades *(fig. 6.10)*.

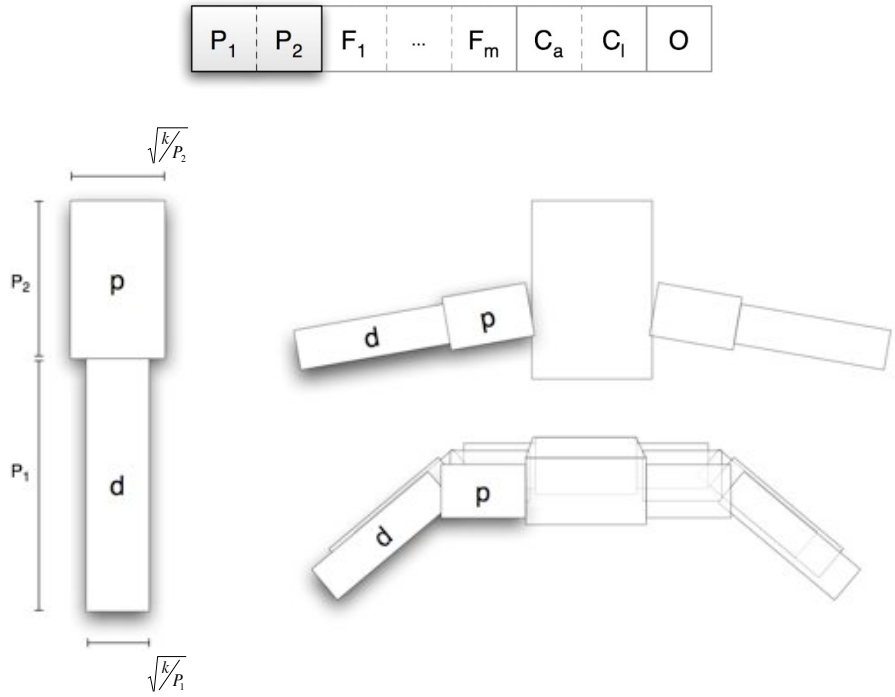


Figura 6.10: *Las posiciones indicadas como P_1 y P_2 codifican valores que determinan, respectivamente, el largo distal y proximal de la pata modelo. El grosor de cada componente se calcula automáticamente en base a estos valores, manteniendo el volumen constante (k).*

6.2.1.2 Variante con patas de tamaños independientes

En esta variante, el tamaño de cada par de patas es independiente, por lo que es necesario codificar *n/2* longitudes proximales y *n/2* longitudes distales, donde *n = cantidad de patas*. Las longitudes proximales y distales se codifican de forma intercalada, para expresar en el genoma la relación física existente entre los componentes de cada pata y la relación entre patas

cercanas. Esto aumentará la probabilidad que las codificaciones de segmentos completos de patas no se vean afectadas por entrecruzamiento durante la reproducción, dado que formarán esquemas de longitud reducida *(fig. 6.11)*.

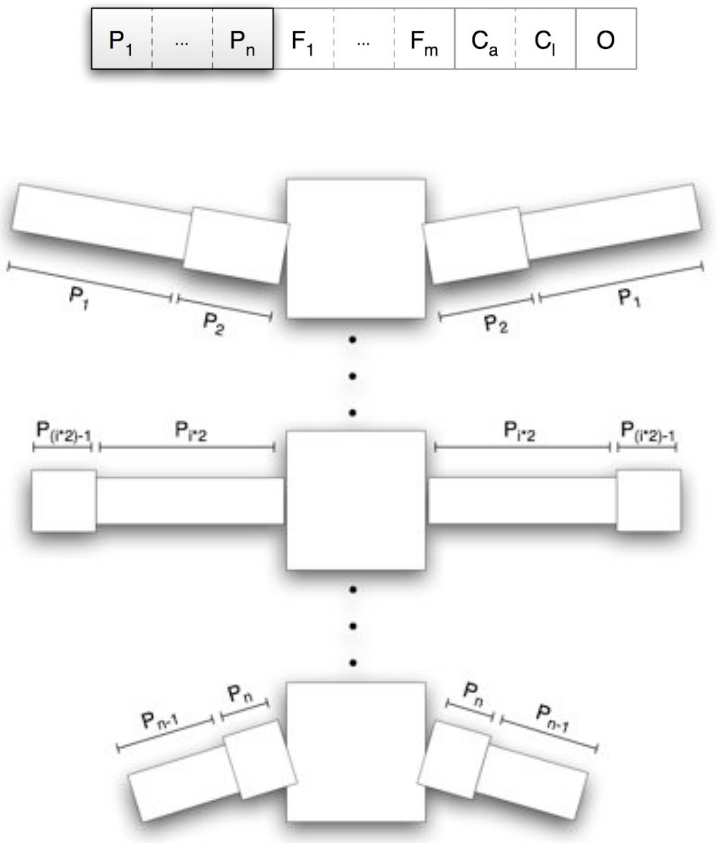


Figura 6.11: *Las posiciones indicadas como P_1 a P_n codifican valores que determinan el largo de los componentes de cada pata. Cada pata tiene dos segmentos: un segmento proximal y otro distal. Estos valores se codifican de forma alternada, con las longitudes proximales en las posiciones pares y las distales en las impares. Los pares patas en lados opuestos del mismo segmento del cuerpo son simétricas, los tamaños pueden variar entre un par de patas y el siguiente.*

6.2.1.3 Variante con cuerpo sólido

En esta variante, el cuerpo del robot está compuesto por un único segmento. A ambos lados a lo largo del mismo se encuentran las patas, conectadas de forma simétrica al segmento central *(fig. 6.12)*.

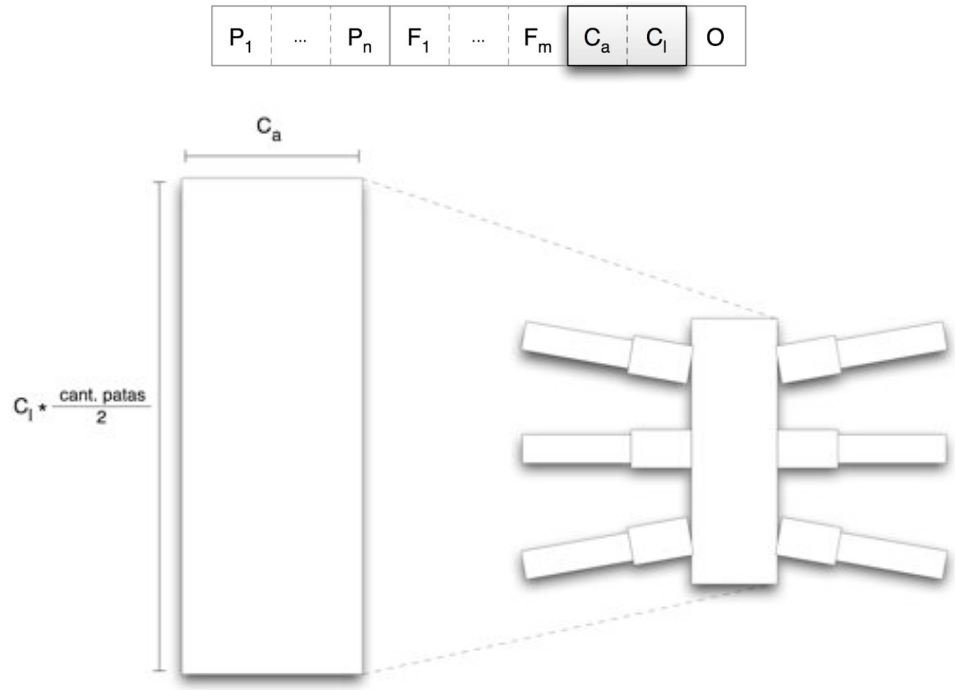


Figura 6.12: *Las posiciones indicadas como C_a y C_l codifican valores que determinan el ancho y largo del cuerpo, respectivamente. El ancho del cuerpo se mapea directamente, mientras que el largo dependerá no sólo de C_l sino también de la cantidad de patas. Este mapeo particular fue diseñado para evitar que un cambio en la cantidad de patas del robot a simular obligue a recalcular el rango de valores permitidos para C_l*

6.2.1.4 Variante con cuerpo articulado

En este caso, el cuerpo del robot está compuesto por múltiples segmentos. Las patas se conectan de forma simétrica, un par por cada segmento *(fig. 6.13)*.

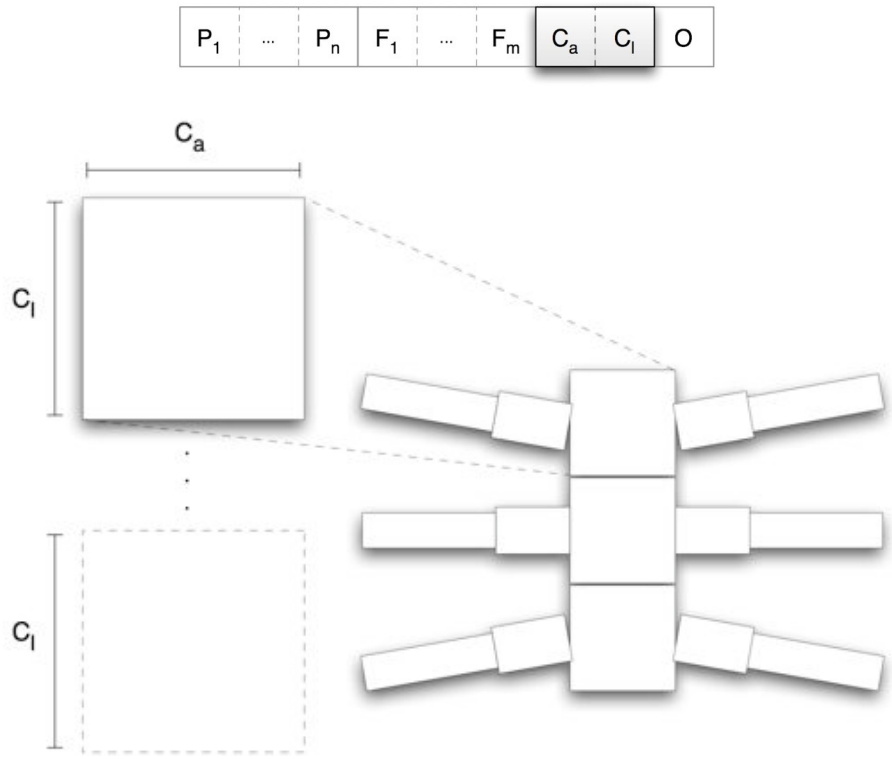


Figura 6.13: *Las posiciones indicadas como C_a y C_l codifican valores que determinan el ancho y largo del cuerpo, respectivamente. El ancho del cuerpo se mapea directamente, mientras que C_l define el largo de cada segmento, por lo que el largo total del cuerpo dependerá no sólo de C_l sino también de la cantidad de segmentos. Como cada par de patas está conectado a un segmento, se puede definir el largo total del cuerpo de manera similar a la variante con cuerpo sólido:* $C_l * \frac{cant.patas}{2}$

6.2.2 Movimiento del cuerpo

En la sección 6.1.2 se describieron las articulaciones y grados de libertad de movimiento de las distintas partes del robot. El desplazamiento del robot se produce a través del movimiento de sus patas. La fuerza, velocidad y sincronización de dichos movimientos están codificados en el genoma.

El componente *O* del genoma codifica la frecuencia de una onda sinusoidal *(fig. 6.14)* la cual será utilizada como parámetro base por otros componentes del genoma. Estos componentes, F1 … Fm codifican una fase dentro de la onda O y el conjunto onda-fase determina el movimiento de cada segmento de cada pata (fig. 6.15).

Existen cuatro diferentes mapeos para los parámetros F que dependerán del grado de independencia y sincronización que se deseen fijar de manera estructural.

En el nivel más bajo, cada uno de los componentes del movimiento de cada pata tendrá una fase completamente independiente, respecto a la onda global. Para obtener una locomoción estable y eficiente el AG deberá evolucionar cada componente y encontrar valores útiles para una serie de n*2 parámetros de fase (donde n = cantidad de patas).

En los siguientes niveles, se utilizan conceptos extraídos de la biología, en particular relacionados con el desplazamiento de los insectos, para establecer sincronizaciones a nivel estructural. Con esto se reduce el número de parámetros de fase a optimizar hasta llegar un mínimo de 4 en el nivel más alto, sin que importe en este caso la cantidad de patas.

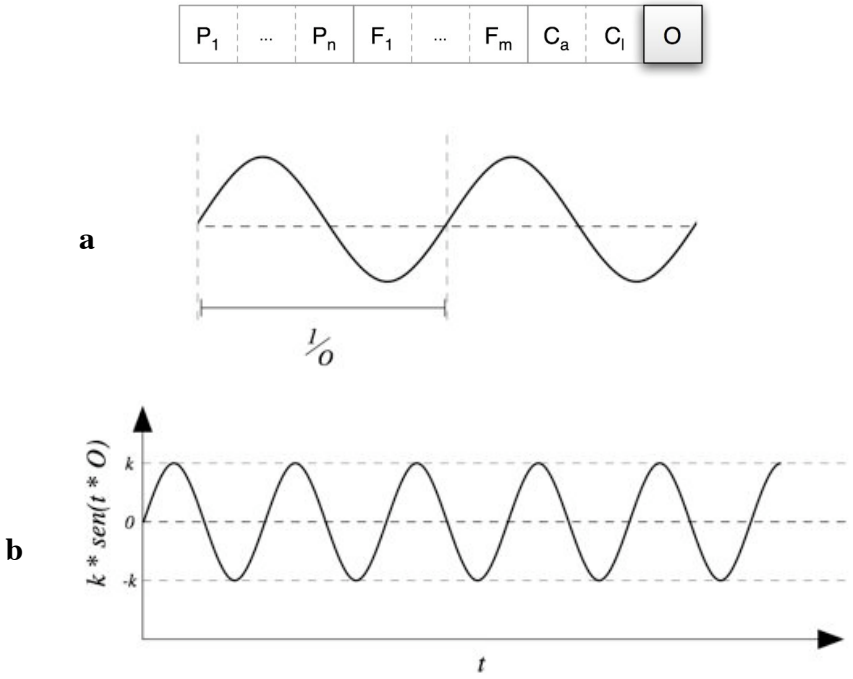


Figura 6.14: *La última posición del genoma, identificada como O, codifica (a) la frecuencia de una onda sinusoidal y, por lo tanto,* $\lambda = 1/O$. *La onda completa (b) se define en función del tiempo t, la frecuencia O y una amplitud constante k.*

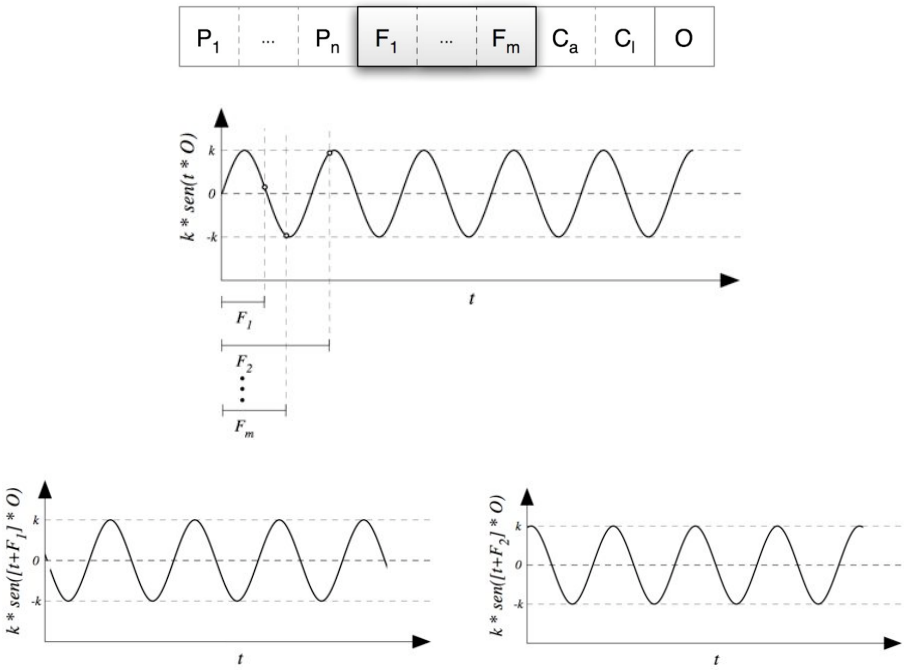


Figura 6.15: *Las posiciones indicadas como* F_1 *a* F_m *codifican desplazamientos de fase dentro de la onda global. Las diferencias de fases entre las ondas generadas por la serie de parámetros F determinarán las relaciones entre los movimientos de las distintas partes del robot.*

6.2.2.1 Variante A: Secciones de patas independientes

El robot posee *n*2* articulaciones móviles activas (*n* = cantidad de patas), dos en cada pata. En la versión con cuerpo articulado, el robot tiene también otras *(n/2)-1* articulaciones, una entre cada segmento del cuerpo, pero estas son pasivas – no están bajo su control directo.

