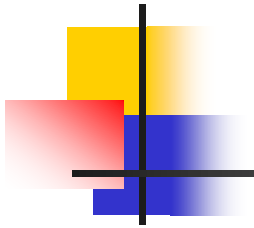
Universidad
Carlos III de Madrid

Programación Automática

MÁSTER EN CIENCIA Y TECNOLOGÍA INFORMÁTICA

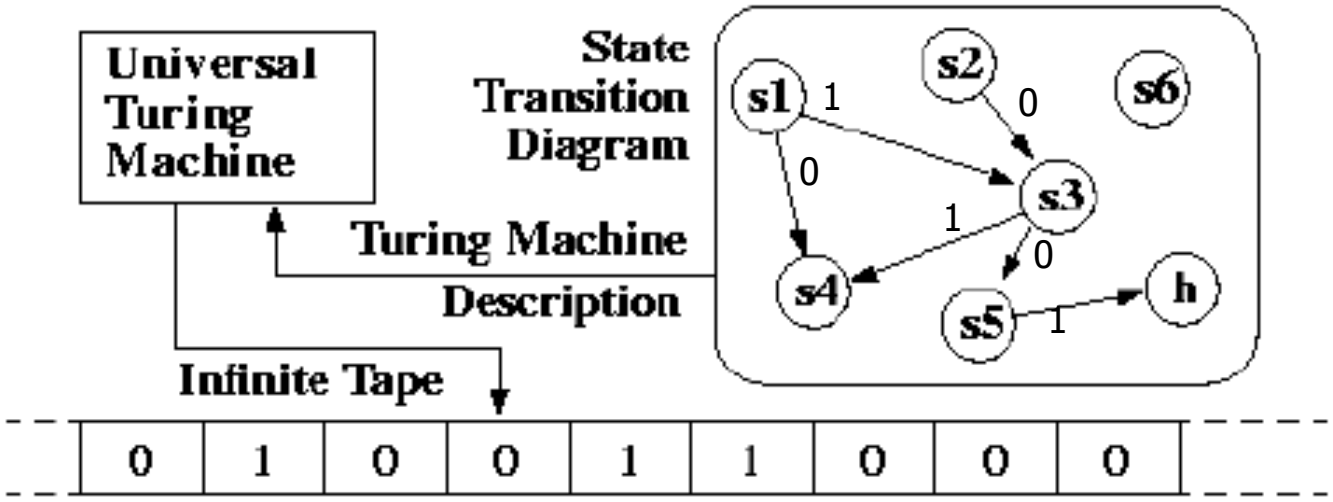
Ricardo Aler Mur, David Quintana Montero



EVOLUCIÓN DE AUTÓMATAS DE ESTADO FINITO (FSA) Y SISTEMAS CLASIFICADORES (LCS)

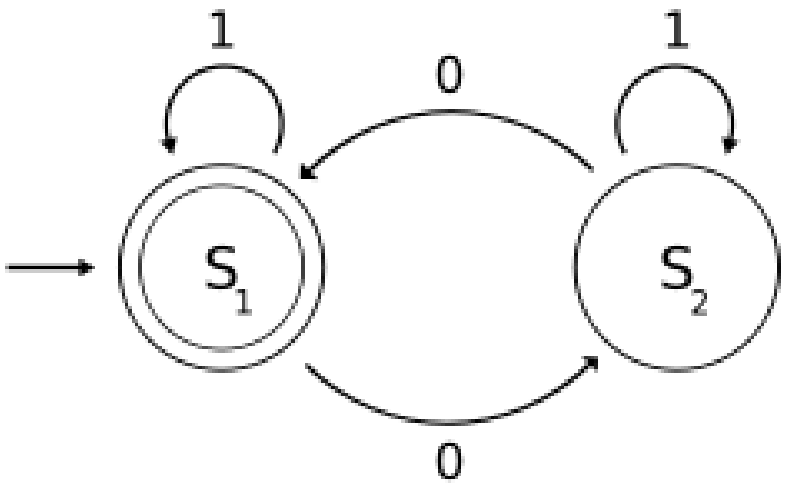
Máquina de Turing

- Es un modelo abstracto de la computación

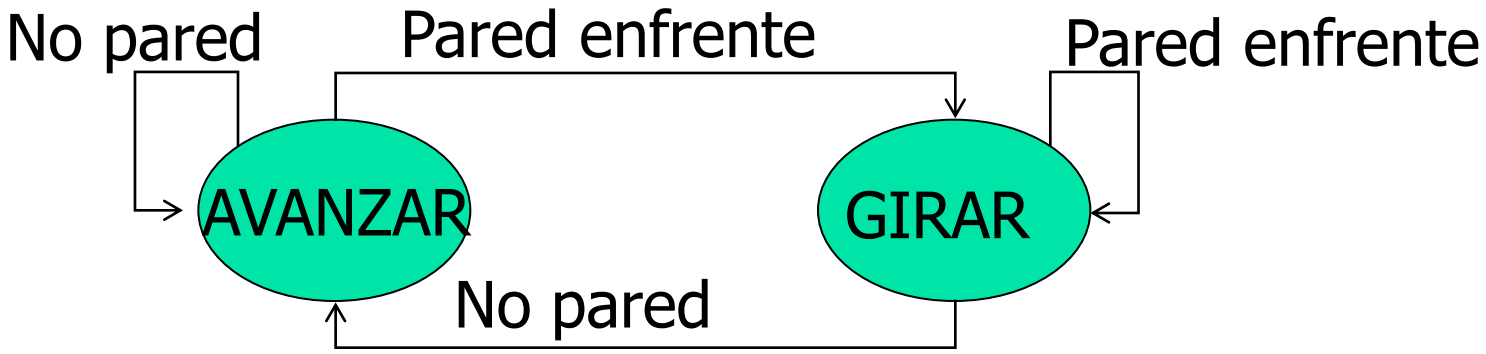


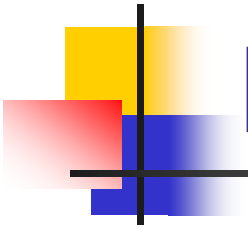
Autómatas deterministas de estado finito (FSA)

- Son modelos de computación restringida
- Reconocen lenguajes regulares:
  - $L=(1^*01^*)^*$
    - 1; 1111 …; 111101111; 1110111101111 …



FSA para control de robots





Evolutionary Programming

- D.B. Fogel. An introduction to simulated evolutionary optimization. IEEE Transactions on Neural Networks 5(1):3-14 (January 1994)

Evolutionary Programming: un algoritmo evolutivo con mutación para FSAs:

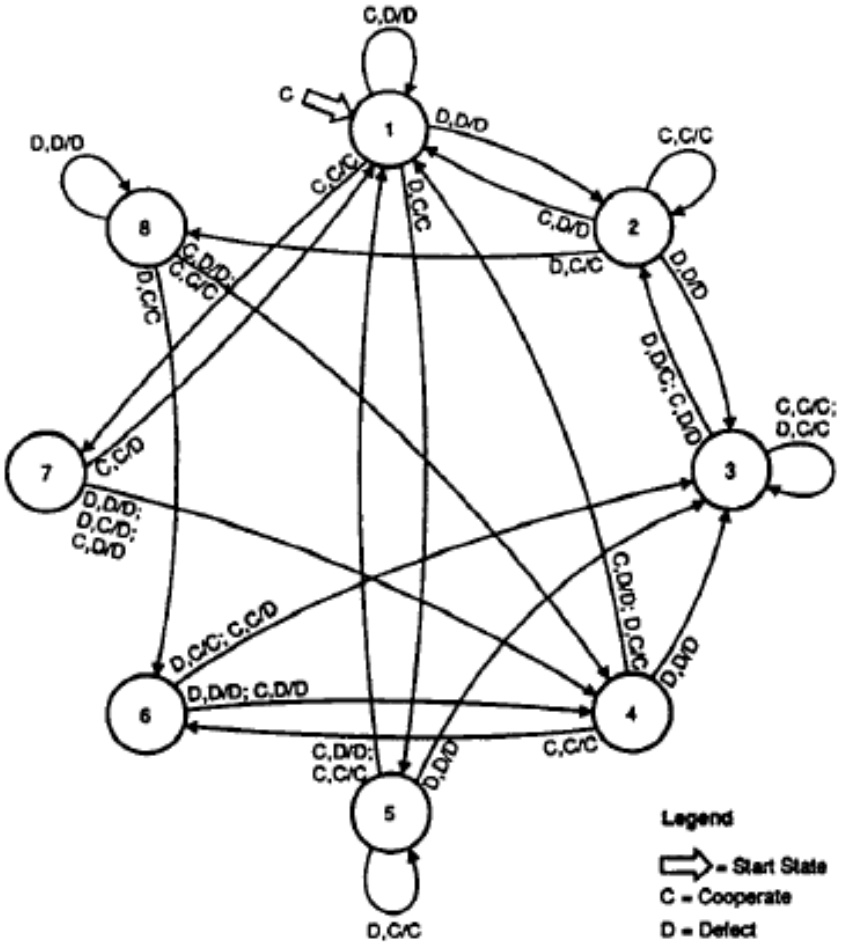
- Añadir / Quitar / Modificar estados
- Añadir / Quitar / Modificar transiciones

- Mutación
  - Opera sobre un único AF
  - Una entre las siguientes
    - a new random transition is created,
    - random transition is deleted,
    - a new state is created (with minimum incoming and outgoing random transitions); in addition, one new transition leading to this state from another state is randomly generated,
    - a random state is deleted as well as all its incident transitions,
    - a random transition is modified: (one of its parts `new_state`, `message_type`, `msg_to_send_out`, `msg_to_send_in` is replaced by an allowed random value),
    - a completely random individual is produced (this operator changes all FSAs),
    - a random transaction is split in two and new state is created in the middle,
    - the initial state number is changed.

Dilema del prisionero

- La policía arresta a dos sospechosos. No hay pruebas suficientes para condenarlos y, tras haberlos separado, los visita a cada uno y les ofrece el mismo trato.
  - Si uno confiesa y su cómplice no, el cómplice será condenado a la pena total, diez años, y el primero será liberado.
  - Si uno calla y el cómplice confiesa, el primero recibirá diez años y será el cómplice quien salga libre.
  - Si ambos confiesan, ambos serán condenados a seis años.
  - Si ambos lo niegan, todo lo que podrán hacer será encerrarlos durante seis meses por un cargo menor.
- Vamos a suponer que ambos prisioneros son completamente egoístas y su única meta es reducir su propia estancia en la cárcel:
  - Si el otro confiesa, lo mejor que puedo hacer es confesar (porque si callo me caeran 10 años)
  - Si el otro calla, lo mejor que puedo hacer es confesar (porque así saldré libre)
- Sin embargo, la solución racional egoísta lleva al peor resultado para ambos

Dilema del prisionero iterado

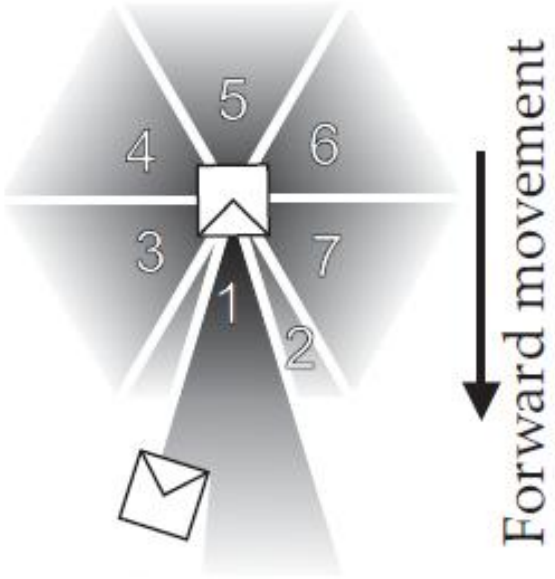


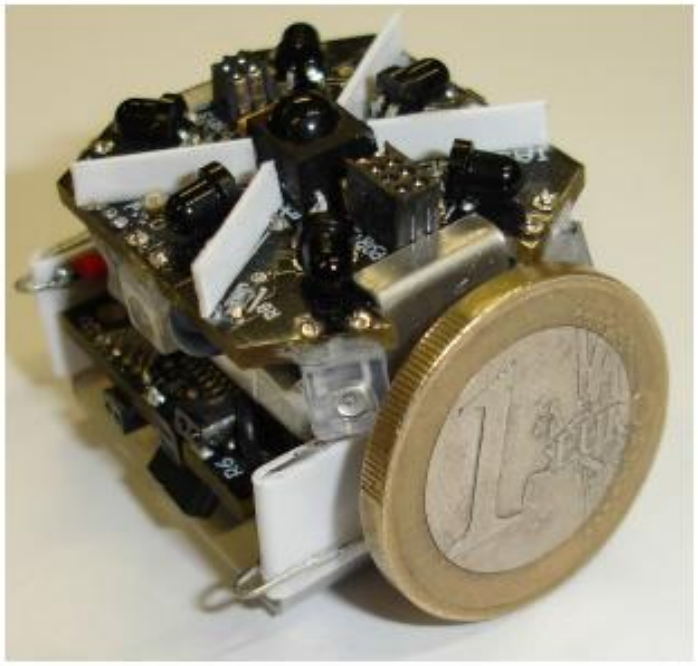
Evolutionary robotics

- König, L., Mostaghim, S., & Schmeck, H. (2009). Decentralized evolution of robotic behavior using finite state machines. *International Journal of Intelligent Computing and Cybernetics*, *2*(4), 695-723.

Jasmine IIIp robot

- Se trata de enjambres de estos robots





Condiciones y acciones. Autómatas

Example conditions are: $true$; $false$; $h_1 < h_2$; $20 > h_7$; $(h_1 \approx h_2\ OR\ h_2 \not\approx 120)$.

- $(Move, X)$: drive forward for at most X mm.
- $(TurnLeft, X)$: turn left for at most X degrees.
- $(TurnRight, X)$: turn right for at most X degrees.
- $(Stop, X)$: stop the current action and do nothing (X is ignored).
- $(Idle, X)$: keep performing the current action (X is ignored).

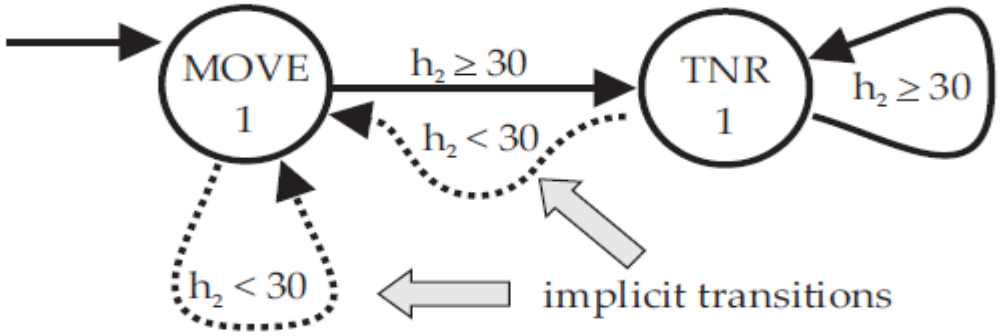


Fig. 2. An example MARB with two states. The dotted transitions are inserted implicitly by the model. The states perform the operations $(Move, 1)$ and $(TurnRight, 1) \in Op$, respectively. The automaton represents a simple collision avoiding behavior moving forward as long as no obstacle is ahead ($h_2 < 30$) and turning right if an obstacle is ahead ($h_2 \geq 30$).

