UNIVERSIDADE DA CORUÑA

facultade de informática da coruña

Robótica evolutiva

Escuela de verano de Inteligencia Artificial 2014
Asociación Española para la Inteligencia Artificial

José Antonio Becerra Permuy

Robótica Evolutiva

Utilización de técnicas de computación evolutiva para la obtención automática de controladores de robots autónomos.

Computación Evolutiva

Algoritmos de optimización estocásticos que se inspiran en la evolución natural:

- *Los individuos de una población representan soluciones candidatas a un problema dado.*
- *El proceso evolutivo (nacimiento, reproducción y muerte) se convierte en un proceso de búsqueda de una solución.*
- *Las reglas que rigen dicho proceso evolutivo guían a la población de forma que los individuos cada vez representan mejores soluciones (➪ los "mejores" individuos tienen más probabilidades de reproducirse y / o viven más)*

Robots autónomos

Robots capaces de reaccionar correctamente, sin supervisión exterior, en situaciones no consideradas explícitamente en su programación.


Robótica basada en comportamiento
Robótica evolutiva
Robótica cognitiva
CDR

UNIVERSIDADE DA CORUÑA

facultade de informática da coruña

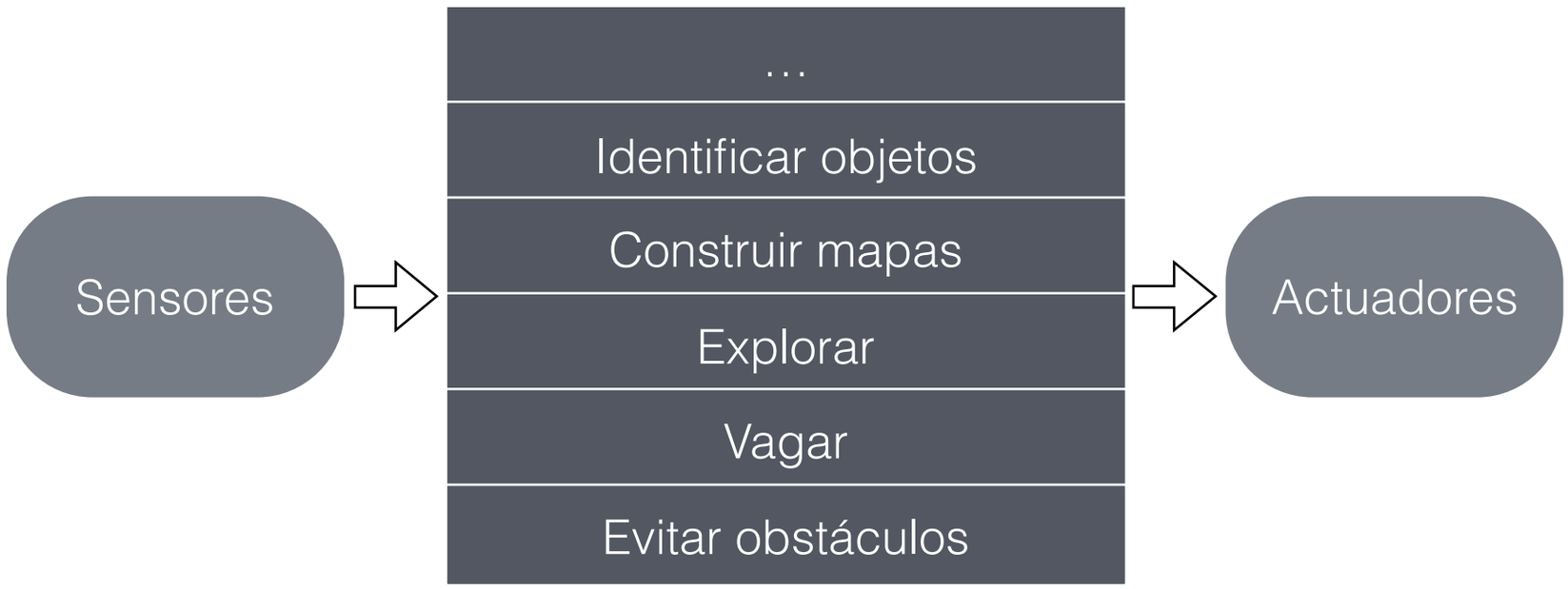
ROBÓTICA BASADA EN COMPORTAMIENTO

Conocimiento vs. comportamiento

- Robótica basada en conocimiento:
  - Modela el conocimiento.
  - Representación interna simbólica de acciones, metas y eventos.
  - Problema cuando el conocimiento del dominio es incompleto.
  - Problema por la diferencia en la percepción del entorno entre el hombre y el robot.
- Robótica basada en comportamiento:
  - Se intentan reproducir los comportamientos.
  - Problema: determinar por qué el sistema se comporta de la manera en la que lo hace.

Conocimiento vs. comportamiento

Control basado en conocimiento



Ejemplo de control basado en comportamiento

Definiciones

- *Comportamiento*: par estímulo/respuesta para una determinada configuración del entorno que está modulado por la atención y determinado por la intención.
    - *Atención*: prioriza las tareas y la utilización de los recursos sensoriales y está determinada por el contexto del entorno en un momento dado.
    - *Intención*: determina que conjunto de comportamientos debería de estar activo en base a los objetivos del robot.
- *Comportamiento emergente*: comportamiento global del robot como consecuencia de la interacción de los comportamientos individuales activos.
- *Comportamiento reactivo*: comportamiento generado mediante conexiones directas entre sensores y actuadores, sin persistencia en la información sensorial ni modelos de mundo explícitos.

Propiedades

- *Contextualización (situatedness)*: El robot es una entidad situada y en contacto directo con el mundo real. No opera sobre representaciones abstractas de la realidad, si no sobre la realidad en sí misma.

- *Embodiment*: El robot tiene una presencia física y esta realidad espacial tiene consecuencias en sus interacciones con el entorno que no pueden ser simuladas fielmente.

- *Emergence*: La inteligencia surge de las interacciones del robot con el entorno. No es una propiedad ni del robot ni del entorno de forma aislada, si no el resultado de la interacción entre ambos.

Consideraciones

- *Grounding in reality*. Los sistemas robóticos basados en conocimiento presentan el symbol grounding problem; el robot “razona” utilizando una abstracción de la realidad. En la robótica basada en comportamiento el robot trabaja directamente sobre el mundo real.

- *Ecological dynamics*. El robot se desenvuelve en un entorno altamente dinámico. Por ello, el control debe de ser adaptativo.

- *Scalability*. ¿Hasta qué punto es escalable un sistema basado en comportamiento? ¿Sistemas híbridos?.

Arquitecturas de control

- En un problema real no trivial:
  - Imposibilidad de implementar a mano todo el sistema de control.
  - Imposibilidad de implementar un controlador monolítico para un comportamiento complejo.
- Necesidad de:
  - Utilizar algoritmos de aprendizaje máquina para obtener automáticamente parte o toda la arquitectura de control.
    - Idealmente, el aprendizaje debe de poder realizarse (no necesariamente de forma exclusiva) en tiempo de ejecución del sistema de control para permitir la adaptación en entornos dinámicos y no estructurados.
  - Dividir el comportamiento global en sub-comportamientos (módulos) que se puedan obtener separadamente de forma incremental.

Arquitecturas de control

- Ejemplos de algoritmos de aprendizaje máquina empleados en BBR:
  - Sistemas clasificadores.
  - Lógica borrosa.
  - Redes de Neuronas Artificiales.
- Clasificación de las arquitecturas de control modulares en función del tipo de coordinación entre módulos:
  - Arquitecturas jerárquicas. Fácil obtener comportamientos complejos de forma incremental. Normalmente, elevado grado de intervención humana en el proceso de diseño.
  - Arquitecturas distribuidas. Menor grado de intervención humana ya que no es necesario establecer niveles de comportamientos, pero mayor dificultad para obtener comportamientos complejos debido a la necesidad de implementar métodos de coordinación automática.

Sistemas clasificadores

- *Learning classifier system (LCS).*

- Algoritmo de aprendizaje máquina ideado por Holland que utiliza aprendizaje por refuerzo y algoritmos genéticos.

- Busca representar explícitamente el conocimiento pero que este se obtenga automáticamente y no extraerlo de expertos.

- El primero en aplicarlo en robótica autónoma fue Marco Dorigo en 1992.

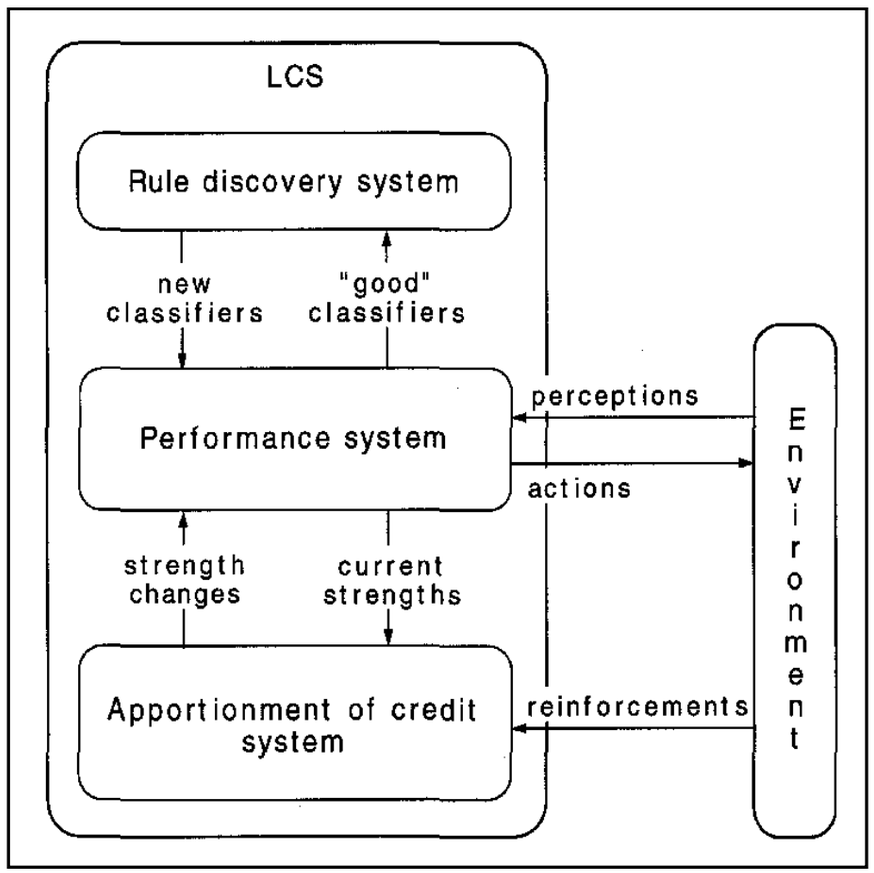
Sistemas clasificadores

- El *sistema de actuación* opera el robot aplicando el conjunto de reglas (*clasificadores*) existente en un momento dado.