Cada articulación activa tiene un grado de libertad. En una pata hay dos articulaciones:

- la articulación entre el segmento distal y el proximal, que puede rotar sobre un plano vertical y
- la articulación entre el segmento proximal y el cuerpo, que puede rotar sobre el plano horizontal y afecta a toda la pata

El mapeo más simple, que se denominará "Variante A" es asignar una fase *F* individual por cada articulación, de forma que todas sean independientes. En este caso, el algoritmo genético deberá optimizar ***n*2*** fases (además del resto de los parámetros, como la frecuencia y las dimensiones físicas) para lograr una locomoción eficiente.

La onda determinada por la combinación de una frecuencia propia del individuo, una amplitud constante y la fase específica asociada a una articulación, se utiliza para asignar a ésta una velocidad y sentido de movimiento en cada instante de tiempo *t*.

Es importante destacar que esta velocidad se asigna de forma indirecta, por medio de la aplicación de fuerzas. De ser necesario, la fuerza se incrementará para tratar de obtener la velocidad requerida, pero cada articulación tiene un límite predefinido de fuerza máxima. La fuerza es aplicada por el motor de física de la simulación y dependiendo de factores como el peso, posición, velocidad en el instante anterior y cualquier otra

fuerza actuando sobre la articulación, podrá o no alcanzar a la velocidad especificada por la onda en un instante dado.

6.2.2.2 Variante B: Patas independientes

El siguiente mapeo, que se denominará "Variante B" se basa en la observación de que en la naturaleza, las patas en lados opuestos del cuerpo normalmente tienen un movimiento similar. Es decir, durante el desplazamiento, la combinación de movimientos de los componentes de las patas utilizada es la misma para una pata del lado izquierdo que para una del lado derecho. Además, en insectos como el ciempiés y otros similares, compuestos por varios segmentos, esta similitud de movimientos también está presente entre patas de un mismo lado.

En este caso, el algoritmo genético deberá optimizar ***n+1*** fases (además del resto de los parámetros, como la frecuencia y las dimensiones físicas) para lograr una locomoción eficiente.

6.2.2.3 Variante C: Segmentos del cuerpo independientes

Esta variante se basa también en la apreciación de que los cuerpos de muchos insectos presentan una serie de segmentos, cada uno con un par de patas, donde el movimiento de cada pata es similar. En particular, en esta variante cada segmento posee la misma sincronización entre patas en lados opuestos y una fase adicional independiente por segmento.

En este caso, el algoritmo genético deberá optimizar ***(n/2)+2*** fases (además del resto de los parámetros, como la frecuencia y las dimensiones físicas) para lograr una locomoción eficiente.

6.2.2.4 Variante D: Segmentos del cuerpo sincronizados

Inspirado en insectos como ciempiés y otros compuestos por series de segmentos, esta variante se basa en la definición de un módulo modelo compuesto de un segmento con dos patas, y una relación de fase relativa entre cada segmento sucesivo.

En este caso, el algoritmo genético deberá optimizar ***4*** fases (además del resto de los parámetros, como la frecuencia y las dimensiones físicas) para lograr una locomoción eficiente.

Capítulo 7

7 Ambiente

Dos de los aspectos más importantes para el proceso evolutivo son el ambiente en el cual este se llevará a cabo y la función de fitness que se utilizará para evaluar a cada individuo. BugSimEvo permite utilizar cuatro ambientes diferentes predefinidos y dos funciones de fitness alternativas.

7.1 Terreno plano

El ambiente más simple consiste en un plano sobre el cual el robot se puede desplazar. La función de fitness utilizada en este caso es la distancia a la que se encuentra el robot luego de un tiempo fijo t_n predeterminado con respecto al punto de partida.

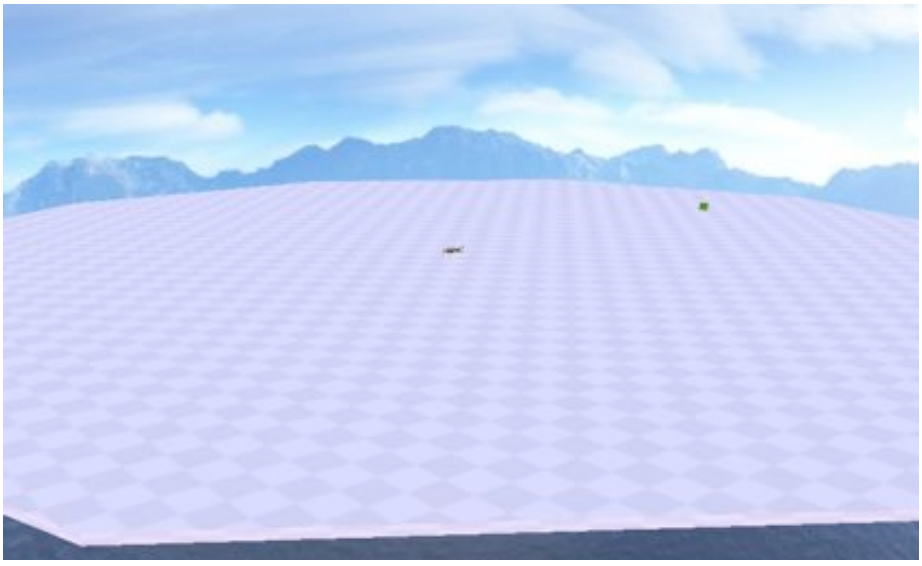


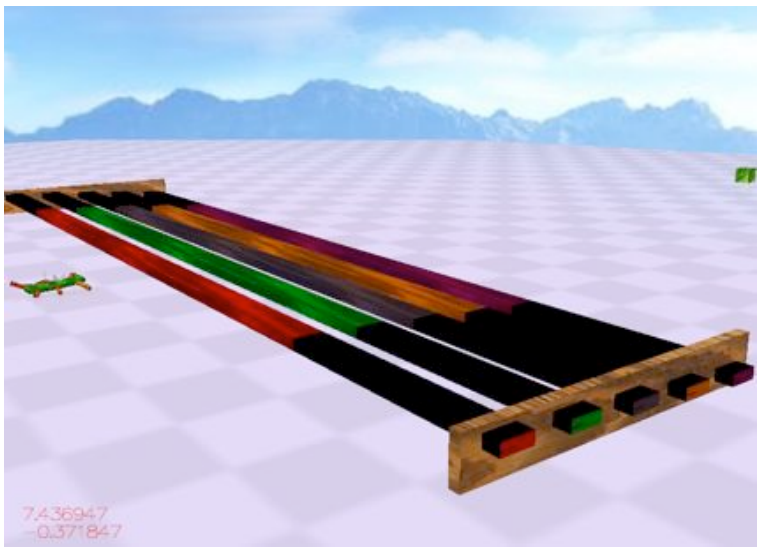
Figura 7.1: *Terreno plano*

Función de fitness basada en distancia al origen:

$$f_{t_n} = \sqrt{(x_{t_n} - x_{t_0})^2 + (y_{t_n} - y_{t_0})^2}$$

7.2 Terreno plano con obstáculos

Una versión más exigente del ambiente consiste en agregar obstáculos fijos, lo que permite estudiar adaptaciones particulares a dichos obstáculo. La función de fitness utilizada en este caso es la distancia a la que se encuentra el robot de un objetivo predefinido (ubicado detrás de los obstáculos) luego de un tiempo predeterminado.



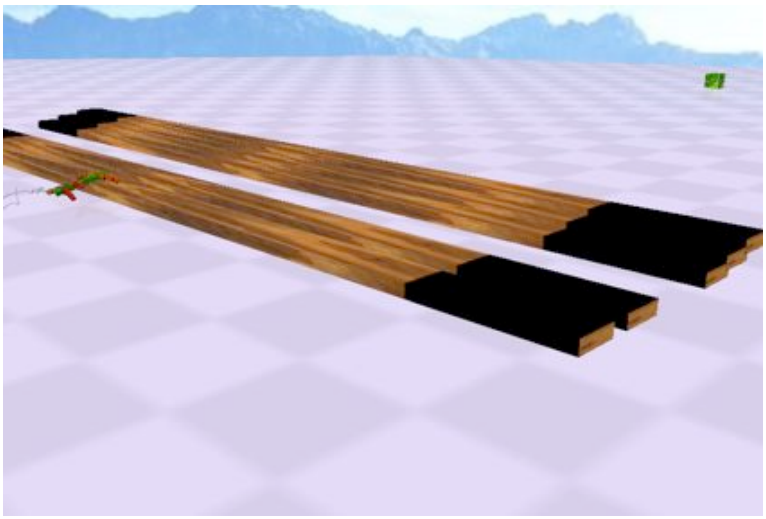


Figura 7.2: *Dos variantes de terreno plano con obstáculos. Los obstáculos se encuentran entre el robot (izquierda en ambas imágenes) y el objetivo (derecha). En el ambiente de la primera imagen, el robot está obligado a pasar por debajo, mientras que en la segunda debe trepar para alcanzar su objetivo.*

Función de fitness basada en distancia al objetivo *(x', y')*:

$$f_{t_n} = \frac{1}{\sqrt{(x'_{t_n} - x_{t_n})^2 + (y'_{t_n} - y_{t_n})^2}}$$

7.3 Terreno irregular

El ambiente plano es un punto de partida interesante ya que tiene la particularidad de que la forma de desplazamiento que es eficiente en un punto del mismo es igual de eficiente en cualquier otro. Sin embargo, los ambientes naturales raramente presentan esta característica.

Para simular mejor la característica de variabilidad en el terreno, el ambiente con terreno irregular usa un mapa de elevación que determina la altura en distintos puntos de la superficie.

El mapa de elevación es una grilla que contiene M x M enteros que determinan la altura a espacios regulares a lo largo de la superficie, y se utiliza la interpolación lineal para obtener valores de altitud en cualquier punto dado de ésta.

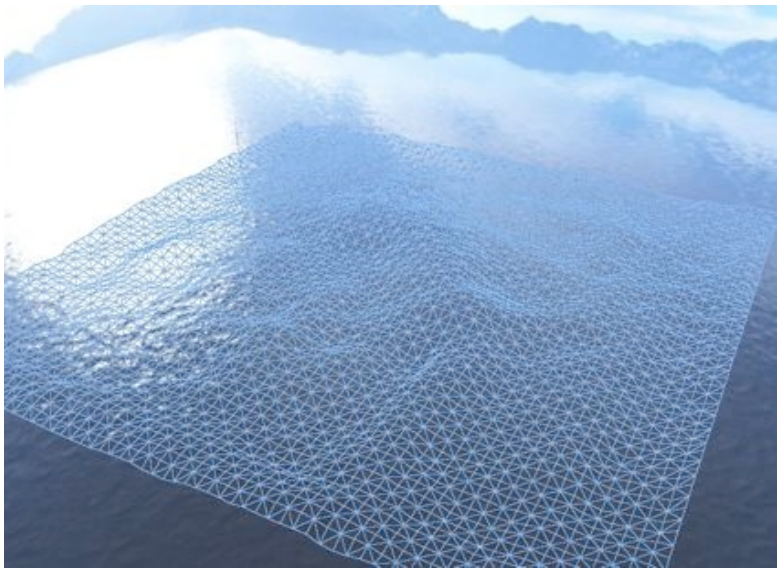


Figura 7.3: *Terreno irregular. Vista en modo grilla.*

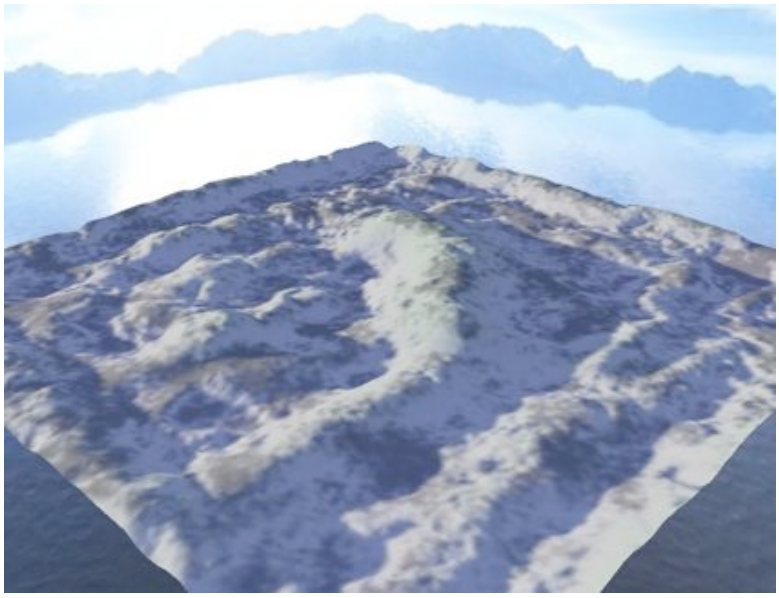


Figura 7.4: *Terreno irregular. Vista en modo normal.*

Capítulo 8

8 Parámetros e Interfaz de control

Para facilitar el uso del sistema BugSimEvo se diseñó una interfaz visual de configuración organizada en tres secciones:

- Sección *"Configuración"*
- Sección *"Batch"*
- Sección *"Individuo"*

La primera permite especificar los parámetros de una simulación individual, ejecutar la misma en el entorno de simulación, guardarla para futuras ejecuciones y cargar simulaciones previamente guardadas.

La segunda sección, "Batch" permite cargar archivos por lotes que especifican series de simulaciones a ejecutar una tras otra. Estas simulaciones deben haber sido primero configuradas en la sección anterior.

La sección final, "Individuo", permite cargar cualquier individuo cuya evolución haya sido resultado de la ejecución de una simulación previa, de forma de poder analizar su comportamiento, tanto en el ambiente original en el que evolucionó como en uno diferente.

8.1 Configuración

Esta pantalla permite especificar los parámetros de una simulación, desde parámetros globales de la población, incluyendo su tamaño, niveles de entrecruzamiento y mutación hasta características físicas y de comportamiento de los individuos de la misma.