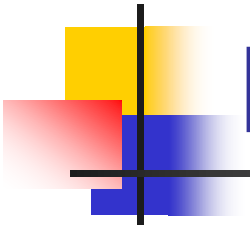
Fitness

- Cada cierto tiempo, se calcula un fitness snapshot
- Hay fitness para:
  - evitar colisiones (CollAvoid)
  - Pasar puertas (GatePass)

$$f := f + snap_X(t)\ (X \in \{CollAvoid, GatePass\}, t \in \mathbb{N}).$$

- Como el autómta sufre mutaciones, es necesario ir disminuyendo las fitnesses antiguas (fitness evaporation)

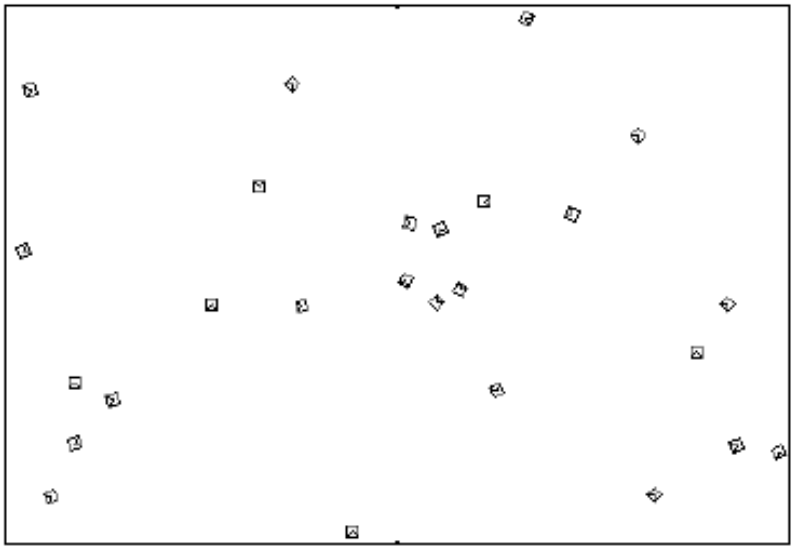
$$f := f / E.$$



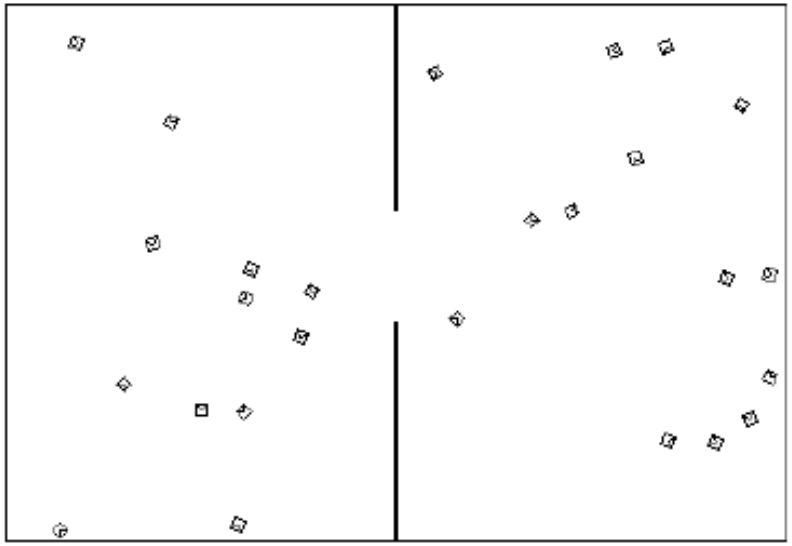
Mutación

- Insertar o quitar estados
- Insertar o quitar transiciones
- Cambiar etiquetas de los nodos
  - La mayoría de las operaciones sólo quitan nodos o transiciones si no afectan al comportamiento
- Hardening: para aquellos estados y transiciones que se comprueba que tienen una buena influencia en el comportamiento final, se hace menos probable que se puedan quitar

Escenarios



(a)



(b)

Fig. 3. Experimental field with 26 robots for Collision Avoidance (a) and for Gate Passing (b).

Fitness para evitar colisiones

Algorithm 3.1: Computation of fitness snapshot $snap_{CollAvoid}$ for Collision Avoidance.

input : Current operation $op \in Op$ of a robot R at a time step t; number of collisions $|Coll|$ of R since last snapshot before t.
output: Fitness snapshot for R at time step t.

int $snap := 0$;
if $op = (Move, X), X \in B$ **then**
  $snap := snap + 1$;
end

$snap := snap - 3 \cdot |Coll|$;

return $snap$;

Fitness para pasar por la puerta

Algorithm 3.2: Computation of fitness snapshot $snap_{GatePass}$ for Gate Passing.

input : Current operation $op \in Op$ of a robot R at a time step t; number of collisions $|Coll|$ of R since last snapshot before t; Boolean value $Gate$ indicating if the gate was passed since the last snapshot before t.

output: Fitness snapshot for R at time step t.

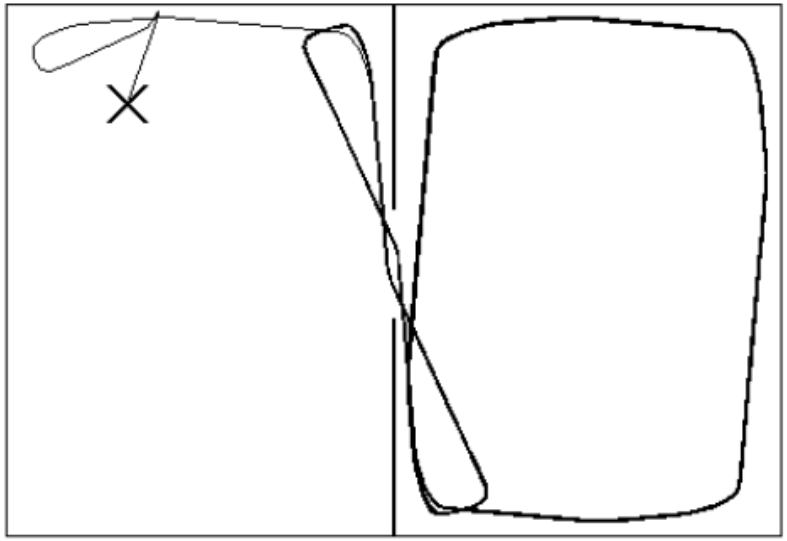
int $snap := snap_{CollAvoid}(op, |Coll|)$;

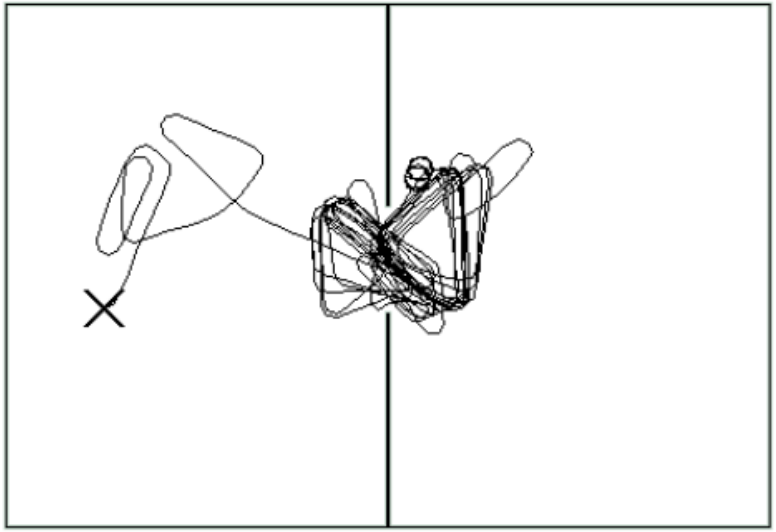
if $Gate$ **then**
  $snap := snap + 10$;
end

return $snap$;

Resultados

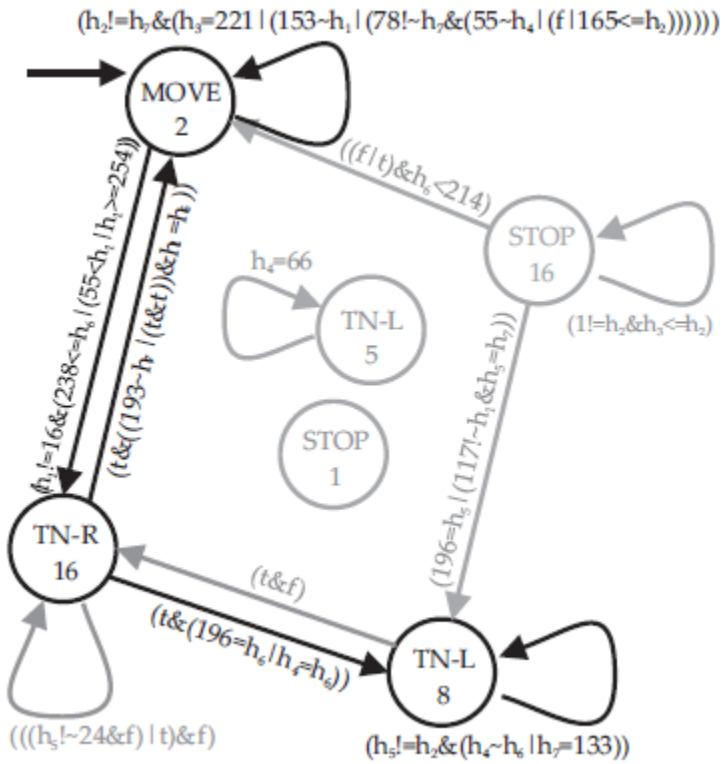
- Sobre un 90% de ejecuciones exitosas con un 75% de robots exitosos

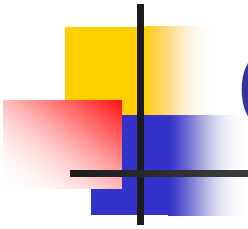




Ejemplo de autómata evolucionado

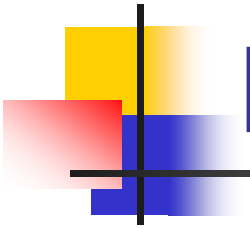
- Difícil de analizar





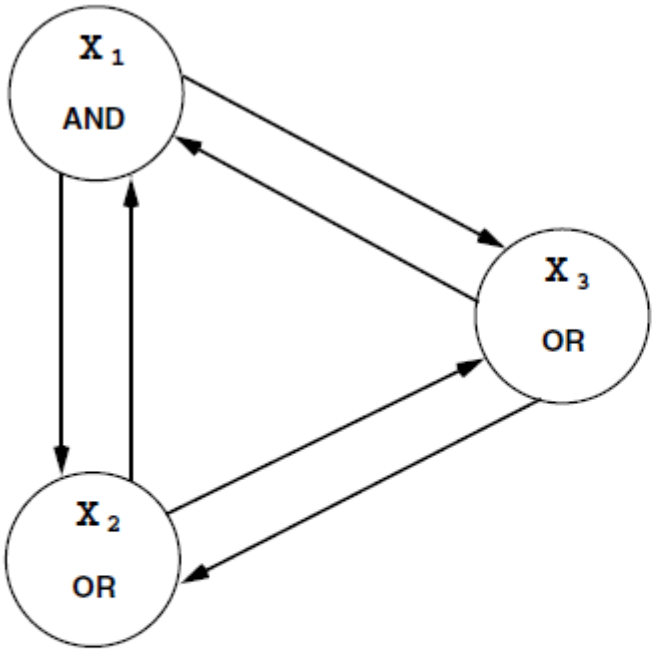
Conclusiones

- Se consiguen evolucionar autómatas, aunque en comportamientos sencillos
- Las operaciones de mutación (y cruce) son complicadas

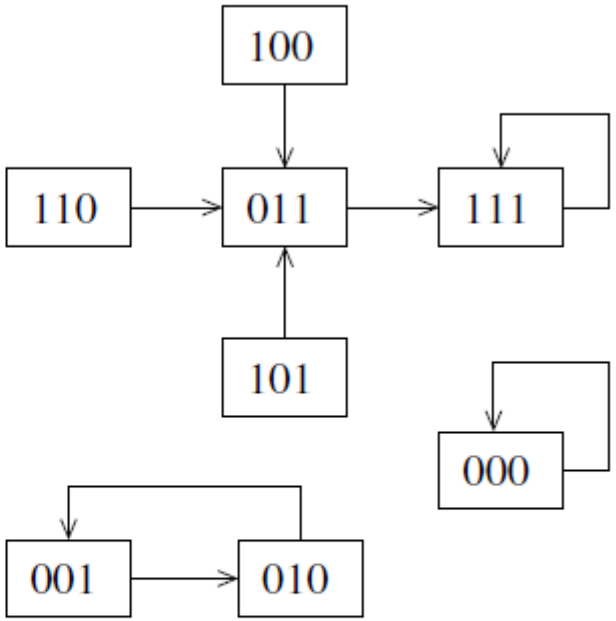


Boolean Network Robotics

- A. Roli, M. Manfroni, C. Pinciroli, and M. Birattari. On the design of boolean network robots. Lecture Notes in Computer Science, pages 43–52, 2011.
- Lorenzo Garattoni. Finite State Automata Synthesis in Boolean Network Robotics. 2012



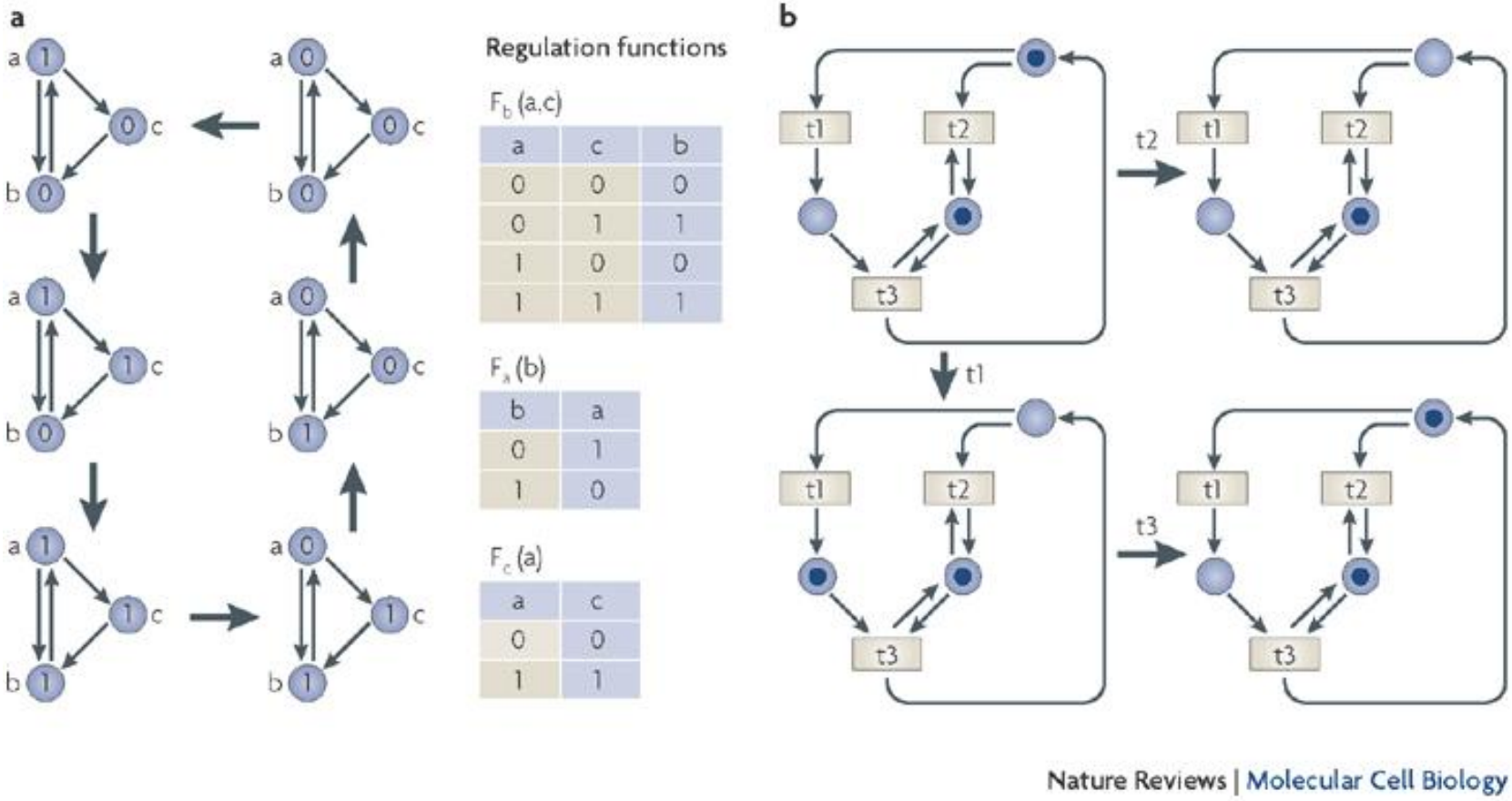
(a) A BN with three nodes



(b) State space

Fig. 1. An example of a BN with three nodes (a) and its corresponding state space under synchronous and deterministic update (b). The network has three attractors: two fixed points, $(0, 0, 0)$ and $(1, 1, 1)$, and a cycle of period 2, $\{(0, 0, 1), (0, 1, 0)\}$.