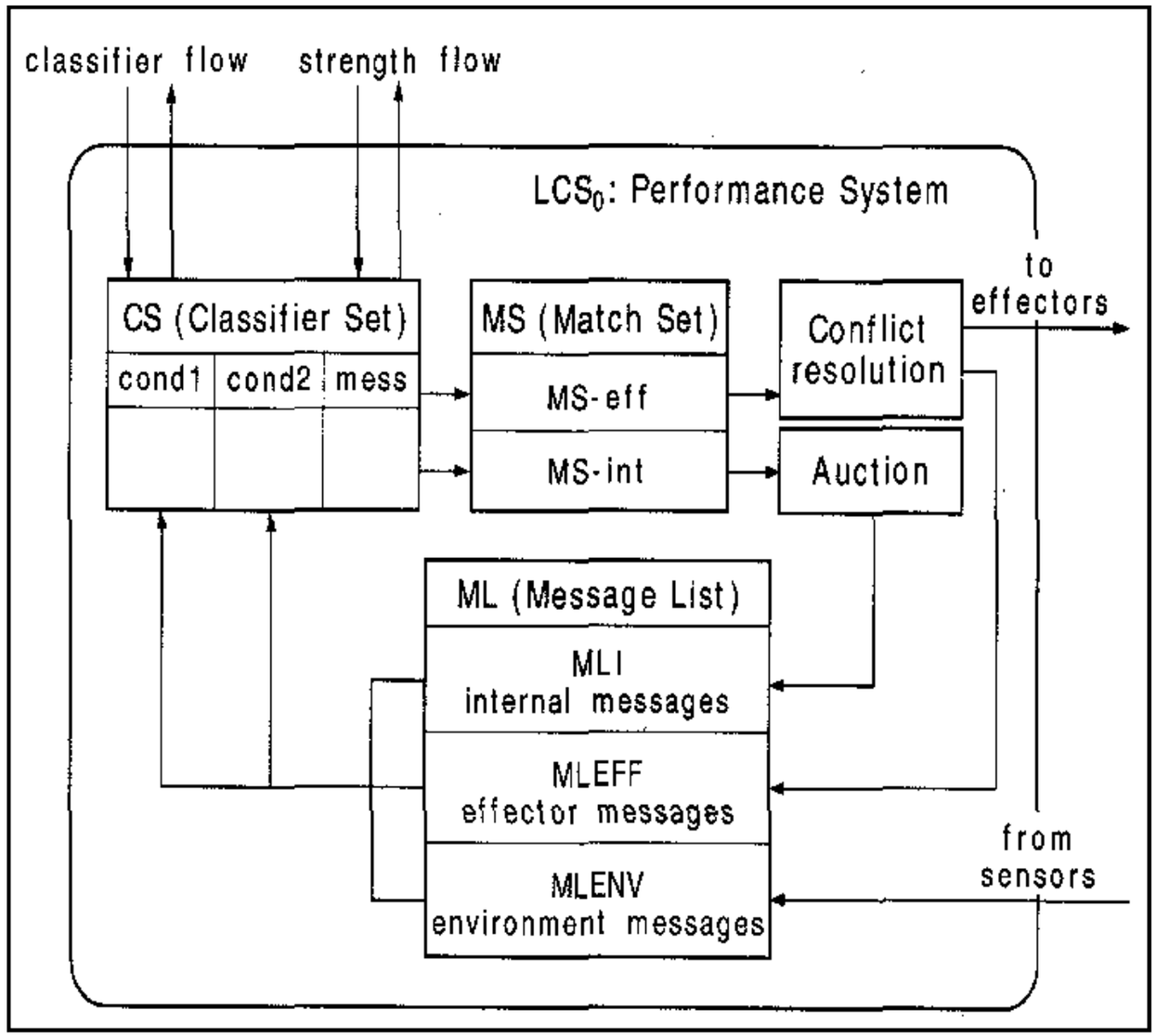
- El *sistema de generación de reglas* crea nuevos clasificadores por medio de un algoritmo genético.

- El *sistema de distribución de crédito* se encarga de distribuir el refuerzo en las reglas de la base de conocimiento. Cada regla posee un peso en función de si ha contribuido o no a un refuerzo positivo. Se usa el *bucket brigade* de Holland para repartir ese refuerzo entre las reglas.

Sistemas clasificadores



Esquema global



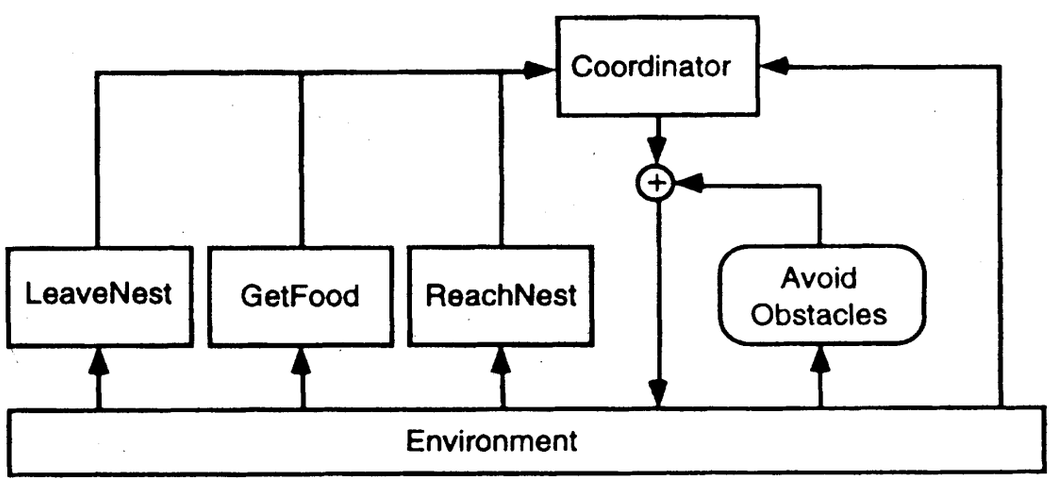
Esquema del sistema de actuación

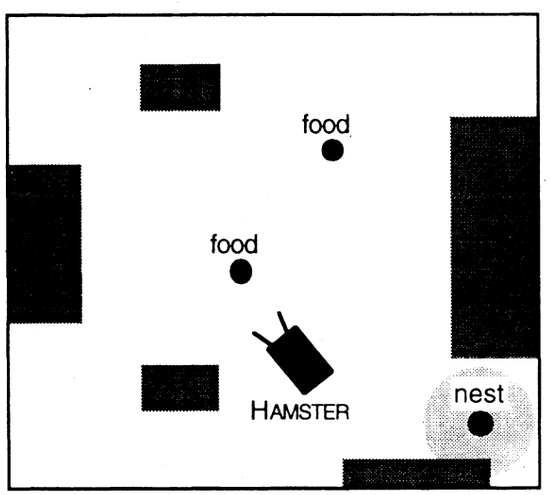
Sistemas clasificadores

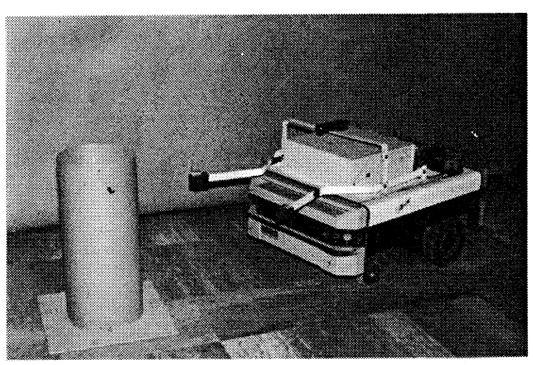
- Componentes del *sistema de actuación*:
  - CS: conjunto de clasificadores. Son reglas con dos condiciones y una acción o mensaje.
  - MS: clasificadores que han sido activados.
  - MS-eff: clasificadores que envían mensajes a los actuadores.
  - MS-int: clasificadores que envían mensajes a la lista de mensajes interna (MLI).
  - Módulo de resolución de conflictos: arbitra conflictos entre reglas en el conjunto MS-eff que proponen acciones contradictorias.
  - Módulo de selección: selecciona que reglas del conjunto MS-int añaden mensajes al conjunto MLI.
  - ML: lista de mensajes con las siguientes sublistas:
    - MLI: mensajes previos.
    - MLENV: mensajes del entorno obtenidos de los sensores.
    - MLEFF: mensajes usados para elegir la acción anterior.

Sistemas clasificadores

- En la arquitectura propuesta por Marco Dorigo los sistemas clasificadores se utilizaban para obtener los distintos módulos automáticamente, pero la interconexión la realizaba manualmente el diseñador utilizando una arquitectura jerárquica.
- Ejemplo: Llevar comida al nido
    - Robot Hamster.
    - 4 módulos de comportamiento en el nivel inferior.
    - 1 nivel de coordinación en un nivel superior.
    - Se usan los sensores del sonar y una cámara para identificar y diferenciar los cilindros que simbolizan comida y nido.





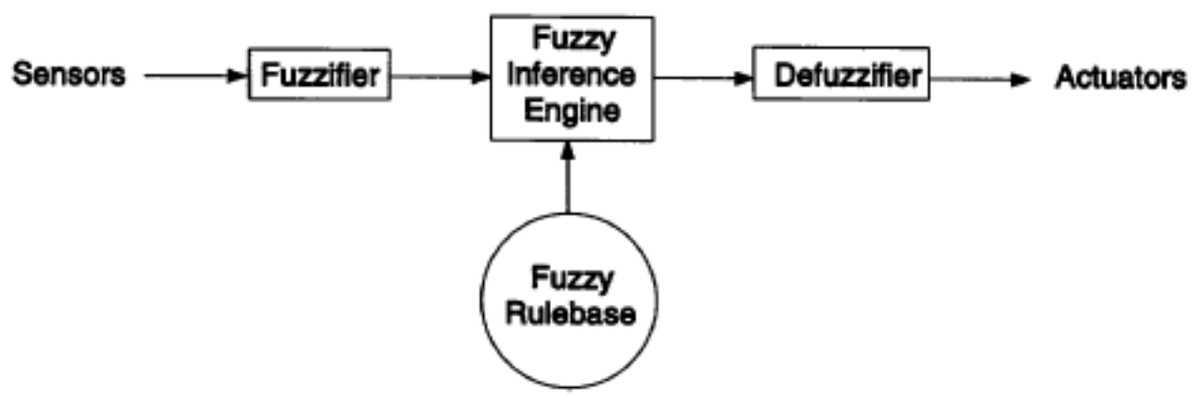


Lógica borrosa

- Al igual que con los sistemas clasificadores, se busca representar explícitamente el conocimiento pero que este se obtenga automáticamente y no extraerlo de expertos.
- La lógica borrosa proporciona una forma de tratar con el comportamiento de sensores y actuadores.
- Los comportamientos se implementan utilizando lógica borrosa:
  - Conjuntos borrosos en sensores y actuadores.
  - Los controladores están formados por reglas borrosas.
  - Las reglas no tienen por qué disparar solo actuaciones simples (meta-reglas).
  - Arquitectura distribuida implícita: Varios comportamientos pueden activarse simultáneamente en diferente grado (*blending*).
  - Se puede forzar una arquitectura jerárquica si introducimos variables adicionales (además de las necesarias para los sensores) y se diseña el sistema para que haya varias capas.
  - Se puede dividir el comportamiento global en varios sistemas borrosos actuando en paralelo, cada uno con su conjunto de reglas y motor de inferencia.

Lógica borrosa

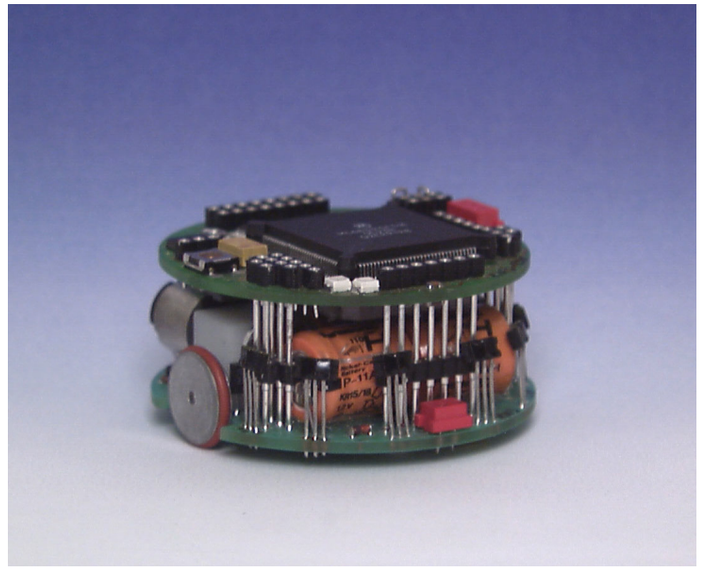
- Primeros autores en aplicarla en BBR:
  - Saffiotti, Ruspini, Konolige (1993). Varias arquitecturas diferentes, una híbrida que incluía planificación.
  - Hoffmann, Pfister (1994). Las reglas se obtienen automáticamente mediante un algoritmo genético.