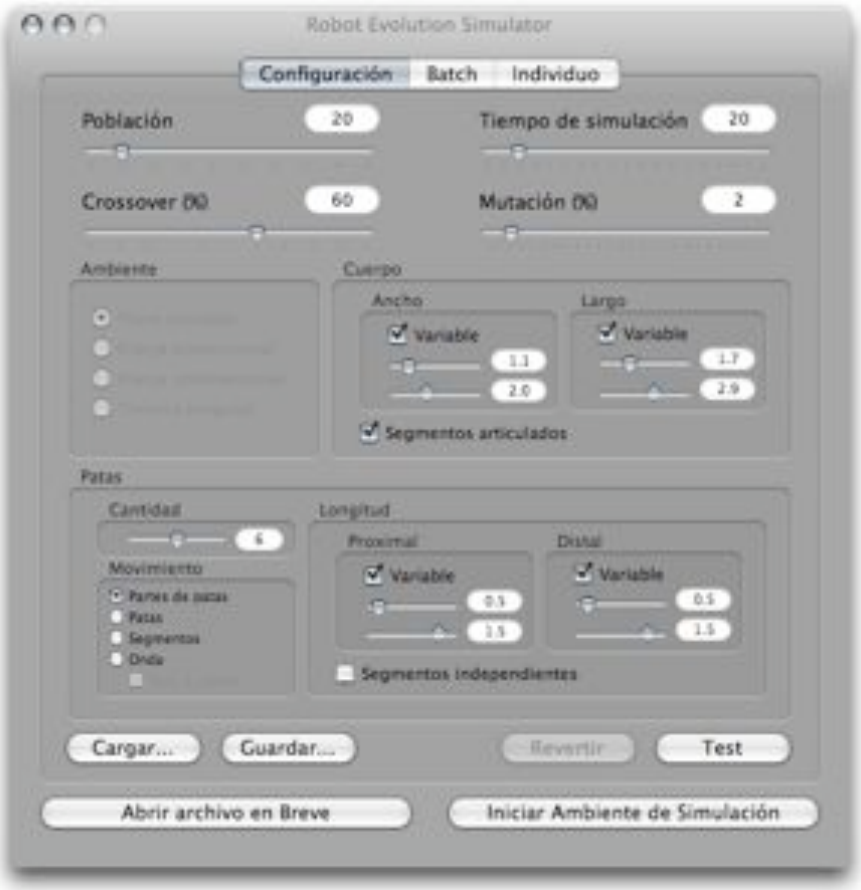


Figura 8.1: *Vista general de la sección Configuración de la interfaz*

Los parámetros están organizados de forma jerárquica, con parámetros globales de la simulación en la sección superior, luego los del ambiente, seguido de parámetros físicos y de comportamiento de los individuos, agrupados de acuerdo a las diferentes partes del robot afectadas.

Hay cuatro parámetros globales:

Figura 8.2: *Grupo de parámetros globales de la sección Configuración*

- ***Población:*** determina la cantidad de individuos que componen cada generación
- ***Tiempo de simulación:*** cantidad de segundos simulados por individuo
- ***Nivel de entrecruzamiento (crossover):*** probabilidad de que se produzca entrecruzamiento simple durante la reproducción
- ***Nivel de mutación:*** probabilidad de mutación de un gen durante la reproducción

El grupo de opciones ***Ambiente*** permite seleccionar el ambiente en el cual se moverán los individuos (ver capítulo 7 para más detalles).

El grupo ***Cuerpo*** permite configurar el rango de valores permitidos para las dimensiones del cuerpo, así como también si este presentará o no articulaciones:

Figura 8.3: *Grupo de parámetros relacionados con el cuerpo del robot*

- ***Ancho / Largo***
    - ***Variable:*** si se selecciona esta opción, este valor podrá variar entre individuos y estará sujeto a los efectos de la mutación y el entrecruzamiento, en caso contrario será constante
    - ***Selectores deslizables:*** establecen los valores mínimo y máximo permitidos
- ***Segmentos articulados:*** si se selecciona esta opción, el tronco del cuerpo estará formado por una serie de segmentos articulados, en caso contrario será sólido. Para más detalles consultar el capítulo 6.

El grupo ***Patas*** abarca aspectos tanto físicos como de comportamiento. Al igual que para el caso anterior, se pueden consultar más detalles en el capítulo 6:

Figura 8.4: *Grupo de parámetros relacionados con las patas del robot*

- ***Cantidad:*** define la cantidad de extremidades que poseerá cada individuo a simular. Dado que se utilizan cadenas genéticas de longitud fija, este parámetro no variará durante el curso de una simulación. En caso de que se haya especificado un cuerpo articulado, este parámetro determinará también la cantidad de segmentos del mismo: cada segmento del cuerpo tendrá asociado un par de patas.
- ***Longitud***
    - ***Proximal / Distal:*** cada extremidad posee dos partes, una proximal, conectada al tronco del cuerpo, y una distal, que se

apoya sobre el suelo y articula con la primera. La longitud de cada parte puede configurarse por separado::

- ***Variable:*** si se selecciona esta opción, la longitud de la sección de la pata podrá variar entre individuos y estará sujeta a los efectos de la mutación y el entrecruzamiento, en caso contrario será constante
- ***Selectores deslizables:*** establecen los valores mínimo y máximo permitidos

- ***Segmentos independientes:*** por defecto, todas las patas comparten el mismo tamaño y proporción para un individuo dado. Si se selecciona esta opción, cada par de patas asociado a un segmento del cuerpo será independiente, conservando la simetría entre patas opuestas (izquierda / derecha).

- ***Movimiento:*** el comportamiento de cada individuo está determinado por el movimiento de sus patas, dado que éstas son los únicos efectores de los que dispone para desplazarse y por la función objetivo. Se presentan diferentes opciones de sincronización de movimientos, con una función objetivo por defecto y una alternativa.
  - ***Partes de patas:*** movimiento no sincronizado, con cada sección de cada pata moviéndose de forma independiente de las demás
  - ***Patas:*** la secuencia de movimientos de una pata completa se utiliza como base para el movimiento de las demás, pero cada pata iniciará su movimiento de manera independiente
  - ***Segmentos:*** la secuencia de movimientos de una pata completa se utiliza como base para el movimiento de la pata opuesta en el mismo segmento y luego este segmento se utiliza como base para los segmentos sucesivos, aunque cada uno iniciará su secuencia de manera independiente
  - ***Onda:*** similar al anterior, con la particularidad que la diferencia de fase entre el primer y segundo segmento se utilizará sin cambios entre el segundo y el tercero y así sucesivamente

- ***Modulación por sensor:*** permite alterar el movimiento de las patas de acuerdo a un sensor simple que determina el ángulo entre la dirección de avance del robot y un objetivo físico ubicado en el ambiente. Si se selecciona esta opción, se altera también la función objetivo: en lugar de distancia al origen se tomará en cuenta la distancia al objetivo, siendo inversamente proporcional en este caso la relación entre el fitness y la distancia

8.2 Modo Batch

A través de esta pantalla se pueden cargar archivos por lotes, los cuales permiten especificar series de simulaciones a ejecutar en orden, una tras otra. El formato de archivo es texto plano, indicando en cada línea el nombre de un archivo xml de configuración (generado en la sección anterior), seguido de la cantidad de generaciones que se desea ejecutar, separados por un espacio.

Figura 8.5: *Vista general de la sección Batch de la interfaz*

8.3 Carga y simulación de individuos

El resultado buscado a través de la simulación de sucesivas generaciones es obtener individuos que se adapten al entorno y a los objetivos especificados en la configuración de la misma. En ocasiones, puede ser útil volver a reproducir un individuo particular, por ejemplo, para estudiar su comportamiento en un ambiente distinto. A través de este módulo se puede recuperar cualquier individuo que se haya evolucionado previamente y realizar una nueva simulación del mismo.

Figura 8.1: *Vista general de la sección Individuo de la interfaz*

Capítulo 9

9 Resultados Experimentales

Los Algoritmos Genéticos permiten generar posibles soluciones a problemas complejos mediante la especificación en forma global de los resultados esperados y una función de evaluación adecuada. El diseño de estructuras de cuerpos y configuración de movimientos para lograr un comportamiento aparentemente simple, como lo es caminar en línea recta es en realidad una tarea compleja que, sin embargo, presenta características que la hacen ideal para un AG.

Se utilizó el sistema desarrollado, BugSimEvo, para explorar la evolución de diferentes características físicas y de locomoción, realizando una serie de experimentos diseñados de forma de evaluar distintos factores y su influencia, así como también para explorar posibles interacciones entre éstos.

Se simularon un total de más de 300000 individuos, utilizando más de 300 horas de CPU, generándose un total aproximado de 50000 archivos de resultados (estadísticas globales, genomas completos y registros de recorridos en 3D).

9.1 Configuración Experimental

Dado el alto uso de recursos necesarios para una simulación física realista, los experimentos se realizaron en forma paralela en varias máquinas, todas con configuraciones idénticas de hardware y software, descriptas a continuación:

- Hardware
    - 4 Mac minis, con procesador Intel Core Duo 1.83GHz, disco rígido de 2.5" a 5400 rpm y 4GB de RAM (667MHz DDR2)
    - 2 Macbooks, con procesador Intel Core Duo 1.83GHz, disco rígido de 2.5" a 5400 rpm y 2GB de RAM (667MHz DDR2)
- Software (configuración idéntica en todas las máquinas)
    - Sistema Operativo: Mac OS X Tiger 10.4.11
    - IDE de simulación: Breve 2.4 (Universal)
    - Sistema de notificación: Growl (mensajes visuales de progreso de simulaciones, por síntesis de voz y por email)

Se realizaron experimentos por lotes e interactivos. Los primeros se realizaron cargando archivos por lotes con conjuntos de configuraciones en formato XML generados por la interfaz visual de configuración de BugSimEvo *(sección Configuración)* y ejecutados también a través de ésta *(sección Batch).* Los experimentos interactivos se realizaron cargando archivos XML conteniendo genomas completos de individuos evolucionados en simulaciones anteriores *(sección Individuo).*

9.2 Simulación Inicial: Parámetros Evolutivos

Uno de los desafíos iniciales en el uso de cualquier variante de algoritmo genético es determinar nivel de mutación y de entrecruzamiento adecuados. En la literatura sobre AG existen recomendaciones generales [Mitchel1998][Goldberg1989] pero su determinación continúa siendo mayormente un proceso intuitivo dado que combinaciones de valores que funcionan para ciertos problemas no necesariamente funcionan bien en otros.

Con el fin de establecer de forma experimental valores adecuados para los niveles de mutación y de entrecruzamiento se procedió a diseñar una serie de simulaciones que permitieran determinar la influencia de dichos factores.

9.2.1 Entrecruzamiento vs. Mutación Uniforme

Se generó una matriz de combinaciones con cuatro niveles de mutación (probabilidades de 0.0 a 0.2) y cinco niveles de entrecruzamiento (0 a 0.8). Por cada combinación se realizaron 18 repeticiones, con el fin de poder determinar posteriormente la influencia de cada factor por separado y cualquier interacción entre estos. El operador de mutación seleccionado fue la mutación con distribución uniforme o rectangular. En la tabla 9.1 se presenta un resumen de la configuración completa del experimento.

Resumen de la simulación	
Título	Parámetros Evolutivos – Entrecruzamiento vs. Mutación Uniforme
Ambiente	Plano sin obstáculos
Parámetros	
Nivel de entrecruzamiento	0% - 20% - 40% - 60% - 80%
Nivel de mutación	0% - 2% - 10% - 20%
Generaciones por simulación	20
Población	20 individuos
Tiempo asignado por individuo	40 segundos simulados
Individuo	
Diseño	6 extremidades, cuerpo articulado
Movimiento	Variante A
Simulaciones Completadas	360 (18 para cada combinación de factores)
Total de Individuos Simulados	144000
Tiempo de Ejecución Total (simulado)	1600 horas (66 días, 16 horas)
Tiempo de Ejecución Total (CPU)	110 horas (4 días, 14 horas)

Tabla 9.1: *Resumen de la simulación "Parámetros Evolutivos – Entrecruzamiento vs. Mutación Uniforme."*

Se procedió a realizar un análisis estadístico de los valores de fitness obtenido, tomando en cuenta la diferencia de fitness entre el mejor individuo de la primer generación y el mejor individuo de la última generación, para reducir la influencia de los factores estocásticos involucrados en la generación de la primer generación.

Se realizó un análisis de varianza completamente aleatorizado para identificar la influencia de los factores y de su interacción, en caso de existir.

Se obtuvo como resultado que la mutación es el factor con mayor influencia en el fitness *($p = 2.37x10^{31}$)*, siendo su influencia altamente significativa. También resultaron significativos el entrecruzamiento *($p = 0.007$)* y la interacción entre mutación y entrecruzamiento *($p = 0.005$)*.

Graficando los resultados se puede observar una distribución unimodal con un máximo en la zona alrededor de la combinación de 40% de entrecruzamiento y 10% de mutación, obteniéndose un aumento de fitness promedio de 60 unidades. Los valores mínimos se concentran en las áreas con menor nivel de mutación, particularmente 0% en combinación con valores extremos (altos y bajos) de entrecruzamiento. *(figs. 9.1 y 9.2)*

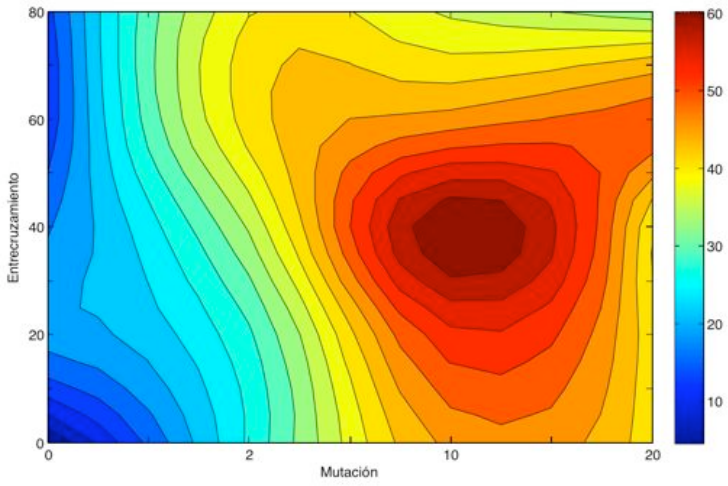


Figura 9.1: *Entrecruzamiento vs. Mutación Uniforme. Puede observarse una distribución unimodal con un valor máximo en la zona alrededor de la combinación de 40% de entrecruzamiento y 10% de mutación.*

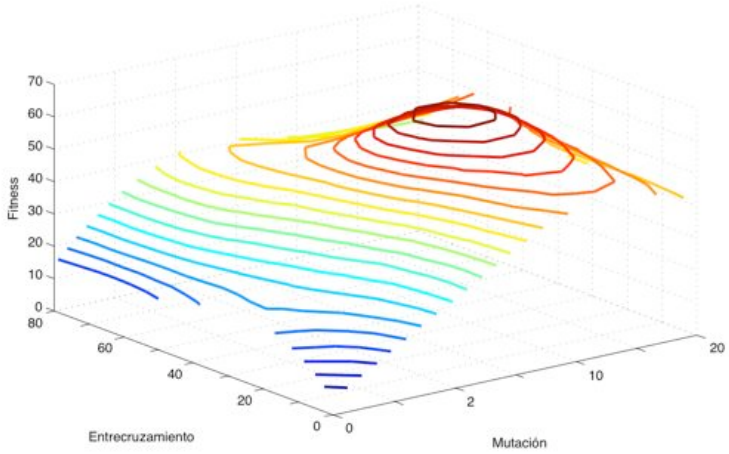


Figura 9.1: *Entrecruzamiento vs. Mutación Uniforme. Vista 3D.*

9.2.2 Entrecruzamiento vs. Mutación Gaussiana

La misma experiencia se repitió utilizando otra variante del operador de mutación: mutación con distribución Gaussiana o Normal. *(tabla 9.2)*

La mutación Gaussiana es una opción comúnmente utilizada en EE, PE y AG de valores reales, por lo cual se implementó como opción de mutación en BugSimEvo.

La configuración del experimento fue similar al experimento anterior, incluyendo las mismas combinaciones de factores, mismo tamaño de población y numero de generaciones, realizándose la misma cantidad de repeticiones por combinación de factores. La configuración completa puede observarse en la tabla 9.2.