Evolución de una BN



BN robotics

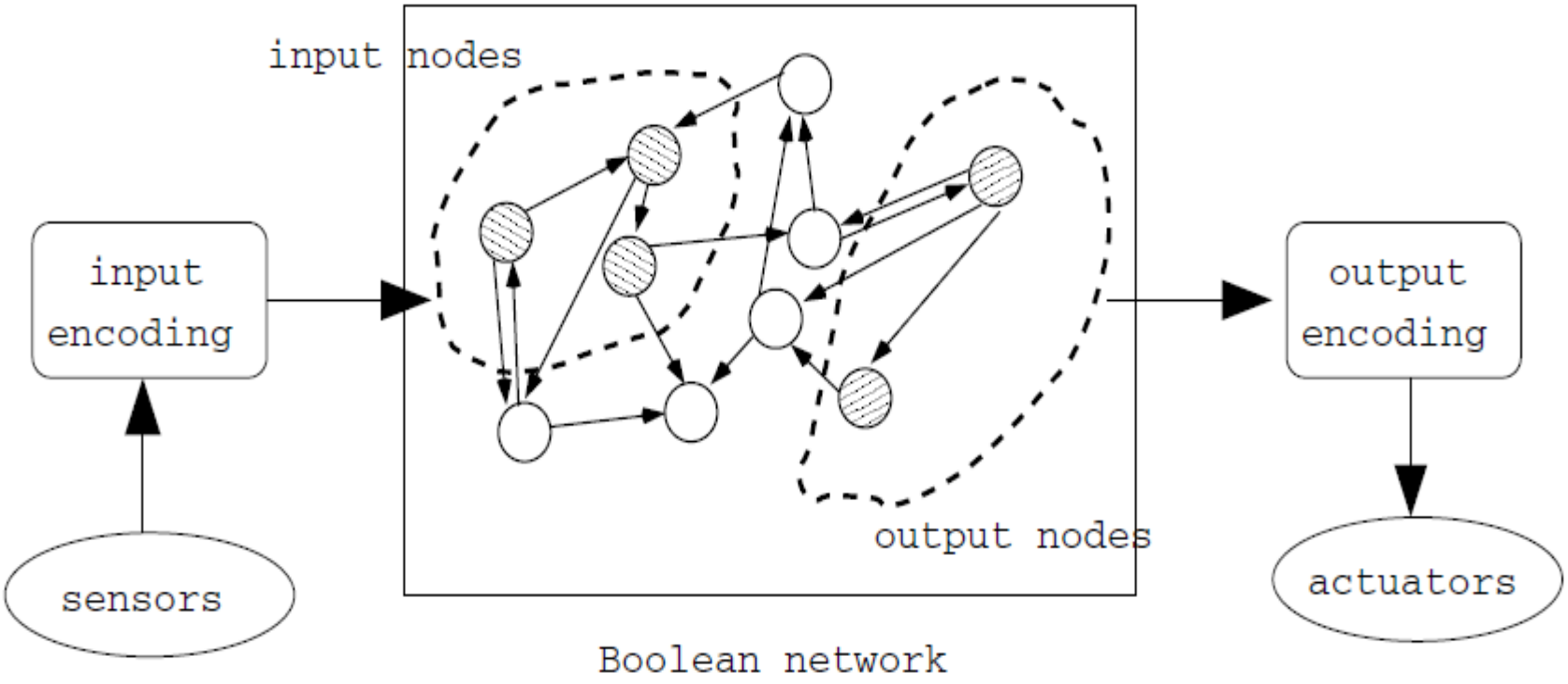


Fig. 2. The coupling between BN and robot

Optimización de una BN

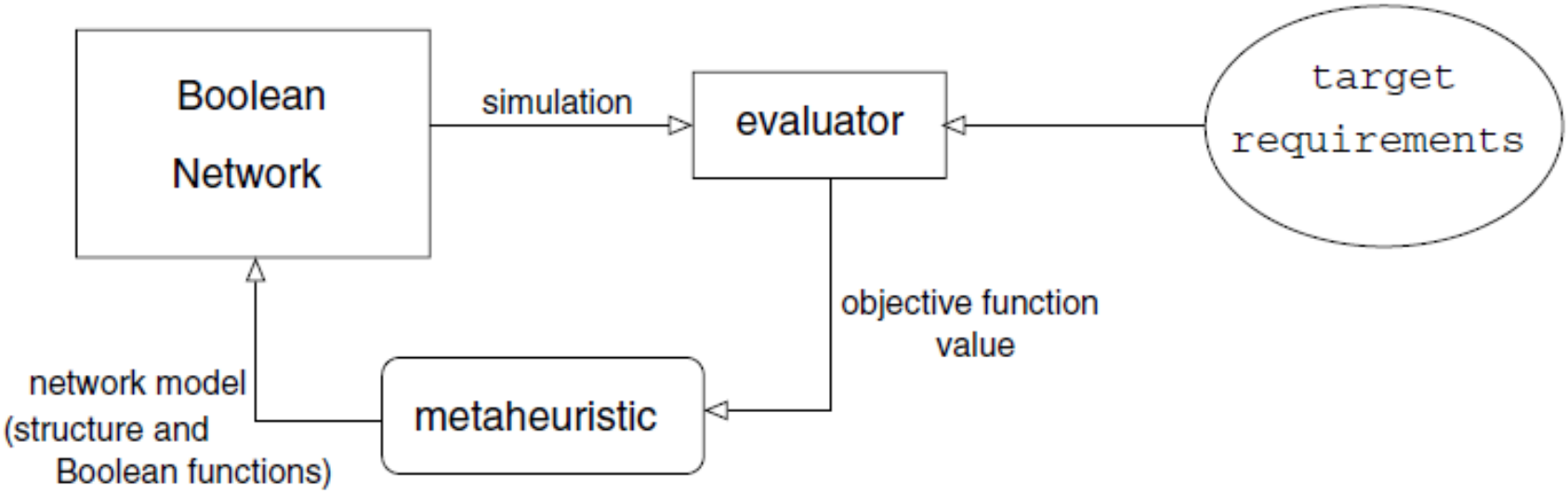


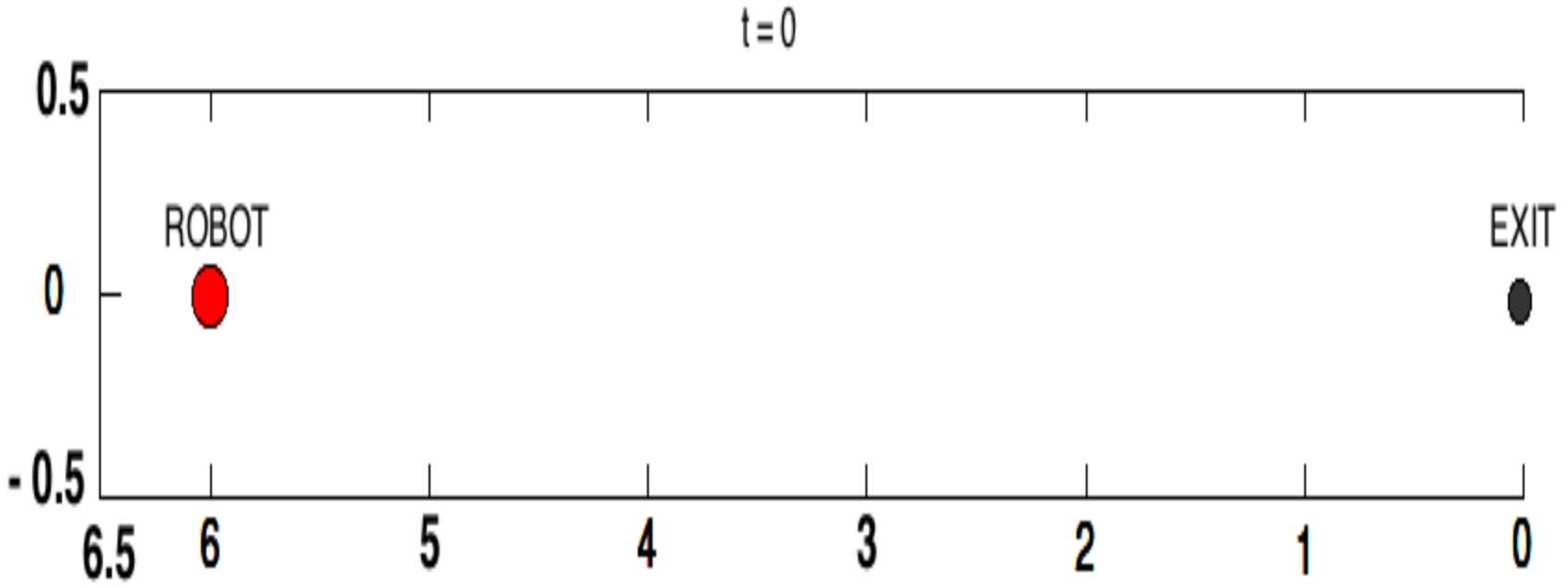
Fig. 3. BN design by metaheuristics

Robot e-puck

- Simulador ARGoS
- 8 sensores infrarrojos alrededor del cuerpo
- 3 sensores infrarrojos al frente apuntando al suelo (pueden ver el color del suelo en tono de grises)
- Actuadores: 2 ruedas y 8 LED rojos

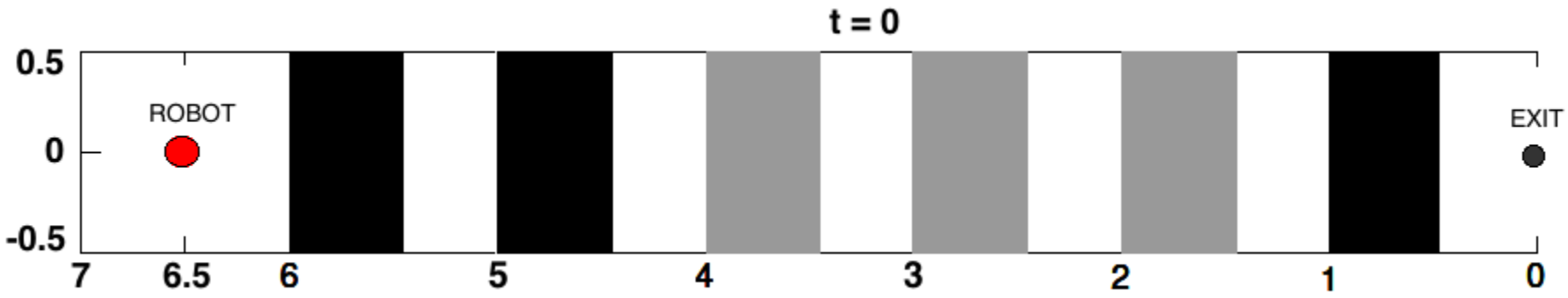
Tarea 1

- Navegación por el pasillo sin chocar, en menos de 120s



Tarea 2

- Reconocimiento de secuencias



Estados visitados

- Redes con 20 o 30 nodos. Número de estados = 2^{20} o 2^{30}
- Se visitan muy pocos estados de entre los posibles

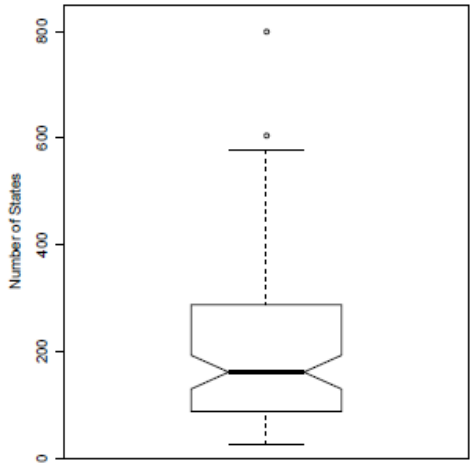
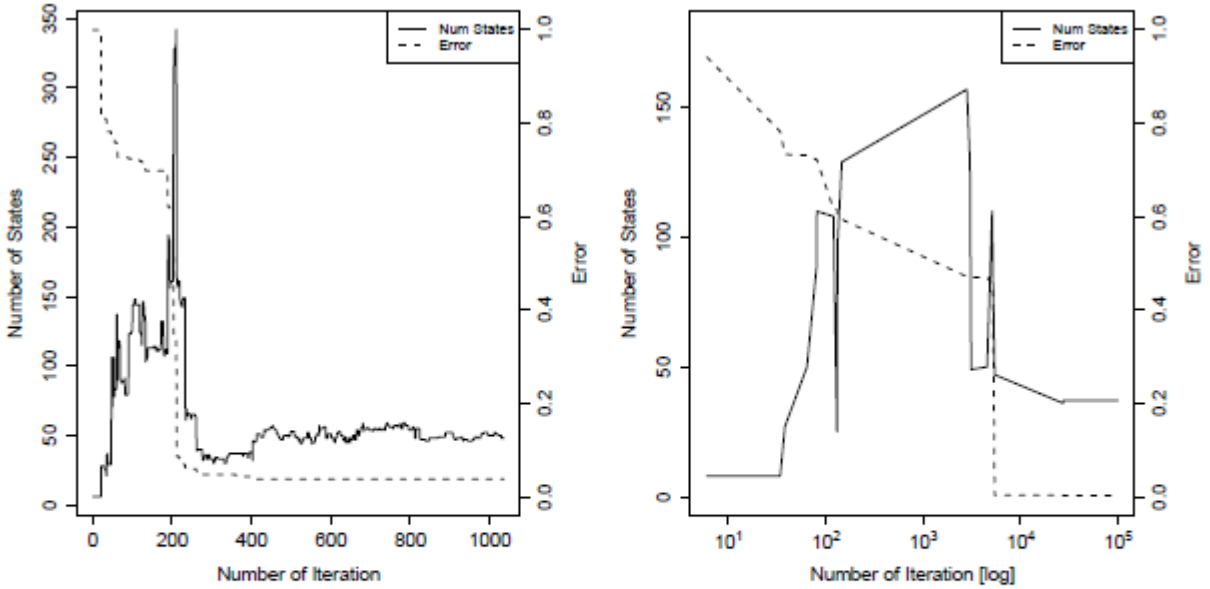


Figure 5: Number of visited states in final networks. Corridor navigation test case

Disminución del número de estados durante la búsqueda



(a) Corridor navigation (b) Sequence recognition

Figure 6: States number and error function trends. Corridor navigation (a), sequence recognition (b).