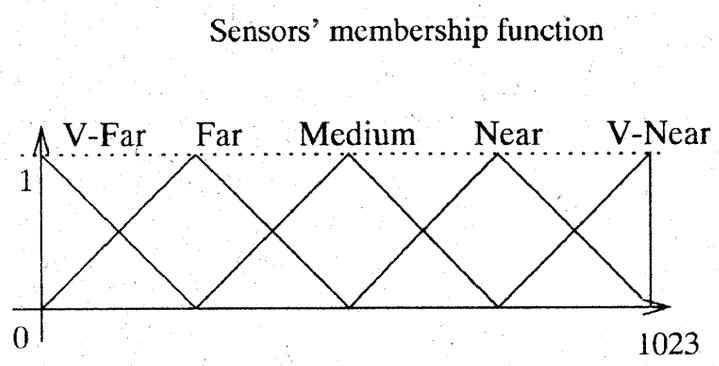


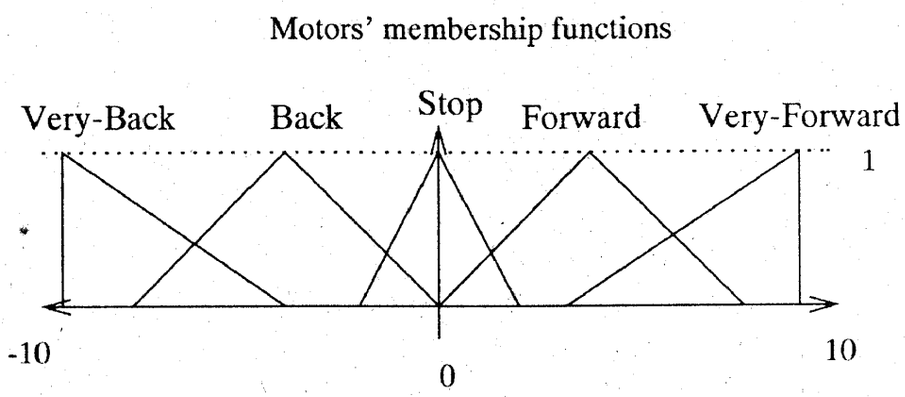
Esquema global de un controlador con lógica borrosa

Lógica borrosa

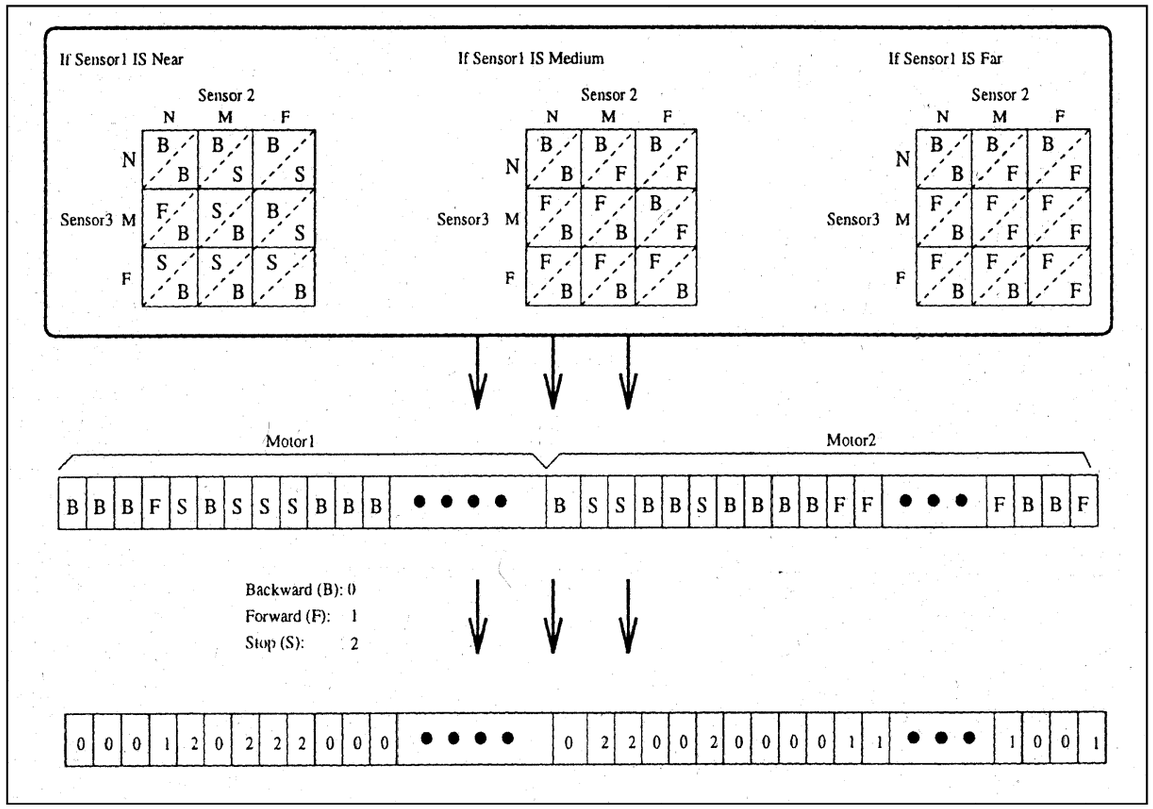
- Ej. Robot Khepera:
  - Sensores de infrarrojos
  - Actuadores motores de corriente continua acoplados a dos ruedas

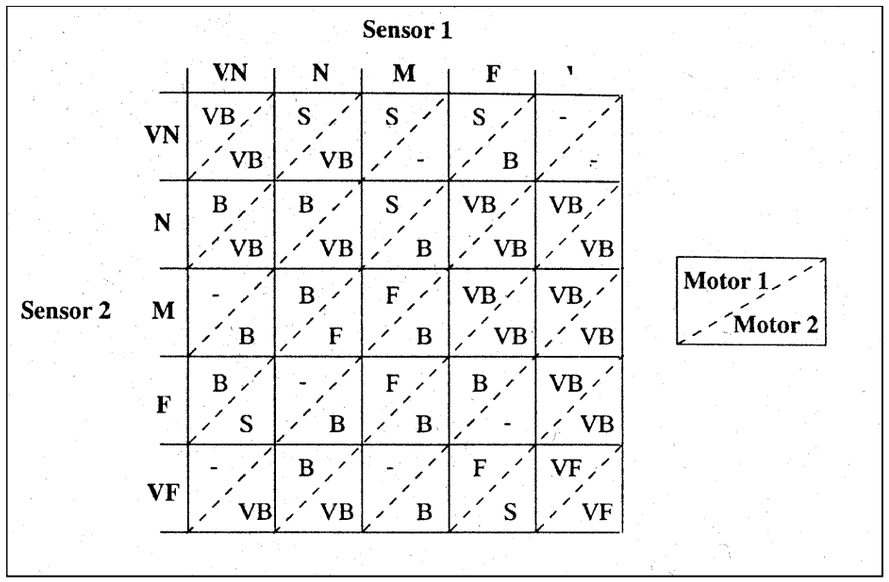






Lógica borrosa





Ejemplo de posible codificación genética de las reglas de un controlador borroso

Redes de Neuronas Artificiales

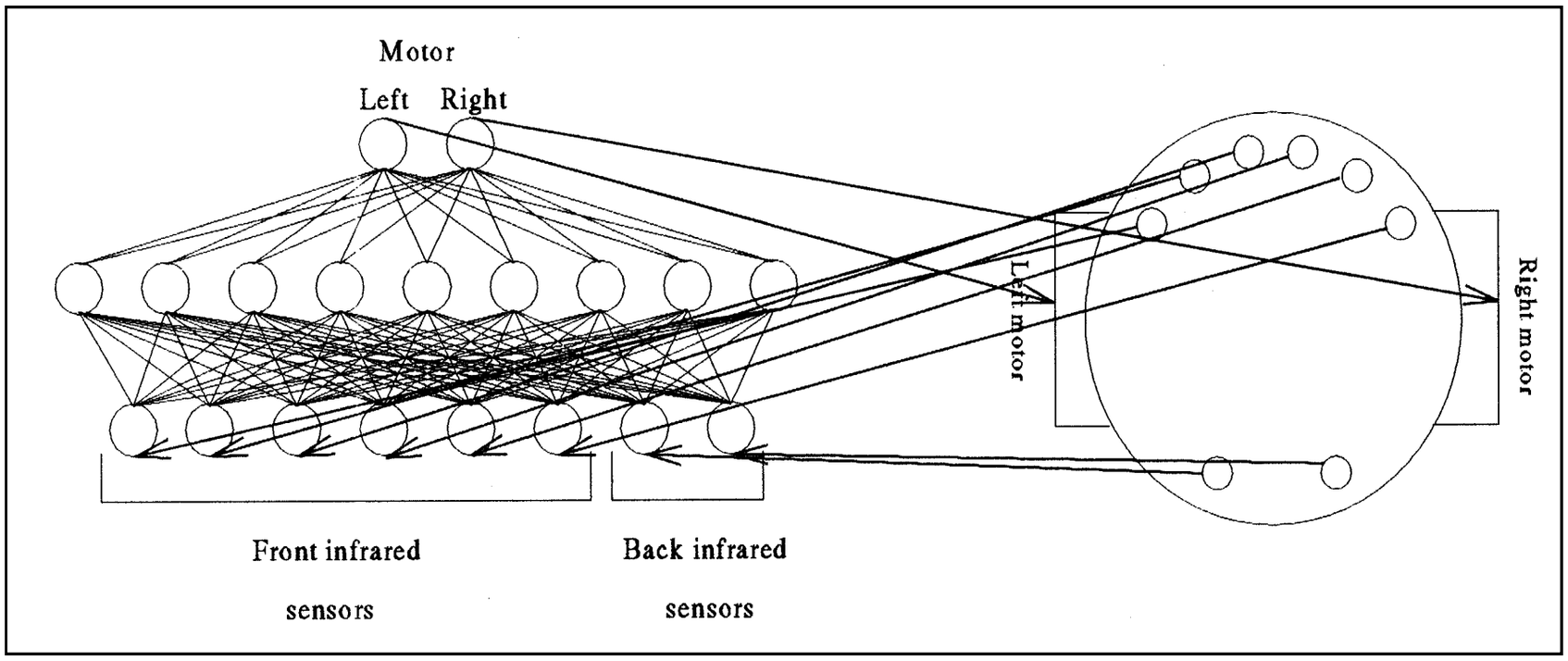
- Aproximadores no lineales universales inspirados biológicamente en el funcionamiento del cerebro.

- Tolerancia a ruido.

- Permiten obtener fácilmente controladores de forma automática mediante entrenamiento o evolución, ya sea de forma previa o en tiempo real.

- Dificultad para extraer conocimiento una vez obtenido el sistema.

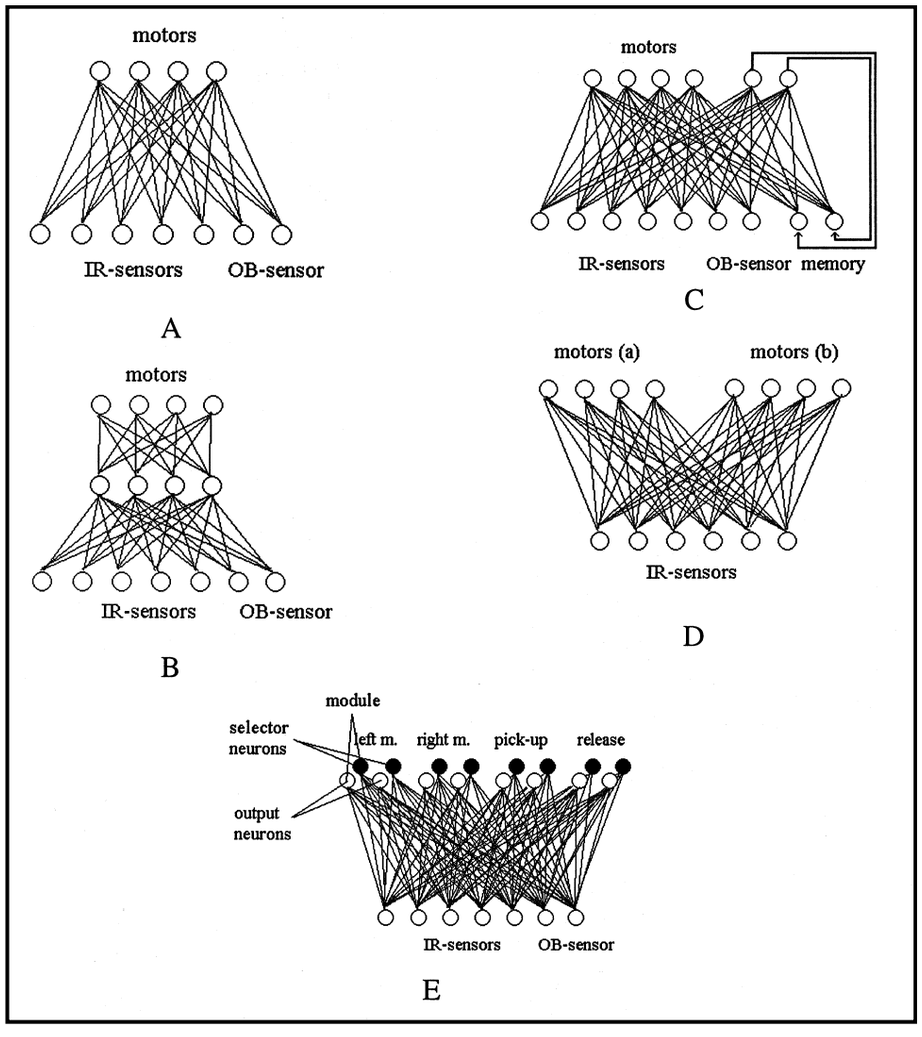
Redes de Neuronas Artificiales



Ejemplo de controlador neuronal para un robot Khepera (Lund 1996)

Redes de Neuronas Artificiales

- En robótica se han utilizado infinidad de tipos de neuronas y tipologías de redes.
- Uno de los primeros en estudiar varias tipologías en robots dentro de BBR fue Stefano Nolfi en 1994.
    - Siempre sin jerarquía de módulos. El mecanismo de selección y los diferentes módulos (cuyo número debe establecerse a priori) se obtienen mediante evolución.
- Diferentes arquitecturas de redes:
    A. Red estándar sin realimentaciones.
    B. Perceptron con capa oculta.
    C. Red recurrente.
    D. Arquitectura modular con 2 módulos prediseñados.
    E. Arquitectura modular emergente. El número de módulos está predefinido, pero no su funcionalidad.