Resumen de la simulación	
Título	Parámetros Evolutivos – Entrecruzamiento vs. Mutación Gaussiana
Ambiente	Plano sin obstáculos
Parámetros	
Nivel de entrecruzamiento	0% - 20% - 40% - 60% - 80%
Nivel de mutación	0% - 2% - 10% - 20%
Generaciones por simulación	20
Población	20 individuos
Tiempo asignado por individuo	40 segundos simulados
Individuo	
Diseño	6 extremidades, cuerpo articulado
Movimiento	Variante A
Simulaciones Completadas	360 (18 para cada combinación de factores)
Total de Individuos Simulados	144000
Tiempo de Ejecución Total (simulado)	1600 horas (66 días, 16 horas)
Tiempo de Ejecución Total (CPU)	110 horas (4 días, 14 horas)

Tabla 9.2: *Resumen de la simulación "Parámetros Evolutivos – Entrecruzamiento vs. Mutación Gaussiana."*

Se realizó nuevamente un análisis de varianza completamente aleatorizado para identificar la influencia de los factores y de su interacción, en caso de existir.

Se obtuvo como resultado que la mutación es el factor con mayor influencia en el fitness $(p = 1.45x10^{36})$, seguido por la interacción de entrecruzamiento y mutación $(p = 0.002)$. También resultó significativa, aunque en menor medida la influencia exclusiva del factor de entrecruzamiento $(p = 0.038)$.

Graficando los resultados se puede observar una distribución multimodal, con un máximos en las zona alrededor de la combinación de 40% de entrecruzamiento y 10% de mutación, 0% de entrecruzamiento y 20% de

mutación y 60% de entrecruzamiento y 20% de mutación, obteniéndose un aumento de fitness promedio de poco más de 50 unidades, consistentemente menor a lo obtenido por la mutación uniforme. Los valores mínimos se concentran en las áreas con menor nivel de mutación, particularmente 0% en combinación con valores bajos de entrecruzamiento. *(figs. 9.3 y 9.4)*

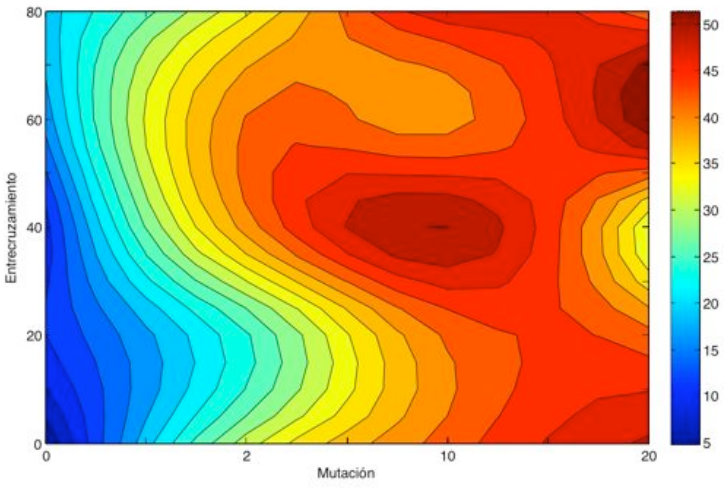


Figura 9.3: *Entrecruzamiento vs. Mutación Gaussiana. Puede observarse una distribución multimodal, con valores máximo en la zona alrededor de la combinación de 40% de entrecruzamiento y 10% de mutación, 0 y 20%, y en 60 y 20%.*

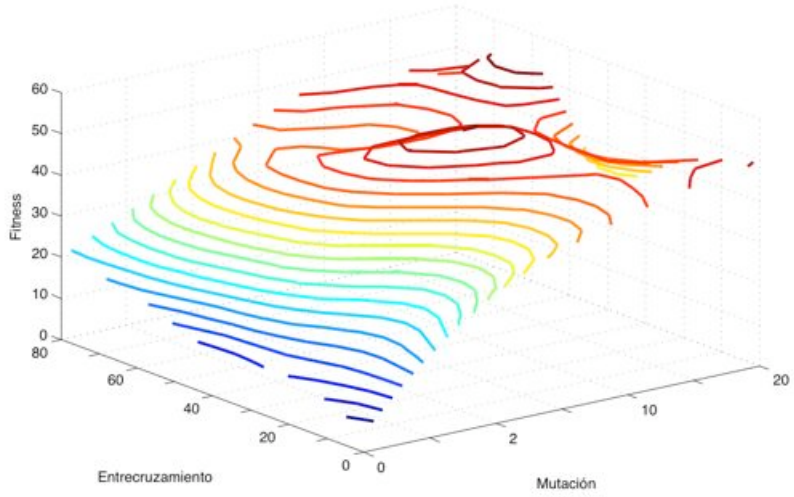


Figura 9.4: *Entrecruzamiento vs. Mutación Gaussiana. Vista 3D.*

9.3 Sistemas de Control y cantidad de patas

Basado en características del movimiento de insectos y animales, BugSimEvo permite evolucionar patrones de locomoción con distintos grados de flexibilidad. En este experimento se evaluaron las distintas variantes de mapeos de genes para locomoción que permiten codificar el grado de independencia y sincronización de los diferentes movimientos (descriptos en detalle en el Capítulo 6, sección 6.2.2).

Se generó una matriz con dos variantes de morfología (6 y 10 patas) y cuatro variantes de movimiento (variantes A - B - C y D, en orden creciente en función de su nivel de sincronización), para evaluar además la influencia de la complejidad de los individuos en la eficiencia y adaptabilidad de cada sistema de control.

Los niveles de mutación y entrecruzamiento se establecieron de acuerdo a los resultados experimentales previos en 10 y 40%, respectivamente. La configuración completa de parámetros de la simulación puede observarse en la tabla 9.3.

Resumen de la simulación	
Título	Sistemas de Control y cantidad de patas
Ambiente	Plano sin obstáculos
Parámetros	
Nivel de entrecruzamiento	40%
Nivel de mutación	10%
Generaciones por simulación	20
Población	20 individuos
Tiempo asignado por individuo	20 segundos simulados
Individuo	
Diseño	6 y 10 extremidades, cuerpo articulado
Movimiento	Variantes A - B - C y D
Simulaciones Completadas	96 (12 para cada combinación de factores)
Total de Individuos Simulados	38400
Tiempo de Ejecución Total (simulado)	213 horas (8 días, 21 horas)
Tiempo de Ejecución Total (CPU)	30 horas (1 día, 6 horas)

Tabla 9.3: *Resumen de la simulación "Sistemas de Control y cantidad de patas"*

Analizando en primer lugar los resultados obtenidos para las distintas variantes de movimientos en individuos de seis patas, se pudo observar que a mayor nivel de sincronización de movimientos, mayor es el fitness (distancia recorrida por el robot), no sólo luego de 20 generaciones sino durante todo el proceso evolutivo. *(fig. 9.5)*

Existe una gran diferencia entre la variante A, donde cada componente de cada pata tiene su propio gen determinante, cuyo valor evoluciona de forma independiente, y el resto de las variantes.

La variante B, en la cual dos genes codifican el movimiento de una pata modelo, replicando luego este movimiento en el resto, (cada pata con una fase independiente), mostró un resultado mejor – intermedio con respecto a las demás variantes.

Los mejores resultados fueron obtenidos con las variantes C y D, con distancias recorridas muy similares, particularmente luego de 8 generaciones. En la variante C, cada segmento del cuerpo tiene una secuencia de movimientos de patas controlada por 3 genes, siendo el resto de los segmentos réplicas de este, cada uno con una fase independiente. La variante D es similar, pero en lugar de fases independiente entre segmentos hay una única fase que se define en forma relativa entre un segmento y el siguiente, favoreciendo la generación de patrones de movimientos en forma ordenada a lo largo del cuerpo.

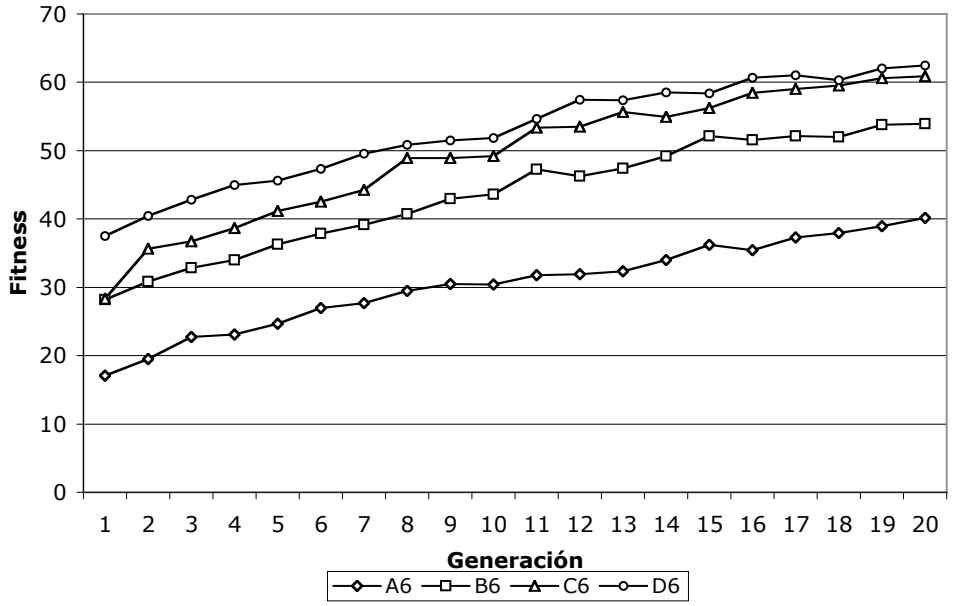


Figura 9.5: *Evolución de fitness (distancia recorrida) a lo largo de 20 generaciones para robots de 6 patas utilizando distintas variantes de mapeos de movimientos. Se puede observar una relación directa entre los niveles de organización y sincronización de movimientos y el fitness obtenido en cada generación. Cada punto mostrado corresponde al valor de fitness máximo en una generación y variante de movimiento particular, promediado a partir de 12 simulaciones.*

Realizando la misma comparación en individuos más complejos, de diez patas, se pudo observar una clara diferencia entre todas las variantes de locomoción. *(fig. 9.6)*

En particular, se puede señalar que el nivel más bajo de organización y sincronización, la variante A, alcanzó al final de 20 generaciones un fitness 12% menor que cuando la misma variante se aplicó a individuos de seis patas. La variante B mantuvo un nivel similar, sólo levemente (6%) mayor, indicando que no fue afectada significativamente por el aumento de seis a diez patas. *(fig. 9.7a y 9.7b)*

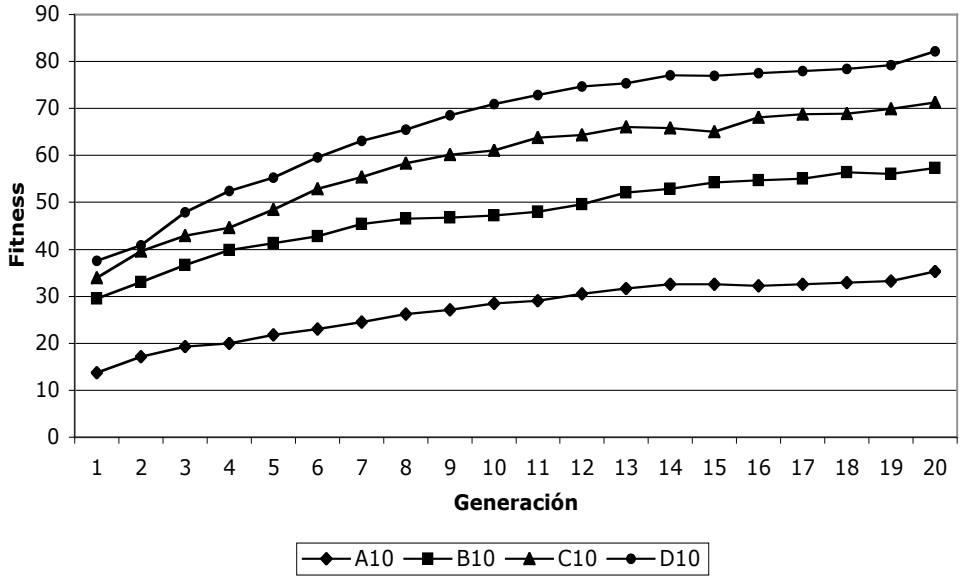


Figura 9.6: *Evolución de fitness (distancia recorrida) a lo largo de 20 generaciones para robots de 10 patas utilizando distintas variantes de mapeos de movimientos. Se puede observar una diferencia aún más marcada entre los distintos mapeos, indicando que al aumentar la complejidad estructural del robot aumenta también la importancia de una organización y sincronización adecuada de movimientos.*

Por otro lado, las variantes C y D mostraron una mejora importante (17% y 31%, respectivamente) indicando que el nivel de organización y sincronización implícitos en sus mapeos de genes tiene un efecto positivo en la capacidad de adaptación de individuos con estructuras corporales complejas. Además, el incremento de seis a diez patas mostró una mayor diferencia de performance entre las variantes C y D, destacándose la ventaja de la sincronización entre segmentos presente en ésta última al incrementarse el número de patas. *(figs. 9.7c y 9.7d)*

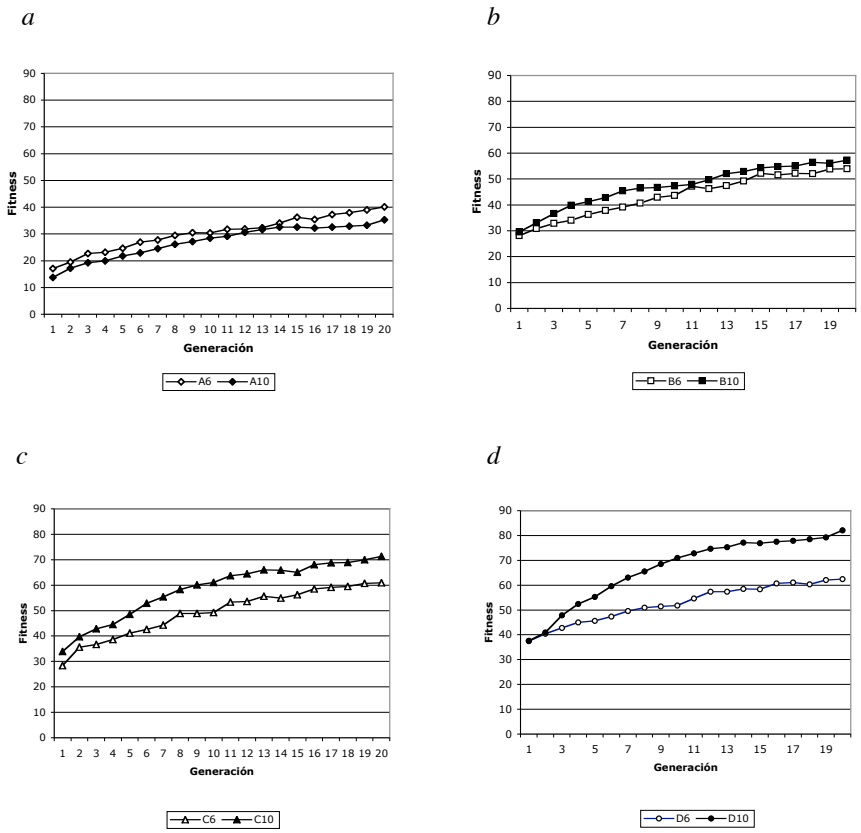


Figura 9.7: *Comparación de performance obtenida por individuos con 6 y 10 patas con distintas variantes de movimientos. a) Variante A: el aumento en la cantidad de patas disminuyó levemente la distancia recorrida b) Variante B: la cantidad de patas no influyó significativamente, mostrando sólo un leve aumento. c) Variante C: mejora significativa al aumentar la cantidad de patas. d) Variante D: mejora muy significativa al aumentar la cantidad de patas, mostrando además una mayor velocidad de aumento de la aptitud para la locomoción durante las primeras generaciones*

9.4 Obstáculos – Deslizamiento por debajo

El objetivo de este experimento fue evaluar que clases de adaptaciones morfológicas evolucionan bajo ambientes particulares. En particular, se estudió como evolucionan las proporciones de los componentes proximal y distal de las patas dado un obstáculo ubicado por sobre el robot.