Unos pocos nodos son muy visitados

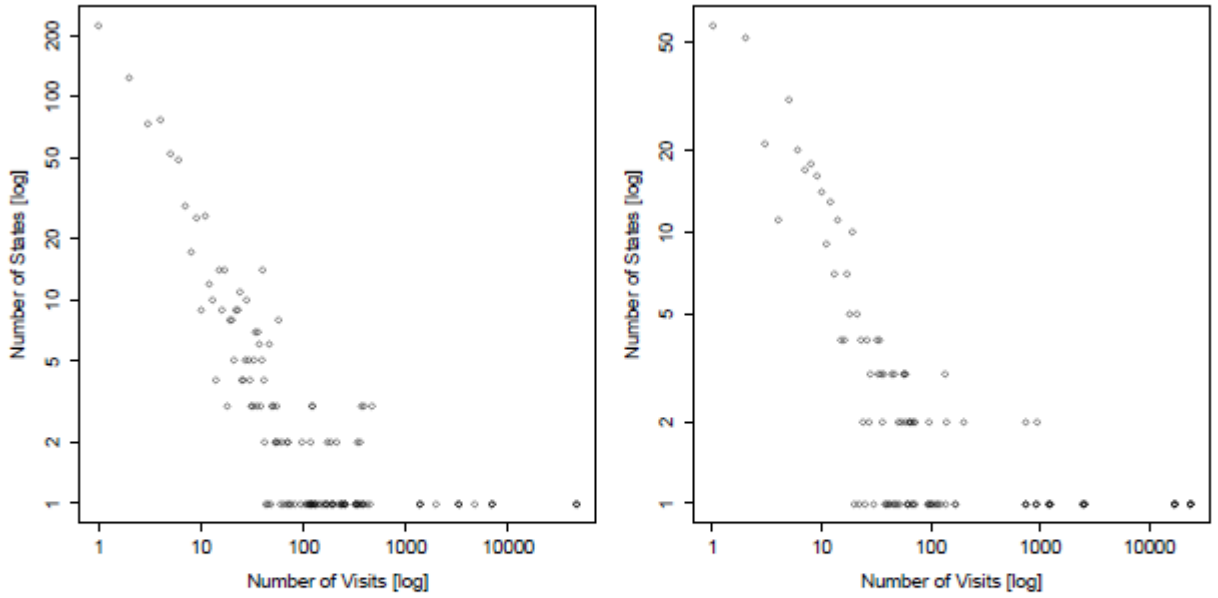


Figure 7: Distribution of the states against the number of visits in two typical cases

De BN a FSA mediante clustering

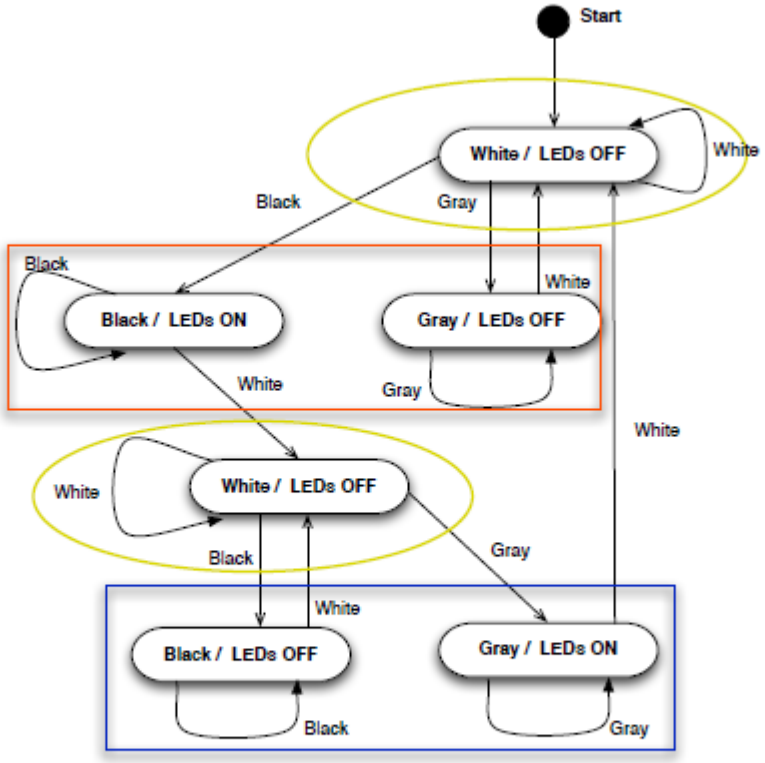


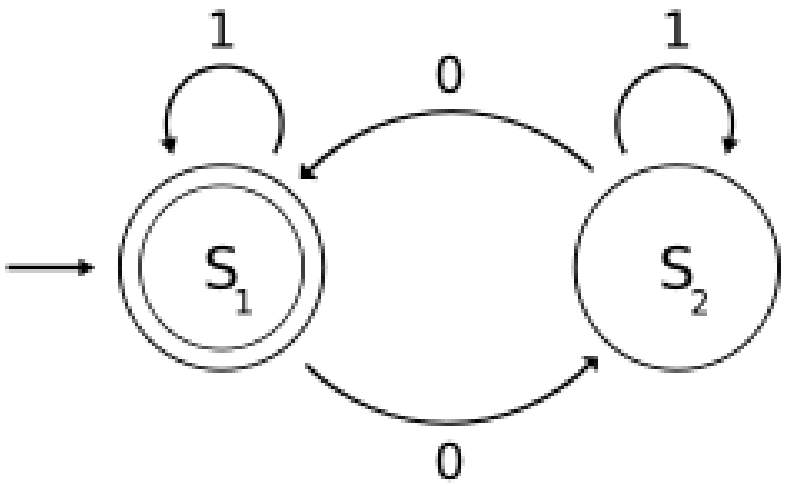
Figure 8: Finite state automaton representation of the state space graph. Sequence learning test case

Evolución de Automátas Finitos

- Ejemplo inspirado en arquitecturas de robots basadas en comportamiento
- P. Petrovic. 2005. Evolving automatons for distributed behavior arbitration. IDI/NTNU Technical Report 05/2005
- P. Petrovic. 2008. Evolving Behavior Coordination for Mobile Robots using Distributed Finite-State Automata. Frontiers in Evolutionary Robotics.

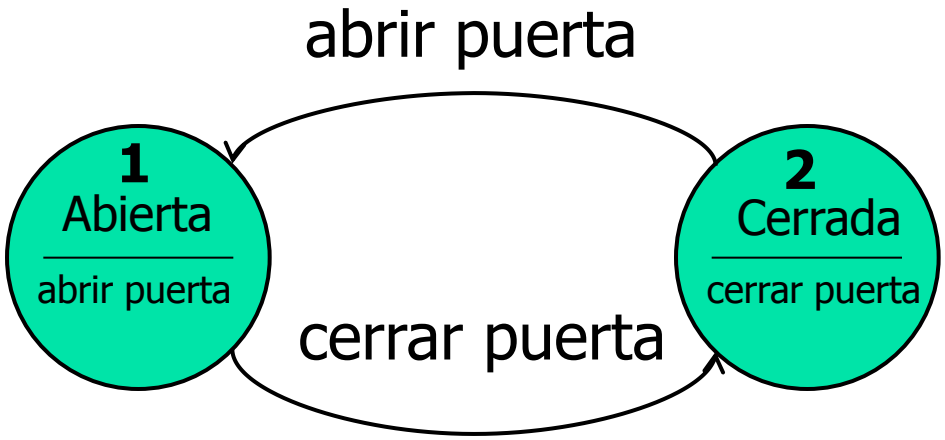
Autómatas deterministas de estado finito (FSA)

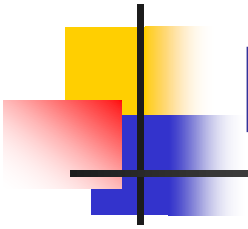
- Son modelos de computación restringida
- Reconocen lenguajes regulares:
  - $L=(1^*01^*)^*$
    - 1; 1111 …; 111101111; 1110111101111 …



Autómatas finitos

- Modelo matemático de comportamiento compuesto de un número finito de estados, acciones y transiciones entre estados



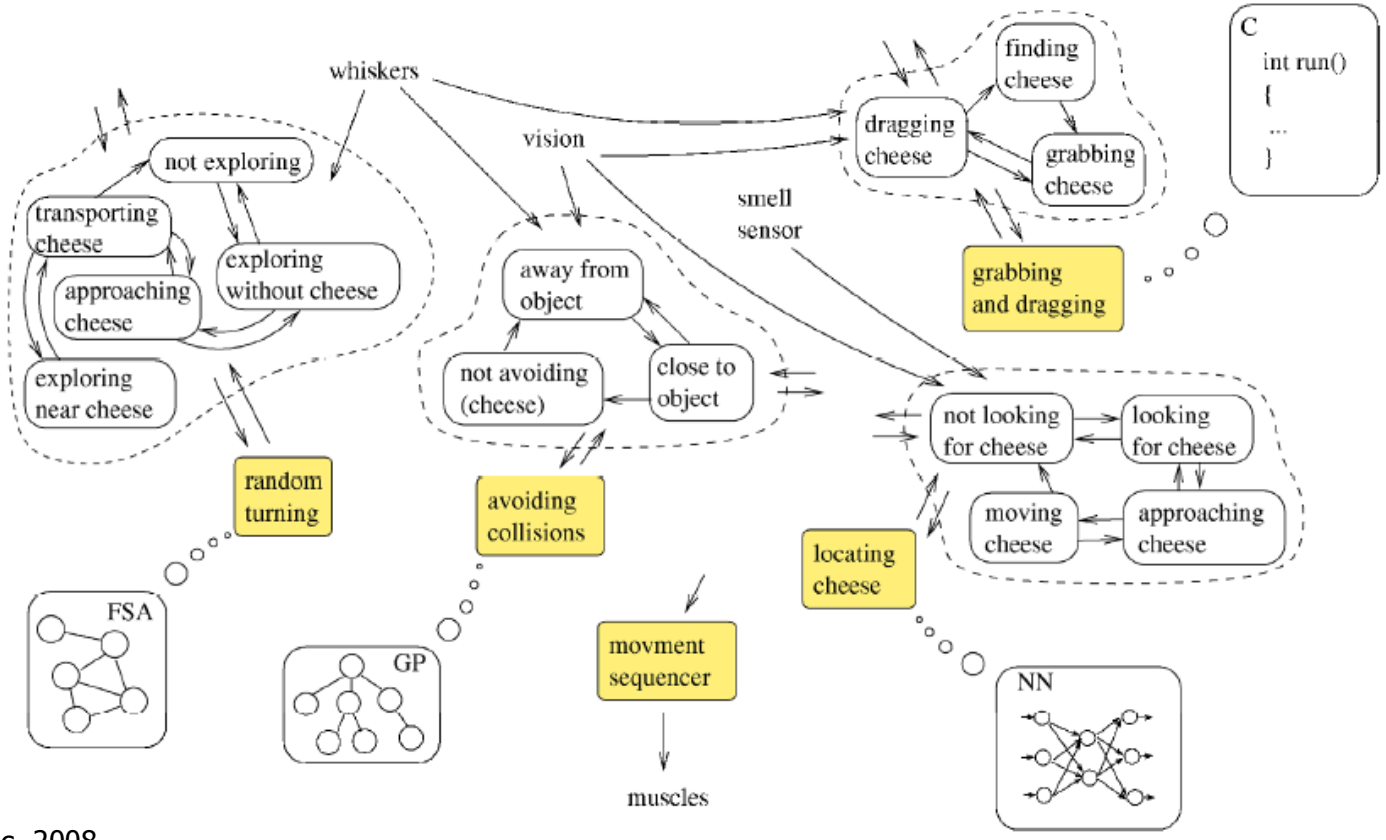


Evolución de AF

- Punto de partida: programación evolutiva (Fogel)
  - Mutación
- Posibilidades de extender el modelo
  - Cruce
  - Encapsulamiento de secciones (*compresión*)

Evolución de AF

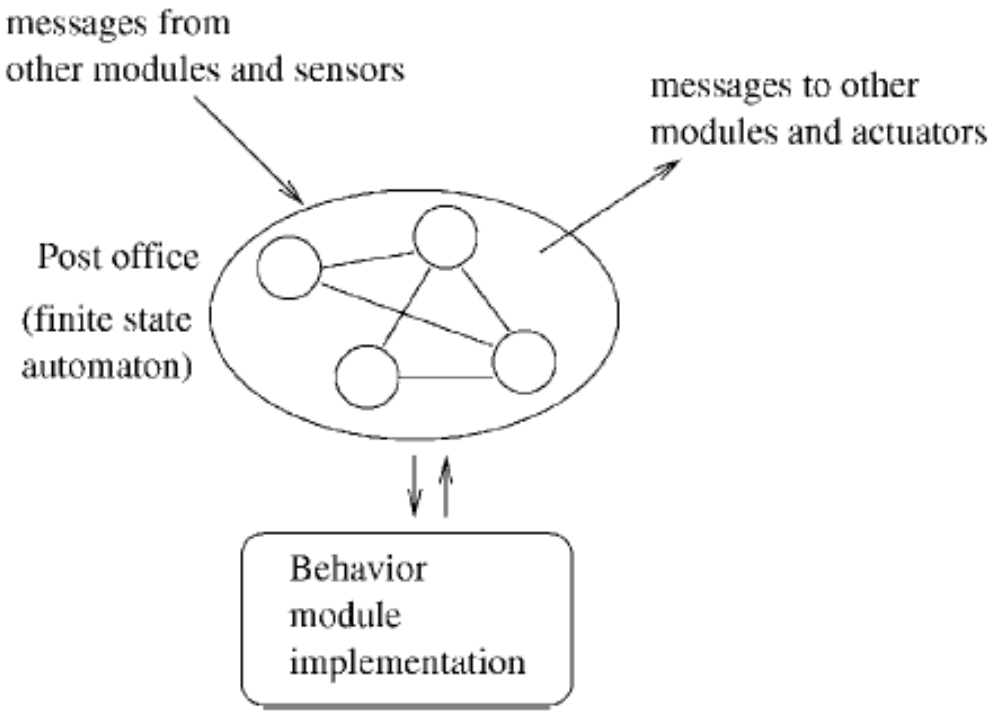
- Controlador de un ratón buscando queso