UNIVERSIDADE DA CORUÑA

facultade de informática da coruña

COMPUTACIÓN EVOLUTIVA

Computación evolutiva

- *Evolutionary Computation / Artificial Evolution*.

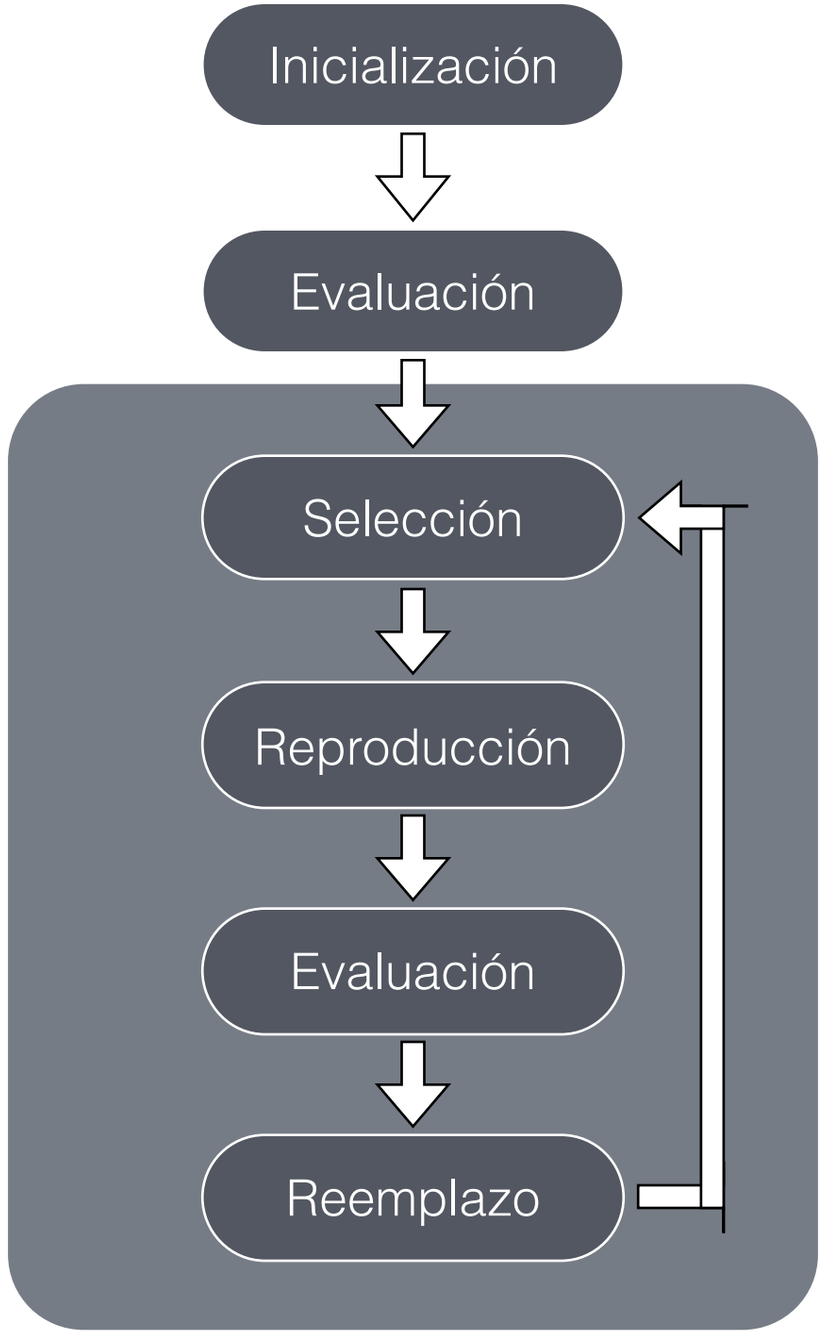
- Algoritmos de optimización estocásticos que se inspiran en la evolución natural:

  - Los individuos de una población representan soluciones candidatas a un problema dado.

  - El proceso evolutivo (nacimiento, reproducción y muerte) se convierte en un proceso de búsqueda de una solución.

  - Las reglas que rigen dicho proceso evolutivo guían a la población de forma que los individuos cada vez representan mejores soluciones (➪ los “mejores” individuos tienen más probabilidades de reproducirse y / o viven más)

Esquema general



ELEMENTOS

Codificación

- Codificación del espacio de soluciones en forma de genes y cromosomas.
  - ¿Codificación directa o desarrollo?
    - Codificación directa ➪ Puede provocar cromosomas muy grandes ➪ Espacios de búsqueda con un elevado número de dimensiones.
    - Desarrollo ➪ Definición de las reglas ➪ Es necesario asegurarse de que se pueden generar todas las posibles soluciones sin bias alguno.
  - ¿Introducimos conocimiento del dominio del problema?
    - No ➪ Potencialmente, la búsqueda puede durar mucho más.
    - Sí ➪ Es necesario asegurarse de que no introducimos bias no deseado en la búsqueda y que el conocimiento introducido es 100% correcto (factor humano). Pérdida de generalidad.

Evaluación

- ¿Cómo se evalúan los individuos?
  - Si la evaluación no es determinista:
    - Es necesario evaluar varias veces un mismo individuo ➪ Cuidadosa selección del número de evaluaciones y de la valoración estadística de cada una de ellas (casos significativos, que cubren el mayor espectro posible, mayor valor a los casos más frecuentes...)
  - Si la evaluación es muy costosa:
    - Tratar de usar simulación si la evaluación es un entorno real, tratar de simplificar el modelo si se trata de una simulación ➪ Es importante que la simulación o el modelo empleado sea lo suficientemente realista de forma que la solución obtenida siga siendo válida en el mundo real.
    - Utilizar un algoritmo evolutivo que trate de minimizar el número de evaluaciones.
    - Utilizar un algoritmo evolutivo distribuido.

Selección

- La probabilidad de reproducirse tiene que ser mayor a mayor calidad del individuo. Muchas formas de establecer esa relación (lineal, logarítmica, escalones...)

- Mucha preponderancia de los buenos individuos ➪ Mayor presión evolutiva ➪ Más rapidez en obtener una solución pero mayor probabilidad de caer en óptimo local.

- Poca preponderancia de los buenos individuos ➪ Menor presión evolutiva ➪ Es necesario más tiempo para obtener la solución y quizás no lleguemos a converger.

Reproducción

- ¿Sexual o asexual?
  - Sexual ➪ El operador fundamental es el cruce. La mutación tiene una incidencia menor y se utiliza para introducir nuevo material genético (ej: GA) ➪ ¿Cómo tiene que ser el cruce? (por un punto, por dos, multipunto, uniforme...)
  - Asexual ➪ El único operador utilizado para cambiar el material genético es la mutación (ej: ES) ➪ ¿Cómo tiene que ser esta? (qué distribución de probabilidad sigue, igual o distinta para cada gen...)
- En la naturaleza los organismos más complejos usan reproducción sexual, pero en la evolución artificial no hay una predominancia clara ➪ sólo estamos ante una analogía.

Reproducción

- Explotación versus exploración.
- Explotación ➪ Utilización de material genético existente en la población para crear nuevos individuos (ej: cruce por uno o dos puntos)
- Exploración ➪ Creación de nuevo material genético de forma aleatoria (ej: mutación con distribución uniforme)
- Predomina la explotación ➪ Más presión evolutiva, la solución se puede alcanzar antes pero hay mayor riesgo de caer en un óptimo local.
- Predomina la exploración ➪ Menos presión evolutiva, evolución más lenta.

Reemplazo

- ¿Síncrono o asíncrono? (los hijos reemplazan a los padres todos al mismo tiempo o no)
  - Síncrono ➪ Opción más habitual.
  - Asíncrono ➪ Más frecuente en embodied evolution y en VA.

- ¿Todos los individuos viven lo mismo? Opciones:
  - Sí ➪ Problema, puede que se pierdan buenos individuos y la población oscile continuamente.
  - Mayoritariamente sí, pero se hace una excepción con el mejor o mejores ➪ elitismo (ej: GA con elitismo)
  - No ➪ Ej: DE (un individuo sólo reemplaza a otro si es mejor), MA...

UNIVERSIDADE DA CORUÑA

facultade de informática da coruña

TÉCNICAS DE COMPUTACIÓN EVOLUTIVA

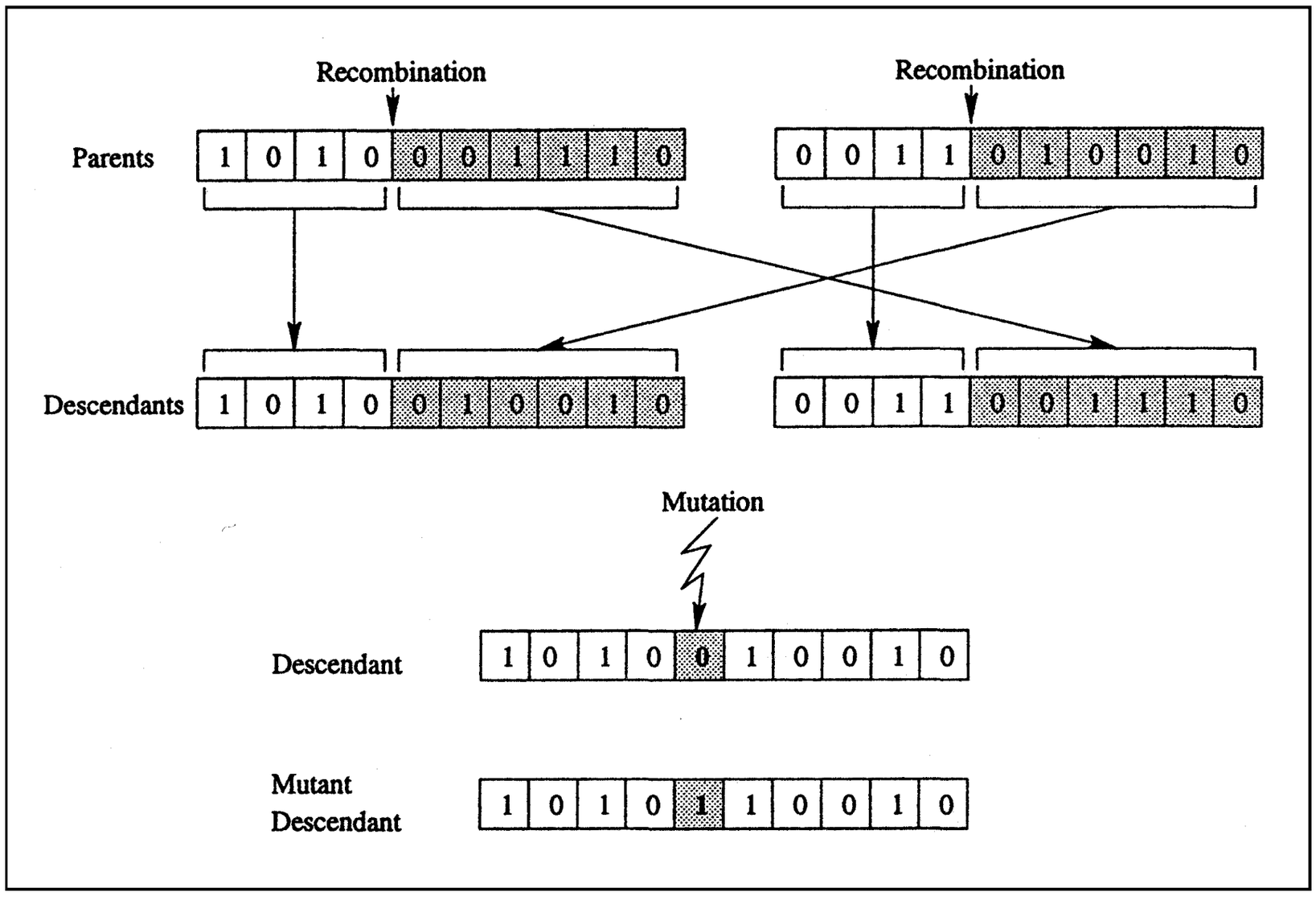
Técnicas

- Evolutionary algorithm (EA)
  - Genetic algorithm (GA)
  - Evolution strategy (ES)
  - CMA-ES
  - Differential Evolution (DE)
  - Genetic programming (GP)
  - Evolutionary programming (EP).
  - Macroevolutionary Algorithms (MA).
- Swarm intelligence (SI)
  - Particle swarm optimization (PSO)
  - Ant colony optimization (ACO)

Algoritmos genéticos

- John Holland (1975)
- Representación
    - Cromosomas binarios (cada gen es un 0 o un 1)
    - Fundamento teórico (controvertido):
        - *Building Block Hypothesis => Schema Theorem*
- Selección
    - Ruleta.
- Reproducción
    - El cruce (por un punto) es el operador genético fundamental.
    - La mutación (cambiar un 0 por un 1 o viceversa) ocurre con poca probabilidad y sirve para introducir nuevo material genético.
- Reemplazo (total).