Se diseñó una ambiente especial, con un terreno plano, una serie de obstáculos posicionados a alturas progresivamente menores y un objetivo, ubicado detrás de los obstáculos. *(fig. 9.8)*

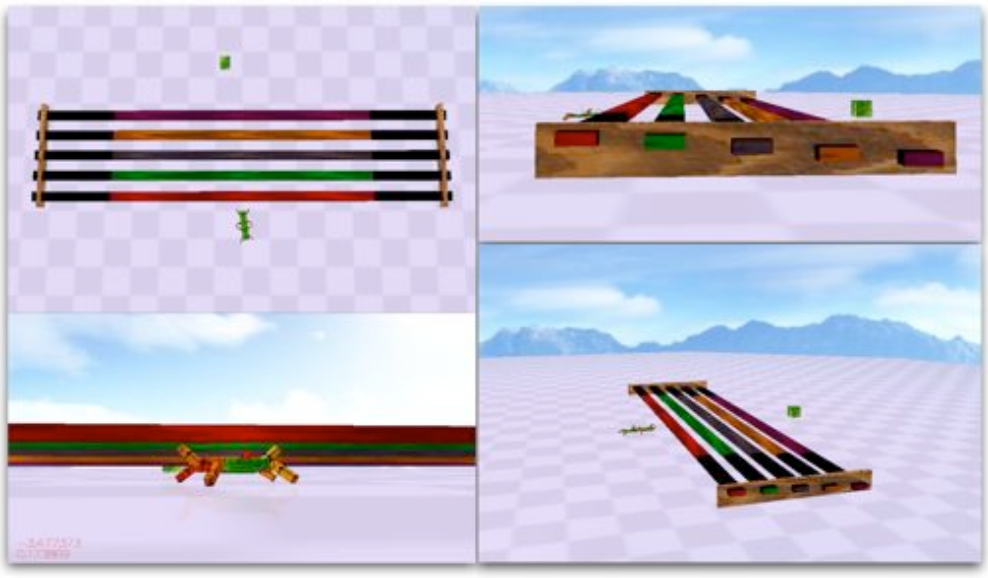


Figura 9.8: *Ambiente con obstáculos posicionados a alturas progresivamente menores, ubicados entre el robot y su objetivo. La función de fitness utilizada es la inversa de la distancia a la que se encuentre el robot de dicho objetivo*[1] *luego de un tiempo predeterminado de simulación.*

Para esta simulación se utilizaron los valores de mutación y entrecruzamiento previamente determinados de forma experimental, 10% y 40%, respectivamente. Como mapeo de genes para locomoción se seleccionó la variante D, con el agregado de un sensor que permite modular el movimiento de las patas de acuerdo a la ubicación (ángulo relativo) del objetivo. La función de fitness se definió como la inversa de la distancia restante al objetivo una vez transcurrido un tiempo predeterminado. Dicho tiempo de simulación por individuo se extendió a 60 segundos, dada la dificultad adicional generada por los obstáculos.

[1] Nota: la ubicación del objetivo en la figura es sólo para propósitos ilustrativos, en las simulaciones realizadas se ubicó a una mayor distancia.

Resumen de la simulación	
Título	Obstáculos – Deslizamiento por debajo
Ambiente	Plano con obstáculos (alturas descendientes)
Parámetros	
Nivel de entrecruzamiento	40%
Nivel de mutación	10%
Generaciones por simulación	20
Población	20 individuos
Tiempo asignado por individuo	60 segundos simulados
Individuo	
Diseño	8 extremidades, cuerpo articulado
Movimiento	Variante D + modulación por sensor
Simulaciones Completadas	18
Total de Individuos Simulados	7200
Tiempo de Ejecución Total (simulado)	120 horas (5 días)
Tiempo de Ejecución Total (CPU)	12 horas

Tabla 9.4: *Resumen de la simulación "Obstáculos – Deslizamiento por debajo"*

En las figuras 9.9 y 9.10 se muestra una corrida típica de una simulación completa en este ambiente. Se puede observar que en las primeras generaciones los individuos tienen problemas en superar los últimos obstáculos, que se encuentran a menor altura. En unas pocas generaciones logran superar un obstáculo adicional, pero luego transcurren diez generaciones sucesivas hasta que finalmente logran atravesarlo y comenzar a acercarse progresivamente al objetivo. Es de destacar también que la simulación con obstáculos produce medidas de fitness variable para un mismo individuo en un nivel mayor que un ambiente plano sin obstáculos. Esto se debe a que al haber un mayor grado de interacción con el ambiente (colisión con obstáculos, fricción, acumulación de fuerzas) se incrementan factores estocásticos propios de la naturaleza necesariamente limitada del entorno simulado.

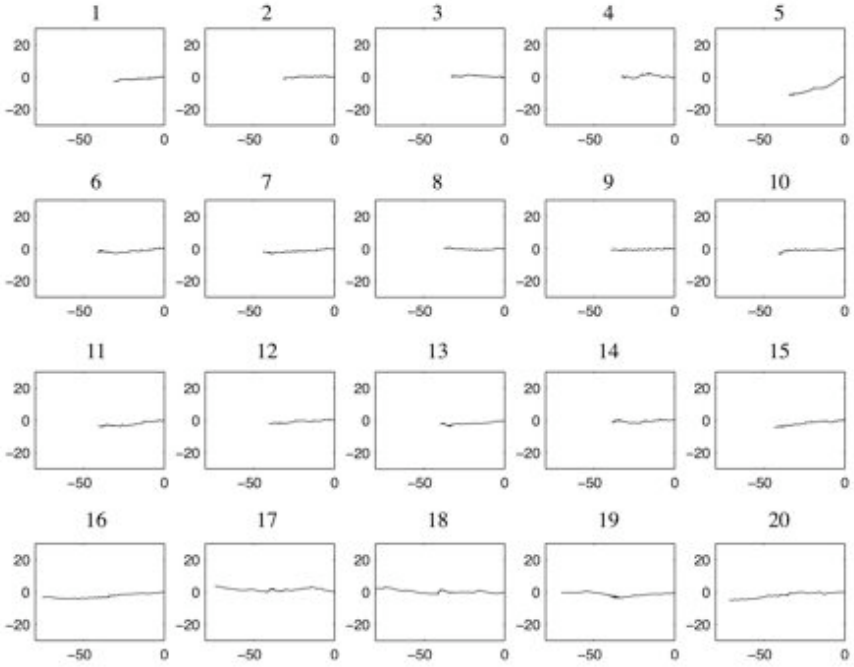


Figura 9.9: *Ejemplo de recorridos reales de los mejores individuos de cada generación en una simulación típica completa (vista superior). En las primeras cinco generaciones los mejores individuos sólo logran superar los tres obstáculos iniciales, quedando detenidos por dos obstáculos de menor altura. En la generación seis logran superar un obstáculo adicional pero quedan detenidos por el siguiente. Finalmente, a partir de la generación 16 logran superar el último obstáculo, avanzando libremente hacia el objetivo.*

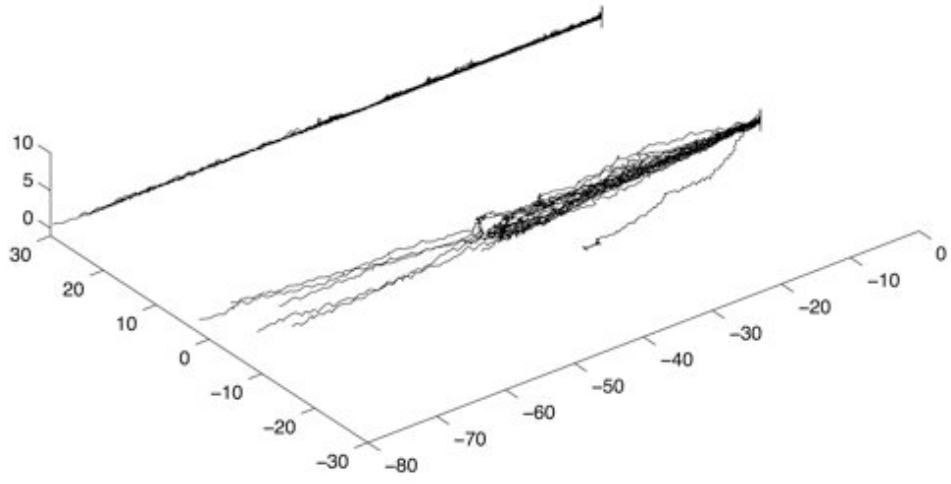


Figura 9.10: *Recorridos de los mejores individuos de cada generación – vista 3D. Puede observarse que durante todo el recorrido la altura se mantiene con pocas variaciones y en un nivel muy bajo. También se puede distinguir una gran concentración de recorridos que no superan la posición del último obstáculo – sólo los últimos cinco recorridos sobrepasan dicho límite.*

El análisis de los resultados obtenidos mostró tendencias claras en la evolución en la longitud de los segmentos proximales y distales de las patas de los robots simulados. Conforme se suceden las generaciones, los individuos se van enfrentando a obstáculos ubicados a alturas cada vez menores. Esto ejerce una presión selectiva que favorece a aquellos individuos de menor altura, es decir, aquellos que tengan patas con un segmento distal pequeño[2]. Por otro lado, los segmentos proximales tienden a aumentar su longitud, favoreciendo un desplazamiento más rápido y compensando en gran medida la reducción de velocidad causada por la necesidad de tener segmentos distales cortos. *(fig 9.11)*

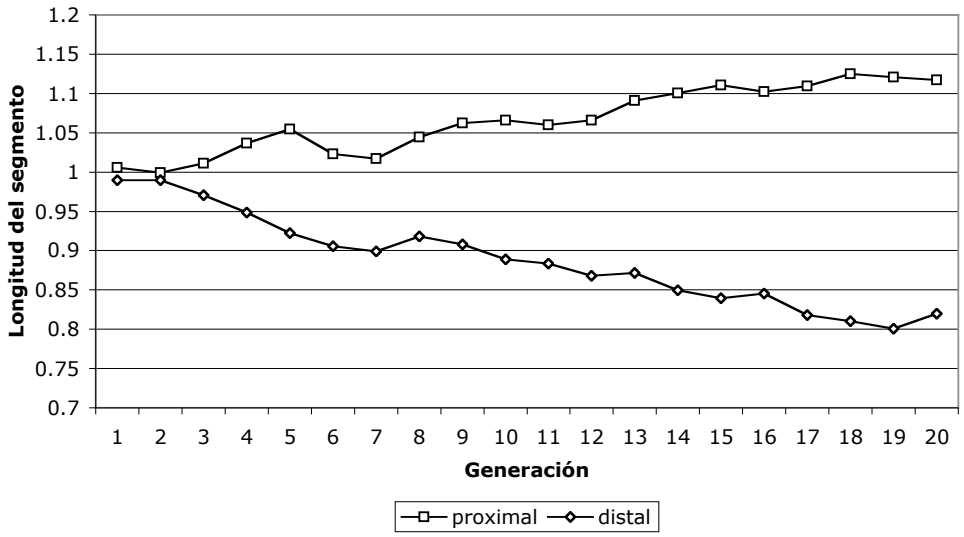


Figura 9.11: *Evolución del tamaño de los segmentos proximal y distal de las patas con el transcurso de las generaciones. El ambiente con obstáculos ubicados a alturas progresivamente decrecientes favorece a aquellos individuos que poseen patas con segmentos proximales largos, que permiten una locomoción rápida, y segmentos distales cortos, que reducen la altura del robot permitiéndole pasar más fácilmente por debajo de los obstáculos.*

[2] En el Capítulo 6 se puede encontrar un esquema detallado de los segmentos proximal y distal de las patas

9.5 Obstáculos – Escalado

El objetivo de este experimento fue, al igual que el experimento anterior, evaluar que tipo de adaptaciones morfológicas evolucionan bajo ambientes particulares. En particular, se estudió cómo evolucionan las proporciones de los componentes proximal y distal de las patas dado un obstáculo por sobre el cual el robot debe trepar.

Se diseñó un ambiente especial, con un terreno plano, una serie de obstáculos posicionados a alturas progresivamente mayores y un objetivo, ubicado detrás de los obstáculos. *(fig. 9.12)*

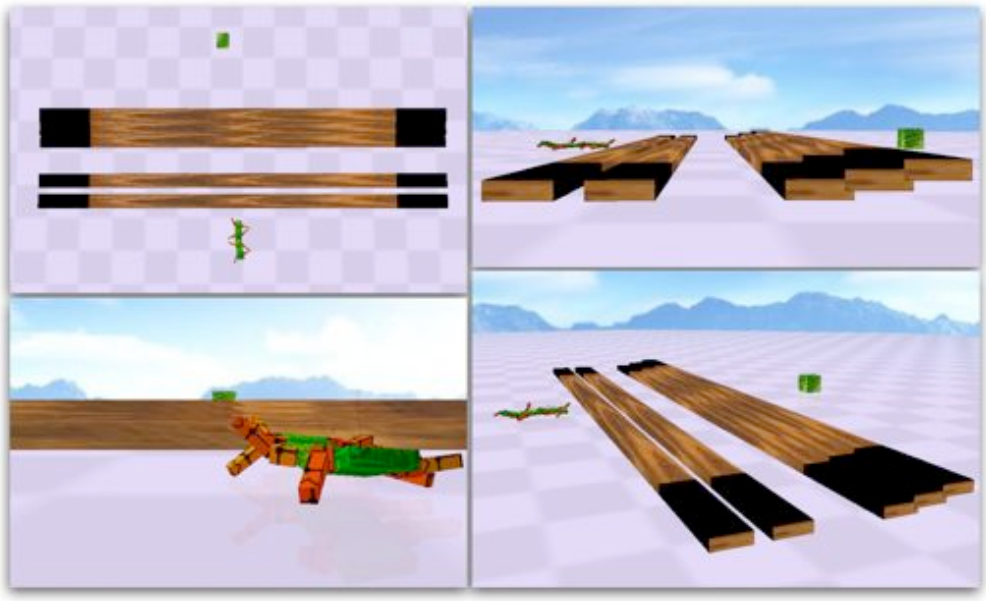


Figura 9.12: *Ambiente con obstáculos posicionados a alturas progresivamente mayores, ubicados entre el robot y su objetivo. La función de fitness utilizada es la inversa de la distancia a la que se encuentre el robot de dicho objetivo luego de un tiempo predeterminado de simulación.*

La configuración general de este experimento es idéntica a la del experimento anterior, pero utilizando obstáculos diferentes. Un resumen se presenta a continuación en la tabla 9.5.

Resumen de la simulación	
Título	Obstáculos – Escalado
Ambiente	Plano con obstáculos (alturas descendientes)
Parámetros	
Nivel de entrecruzamiento	40%
Nivel de mutación	10%
Generaciones por simulación	20
Población	20 individuos
Tiempo asignado por individuo	60 segundos simulados
Individuo	
Diseño	8 extremidades, cuerpo articulado
Movimiento	Variante D + modulación por sensor
Simulaciones Completadas	18
Total de Individuos Simulados	7200
Tiempo de Ejecución Total (simulado)	120 horas (5 días)
Tiempo de Ejecución Total (CPU)	12 horas

Tabla 9.5: *Resumen de la simulación "Obstáculos – Escalado"*

En las figuras 9.13 y 9.14 se muestra una corrida típica de una simulación completa en este ambiente. Se puede observar que en las generaciones iniciales los individuos tienen problemas en superar los primeros obstáculos, de baja altura. En unas pocas generaciones logran superar el primer grupo de obstáculos y a partir de la generación ocho empiezan a trepar el segundo grupo. Finalmente, a partir de la generaciones 13 a 15 logran superarlo y caminar libremente hacia el objetivo. Al igual que en el caso anterior, la simulación con obstáculos produce medidas de fitness variable para un mismo individuo en un nivel mayor que un ambiente plano sin obstáculos.