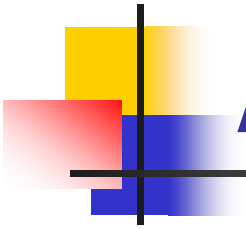
Fuente: Petrovic, 2008

Evolución de AF

- Módulo arbitrado por post-office



Fuente: Petrovic, 2008

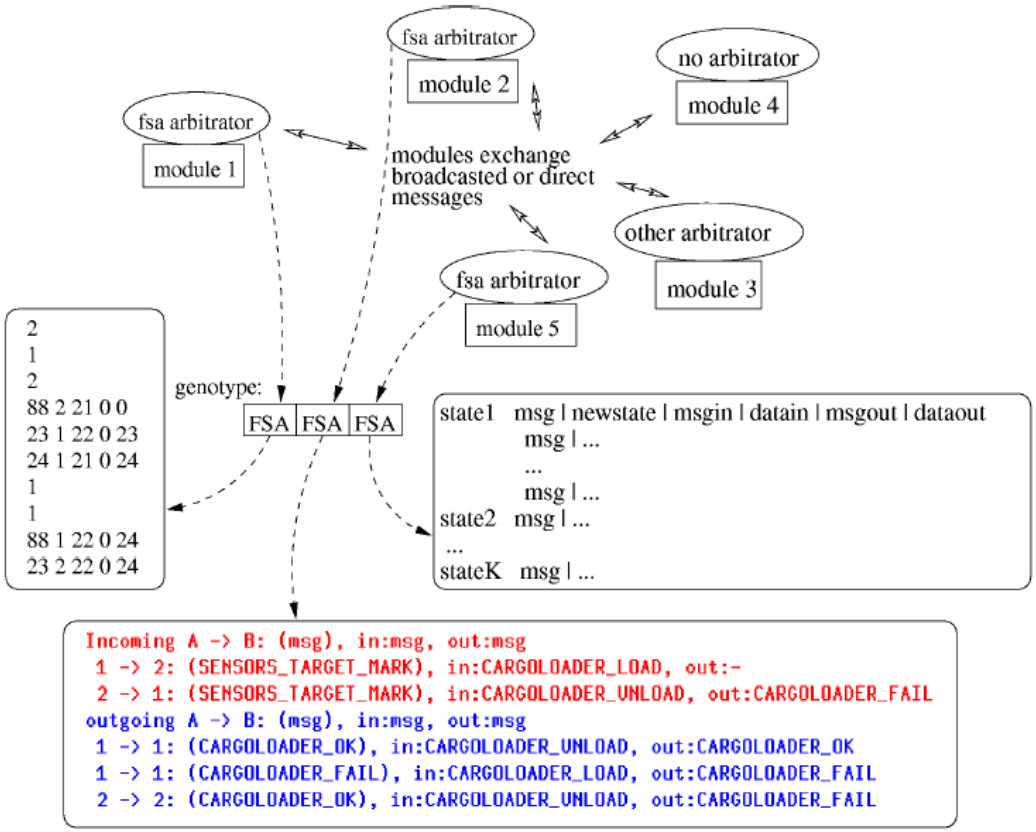


Autómatas finitos

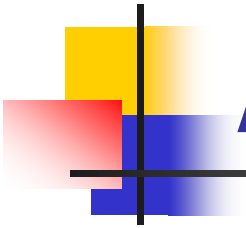
- Objetivo
  - Evolución automática de un conjunto de AFs que regulen el comp. de las post-offices
- Trabajo del diseñador
  - Determinar los módulos
  - Especificar la interfaz de cada módulo
    - Mensajes de entrada y salida
  - Fijar listas de mensajes por módulo capaces de provocar transiciones de estado

Autómatas finitos

- Genotipo



Fuente: Petrovic, 2008

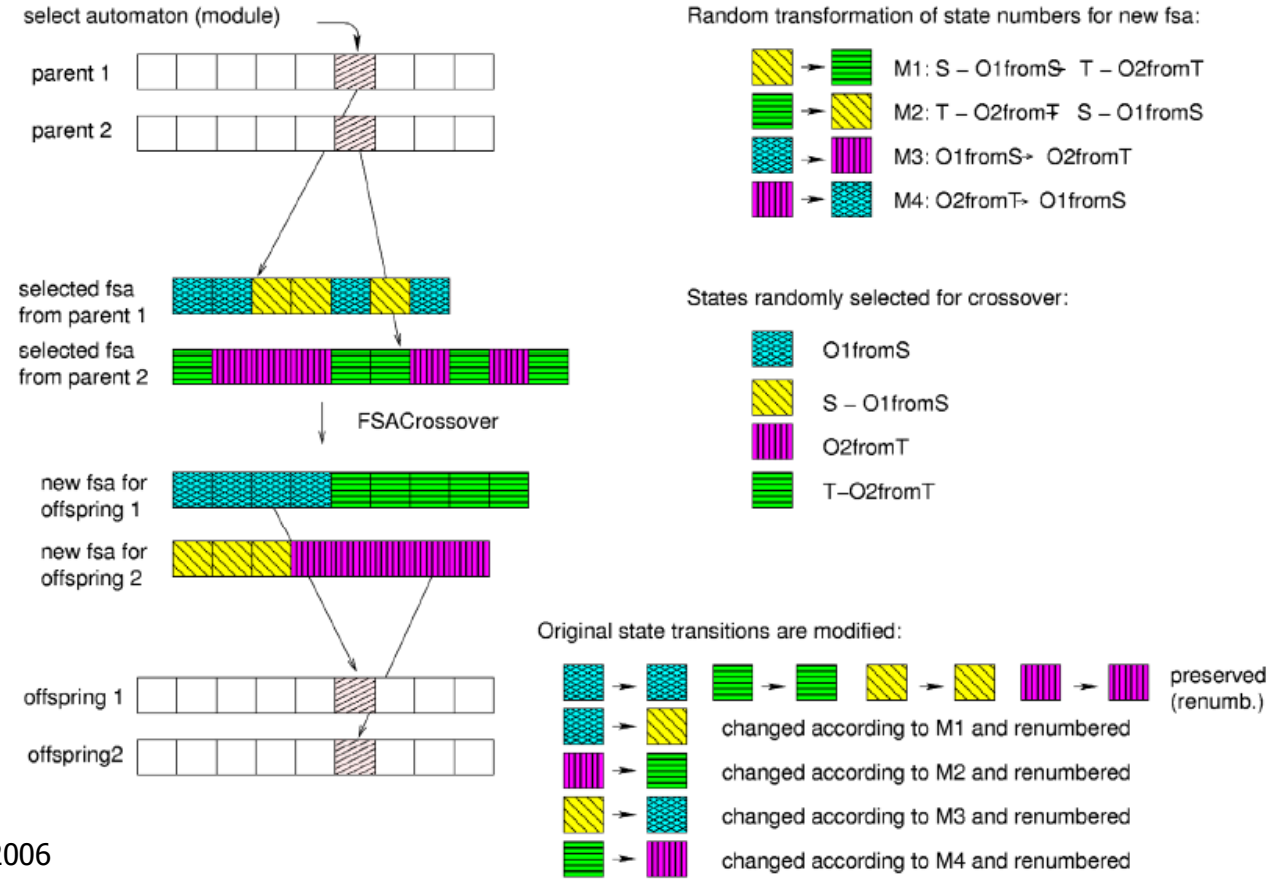


Autómatas finitos

- Cruce
  - Opera sobre sobre el AF de un solo módulo escogido al azar
  - Divide los estados de los AF en dos partes y los combina en dos nuevos AF
  - Posible reajuste aleatorio de transiciones

Autómatas finitos

- Cruce



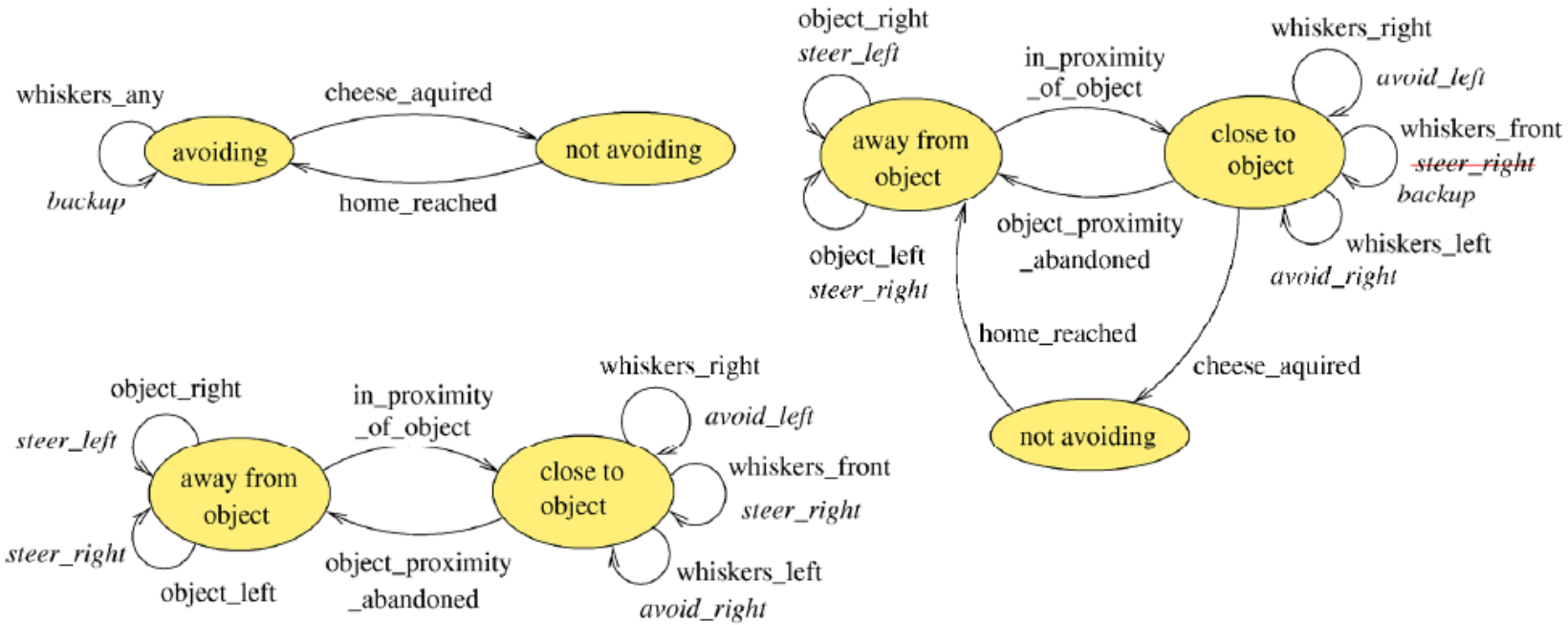
Fuente: Petrovic, 2006

Autómatas finitos

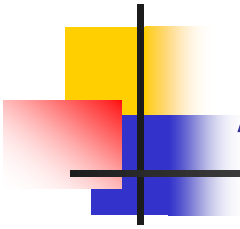
- Mutación
  - Opera sobre un único AF
  - Una entre las siguientes
    - a new random transition is created,
    - random transition is deleted,
    - a new state is created (with minimum incoming and outgoing random transitions); in addition, one new transition leading to this state from another state is randomly generated,
    - a random state is deleted as well as all its incident transitions,
    - a random transition is modified: (one of its parts `new_state`, `message_type`, `msg_to_send_out`, `msg_to_send_in` is replaced by an allowed random value),
    - a completely random individual is produced (this operator changes all FSAs),
    - a random transaction is split in two and new state is created in the middle,
    - the initial state number is changed.

Autómatas finitos

- Ejemplo de cruce y mutación



Fuente: Petrovic, 2008



AF vs PG

- Hay problemas de evolución de programas abordables tanto evolucionando AF como con PG.
- P. Petrovic. 2006. Comparing Finite State Automata Representation with GP-trees. IDI Technical Report 04/06

AF vs PG

- Representación PG

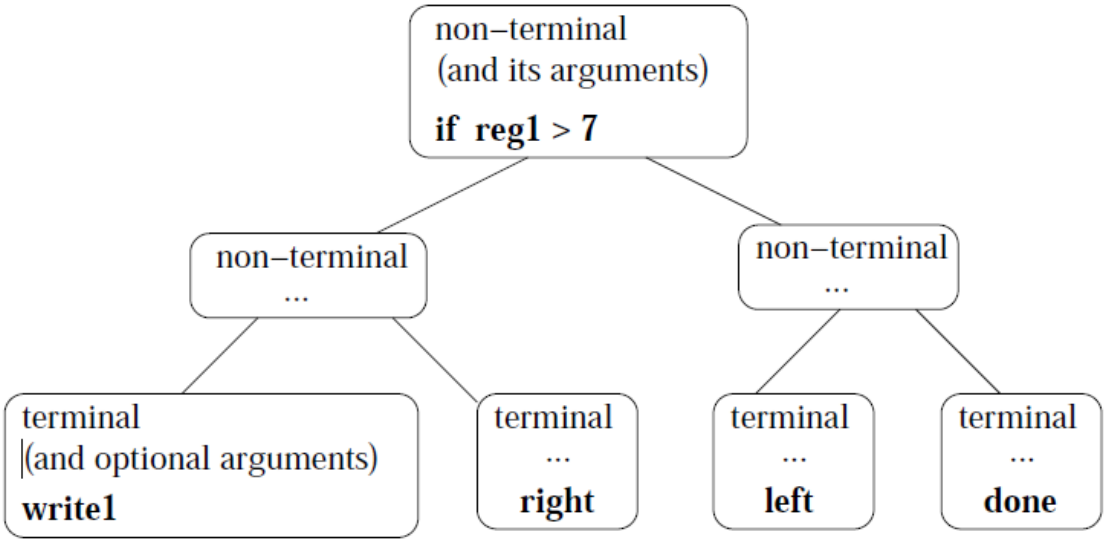


Figure 2: Illustration of GP-tree representation: nodes contain non-terminals that have two sub-trees, terminals are in the leaves.

Fuente: Petrovic, 2006

AF vs PG

- Representación AF

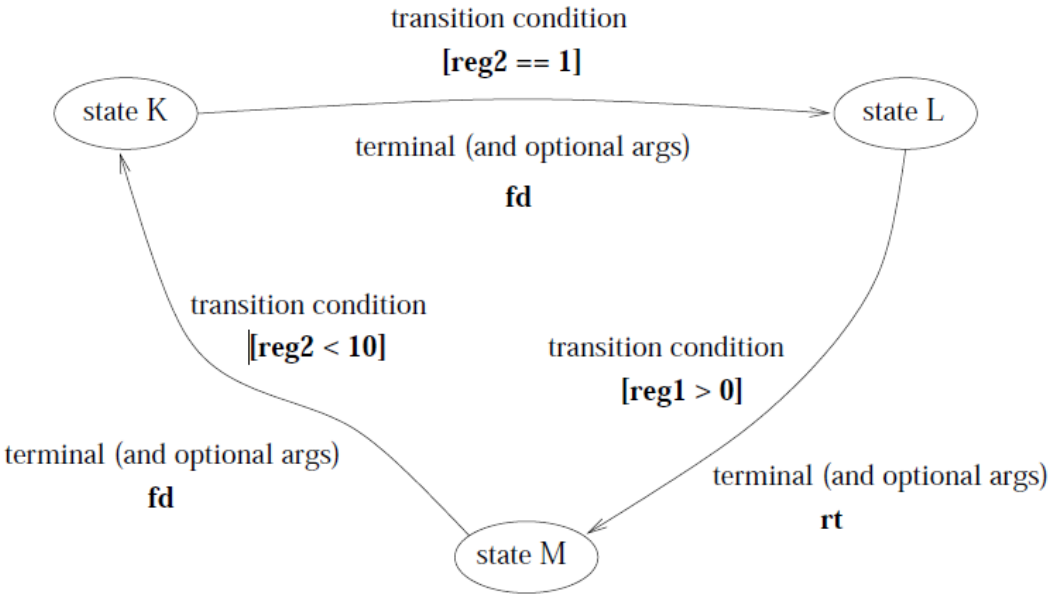
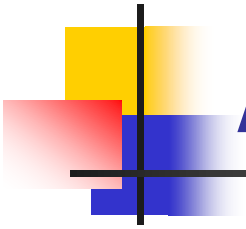


Figure 3: Illustration of FSA representation: transition conditions correspond to GP-tree non-terminals, however each transition has also an associated terminal. The state transitions can lead to the same state where they originate, and multiple transitions between the same two nodes are allowed (although only one is used). Sequences of state transitions can create loops.