Algoritmos genéticos



Algoritmos genéticos

- Múltiples variaciones (para las cuales el formulismo de Holland deja de ser válido):
  - Representación
    - Número reales en vez de bits
  - Selección
    - Ranking o torneo (para evitar convergencia prematura)
- Reproducción
  - Cruce ➪ Cruce por dos puntos, uniforme (BLX-α, SBX, BGA...)
  - Mutación (distribuciones distintas a la uniforme, ej: normal)
- Reemplazo
  - Elitismo

Estrategias evolutivas

- Ingo Rechenberg y Hans-Paul Schwefel (1971)
- Representación
    - Los genes son números reales.
- Selección ➪ Ranking.
- Reproducción
    - No hay cruce (en la versión original), la mutación es el único operador genético. El valor de la mutación sigue una distribución normal.
    - La desviación típica de la distribución normal suele ajustarse automáticamente en el propio proceso evolutivo (genes adicionales)
- Reemplazo
    - Varias posibles estrategias: (1+ λ)-ES, (1, λ)-ES, (μ, λ)-ES, etc. Uno o varios padres generan por mutación uno o varios descendientes y estos los reemplazan en la siguiente generación (en algunos casos sólo si mejoran a los padres)

Estrategias evolutivas: CMA-ES

- *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) ➪ Nikolaus Hansen y Andreas Ostermeier (2001)

- El espacio de búsqueda se modela con una distribución normal multivariada cuya forma va haciéndose más precisa a lo largo del tiempo.

- En cada generación, los μ mejores individuos se utilizan para actualizar los parámetros de dicha distribución normal multivariada, que generará por mutación a partir de su mejor individuo (representado por la media) los λ individuos de la generación siguiente.

Estrategias evolutivas: CMA-ES

- Si considerásemos cada gen individualmente, la mutación para cada uno seguiría una distribución distinta.

- La adaptación de la mutación es automática y se consigue sin incrementar el número de genes en el cromosoma, ya que se estima en cada generación utilizando los μ mejores individuos.

- Algoritmo muy rápido (en términos de número de evaluaciones) comparado con otros EA.

- Inadecuado ante óptimos parecidos muy separados en el espacio de búsqueda (modelado malo con una normal).

Estrategias evolutivas: DE

- *Differential Evolution* (DE) ➪ Kenneth Price y Rainer Storn (1995)

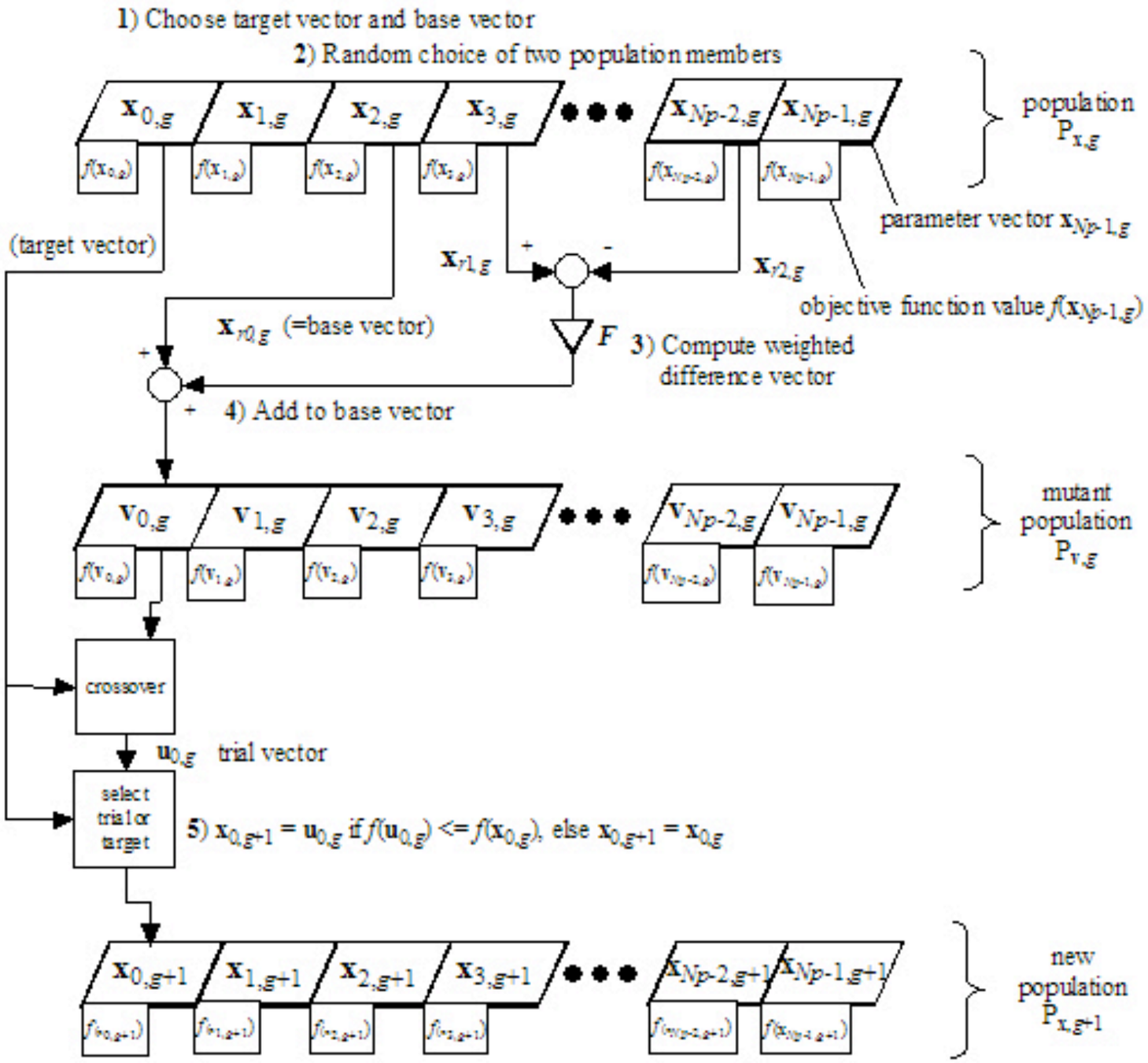
- La diferencia ponderada entre dos individuos escogidos aleatoriamente se añade a un tercero (*base*), bien el mejor o bien escogido aleatoriamente, para generar un nuevo individuo que se cruza, dando lugar al *trial*, con un individuo preseleccionado (*target*). Si el *trial* es mejor que el *target*, lo reemplaza en la población.

- Parámetros:

  - F ➪ Ponderación de la diferencia (lo más normal $\in$ [0.5,1])

  - CR ➪ Ratio de cruce (mejor bajo para funciones sep.) Funciona distinto según el tipo de cruce. En el binomial refleja la probabilidad de escoger un gen del *trial* (CR) o del *target* (1-CR). En el exponencial indica cuántos genes se mutarán.

Estrategias evolutivas: DE



Algorithm 2 Binomial crossover

```
1: crossoverBin (x,y)
2: k ← irand({1,...,n})
3: for j = 1,n do
4:     if rand(0,1) < CR or j = k then
5:         z^j ← y^j
6:     else
7:         z^j ← x^j
8:     end if
9: end for
10: return z
```

Algorithm 3 Exponential crossover

```
1: crossoverExp (x,y)
2: z ← x; k ← irand({1,...,n}); j ← k; L ← 0
3: repeat
4:     z^j ← y^j; j ← ⟨j+1⟩_n; L ← L+1
5: until rand(0,1) > CR or L = n
6: return z
```

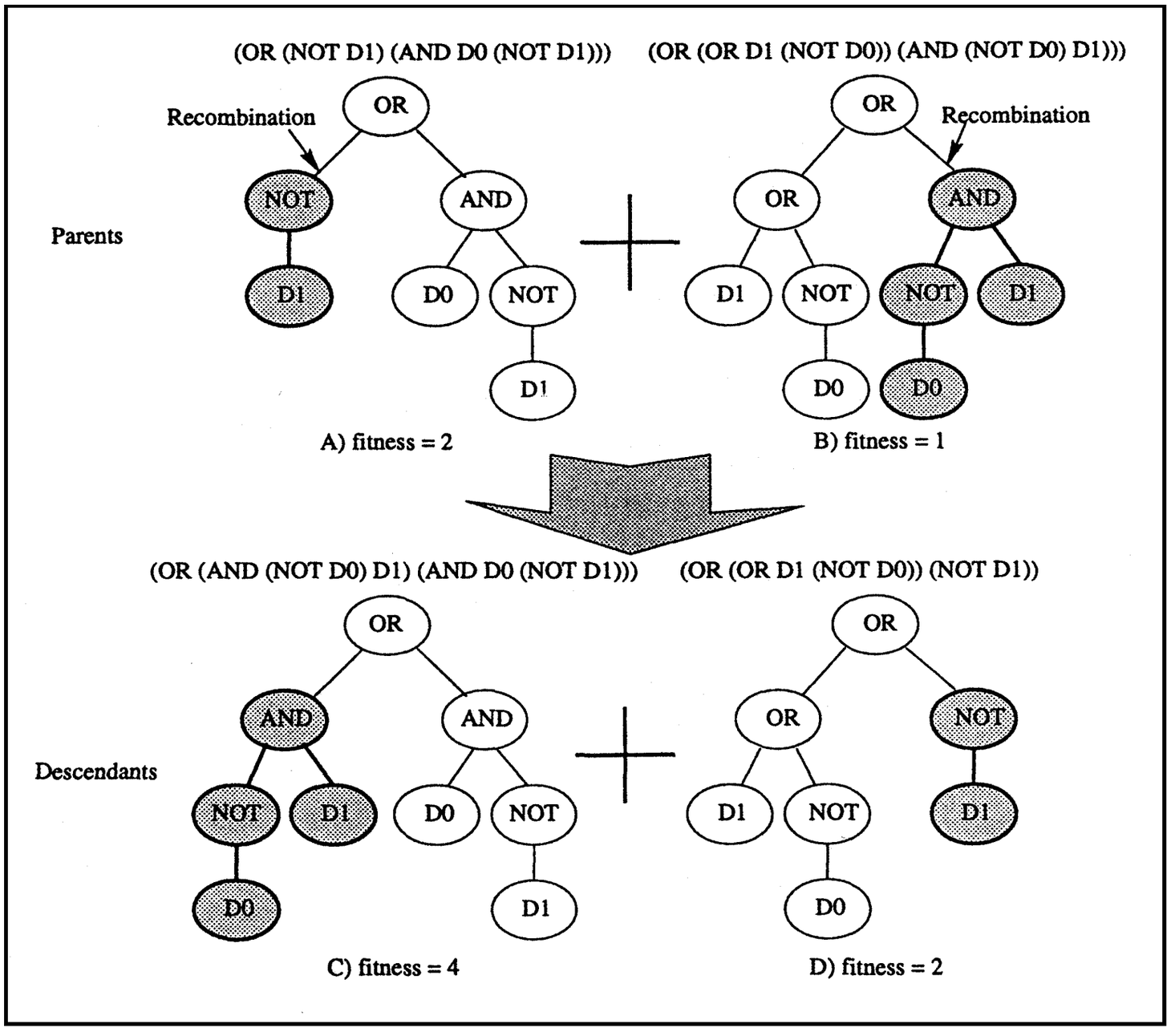
Estrategias evolutivas: DE

- No se maneja de forma explícita la distribución de probabilidad de mutación de los genes, pero está implícita en las operaciones.
- En función de la distribución de los individuos en el espacio de búsqueda, así serán las mutaciones (si la población está concentrada serán pequeñas, si está dispersa serán grandes)
- Equilibrio automático entre exploración y explotación.
- Suele presentar muy buenos resultados en general.