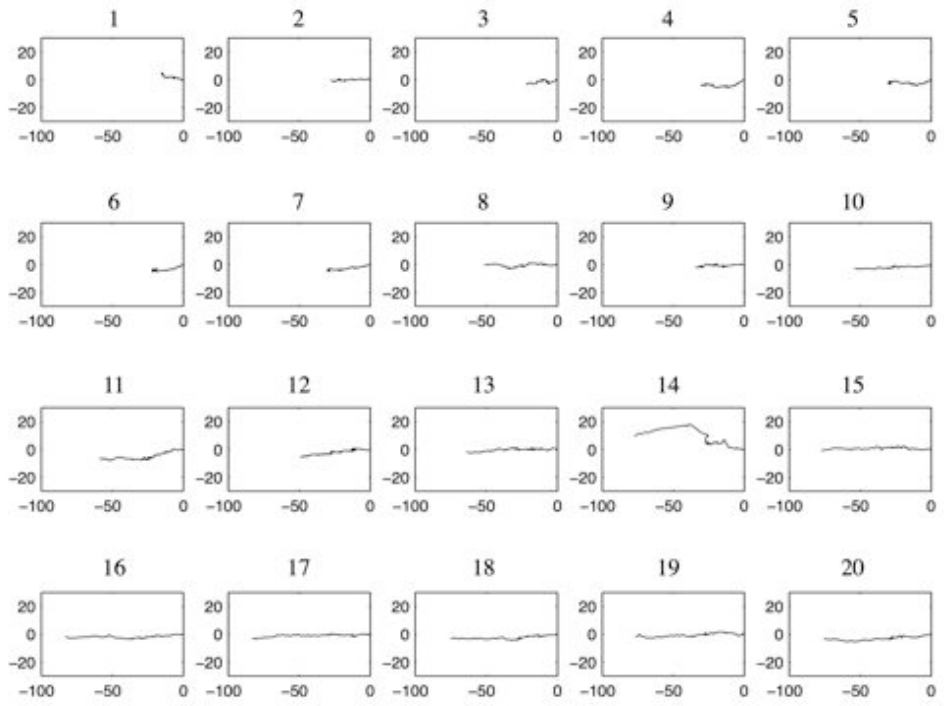


Figura 9.13: *Ejemplo de recorridos reales de los mejores individuos de cada generación en una simulación típica completa (vista superior). En las primeras siete generaciones los mejores individuos sólo logran superar los dos obstáculos iniciales, quedando detenidos por tres obstáculos de mayor altura. En la generación ocho logran trepar pero no superar completamente los últimos obstáculos. Finalmente, a partir de las generaciones 13 a 15 logran superar el último obstáculo, avanzando libremente hacia el objetivo.*

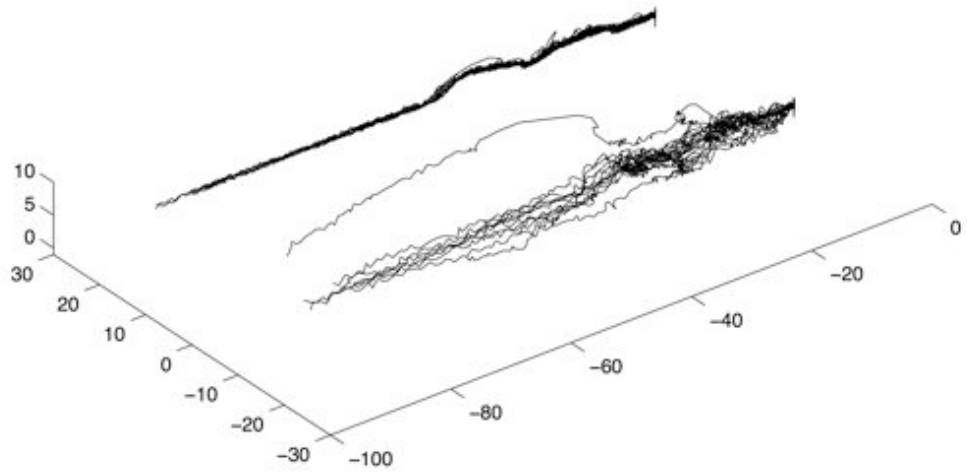


Figura 9.14: *Recorridos de los mejores individuos de cada generación – vista 3D. Pueden observarse dos zonas en las cuales la altura alcanzada aumenta, al trepar los robots por sobre los dos grupos de obstáculos. Se puede distinguir también una gran concentración de recorridos que terminan en estas zonas, debida a la dificultad de los individuos en generaciones iniciales para sobrepasarlas antes de que el tiempo máximo asignado a cada uno de ellos expire.*

El análisis de los resultados obtenidos mostró tendencias claras en la evolución en la longitud de los segmentos distales de las patas de los robots simulados. Conforme se suceden las generaciones, los individuos se van enfrentando a obstáculos de altura creciente. Esto ejerce una presión selectiva que favorece a aquellos individuos que tengan patas con un segmento distal largo, lo que les permite sobrepasar diferencias de nivel mayores.

Por otro lado, los segmentos proximales mantuvieron una longitud relativamente constante durante las primeras 15 generaciones, reduciendo su longitud sólo al final. Una posible explicación para este comportamiento es que durante las primeras 15 generaciones la presión selectiva se basaba casi exclusivamente en la capacidad de superar obstáculos. Una vez que los robots alcanzan un nivel de aptitud tal que superan todos los obstáculos, el factor de la velocidad de desplazamiento sobre terreno plano vuelve a cobrar importancia, originando un cambio en las proporciones proximales y distales óptimas. *(fig 9.13)*

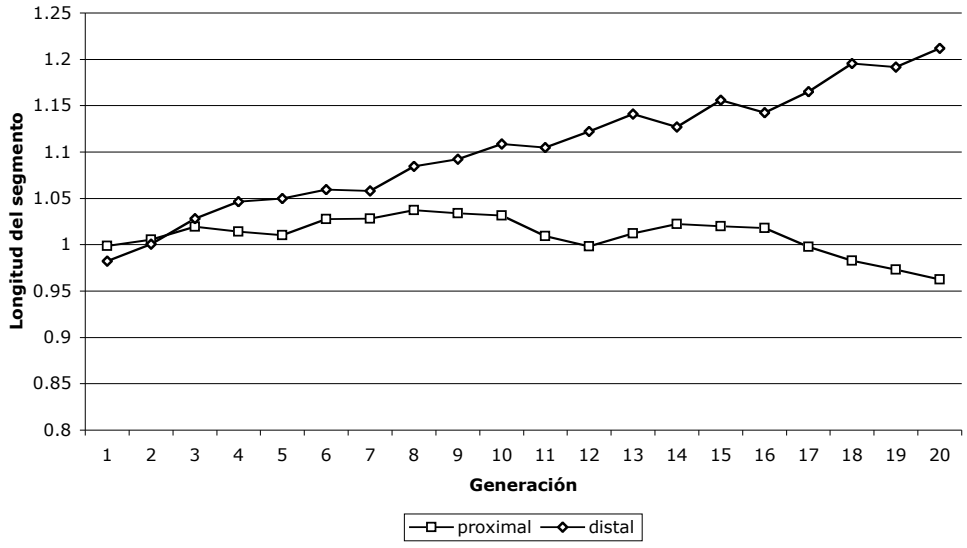


Figura 9.13: *Evolución del tamaño de los segmentos proximal y distal de las patas con el transcurso de las generaciones. El ambiente con obstáculos ubicados a alturas progresivamente mayores favorece a aquellos individuos que poseen patas con segmentos distales largos, que permiten trepar sobre obstáculos de mayor tamaño. Los segmentos proximales mantienen una longitud relativamente constante hasta llegar a las últimas 5 generaciones, donde comienzan a reducir su tamaño.*

Conclusiones

La Robótica Evolutiva mostró ser una forma eficaz para el diseño automatizado de individuos con formas de locomoción y morfologías adaptadas a entornos específicos.

Se aplicaron principios de bio-mimética, resultando éstos en una mejora sustancial de las capacidades de locomoción de los robots simulados. Dicha mejora se manifestó particularmente al incrementar el número de patas en los robots simulados, indicando la importancia del estudio de las formas de locomoción en animales e insectos para obtener comportamientos eficientes.

Se observó la influencia selectiva de diversos ambientes, causando ésta alteraciones en las proporciones de los diversos componentes morfológicos del robot.

Se pudieron establecer experimentalmente parámetros adecuados de mutación y entrecruzamiento, determinando además las ventajas del método de mutación con distribución uniforme por sobre el uso de mutaciones con distribución Gaussiana.

Finalmente, se logró evaluar experimentalmente el entorno de simulación evolutiva desarrollado. Este mostró ser una plataforma práctica para la realización de experiencias en Robótica Evolutiva, permitiendo la programación de tareas de simulación por lotes y el almacenamiento automático de resultados en forma organizada, para su posterior análisis.

Futuras Líneas de Investigación

La plataforma desarrollada permite la realización de experimentos de Robótica Evolutiva. Planes futuros incluyen el agregado de nuevos entornos de simulación, de complejidad creciente, con el objeto de estudiar distintas clases de adaptaciones. También se estudia la posibilidad de agregar sensores adicionales, permitiendo evolucionar comportamientos complejos.

La transferencia al mundo real de estructuras morfológicas y de formas de locomoción obtenidas en el simulador, implementando éstas en un robot físico es otra área de investigación potencial.

También, existen planes de ampliar las capacidades del algoritmo genético utilizado, incluyendo la posibilidad de simular procesos co-evolutivos predador-presa, operadores genéticos alternativos como inversión y entrecruzamiento uniforme y poblaciones distribuidas.

Bibliografía

[Altenberg1995] Altenberg, L., *"The Schema Theorem and Price's Theorem"*, en D. Whitley, M. Vose (editores), Foundations of Genetic Algorithms 3, Morgan Kaufmann, 1995

[Angeline1993] Angeline, P. J., Pollack, J. B., *"Competitive Environments Evolve Better Solutions for Complex Tasks"*, en S. Forrest (editor), Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, 1993

[Antonisse1989] Antonisse, J., *"A New Interpretation of Schema Notation That Overturns the Binary Encoding Constraint"*, en J. D. Schaffer (editor), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, 1989

[Auerbach2009a] Auerbach, J., Bongard, J. C., *"Evolution of Functional Specialization in a Morphologically Homogenous Robot"*, Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, ACM, 2009

[Auerbach2009b] Auerbach, J, Bongard, J. C., *"How Robot Morphology and Training Order Affect the Learning of Multiple Behaviors"*, 2009 IEEE Congress on Evolutionary Computation (IEEE CEC 2009), Trondeim, Noruega, 2009

[Augustsson2002] Augustsson, P., Wolff, K., Nordin, P., *"Creation of a Learning, Flying Robot by Means of Evolution"*, GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan-Kaufmann, 2002

[Bäck1991] Bäck, T., Hoffmeister, F., Schwefel, H. P., *"A Survey of Evolution Strategies"*, Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, 1991

[Bäck1993] Bäck, T., Rudolph, G., Schwefel, H. P., *"Evolutionary Programming and Evolution Strategies: Similarities and Differences"*, Proceedings of the Second Annual Conference on Evolutionary Programming, Evolutionary Programming Society, 1993

[Bäck1996] Bäck, T., *"Evolutionary Algorithms in Theory and Practice"*, Oxford University Press, 1996

[Bäck2000] Bäck, T., Fogel, D.B., Michaelwicz, Z., *"Evolutionary Computation: Basic Algorithms and Operators"*, Taylor & Francis Group, 2000

[Baker1985] Baker, J. E., *"Adaptive Selection Methods for Genetic Algorithms"*, Proceedings of the First International Conference on Genetic Algorithms and Their Applications, Erlbaum, 1985

[Banzhaf1998] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D., *"Genetic Programming - An Introduction: On the Automatic Evolution of Computer Programs and Its Applications"*, Morgan Kaufmann Publishers, 1998

[Bauer1994] Bauer, R. J., Jr., *"Genetic Algorithms and Investment Strategies"*, Wiley, 1994

[Beyer2001] Beyer, H. G., *"The Theory of Evolution Strategies"*, Springer, 2001

[Bongard2000] Bongard, J. C., Paul, C., *"Investigating Morphological Symmetry and Locomotive Efficiency Using Virtual Embodied Evolution"*, From Animals to Animats: The Sixt International Conference on the Simulation of Adaptive Behaviour", MIT Press, 2000

[Bongard2008] Bongard, J., *"Behavior Chaining: Incremental Behavior Integration for Evolutionary Robotics"*, en S. Bullock, J. Noble, R. Watson, M. A. Bedau (editores), Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems, MIT Press, 2008

[Brameier2004] Brameier, M., *"On Linear Genetic Programming"*, Department of Computer Science, University of Dortmund, 2004

[Brooks1986] Brooks, R. A., *"A Robust Layered Control System for a Mobile Robot"*, IEEE Jounal of Robotics and Automation, Vol. 2, Issue 1, IEEE, 1986

[Brooks1991] Brooks, R. A., *"New Approaches to Robotics"*, Science, Vol. 253, 1991

[Busch2002] Busch, J., Ziegler, J., Aue, C., Ross, A. Sawitzki, D., Banzhaf, W., *"Automatic Generation of Control Programs for Walking Robots Using Genetic Programming"*, EuroGP 2002, LNCS 2278, Springer, 2002

[Calabretta1998] Calabretta, R., Nolfi, S., Parisi, D., Wagner, G. P., *"Emergence of Functional Modularity in Robots"*, en R. Pfeifer, B. Blumberg, J. A. Meyer, S. W. Wilson (editores), Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior - From Animals to Animats 5, MIT Press, 1998

[Capi2005] Capi, G., Doya, K., *"Evolutiono of Neural Architectur Fitting Environmental Dynamics"*, Adaptive Behavior, Vol. 13, ISAB, 2005

[Caruana1988] Caruana, R. A., Schaffer, J. D., *"Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms"*, Proceedings of the 5th International Conference on Machine Learning, Morgan Kaufmann, 1988

[Castellani2003] Castellani, F., Franceschini, G., *"Use of Genetic Algorithms as an Innovative Tool for Race Car Design"*, SAE International, 2004

[Cham2002] Cham, J. G., Bailey, S. A., Clark, J. E., Full, R. J., Cutkosky, M. R., *"Fast and Robust: Hexapedal Robots via Shape Deposition Manufacturing"*, The International Journal of Robotics Research, Vol. 21, No. 10-11, Sage Publications, 2002

[Chen1997] Chen, C., Quinn, R. D., Ritzmann, R. E., *"A Crash Avoidance System Based Upon the Cockroach Escape Response Circuit"*, Proceedings of the 1997 IEEE International Conference on Robotics and Automation, 1997

[Davis1991] Davis, L. D., *"Handbook of Genetic Algorithms"*, Van Nostrad Reinhold, 1991

[Dawkins1976] Dawkins, R., *"The Selfish Gene"*, Oxford University Press, 1976

[De Jong1975] De Jong, K. A., *"Analysis of the Behavior of a Class of Genetic Adaptive Systems"*, Technical Report, Computer and Comunication Sciences Department, The University of Michigan, 1975

[De Jong1993] De Jong, K. A., Sarma, J., *"Generation Gaps Revisited"*, en L. D. Whitley (editor), Foundations of Genetic Algorithms 2, Morgan Kaufmann, 1993