Fuente: Petrovic, 2006



AF vs PG

- Experimento bit_collect
  - Estructuras algorítmicas
  - Dado una cadena el binario y una serie de operadores (left, right, write0, write1, done)
    - Básica
      - Reemplazar los ceros con unos
      - 10111001010001->11111111111111
    - Difícil
      - Desplazar los unos al principio de la cadena
      - 10111001010001->11111110000000

AF vs PG

- Experimento bit_collect

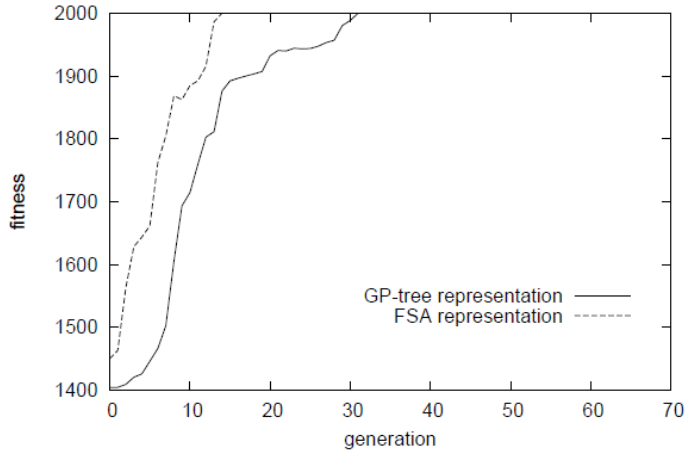


Figure 15: Performance of the FSA and GP-tree representations on simple version of task "bit_collect". Average of 19 (GP-tree) and 22 (FSA) runs. The programs were presented 20 input words of length 5 to 30, containing about 75% of ones, and were allowed to make at most 200 execution steps. Other parameters: population size: 250, prob. crossover: 0.5, brooding crossover (number of non-strict broods 3, 30% of training samples used for brooding), combining crossover (GP-trees): 0.25, prob. mutation: 0.7, 7 elite individuals, tournament selection (tournament size 4, probability 0.8), max. GP-tree depth: 12, max. number of FSA states/transitions: 22/10, FSA shuffle mutation: 0.4.

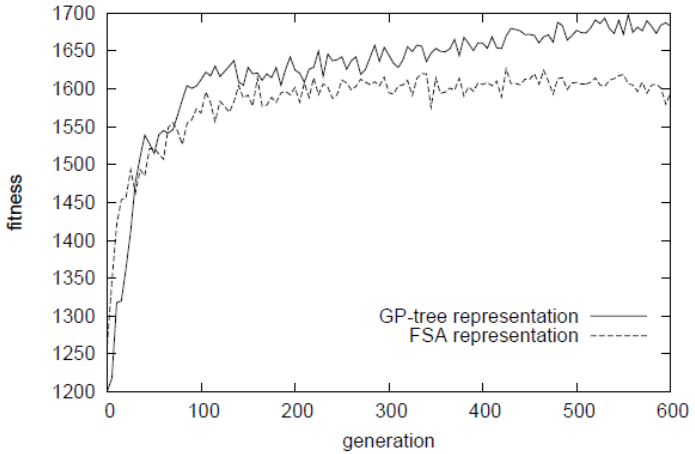


Figure 16: Performance of the FSA and GP-tree representations on more complex version of task "bit_collect". Average of best individuals in each generation from 9 (GP-tree) and 13 (FSA) runs. Other parameters were the same as in the simple version of the task, we terminated the runs after 600 generations if the solution did not evolve.

Fuente: Petrovic, 2006

AF vs PG

- Experimento $abcd^n$
  - Repeticiones
  - Reemplazar los unos de una secuencia continua con la secuencia a, b, c, d

11111111111111111111

abcdabcdabcdabcdabcd

AF vs PG

- Experimento $abcd^n$

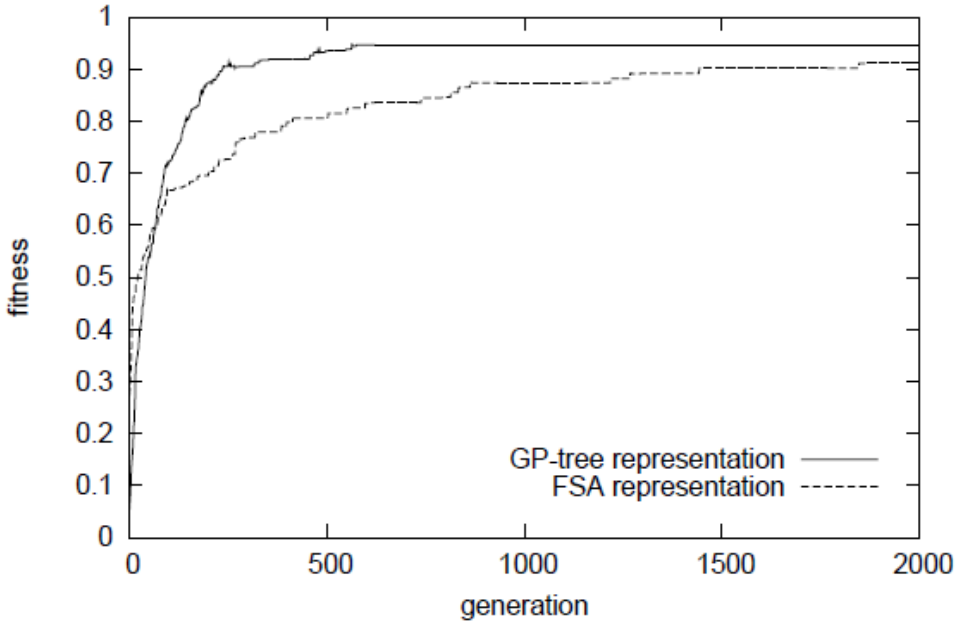


Figure 17: Performance of the FSA and GP-tree representations on task "$abcd^n$". Average of 25 (GP-tree) and 23 (FSA) runs. The programs were presented with input word containing 32 ones, and had 150 execution steps for writing the output word. Population size 300, prob. crossover 0.6, 15 strict-elite individuals. Other parameters were the same as in the "bit_collect" task, we terminated the runs after 2000 generations if the solution did not evolve.

Fuente: Petrovic, 2006

AF vs PG

- Experimento switch
  - Reacción
  - Reemplazar todos los ceros de una cadena con el símbolo correspondiente al valor distinto de cero a la izquierda

10004030000200130004

11114433333222213334

AF vs PG

- Experimento switch

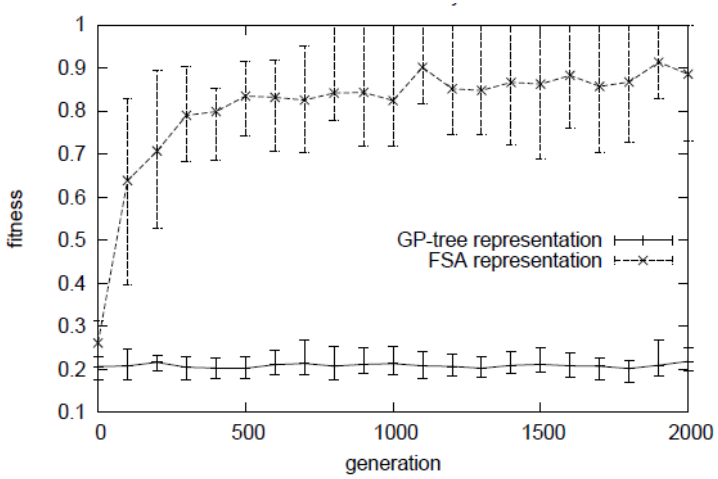


Figure 18: Average of the best fitness from 14 (GP), or 10 runs (FSA) on a switch task witl tournament (FSA), or fitness proportionate (GP-trees) selection, population size 300, prob. o crossover 0.5, crossover brooding of size 3, with 0.3 starting locations used to evaluate broodin individuals, probability of mutation 0.9, 15 elites, each individual evaluated on 10 randon strings, input word length randomly varying from 10 to 60 with maximum 10 continuous 0 symbols, maximum number of GP-tree or FSA execution steps 300, FSA: pshuffle=0.4, numbe of states 1–15, number of transitions: 1–15, GP: pcross_combine=0.25, maximum tree depth=15 The number of evaluations is proportional to the generation number. The error bars show th range of fitness progress in all runs.

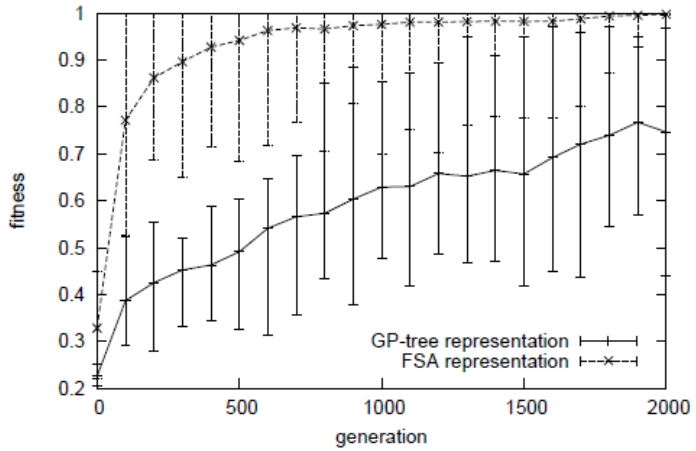
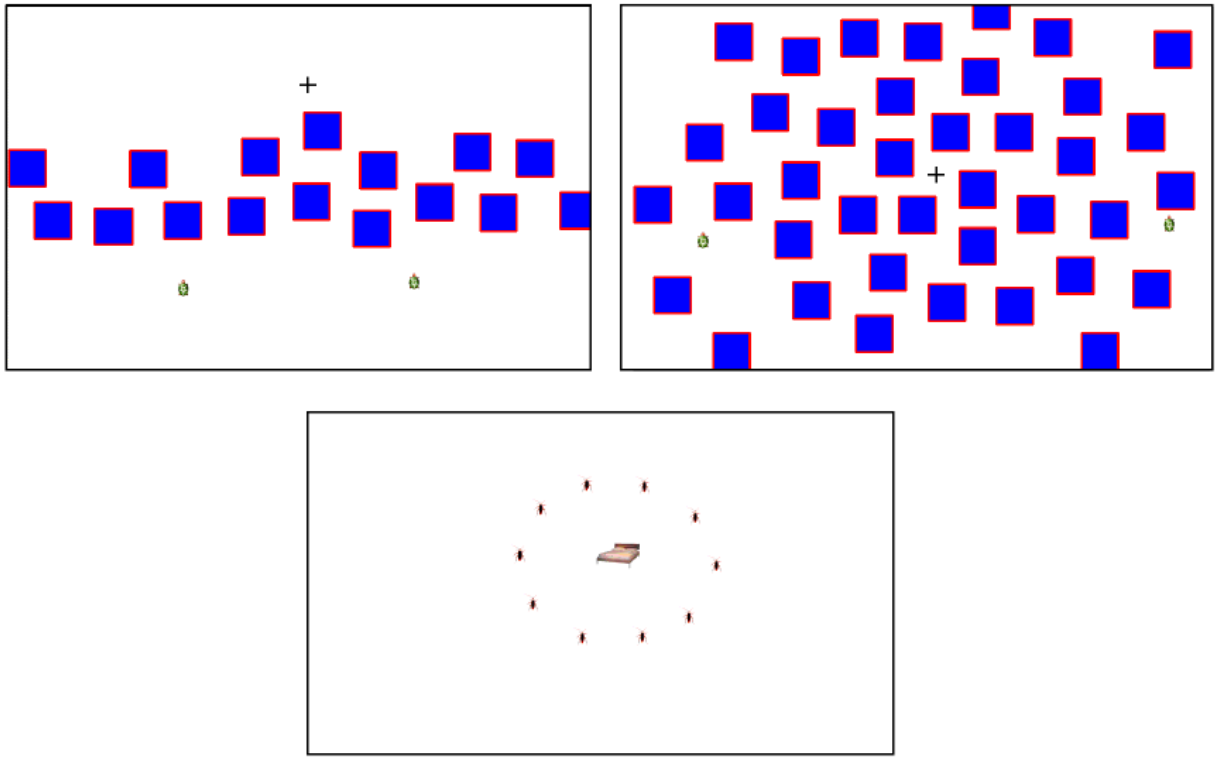


Figure 19: Average of the best fitness from 15 (GP), or 34 runs (FSA) on a switch task with tournament selection, population size 300, prob. of crossover 0.5, crossover brooding of size 3, with 0.3 starting locations used to evaluate brooding individuals, probability of mutation 0.9, 15 elites, each individual evaluated on 10 random strings, input word length randomly varying from 10 to 60 with maximum 10 continuous 0-symbols, maximum number of GP-tree or FSA execution steps 300, FSA: pshuffle=0.4, number of states 1–15, number of transitions: 1–15, GP: pcross_combine=0.25, maximum tree depth=15. The number of evaluations is proportional to the generation number. Also notice that due to the randomness of the testing input strings, the performance in the succeeding generation can decrease, even though the quality of the individual remains or even increases. The error bars show the range of fitness progress in all runs, notice that the partial overlap is only due to the randomness of strings, but the evolved individuals in all FSA runs outperform those with GP-trees.

Fuente: Petrovic, 2006

AF vs PG

- Experimento find_target



Fuente: Petrovic, 2006

AF vs PG

- Experimento find_target

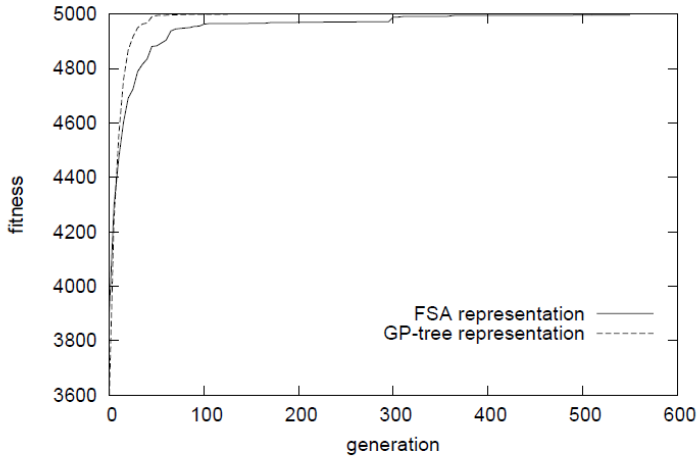


Figure 7: Average of the fitness of the best individuals in the "find_target" task, environment `experiment_fence`, comparison from 13 FSA and 17 GP-tree runs. The maximum possible fitness is 5000.