Programación genética

- Otros autores lo hicieron antes, pero John R. Koza (1989) le dio su forma actual.
- Se evolucionan, fundamentalmente, programas en un determinado lenguaje.
- Normalmente, el cromosoma representa un árbol.
- La preponderancia y motivación de los operadores es similar a un GA.
- El cruce consiste en el intercambio de nodos de individuos distintos.
- La mutación consiste en cambiar un nodo por otro escogido al azar.
- Tanto en el cruce como en la mutación las operaciones tienen que limitarse a aquellas que son sintácticamente correctas.

Programación genética



Programación evolutiva

- Lawrence Fogel (1960)
- Inicialmente se evolucionaban autómatas y el cromosoma representaba un grafo.
- La mutación era el único operador.
- Un individuo sólo reemplazaba a su progenitor si era mejor.
- La programación evolutiva se ha ido diluyendo / confundiendo con otros EA (ES o GP)

Algoritmos macroevolutivos

- J. Marín y R. V. Solé (1999)
- La analogía se hace con especies en vez de con individuos. Todas las implementaciones asumen que una especie equivale a un individuo en otro EA, pero eso no tendría por qué ser así.
- Una especie sobrevive si su calidad es mejor que el promedio de las calidades de todas las especies, pero ponderando por las distancias que las separan.
- Las peores especies se extinguen y su nicho es ocupado por especies derivadas de las existentes o totalmente nuevas (generadas aleatoriamente) en función de un parámetro τ.
- Si un nicho no es ocupado por una especie generada aleatoriamente, se genera a partir de la extinta y de una superviviente al azar (similar al cruce, controlado por ρ).

Algoritmos macroevolutivos

- Modificando τ se puede controlar fácilmente el balance entre exploración y explotación. Lo más normal es realizar annealing.

- Algoritmo con poca presión evolutiva ➪ Resistencia a caer en óptimos locales.

- Facilita la clusterización de los individuos / especies en el espacio de búsqueda.

Particle Swarm Optimization

- James Kennedy y Russell Eberhart (1995)
- Cada individuo es una partícula que, además de su posición en el espacio de búsqueda, tiene una velocidad (vector). También es consciente de su mejor posición (con mayor calidad) y de la mejor de sus vecinos o, incluso, de toda la población.
- No hay selección ni reemplazo, los individuos se mueven a medida que su vector de velocidad va cambiando, influenciado por su mejor posición, la de sus vecinos y la del mejor individuo.
- El problema típico es una convergencia prematura ➪ variación dinámica de las inercias (en función de la generación o de alguna característica detectable en el espacio de búsqueda, reinicio de las posiciones, repulsión…).

Ant Colony Optimization

- Marco Dorigo (1992)
- Analogía con las hormigas en la naturaleza.
- Orientado a la resolución de caminos óptimos en problemas de rutas (aunque otros problemas se pueden reformular para que entren en este paradigma)
- Las hormigas siguen rutas, en principio aleatorias, por los diferentes caminos que unen el “nido” con la “comida”, dejando feromonas por donde pasan a partir del momento en el que encuentran “comida”.
- Las feromonas se van evaporando a medida que pasa el tiempo.
- Las feromonas atraen a las hormigas de forma que estas tienden a seguir los caminos con más feromonas, los cuales, dada la evaporación, serán los más cortos.

CONSIDERACIONES

Evolución natural vs. artificial

- “Objetivo” de la evolución natural ➪ Supervivencia de los más aptos:
  - Múltiples soluciones igual de válidas.
  - Entorno co-evolutivo (una “solución” es buena o mala de forma comparativa con el resto de “soluciones”).
  - No hay restricciones temporales. Cambios en el entorno o en otros individuos disparan cambios en los individuos.
  - Mutación relativamente poco frecuente. Normalmente no produce resultados deseables, pero en ocasiones desemboca en importantes saltos evolutivos.

Evolución natural vs. artificial

- Objetivo de la evolución artificial ➪ Encontrar la solución (o soluciones) a un problema concreto y acotado:
  - Aunque haya varias soluciones, estas suelen ser una minoría de entre el total de posibles individuos.
- La mayor parte de los problemas no son co-evolutivos ➪ Una solución es buena o mala por sí misma.
- ¡Queremos la solución lo antes posible y deseamos convergencia! ➪ Mucha mayor presión evolutiva en la evolución artificial.
- La mutación puede llegar a convertirse en el operador genético fundamental (se pierde la analogía con la naturaleza)

Evolución natural vs. artificial

- Conclusión:
  - La analogía con la naturaleza es interesante y permite inspirarse en algo que funciona muy bien a la hora de buscar cómo solucionar un problema, PERO la evolución artificial NO comparte objetivo con la natural ➪ No todo lo que vale para una, vale para la otra.

Aplicación

- ¿Cuándo es útil un algoritmo evolutivo?
  - Cuando el espacio de búsqueda es desconocido y no hay una forma analítica o determinista de solucionar el problema en un tiempo razonable.
- ¿Por qué hay tantos métodos distintos de computación evolutiva?
  - *No free lunch theorem* (Wolpert y Macready, 1997)
  - Diversos balances entre exploración y explotación.
  - Falta de estudio matemático en muchos de los algoritmos evolutivos (muchas veces primero va la idea, después la aplicación, después el estudio).

Combinación con otras técnicas

- La fase de evaluación puede ser todo lo complicada y puede extenderse todo lo que se quiera ➪ Se pueden utilizar técnicas de búsqueda local / aprendizaje en la evaluación para:
  - Refinar soluciones con mayor precisión de lo que permitan los operadores genéticos, revirtiendo los cambios producidos por el aprendizaje en el genotipo (Lamarck).
  - Acelerar la búsqueda.
  - Evolucionar para aprender. Los cambios del aprendizaje no revierten en el genotipo, pero un individuo que aprenda rápidamente tiene ventaja evolutiva (efecto Baldwin)
  - Adaptarse a entornos cambiantes en *embodied evolution*.

UNIVERSIDADE DA CORUÑA

facultade de informática da coruña

ROBÓTICA EVOLUTIVA

Motivación

- Obtención de controladores de forma manual:
  - Labor muy compleja para comportamientos no triviales.
  - Visión del entorno muy distinta por parte del humano y del robot.
  - Dificultad para establecer estrategia óptima.
- Posibilidades para obtenerlos automáticamente:
  - Si el controlador del robot es una RNA: entrenamiento.
    - Normalmente, es necesario conocer conjuntos de entrada-salida.
    - Es específico para una arquitectura o topología de red.
    - Optimización por movimiento de un solo punto en el espacio de búsqueda.
  - Algoritmos evolutivos.
    - Sólo es necesario establecer un orden en la calidad de los individuos.
    - Optimización por movimiento de múltiples puntos en el espacio de búsqueda.
    - Si el controlador es una RNA: cualquier arquitectura de red puede codificarse en forma de cromosoma.
    - ¡Se puede obtener automáticamente también la morfología!.

Ejemplos

Tipo de algoritmo evolutivo	Autores	Características básicas	Aplicación en robótica
Algoritmos genéticos	Holland (1975)	Cruce operador fundamental. Mutación para introducir variedad.	Miglino (1995), Lund (1995), Nolfi (1994), Floreano (1994), Mondada (1994)
Estrategias evolutivas	Rechenberg y Schwefel (1971)	Solo mutación.	Salomon (1997), Meeden (1996), Harvey (1993)
Programación genética	Koza (1989)	Se evolucionan programas.	Dain (1998), Reynolds (1994), Meyer (1995)

Codificación

- Codificación directa del controlador en el genotipo ➪ La traslación del genotipo al fenotipo es inmediata.

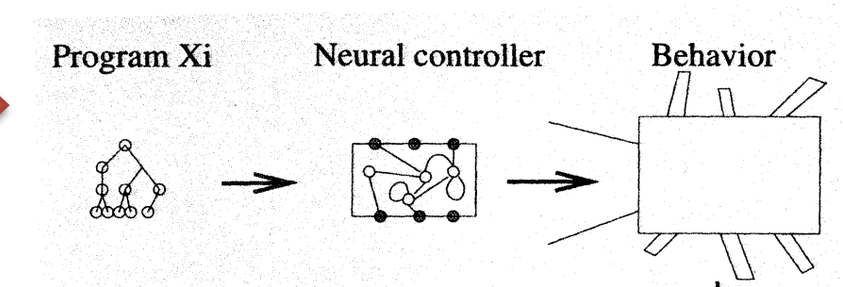
- Desarrollo del fenotipo ➪ El genotipo codifica el desarrollo del fenotipo.

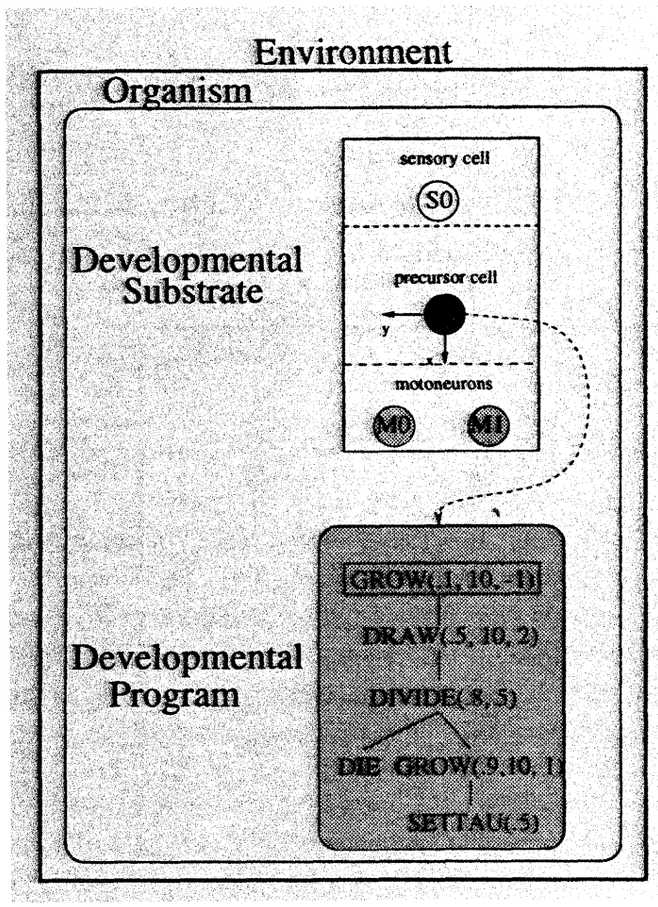
"Nature has invented three automatic design procedures, namely those of evolution, development and learning"