[Dréo2006] Dréo, J., Siarry, P., Pétrowski, A., Taillard, E., *"Metaheuristics for Hard Optimization: Methods and Case Studies"*, Springer, 2006

[Dumitrescu2000] Dumitrescu, D., Lazzerini, B., Jain, L. C., Dumitrescu, A., *"Evolutionary Computation"*, CRC Press, 2000

[Eiben2003] Eiben, A. E., *"Introduction To Evolutionary Computing"*, Springer, 2004

[Espenschied1993] Espenshied, K. S., Quinn, R. D., Chiel, H. J., Beer, R. D., *"Leg Coordination Mechanism in the Stick Insect Applied to Hexapod Robot Locomotion"*, Adaptive Behavior, Vol 1, No. 4, MIT, 1993

[Espenschied1996] Espenshied, K. S., Quinn, R. D., Beer, R. D., Chiel, H. J., *"Biologically Based Distributed Control and Local Reflexes Improve Rough Terrain Locomotion in a Hexapod Robot"*, Robotics and Autonomous Systems, Elsevier Science B. V., 1996

[Ferrell1995] Ferrell, C. L., *"A Comparison of Three Insect Inspired Locomotion Controllers"*, Paper, Artificial Intelligence Laboratory, MIT, 1995

[Ficici2000] Ficici, S. G., Watson, R. A., Pollack, J. B., *"Embodied Evolution: A Response to Challenges in Evolutionary Robotics"*, en J. L. Wyatt, J. Demiris (editores), Proceedings of the Eighth European Workshop on Learning Robots: Advances in Robot Learning (EWLR'99), Springer, 2000

[Fogel1966] Fogel, L. J., Owens, A.J., Walsh, M.J., *"Artificial Intelligence Through Simulated Evolution"*, Wiley, 1966

[Fogel1999] Fogel, L. J., *"Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming"*, Wiley, 1999

[Goldberg1989] Goldberg, D. E., *"Genetic Algorithms in Search, Optimization and Machine Learning"*, Addison-Wesley, 1989

[Goldberg1990] Goldberg, D. E., *"Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking"*, Complex Systems, Volume 5, 1990

[Goldberg1991] Goldberg, D. E., Deb, K., *"A Comparative Analysis of Selection Schemes Used in Genetic Algorithms"*, Foundations of Genetic Algorithms, Morgan Kaufmann, 1991

[Greiner2005] Greiner, D., Winter, G., Emperador, J. M., Galván, B., *"Gray Coding in Evolutionary Multicriteria Optimization: Application in Frame Structural Optimum Design"*, Evolutionary Multi-Criterion Optimization, Third International Conference, EMO 2005, Springer, 2005

[Hancock1997] Hancock, P. J. B., *"A Comparison of Selection Mechanisms"*, en T. Bäck, D. B. Fogel, Z. Michalewicz (editores), Handbook of Evolutionary Computation, Taylor & Francis Group, 1997

[Hapke2003] Hapke, M., Komosinski, M., Waclawski, D., *"Application of Evolutionarily Optimized Fuzzy Controllers for Virtual Robots"*, Proceedings of the 7th Joint Conference on Information Sciences, Association for Intelligent Machinery, 2003

[Hillis1990] Hillis, W. D., "*Co-evolving Parasites Improve Simulated Evolution As An Optimization Procedure"*, Physica D, Volume 42, 1990

[Holland1975] Holland, John H., *"Adaptation in Natural and Artificial Systems"*, University of Michigan, 1975 - 2da edición MIT Press., 1992

[Hornby2000a] Hornby, G. S., Takamura, S., Yokono, J., Hanagata, O., Yamamoto, T., Fujita, M., *"Evolving Robust Gaits with AIBO"*, Proceedings of the 2000 IEEE International Conference on Robotics and Automation, IEEE Press, 2000

[Hornby2000b] Hornby, G. S., Takamura, S., J., Hanagata, O., Fujita, M., Pollack, J., *"Evolution of Controllers from a High-Level Simulator to a High DOF Robot"*, Proceedings of the 2000 IEEE International Conference on Robotics and Automation, IEEE Press, 2000

[Hornby2001] Hornby, G. S., Lipson, H, Pollack, J. B., *"Evolution of Generative Design Systems for Modular Physical Robots"*, Proceedings of the 2001 IEEE International Conference on Robotics and Automation, Vol. 4, IEEE Press, 2001

[Hornby2002a] Hornby, G. S., *"Generative Representations for Evolutionary Design Automation"*, PhD. Dissertation, Department of Computer Sciences, Brandeis University, Waltham, MA, 2002

[Hornby2002b] Hornby, G. S., Pollack, J. B., *"Creating High-Level Components with a Generative Representation for Body-Brain Evolution"*, Artificial Life, Volume 8, Issue 3, MIT Press, 2002

[Jacob2001] Jacob, C., *"Illustrating Evolutionary Computation with Mathematica"*, Morgan Kaufmann Publishers, 2001

[Jakobi1997] Jakobi, N., *"Evolutionary Robotics and the Radical Envelope of Noise Hypothesis"*, Journal of Adaptive Behavior, Vol. 6, No, 2, 1993

[Janikow1991] Janikow, C. Z., Michalewicz, Z., *"An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms"*, en R. K. Belew, L. S. Booker (editores), Proceedings of the 4th International Conference on Genetic Algorithms, Morgan Kaufmann, 1991

[Jindrich2002] Jindrich, D. L., Full, R. J., *"Dynamic Stabilization of Rapid Hexapedal Locomotion",* The Journal of Experimental Biology 205, The Company of Biologists Limited, 2002

[Kaiser1997] Kaiser, C. E., Jr., Lamont, G. B., Merkle, L. D., Gates, G. H., Jr., Pachter, R., *"Polypeptide Structure Prediction: Real-valued Versus Binary Hybrid Genetic Algorithms"*, Proceedings of the 1997 ACM Symposium on Applied Computing, 1997

[Kennedy2001] Kennedy, J., Eberhart, R. C., *"Swarm Intelligence"*, Morgan Kaufmann Publishers, 2001

[Klein2002] Klein, J., *"BREVE: a 3D Environment for the Simulation of Decentralized Systems and Artificial Life"*, Complex Systems Group, Department of Physical Resource Theory, Chalmers University of Technology and Göteborg University SE-412 96 Göteborg, Suecia y Cognitive Science Hampshire College, Amherst, MA, 2002

[Knuth1973] Knuth, D. E., *"The Art of Computer Programming. Volume 3: Searching and Sorting"*, Addison-Wesley, 1973

[Kolen2001] Kolen, J. F., Kremer, S. C., *"A Field Guide to Dynamical Recurrent Networks"*, IEEE, 2001

[Koza1989] Koza, J. R., *"Hierachical Genetic Algorithms Operating on Populations of Computer Programs"*, Proceedings of the 11th International Joint Conference on Artificial Intelligence, Morgan Kaufman Publishers, 1989

[Koza1990] Koza, J. R., *"Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems"*, Computer Science Department, Stanford University, 1990

[Koza1991] Koza, J. R., *"Evolution and Co-Evolution of Computer Programs to Control Independently-Acting Agents"*, Computer Science Department, Stanford University, 1991

[Koza1992] Koza, J. R., *"Genetic Programming: On the Programming of Computers by Means of Natural Selection"*, MIT Press, 1992

[Koza1993] Koza, J. R., *"Evolution of Subsumption Using Genetic Programming"*, Proceedings of the First European Conference on Artificial Life, MIT Press, 1993

[Koza1994] Koza, J. R., *"Genetic Programming II: Automatic Discover´y of Reusable Programs"*, MIT Press, 1994

[Lipson2000] Lipson, H., Pollack, J. B, *"Automatic Design and Manufacture of Robotic Lifeforms"*, Nature, Vol. 406, 2000

[Lipson2006] Lipshon, H., Bongard, J., Zykov, V., Malone, E., *"Evolutionary Robotics for Legged Machines: From Simulation to Physical Reality"*, Proceedings of the 9th International Conference on Intelligent Autonomous Systems, University of Tokyo, Japan, 2006

[Lohn2005] Lohn, J. D., Hornby, G. S., Linden, D. S., *"An Evolved Antena for Deployment on NASA's Space Technology 5 Mission"*, Genetic Programming Theory and Practice II, Springer, 2005

[Luke1998] Luke, S., Spector, L., *"A Revised Comparison of Crossover and Mutation in Genetic Programming"*, Genetic Programming 1998: Proceedings of the Third Anual Conference, University of Wisconsin, Morgan Kaufman, 1998

[Macinnes2004] Macinnes, I., Di Paolo, E., *"Crawling Out of the Simulation: Evolving Real Robot Morphologies Using Cheap Reusable Modules"*, en J. Pollack, M. Bedau, P. Husbands, T. Ikegami, R. Watson (editores), Artificial Life IX: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Life, MIT Press, 2004

[Malone2004] Malone, E., Lipson, H., *"Functional Freeform Fabrication for Physiscal Artificial Life"*, Proceedings of the 9th International Conference on Artificial Life (ALIFE IX), MIT Press, 2004

[McDonnell1995] McDonnell, J. R., Reynolds, R. G., Fogel, D. B., *"Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming"*, MIT Press, 1995

[Mehta1999] Mehta, K., Bhattacharyya, S., *"Combining Rules Learnt Using Genetic Algorithms for Financial Forecasting"*, Proceedings of the 1999 Congress on Evolutionary Computation, Volume 2, IEEE, 1999

[Miglino1996] Miglino, O., Lund, H.H., Nolfi, S. *"Evolving Mobile Robots in Simulated and Real Environments"*, Artificial Life #2, 1996

[Miller1995] Miller, B. L., Goldber, D. E., *"Genetic Algorithms, Tournament Selection and the Effects of Noise"*, Complex Systems, Vol. 9, 1995

[Mitchel1998] Mitchel, M., *"An Introduction to Genetic Algorithms"*, MIT Press, 1998

[Mondada1995] Mondad, F., Floreano, D., *"Evolution of Neural Control Structures: Some Experiments on Mobile Robots"*, Robotics and Autonomous Systems, Elsevier, 1995

[Montana1989] Montana, D., Davis, L., *"Training Feedforward Neural Networks Using Genetic Algorithms"*, Proceedings of the International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 1989

[Nolfi2000] Nolfi, S., Floreano, D., *"Evolutionary Robotics: The Biology, Intelligence and Technology of Self-Organizing Machines"*, MIT Press, 2000

[Paul2001] Paul, C., Bongard, J. C., *"The Road Less Travelled: Morphology in the Optimization of Biped Robot Locomotion"*, Proceedings of the IEEE/RSJ International Conference on Itelligent Robots and Systemas (IROS2001), IEEE Press, 2001

[Poli2008] Poli, R., Langdon, W. B., McPhee, N. F., Koza, J. R., *"A Field Guide to Genetic Programming"*, Lulu Enterprises UK Ltd., 2008

[Radcliffe1992] Radcliffe, N. J., *"Non-Linear Genetic Representations"*, Parallel Problem Solving From Nature 2: Proceedings of the Second Conference on Parallel Problem Solving From Nature, Bruselas, Bélgica, Elsevier Science Publishers, 1992

[Rechenberg1973] Rechemberg, I., *"Evolutionsstrategie–Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution",* Frommann-Holzboog, 1973

[Rosin1995] Rosin, C. D., Belew, R. K., *"Methods for Competitive Co-evolution: Finding Opponents Worth Beating"*, en L. Eshelman (editor), Proceedings of the Sixt International Conference on Genetic Algorithms, Morgan Kaufmann, 1995

[Sarma1997] Sarma, J., De Jong, K., *"Generation Gap Methods"*, en T. Bäck, D. B. Fogel, Z. Michalewicz (editores), Handbook of Evolutionary Computation, Taylor & Francis Group, 1997

[Schwefel1995] Schwefel, H. P., *"Evolution and Optimum Seeking"*, Wiley, 1995

[Seipel2004] Seipel, J. E., Holmes, P. J., Full, R. J., *"Dynamics and Stability of Insect Locomotion: A Hexapedal Model for Horizontal Plane Motions",* Biological Cybernetics 91, Springer, 2004

[Sims1994a] Sims, K., *"Evolving Virtual Creatures"*, Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94, ACM, 1994

[Sims1994b] Sims, K., *"Evolving 3D Morphology and Behavior by Competition"*, Artificial Life, Vol. 1, Issue 4, MIT Press, 1994

[Sprinkhuizen-Kuyper2000] Sprinkhuizen-Kuyper, I. G., Kortmann, R., Postma, E. O., *"Fitness Functions for Evolving Box-Pushing Behavior"*, Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference, 2000

[Syswerda1991] Syswerda, G., *"A Study of Reproduction in Generational and Steady-State Genetic Algorithms"*, Foundations of Genetic Algorithms, Morgan Kaufmann, 1991

[Tanese1989] Tanese, R., *"Distributed Genetic Algorithms for Function Optimization"*, Ph.D. Thesis, Departmente of Electrical Engineering and Computer Science, University of Michigan, 1989

[Tate1993] Tate, D. M., Smith, A. E., *"Expected Allele Coverage and the Role of Mutation in Genetic Algorithms"*, Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, 1993

[Thompson1998] Thompson, A., *"Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution"*, Springer-Verlang, 1998

[van Waveren2001] van Waveren, J. M. P., *"The Quake III Arena Bot"*, University of Technology Delft, Faculty ITS, 2001

[Wahde2005] Wahde, M., *"Evolutionary Robotics - The Use of Artificial Evolution in Robotics"*, Technical Report, Department of Applied Mechanics, Chalmers University of Technology, Göteborg, Suecia, 2005

[Watson1999] Watson, R. A., Ficici, S. G., Pollack, J. B., *"Embodied Evolution: Embodying an Evolutionary Algorithm in a Population of Robots"*, en P.J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, A. Zalzala (editores), Proceedings of the 1999 Congress on Evolutionary Computation (CEC99), IEEE Press, 1999

[Watson2002] Watson, R. A., Ficici, S. G., Pollack, J. B., *"Embodied Evolution: Distributing an Evolutionary Algorithm in a Population of Robots"*, Robotics and Autonomous Systems, Volume 39, Issue 1, Elsevier Science B.V., 2002

[Whitley1989] Whitley, D., *"The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best"*, Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, 1989

[Whitley1997] Withley, D., Rana, S., Heckendorn, R., *"Representation Issues in Neighborhood Search and Evolutionary Algorithms",* Genetic Algorithms and Evolution Strategies in Engineering and Computer Science, Wiley, 1997

[Whitley1999] Whitley, D., *"A Free Lunch Proof for Gray versus Binary Encodings"*, Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 1, Morgan Kaufmann, 1999

[Wolff2002] Wolff, K., Nordin, P., *"Evolution of Efficient Gait With an Autonomous Biped Robot Using Visual Feedback"*, Proceedings of the Mechatronics Conference, Morgan-Kaufmann, 2002

[Wolpert1995] Wolpert, D. H., Macready, W. G., *"No Free Lunch Theorems for Search"*, Technical Report, Santa Fe Institute, 1995

[Wolpert1997] Wolpert, D. H., Macready, W. G., *"No Free Lunch Theorems for Optimization"*, IEEE Transactions on Evolutionary Computation, 1997

[Wright1991] Wright, A. H., *"Genetic Algorithms for Real Parameter Optimization"*, en G. J. E. Rawlins (editor), Foundations of Genetic Algorithms, Volume 1, Morgan Kaufmann, 1991

[Yao1999] Yao, X., *"Evolutionary Computation: Theory and Applications"*, World Scientific Publishing, 1999

[Zykov2004] Zykov, V., Bongard, J., Lipson, H., *"Evolving Dynamic Gaits on a Physical Robot"*, Proceedings of The 2004 Genetic and Evolutionary Computation Conference, ACM, 2004

Anexo A

1 Breve

Breve es un entorno de simulación de agentes en un mundo continuo en 3D, multiplataforma y de código abierto. Entre sus características más destacables se encuentran:

- lenguaje de programación orientado a objetos para el desarrollo de scripts de control, con una extensa biblioteca de clases orientadas a la simulación
- soporte para simulaciones físicas realistas: detección de colisiones, fricción, articulaciones, gravedad, etc.
- motor de gráficos 3D (OpenGL)
- funciones de comunicación por red
- posibilidad de almacenar a disco simulaciones en progreso y restaurarlas posteriormente, continuando su ejecución
- entorno de desarrollo que facilita la programación, visualización, depuración e interacción con las simulaciones

El sistema fué creado por Jon Klein inicialmente como un proyecto de Tesis en el Hampshire College y posteriormente con desarrollos adicionales para su Tesis de Maestría en la Chalmers University [Klein2002, Klein2004]. El objetivo era crear un framework que permitiera el desarrollo de simulaciones realistas, en un espacio tridimensional continuo y con tiempo continuo. Muchas simulaciones de vida artificial están diseñadas en entornos de simulación discretos, ya sea espacial o temporalmente.