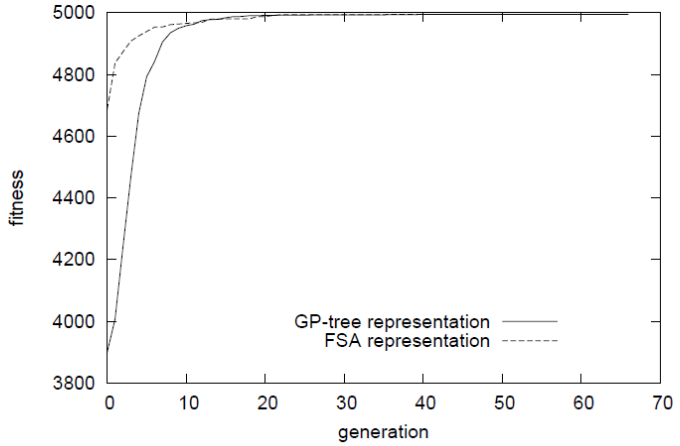
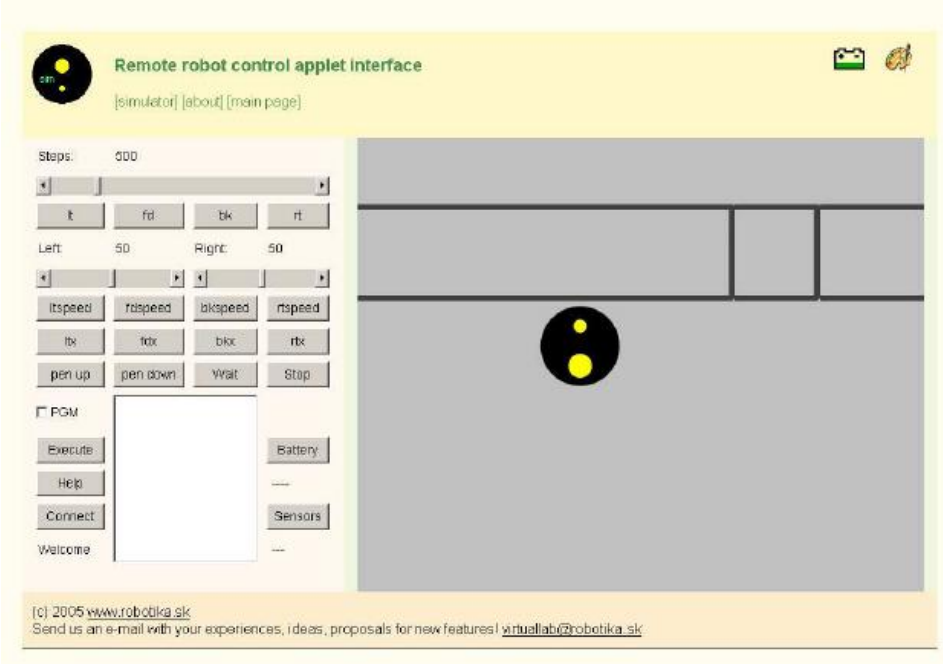


Figure 12: Performance of the FSA and GP-tree representations on task "find_target", environment `ten_around`. Average of 25 (GP-tree) and 23 (FSA) runs. FSA individuals quickly learn to arrive close to the target, but take longer time to fine-tune the solution to arrive exactly to the target location than GP-tree individuals.

Fuente: Petrovic, 2006

AF vs PG

- Experimento dock
  - Llevar el robot al centro del cuadrado desde una posición aleatoria del cuadrante inferior y a la izquierda



Fuente: Petrovic, 2006

AF vs PG

- Experimento dock

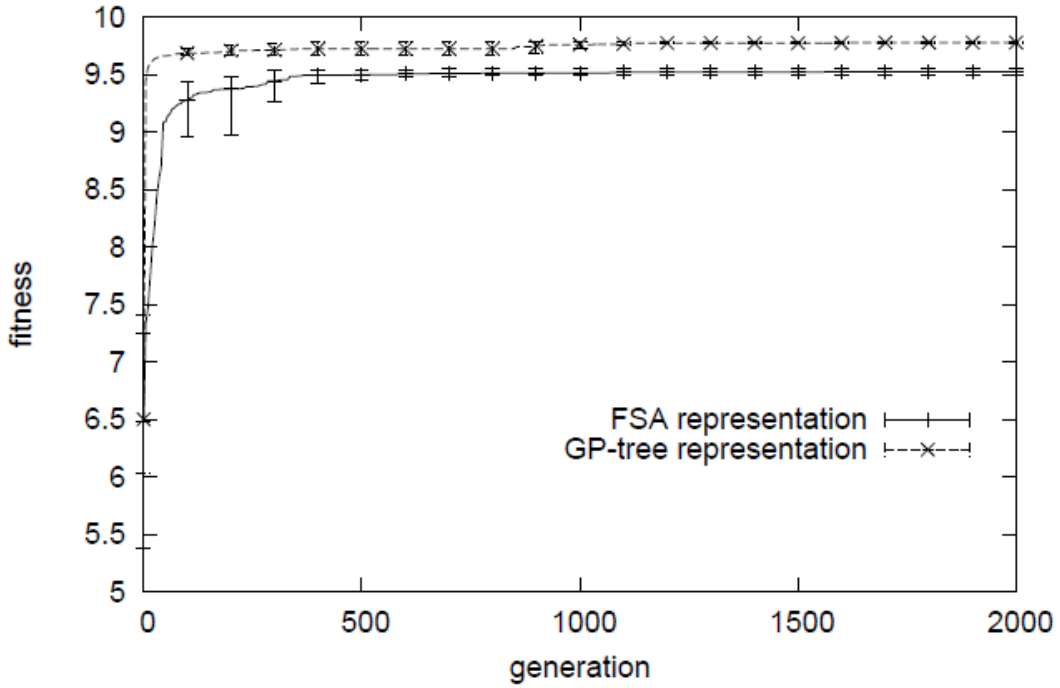
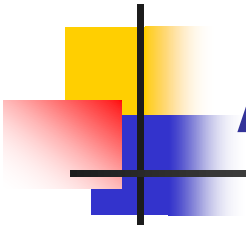


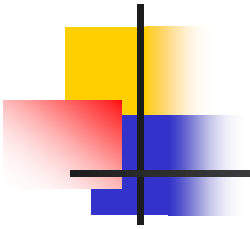
Figure 25: Performance of the GP-tree and FSA representations on simulated "dock" task. Best individual fitness from each generation, average from 10 runs. Errorbars show range.

Fuente: Petrovic, 2006



AF vs PG

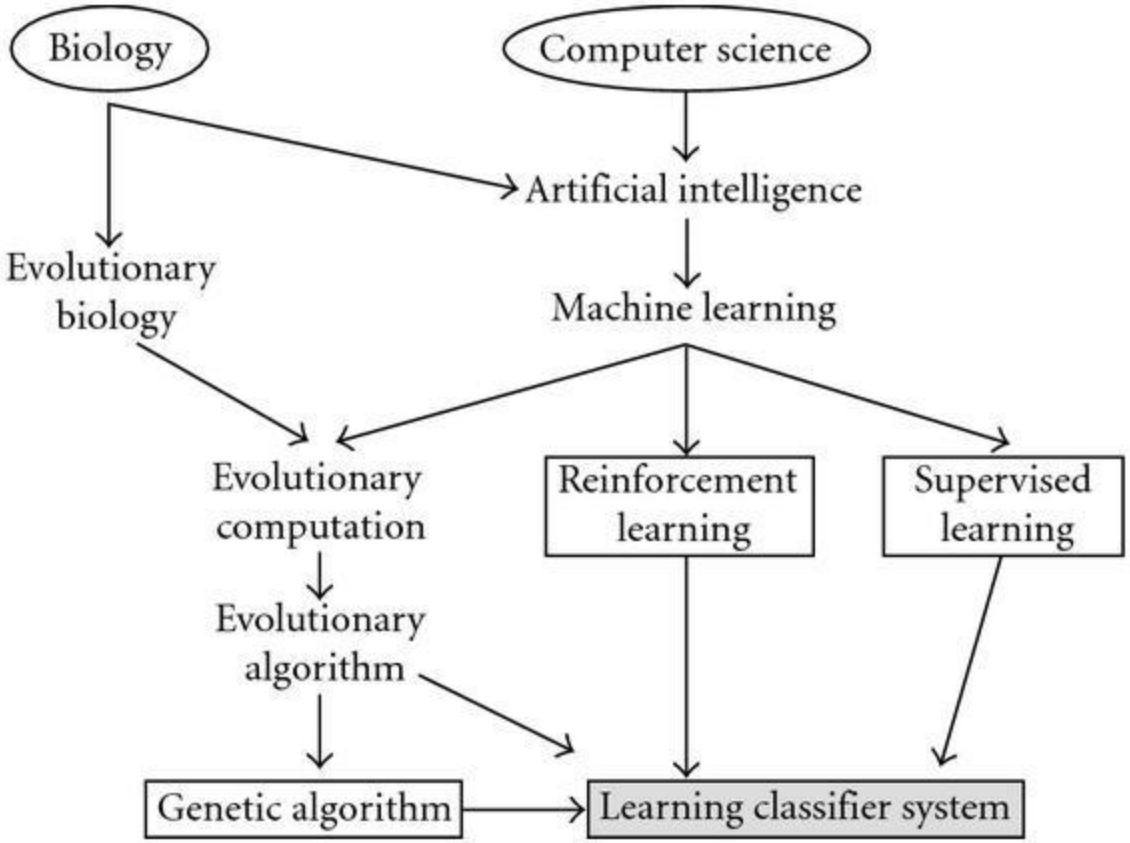
- Conclusiones
  - Los programas de PG tienden presentar patrones de ejecución más lineal
  - Los resultados son variables en función de las estructuras de interacciones necesarias
  - Los individuos AF requieren condiciones para que se den dos acciones (no en PG)
  - Las soluciones FSA más sensibles a entradas de sensores
  - La evolución incremental es una vía a valorar



Sistemas Clasificadores

- O Learning Classifier Systems (LCS)
- Son sistemas que aprenden reglas mediante un algoritmo genético para que un agente se desenvuelva en un entorno arbitrario

Sistemas Clasificadores

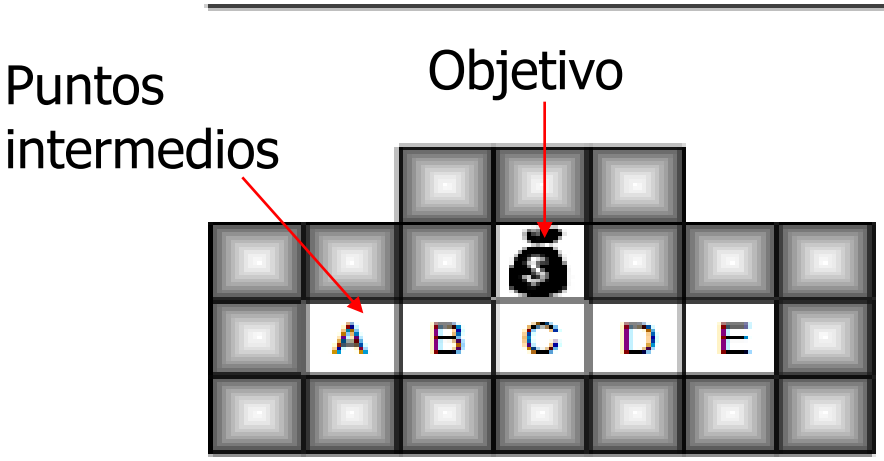


Refuerzo (Reinforcement Learning (RL))

- Un agente interacciona con su entorno, del que recibe premios y castigos (refuerzo) y debe de aprender una política (comportamiento) para maximizar su refuerzo

Ejemplo de problema de aprendizaje por Refuerzo (RL)

- El robot sólo ve las 8 paredes que le rodean
- Sus sensaciones se pueden representar con un vector de 8 componentes



stat e	sensation
A	11011111
B	10011101
C	01011101
D	11011100
E	11111101

N, NE, E, SE, S, SO, O, NO

Ejemplo de problema de aprendizaje por Refuerzo (RL)

- El robot debe aprender que acción realizar (ir norte, ir nor-este, ir este, …) en cada posición (A, B, C, D, E)

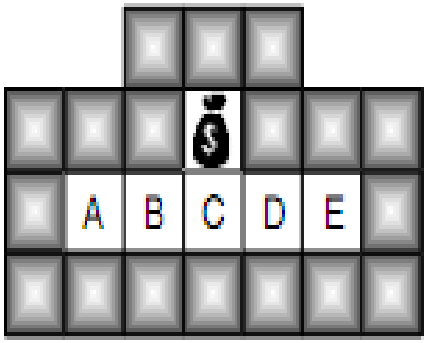
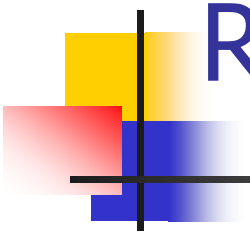


Tabla de valores-Q y Política (o estrategia)

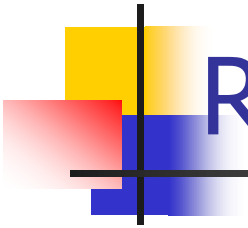
stat	e sensation	↑	↗	→	↘	↓	↙	←	↖
A	11011111	810	810	**900**	810	810	810	810	810
B	10011101	900	**1000**	900	900	900	900	810	900
C	01011101	**1000**	900	900	900	900	900	900	900
D	11011100	900	900	810	900	900	900	900	**1000**
E	11111101	810	810	810	810	810	810	**900**	810

N, NE, E, SE, S, SO, O, NO

RL es multi-paso (semisupervisado)



- Supongamos que el robot sólo recibe refuerzo (1000) cuando alcanza el objetivo y 0 en cualquier otra situación
- Para aprender la estrategia, el robot debe pasear mas o menos aleatoriamente hasta que llega al objetivo. En ese caso recibe refuerzo y puede actualizar su "política"
- La dificultad principal de RL es que se trata de problemas multi-paso

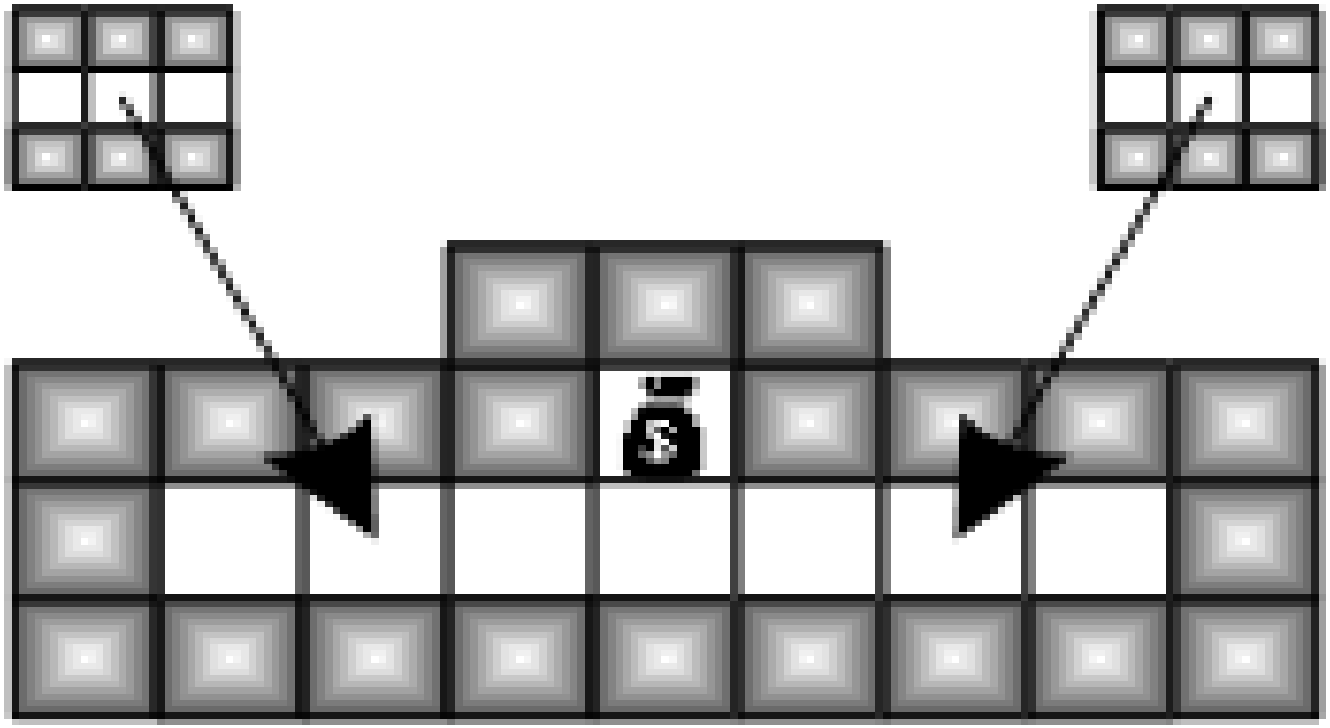


RL es multi-paso (semisupervisado)