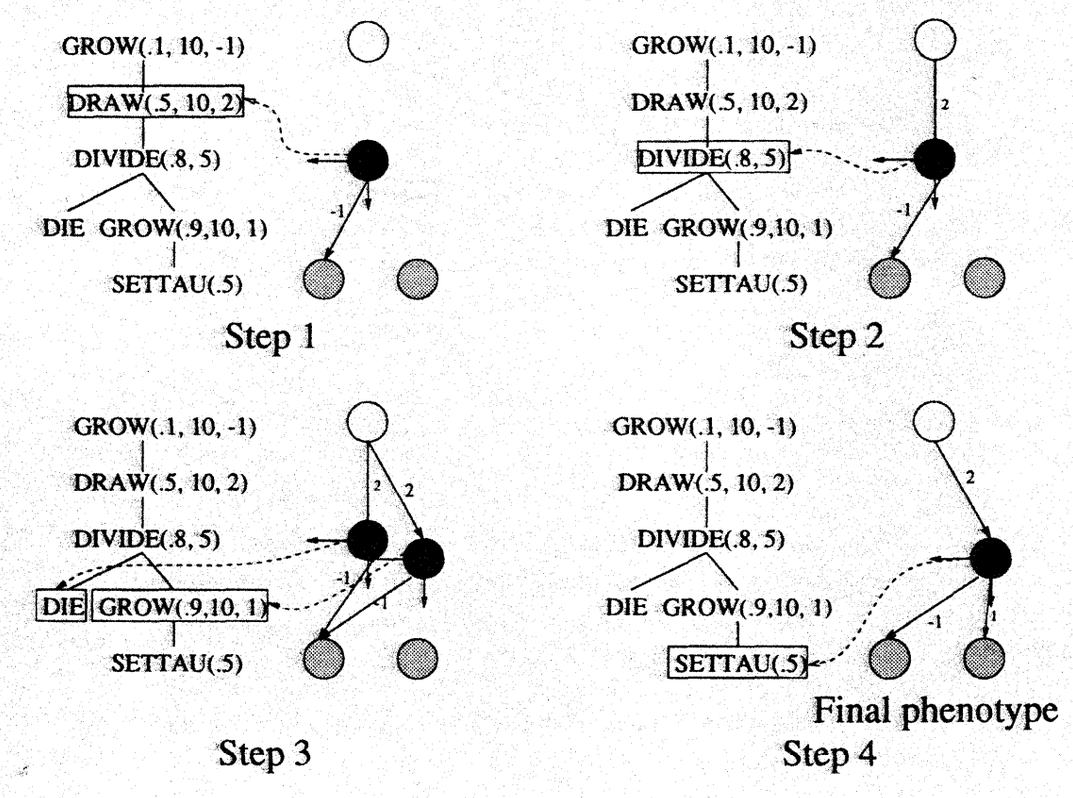
[Kodjabachian & Meyer 98]

Codificación

Estados en la evaluación de calidad partiendo de un programa que desarrolla una RNA







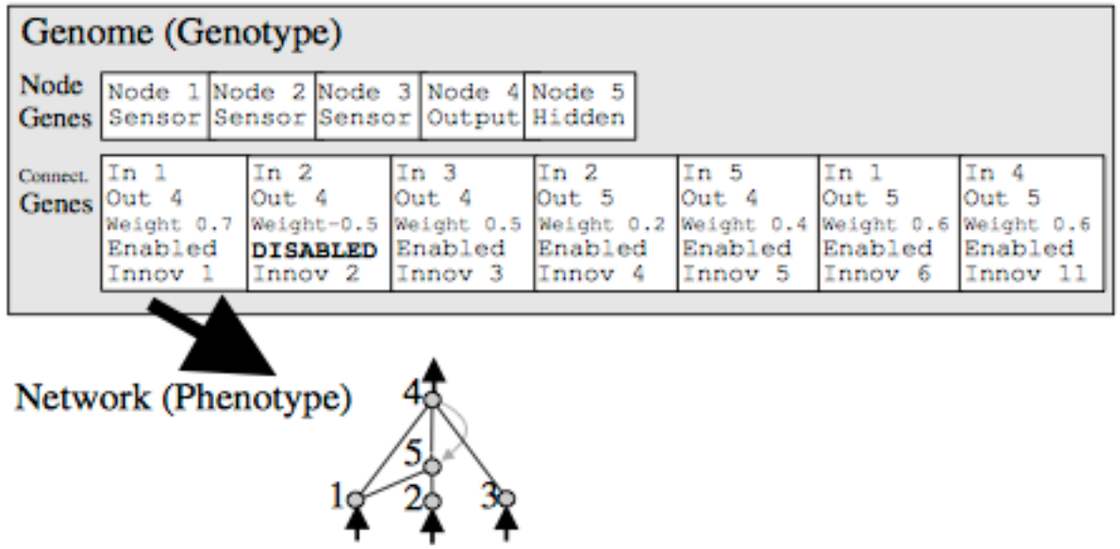
Esquema de desarrollo usado por Kodjabachian y Meyer

Codificación

- NeuroEvolution of Augmenting Topologies (NEAT):
  - Los genes representan conexiones entre dos neuronas numeradas. Cada uno tiene:
    - Neurona de entrada.
    - Neurona de salida.
    - Peso.
    - Habilitada / deshabilitada.
    - *Innovation number* (funciona como id, cada vez que aparece un gen nuevo, éste recibe uno).
- Los cromosomas son de longitud variable.
- La población inicial es uniforme con individuos que representan una red con únicamente neuronas de entrada y salida y conexiones de las primeras a las segundas.
- La población se divide en especies formadas por individuos con topologías similares.

Codificación

- NeuroEvolution of Augmenting Topologies (NEAT)



Ejemplo de mapeado genotipo a fenotipo

Codificación

- NeuroEvolution of Augmenting Topologies (NEAT)
  - Selección:
    - La calidad de cada individuo se divide por el número de individuos de su especie.
    - El nuevo número de individuos de una especie se calcula en base a la calidad de sus individuos frente a la media de la población.

$$N_j^{'} = \frac{\sum_{i=1}^{N_j} f_{ij}}{\overline{f}}$$

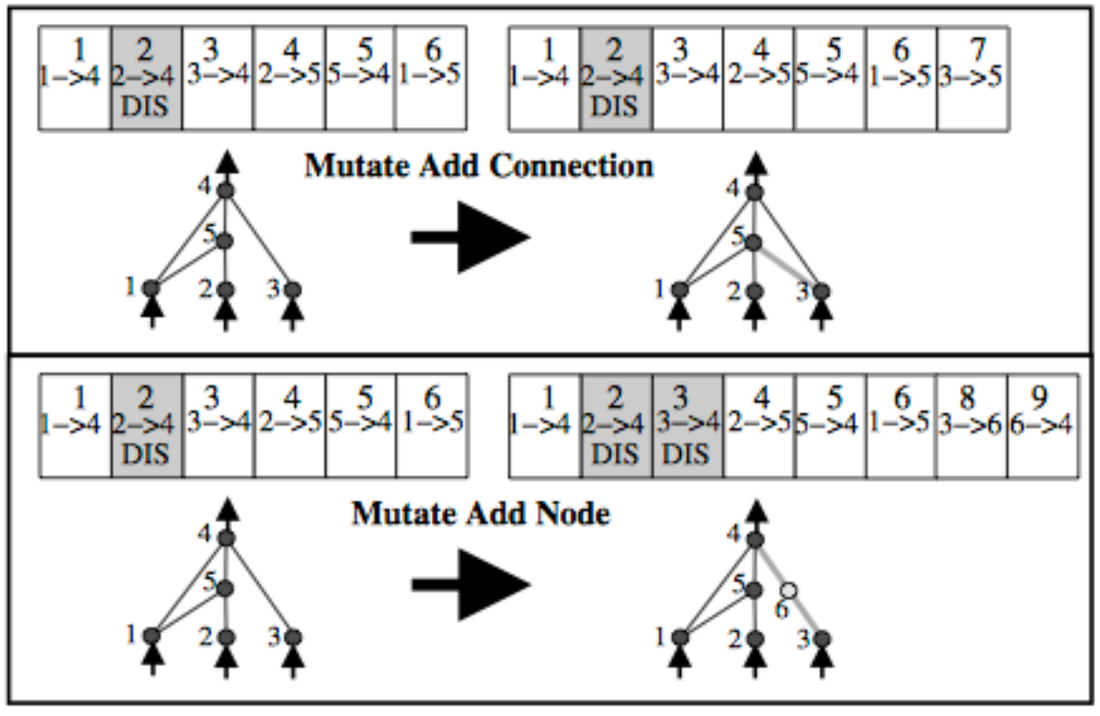
    - Los *r*% mejores de cada especie se reproducen aleatoriamente hasta tener $N_j^{'}$ descendientes que reemplazan a los padres.
    - Si la calidad de la población no mejora en 20 generaciones, solo se reproducen las 2 mejores especies.

Codificación

- NeuroEvolution of Augmenting Topologies (NEAT)
  - Reproducción:
    - Mutación:
      - Cambio de un peso.
      - Cambio de la morfología:
        - Añade una conexión entre dos neuronas existentes (+ 1 gen).
        - Añade una neurona que reemplaza a una conexión (+ 2 genes y 1 gen a *disabled*).
  - Cruce:
    - Los genes con mismo identificador en ambos padres ➪ se selecciona uno aleatoriamente (aunque el identificador sea el mismo el peso puede ser diferente).
    - Los genes con identificador único ➪ se cogen todos los del padre con mayor calidad o los de ambos si tienen la misma.

Codificación

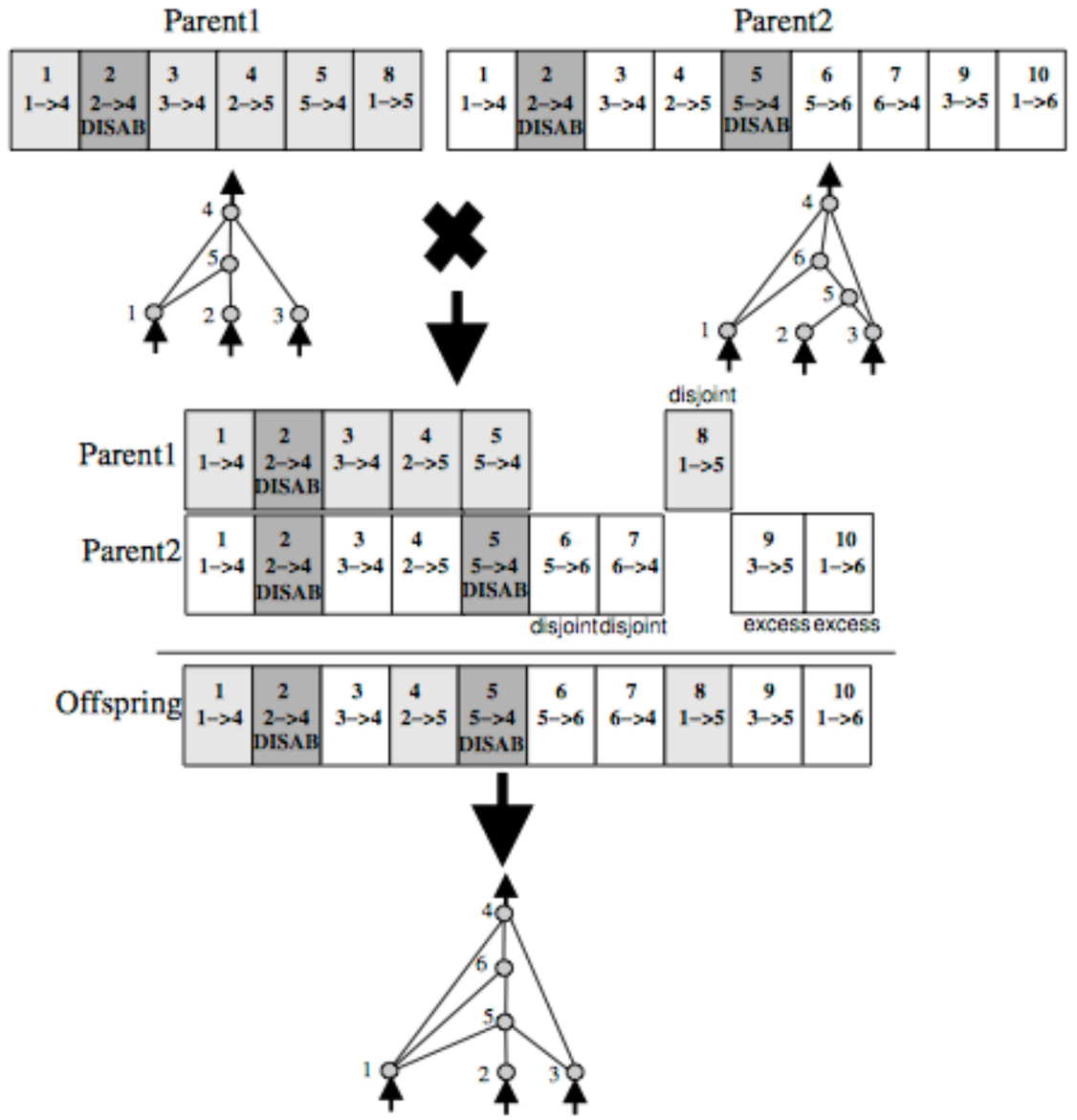
- NeuroEvolution of Augmenting Topologies (NEAT)



Ejemplos de mutación

Codificación

- NeuroEvolution of Augmenting Topologies (NEAT)



Ejemplo de cruce con padres de igual calidad

Codificación

- NeuroEvolution of Augmenting Topologies (NEAT)
  - Reemplazo:
    - Dados dos individuos:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W}$$

      - E = número de genes que solo posee uno de ellos
      - D = número de genes disjuntos
      - N = número de genes del cromosoma más largo
      - $\bar{W}$ = media de las diferencias de los pesos correspondientes a los genes comunes
      - c_1, c_2, c_3 = constantes
- Cada individuo nuevo es comparado con uno seleccionado al azar de cada especie, de forma que si $\delta < \delta_t$ entonces es asignado a dicha especie (la primera que lo verifique).