El sistema presenta dos variantes, ambas disponibles para plataformas Mac OS X, Linux y Windows:

- **Breve IDE (Integrated Development Environment):** Incluye un editor de scripts, una ventana de visualización la cual permite

interactuar con la simulación en tiempo real, un log y una ventana de inspección que muestra información sobre la simulación y/o sobre los objetos seleccionados.

- **Breve CLI (Command Line Interface):** Es una versión de línea de comandos, la cual acepta varios parámetros, entre ellos el script a ejecutar. Permite realizar simulaciones con o sin gráficos. En caso de habilitarse la generación de gráficos, al comenzar la ejecución se abre una ventana gráfica simple, la cual muestra la simulación en tiempo real y permite la interacción con la misma por medio del mouse y el teclado (no presenta controles visuales como botones o menús).

2 Breve IDE

Esta interfaz permite el desarrollo, ejecución y depuración de simulaciones de forma visual e interactiva.

Los principales componentes del IDE son:

- *Ventana gráfica interactiva*, que permite visualizar y controlar la simulación
- *Editor de código fuente*, el cual soporta el coloreado de sintaxis
- *Log interactivo*, utilizado para mostrar datos alfanumericos y para ejecutar sentencias de programa individuales
- *Inspector interactivo*, el cual permite observar y editar las propiedades de los objetos presentes en la simulación

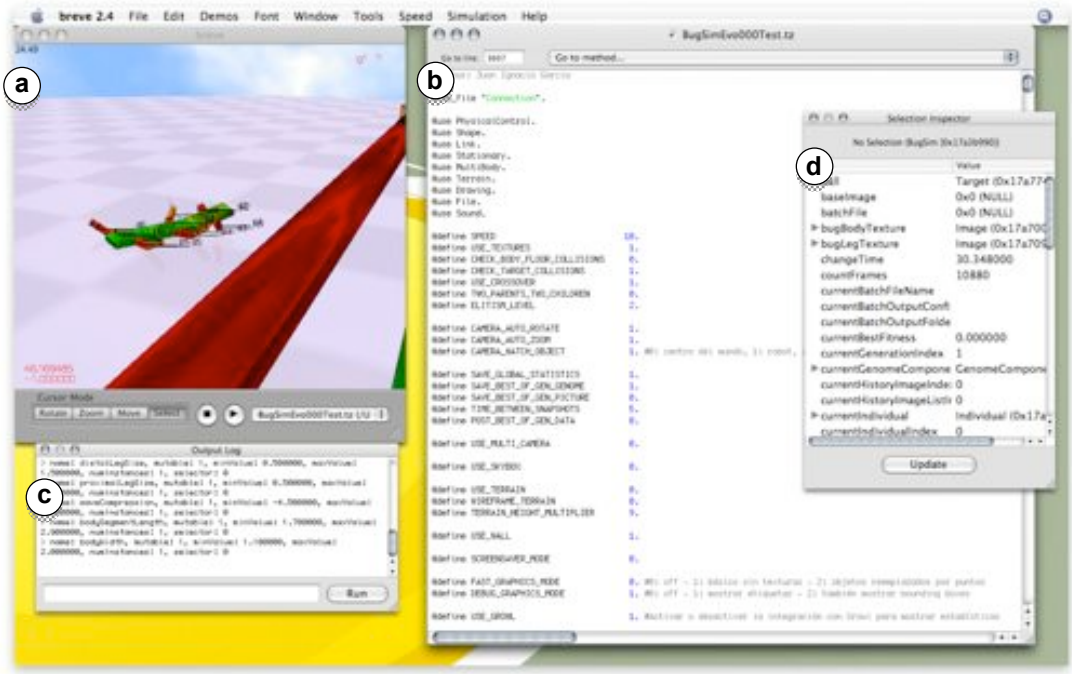


Figura 1: *Breve IDE, mostrando la ventana gráfica interactiva (a), el editor de código fuente (b), la ventana de log y ejecución de comandos (c) y el inspector interactivo (d).*

3 Breve CLI

La versión de línea de comandos de Breve permite ejecutar simulaciones directamente desde la consola, tanto con como sin gráficos. Para invocarla, se debe utilizar el siguiente formato:

./breve [opciones] archivo_simulación

Las opciones principales son:

- x Ejecutar la simulación sin abrir una ventana gráfica
- f Ejecutar la simulación con gráficos en modo pantalla completa
- sX,Y Establece el ancho y alto de la ventana gráfica en pixels

- a<archivo_xml> Permite continuar la ejecución de una simulación suspendida identificada por archivo_simulación, restaurando su estado al almacenado en archivo_xml.

4 Lenguaje

Las simulaciones de Breve se programan en un lenguaje denominado *"Steve"*, el cual fue diseñado específicamente para la simulación de agentes que interactúan en un ambiente simulado. Posee ademas una extensa colección de clases orientadas a la simulación de ambientes continuos y a la interacción de agentes con estos.

El lenguaje presenta características típicas de lenguajes orientados a objetos como C++ o Java, mientras que la sintaxis presenta cierta similitud con Objective-C, en particular su uso de parámetros con nombre y forma de referencia a métodos.

A continuación se describen sus principales elementos.

4.1 Operadores

Operador	*Descripción*
=	asignación, puede combinarse con otros operadores: x += y es equivalente a x = x+y
+	suma
-	resta
++	incrementa en 1 el valor de una variable. Importante: es inmediato, si bien se escribe x++ equivale al ++x de lenguajes como C, Perl, etc
--	decrementa en 1 el valor de una variable (también es inmediato)
*	multiplicación
/	división
%	módulo
^	potencia
| número |	valor absoluto de un número
| texto |	longitud de un texto
| vector |	longitud de un vector

4.2 Funciones matemáticas

Función	*Descripción*
sin(x)	seno de x
cos(x)	cos de x
tan(x)	tangente de x
asin(x)	arcoseno de x
acos(x)	arcocoseno de x
atan(x)	arcotangente de x
sqrt(x)	raiz cuadrada de x
log(x)	logaritmo natural de x
max(a, b)	máximo entre a y b
min(a, b)	Mínimo entre a y b
angle(v1, v2)	ángulo entre los vectores v1 y v2
cross(v1, v2)	producto vectorial entre los vectores v1 y v2
dot(v1, v2)	producto punto entre los vectores v1 y v2
transpose(m)	retorna la transpuesta de la matriz m
random(x)	retorna un número aleatorio entre 0 y x (inclusive). Si x es un vector, retorna un vector entre (0, 0, 0) y dicho vector
randomGauss()	retorna un número aleatorio de punto flotante con una distribución gausiana

4.3 Tipos de Datos

Tipo de Dato	*Descripción*	*Ejemplo*
int	Número Entero (32 bits)	185
float	Número Real de punto flotante (64 bits)	23.55
object	Instancia de una Clase	miObjeto = new MiClase
vector	Vector de 3 componentes (números reales)	(1.25, 2.0, 1.83)
matrix	Matriz de 3x3 componentes (números reales)	[(1.0,1.5,2.9), (0.5,3.4,5.3), (2.9,1.8,9.7)]
list	Lista de elementos, pueden ser de cualquier tipo soportado por el lenguaje, incluso otras listas. Se pueden combinar diferentes tipos en una misma lista.	{ 1, “hola”, miObjeto, 3.65 } { “lista anidada”, {1, 2, “a”} }
string	Cadena de caracteres	“hola”
hash	Lista con mapeo clave-valor. Tanto las claves como los valores pueden ser de cualquier tipo soportado por el lenguaje.	miHash{“clave 1”} = “valor 1” miHash{“clave 2”} = “valor 2”
[] (array)	Lista de elementos de un mismo tipo.	

4.4 Sentencias y Expresiones

Las *sentencias* son las mínimas unidades de ejecución de un programa. En Steve, el fin de cada sentencia se indica por medio de un punto al final de la misma. Una sentencia puede estar compuesta por varias *expresiones* (combinaciones de variables, operadores y funciones), las cuales se combinan de acuerdo a reglas de precedencia. Dicha combinación puede indicarse de

forma explícita por medio del uso de paréntesis. Se denomina *expresiones condicionales* a aquellas que evaluan a un valor booleano (verdadero o falso).

Entre las sentencias posibles en Steve se encuentran:

- Asignación, que permite establecer el valor de una variable
  Ej: *variable = expresión.*
- Llamada a una subrutina
  Ej: *print expresión.*
- Llamada a un método
  Ej: *(miObjeto miMétodo).*
- Sentencias compuestas (por medio del uso de estructuras de control)
  Ej: *if expresión condicional: sentencia_1. else sentecia_2.*

Las sentencias pueden agruparse en bloques indicando el comienzo y fin de los mismos por medio de llaves:

```
{
  sentencia_1.
  sentencia_2.
  …
  sentencia_n.
}
```

4.5 Estructuras de Control

Las estructuras de control permiten alterar el flujo del programa por medio de la ejecución condicional de bloques de código y son idénticas en funcionamiento a sus equivalentes en C y lenguajes similares, con una mínima diferencia de sintaxis. En C, la condición a evaluar debe siempre estar entre paréntesis, mientras que en Steve, estos son opcionales debiéndose agregar dos puntos al final de la misma. Por ejemplo:

C/C++/Java:

if (condición) { sentencia_1; sentencia_2; … sentencia_n;} else { … }

Steve:

if condición: { expresión_1. expresión_2. … expresión_n.} else { … }

El flujo normal de ejecución de las expresiones es secuencial, en el orden en el que se ingresaron. Existen cuatro estructuras de control que permiten alterar dicho flujo:

- if - else
- while
- for
- foreach

4.5.1 If - Else

Permite la ejecución condicional de una sentencia o un bloque de sentencias, opcionalmente ejecutando una segunda sentencia o bloque en caso de no cumplirse la condición:

if *expresión condicional****:***
    (sentencia | bloque)
*[****else***
    (sentecia | bloque)]

Ejemplo:

if *x == 5****:***
    print "cinco".
else
    print "otro valor".

4.5.2 While

Permite la ejecución de una sentencia o un bloque de sentencias de forma repetitiva, mientras se cumpla una condición, la cual se vuelve a evaluar con cada repetición:

while *expresión condicional:*

(sentencia | bloque)

Ejemplo:

***while** x < y**:** {*
    print x.
    *x *= 3.*
}

4.5.3 For

Similar al while, pero diseñado para simplificar un caso de uso común, permite repetir la ejecución de una sentencia o bloque un número determinado de veces, indicando no sólo una condición, sino también la inicialización de una variable y una operación de modificación del valor de esta. La inicialización se ejecutará una única vez, mientras que tanto la modificación del valor de la variable como la evaluación de la condición se realizarán en cada repetición:

for** expresión inicialización**,** expresión condicional**,
*sentencia_modificación**:***
    (sentencia | bloque)

Ejemplo:

for** x = 0, x < 10, x++**:
    print x.

4.5.4 Foreach

Permite recorrer una lista, almacenando temporariamente cada elemento de la misma en una variable y ejecutando una sentencia o bloque, que podrá operar con dicho elemento:

foreach** variable temporaria **in** lista**:
    (sentencia | bloque)

Ejemplo:

foreach** individuo **in** poblacion**:

print (individuo get-fitness).

4.6 Clases y Objetos

Al igual que otros lenguajes orientados a objetos, steve permite la declaración de *clases* y la creación de instancias de estas (*objetos*).
Una declaración mínima de clase tiene la siguiente forma:

```
ClasePadre : MiClase {
    …
}
```

Se permite la herencia simple, es decir, una clase dada sólo puede heredar de una única clase, no de una combinación de estas. En el ejemplo anterior, MiClase heredará sus variables y métodos de la clase ClasePadre.

La creación de *instancias* tiene dos formas, dependiendo de si se creará una instancia única o una lista:

```
instancia = new Clase.
listaInstancias = cantidad new (Clase|PluralClase).
```

En el primer caso *instancia* es una variable de tipo "Clase" o una clase en un nivel superior de la jerarquía. En el segundo caso, la variable *listaInstancias* debe ser del tipo *list*.

Ejemplo:

```
miLibro (Object).
misLibros (list).

miLibro = new Libro.
misLibros = 5 new Libros.
```

Cabe destacar que en el lenguaje Steve, *Object* es la clase base de la cual heredan todas las demás. Además, en la última sentencia se puede notar que se utiliza "*Libros*" en plural, en lugar de *Libro*. Esto es posible porque Steve pemite la declaración de alias para los nombres de clases, especificamente

pensados para ser utilizados en la declaración de listas de instancias, facilitando así la lectura:

ClasePadre : MiClase ***(aka MiClasePlural) {***
…
}

En el ejemplo previo, la clase Libro tendría la siguiente declaración:

Object : Libro ***(aka Libros) {***
…
}

4.6.1 Variables

Una clase puede tener *variables de instancia*, las que se deben declarar debajo de un encabezado "*+ variables:*" y con el formato *nombreVar (tipoVar)*, donde tipoVar puede ser cualquiera de los descriptos en la sección 4.3 de este Anexo. Por ejemplo:

Vehiculo : Camion {
+ variables:
peso (float).
longitud (float).
numRuedas (int).
patente (string).
}

4.6.2 Métodos

Además de variables, una clase puede poseer de una serie de *métodos*, tanto *públicos* (accesibles desde fuera de la clase) como *privados* (sólo podrán ser referenciados por otros métodos dentro de la misma clase).

Los *métodos públicos* se declararán con el siguiente formato:

+ to *nombre-metodo [nombre-parametro1 nombre-var1 (tipo_var1) …]* ***:***
sentencia1.
[…]

Los *métodos privados* se declararán de forma similar, sólo que en lugar de comenzar la declaración con "**+**" se comenzará con "**-**".

Para ejecutar un método en una instancia de una clase se debe utilizar el siguiente formato:

(objeto metodo [nombre-parametro1 valor-parametro1 …]).

Además, hay dos objetos especiales: *self*, que permite hacer referencia a la instancia actual, y *super*, que permite acceder a los métodos declarados en la clase padre.

```
Vehiculo : Camion {
    + variables:
            peso (float).
            longitud (float).
            numRuedas (int).
            patente (string).

    + to get-patente:
            return patente.

    + to set-patente to-value valor (string):
            patente = valor.

    + to get-peso:
            return peso.

    + to get-peso-estimado:
            return (self estimar-peso basado-en-largo:largo
ruedas:numRuedas).

    - to estimar-peso basado-en-longitud metros (float) ruedas cantidad
(int):
            return metros * 1000 + cantidad*30.

    …
}
```

Nota: no es necesario, incluso en el caso de que un método contenga varias sentencias, declararlas dentro de un bloque. El comienzo de otro método o el final de la clase determina la extensión del mismo.