Evaluación

- Evaluación en entornos reales:
  - Tiempo de evaluación muy elevado.
  - Dificultades en el proceso de evaluación si el robot es incapaz de salir de una situación de atasco.
    - Riesgo de daños físicos en el robot, especialmente en las primeras generaciones.
- Evaluación en entornos simulados:
  - Mayor flexibilidad en la evaluación (entornos y funciones de calidad).
  - Problema de la transferencia de los controladores al robot real.
- Evaluación mixta:
  - Mismos inconvenientes que evaluación en entornos reales aunque atenuados.

Evaluación

- Jakobi (1995) plantea la siguiente metodología para transferir controladores obtenidos en entornos simulados a entornos reales:
  - Definir con precisión el comportamiento.
  - Identificar el “conjunto base” de características del entorno real.
  - Todos los aspectos del conjunto base deben ser variados aleatoriamente dentro del rango para el cual queremos que el comportamiento funcione correctamente.
  - Todos los aspectos de la implementación que no formen parte del conjunto base deben ser variados aleatoriamente entre evaluación y evaluación.
  - Definir un modelo de las interacciones de los miembros del conjunto base con el robot.
  - Definir un test de calidad adecuado.

Evaluación

- Es necesario introducir ruido en la simulación para cumplir los criterios de Jakobi y eliminar las discrepancias entre la realidad y la simulación:
  - Ruido aditivo de media cero en valores de sensores y actuadores durante la evaluación.
  - Perturbaciones persistentes en algunos parámetros del entorno ➪ ruido sistemático:
    - Ruido de generalización. Hacer los comportamientos más robustos ante cambios en la naturaleza del entorno o en la configuración del robot.
    - Ruido sistémico. El controlador debe funcionar adecuadamente en distintas unidades del mismo robot.
    - Ruido temporal. Tolerar cambios en el tiempo transcurrido entre eventos temporales.

Evaluación

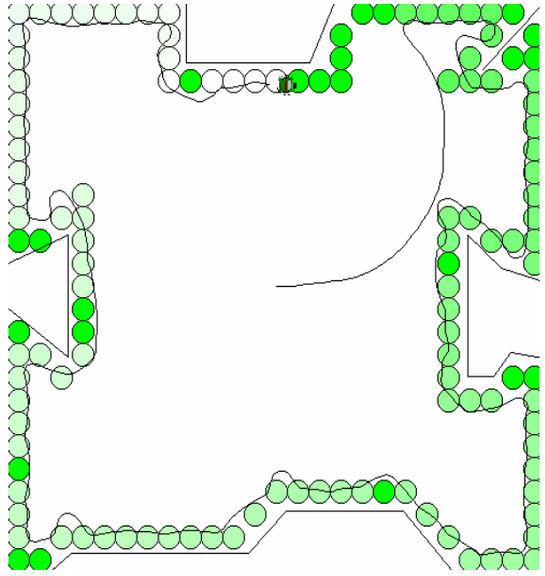
- Entornos de desarrollo para robots con simuladores 3D:
  - Webots => http://www.cyberbotics.com
  - Microsoft Robotics Developer Studio => http://msdn.microsoft.com/en-us/library/bb648760.aspx
  - Gazebo (incluído en ROS) => http://gazebosim.org
  - Robot Virtual Worlds (acompaña a RobotC) => http://www.robotvirtualworlds.com

Evaluación

- Cálculo de la calidad de cada individuo:
  - Perspectiva local. El robot recibe una puntuación en cada movimiento en función de lo bien que lo ha realizado y tomando al final como su calidad la suma de las puntuaciones obtenidas en cada paso. Problemas:
    - Determinar la calidad en cada movimiento.
    - Elevada intervención del diseñador en el proceso.
  - Perspectiva global. Al finalizar el periodo de evaluación se le asigna una calidad en función de su comportamiento global en el conjunto del periodo de evaluación. Problemas:
    - Cómo calcular esa calidad.
    - El proceso evolutivo puede explotar debilidades en la definición de la función de calidad.

Evaluación

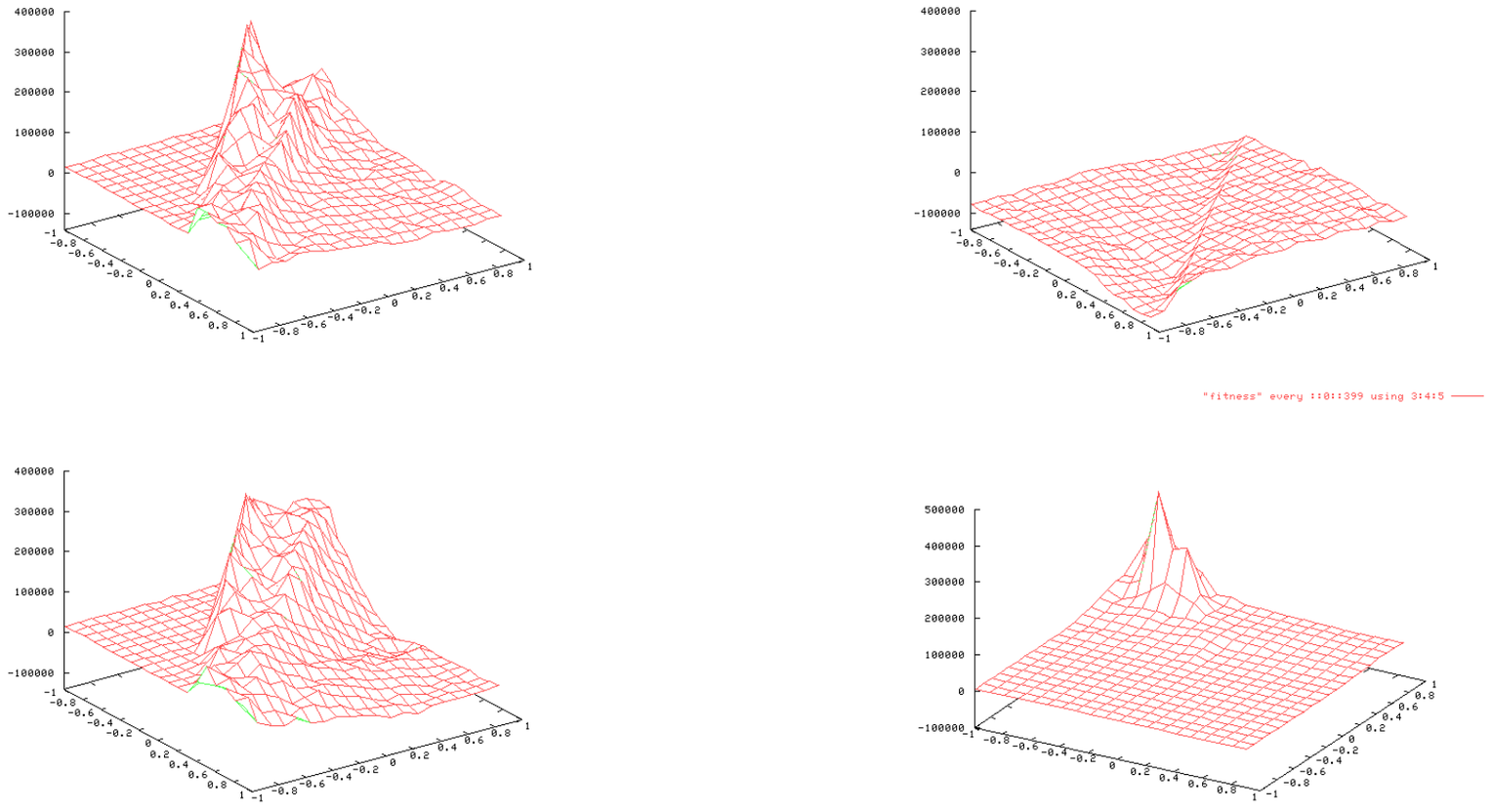
- Cálculo de la calidad de cada individuo:
  - Modelo energético. El individuo, a medida que se desenvuelve por el entorno, puede perder o ganar energía según sus acciones, y su calidad viene determinada únicamente por el nivel de energía al finalizar su periodo de evaluación.



Ej: Función de calidad utilizando modelo energético para seguir paredes con Rug Warrior.

Evaluación

- *Epistasis*: La bondad de un gen no depende únicamente de su valor, si no del valor de otros genes del mismo cromosoma.



Cambio en la calidad (eje *z*) variando 3 genes (eje *x*, eje *y*, y tiempo) en un cromosoma de 70 genes

Evaluación

- La evaluación es muy costosa pese a usar simulación ➪ poblaciones pequeñas ➪ posibles convergencias prematuras.

- Dividir el global de la población en subpoblaciones y limitar el intercambio de material genético entre ellas:
  - Reduce el riesgo de caer en máximos locales.
  - Favorece la parelelización y distribución del algoritmo.
  - Los individuos de una subpoblación sólo pueden reproducirse con individuos de su misma subpoblación.
  - El intercambio de material genético entre las subpoblaciones se produce mediante la migración, proceso en el cual el mejor o mejores individuos de cada subpoblación son copiados a otras subpoblaciones.

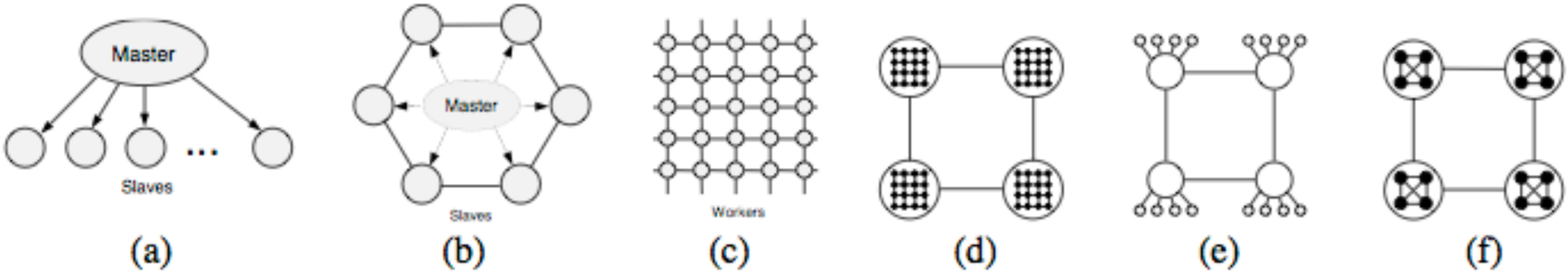
Evaluación

- Paralelización:
  - Algoritmos genéticos de micrograno (mgGA):
    - Un proceso maestro envía los individuos a los procesos esclavos.
    - Los esclavos evalúan los individuos.
    - El maestro recibe los resultados de dicha evaluación y realiza todos los demás pasos.
  - Algoritmos genéticos de grano fino (fgGA):
    - La población se divide en varias subpoblaciones que se disponen en una red n-dimensional y las fronteras entre las poblaciones se diseñan de forma que se solapen algunos individuos, que pertenecerán así a más de una población.
    - La reproducción sólo puede producirse entre individuos de una misma población, pero, al haber individuos que forman parte de más de una, se produce una migración implícita.

Evaluación

- Paralelización:
  - Algoritmos genéticos de grano grueso (cgGA):
    - La población se divide en subpoblaciones.
    - No hay individuos comunes.
    - El intercambio de material genético entre poblaciones se produce mediante migraciones explícitas.
    - Las migraciones son periódicas, tienen lugar cada “x” generaciones.
    - Variantes:
      - Islas aisladas. No hay migración.
      - Islas síncronas. La migración se produce siempre en las mismas generaciones en todas las poblaciones, es decir, éstas se sincronizan en el momento de realizar la migración.
      - Islas asíncronas. No se requiere que las islas implicadas estén en el mismo número de generación, con lo que es más adecuado para entornos distribuidos con procesadores heterogéneos.

Evaluación



A. mgGA

B. cgGA

C. fgGA

D. cgGA + fgGA

E. cgGA + mgGA

F. cgGA + cgGA