- La cuestión en los problemas multi-paso es saber que valor tuvo una acción que hice en el pasado para conseguir el objetivo actual
- Ejemplo del ajedrez: ¿qué influencia (valor) tuvo el que yo moviera el alfil hace 50 jugadas para conseguir hacer ahora jaque mate?
- A esta cuestión se la denomina asignación de crédito

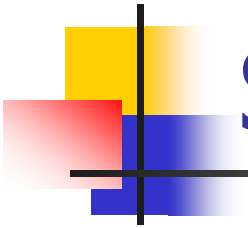
RL es a veces parcialmente observable

- Distintas situaciones resultan en la misma percepción



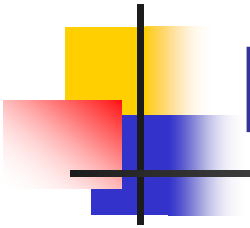
Sistemas Clasificadores

- J. Holland y J. Reitman. 1978. Cognitive systems based in algorithms. Machine learning, an artificial intelligence approach
- J. Holland. 1980. Adaptive algorithms for discovering and using general patterns in growing knowledge bases. International Journal of Policy Analysis and Information Systems



Sistemas Clasificadores

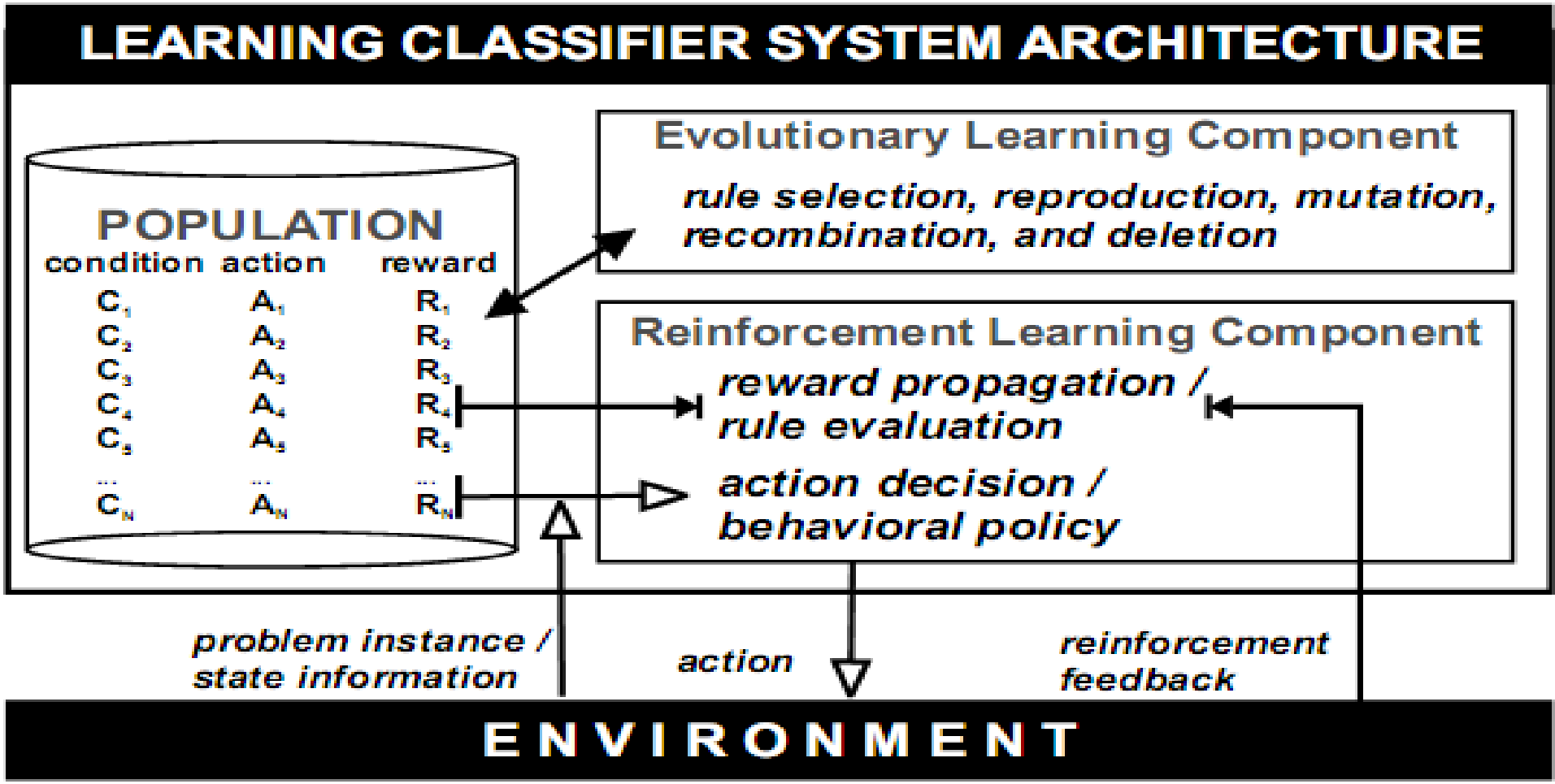
- Tres componentes:
  - Reglas (o clasificadores)
  - Asignación de crédito (propagación del refuerzo)
  - Descubrimiento de nuevas reglas mediante un algoritmo genético

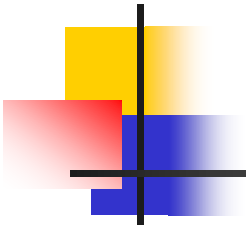


Reglas

- Usan tres símbolos: 0, 1 y #
- El # significa "don't care" y es un comodín que puede valer 0 o 1.
- Ejemplo:
  - la regla "######1# → Este" significa que si hay pared hacia el oeste, hay que ir hacia el este, independientemente del resto de paredes
- Cada regla tiene además el refuerzo esperado. Es decir, la *fitness* de las reglas es el refuerzo que se espera recibir si ejecutan su acción

LCS1: Arquitectura





LCS1: funcionamiento

1. Llega una percepción: 1001
2. Se calculan todas las reglas cuyas condiciones encajan con la percepción (*match set*)
3. Para cada posible acción, se calcula cuantas reglas del *match set* están a favor, calculando el promedio de sus *fitness*
4. Se ejecuta la acción que tiene mayor promedio de refuerzo
5. Se actualizan las *fitness* (refuerzos, *reward*) de las reglas involucradas
6. Se ejecuta el algoritmo genético para crear y aprender nuevas reglas

LCS1: cálculo del *match set*

instance	matching conditions
1001	1001
	100# 10#1 1#01 #001
	10## 1#0# #00#
	1##1 #0#1 ##01
	###1 ##0# #0## 1###
	####

matching problem instances				condition
			1001	1001
		1001	1000	100#
	1011 1010	1001	1000	10##
1111 1101	...	0011	0001	###1
1111 1110	...	0001	0000	####

LCS1: Acción a ejecutar

$$\pi_{LCS1}(s) = \begin{cases} \arg\max_a \frac{\sum_{\{cl \in [M] | cl.A = a\}} cl.R}{|\{cl \in [M] | cl.A = a\}|} & \text{with prob. } 1-\epsilon \\ \text{rand}(a) & \text{otherwise} \end{cases},$$

LCS1: refuerzo

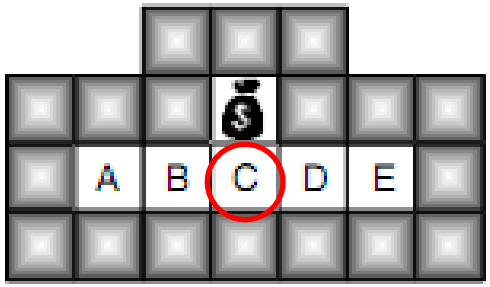
- Utiliza el algoritmo de aprendizaje por refuerzo *bucket brigade*
- En cada iteración *t* los clasificadores *C* aplicables usan su fitness *F* y su especificidad *S* para pujar para su acción sea la elegida

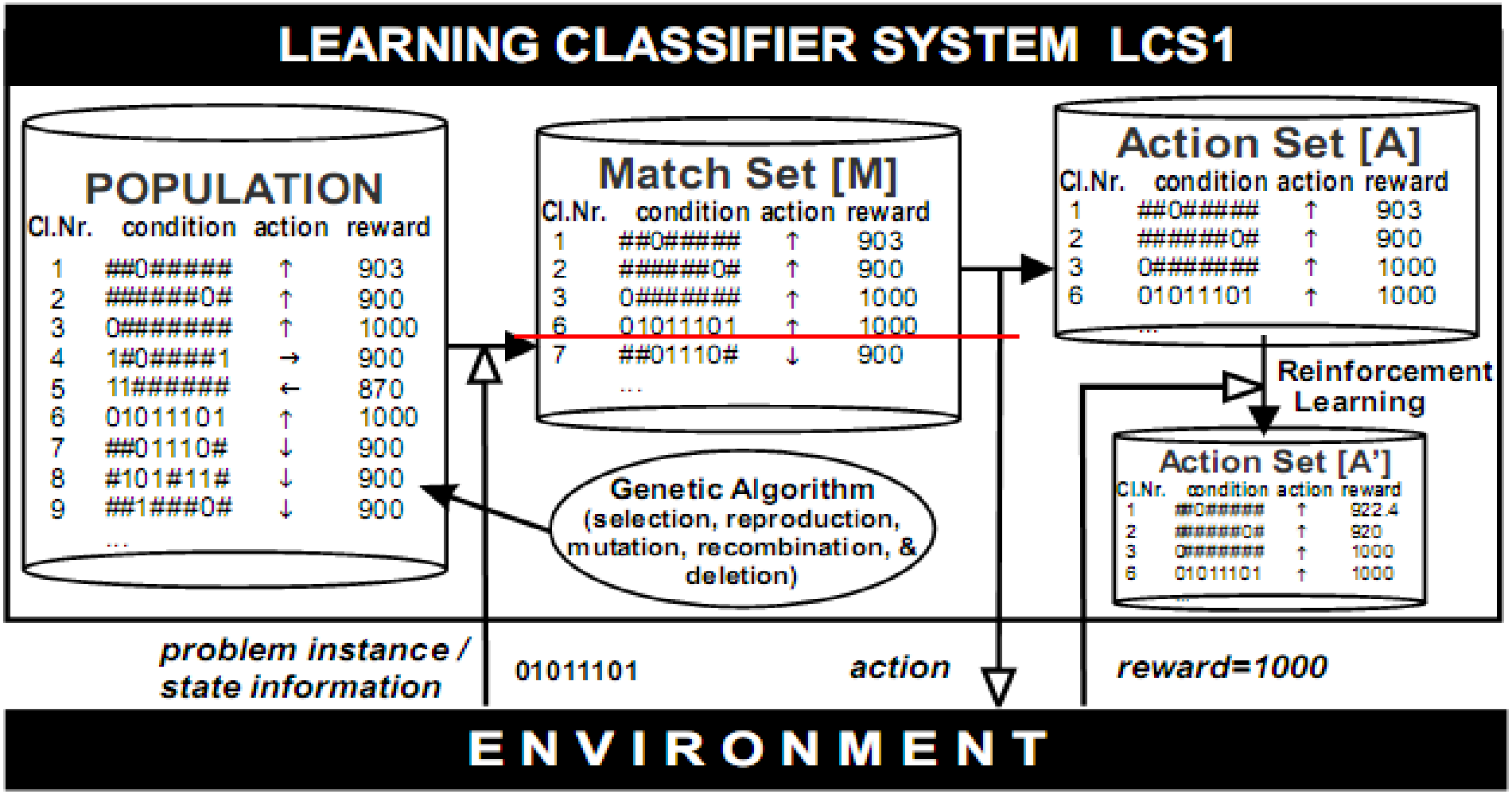
$$B^t{}_C = k * SC * FtC \qquad k \ll 1$$

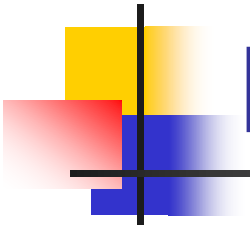
- El ajuste en las reglas de las iteraciones anteriores se hace de la forma

$$F^{t+1}{}_{C'} = F^t{}_{C'} + \sum B^t / x$$

LCS1: funcionamiento





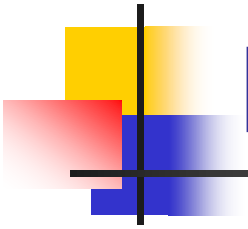


LCS1: aplicación del GA

- Periódicamente se ejecuta un GA steady state para generar nuevas reglas reemplazando a las peores
- La búsqueda está guiada por las mejores reglas, que son aquellas que reciben más refuerzo del entorno
- Con el tiempo, el sistema de reglas converge a un conjunto de reglas que de forma conjunta resultan efectivas
- El uso de un mecanismo de fitness sharing permite el mantenimiento de nichos

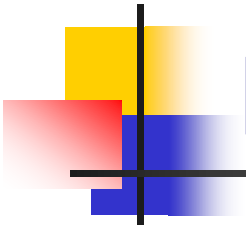
LCS1: aplicación del GA

- Fase de aprendizaje: el genético selecciona dos clasificadores, los recombina y los muta:
  - Selección de los clasificadores 3 y 6 (notese su alta fitness de 1000)
  - 3 = (0########,↑)
  - 6 = (01011101,↑)
  - 3 mutado = (0####1###,↑)
  - 6 mutado = (0101#101,↑)
  - 3 cruzado = (0####1101,↑)
  - 6 cruzado = (0101####,↑)
- A continuación los introduce, reemplazando algún clasificador que ya existiera (en este caso, se reemplazan los clasificadores 2 y 5 (baja fitness))



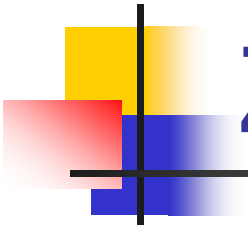
LCS1: covering

- Operación a realizar en dos situaciones
  - No hay ninguna regla que encaje
  - La fuerza total de las reglas que encajan es inferior a una cota
- Se genera una regla que encaje
  - Número aleatorio de #
  - Fuerza inicial: media de la población
  - Acción aleatoria
- Se introduce la nueva regla en la población reemplazando una existente



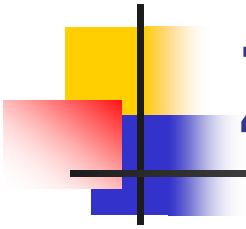
LCS1

- Éxitos iniciales en aplicaciones de mundo real
  - Control de gaseoductos (Goldberg 1983)
  - Control de videocámaras (Wilson 1985)
  - Modelado de mercados financieros (Marimon et. Al (1991)
- Caída del interés al detectar problemas en la formación de cadenas (Wilson y Goldberg, 1989)



ZCS

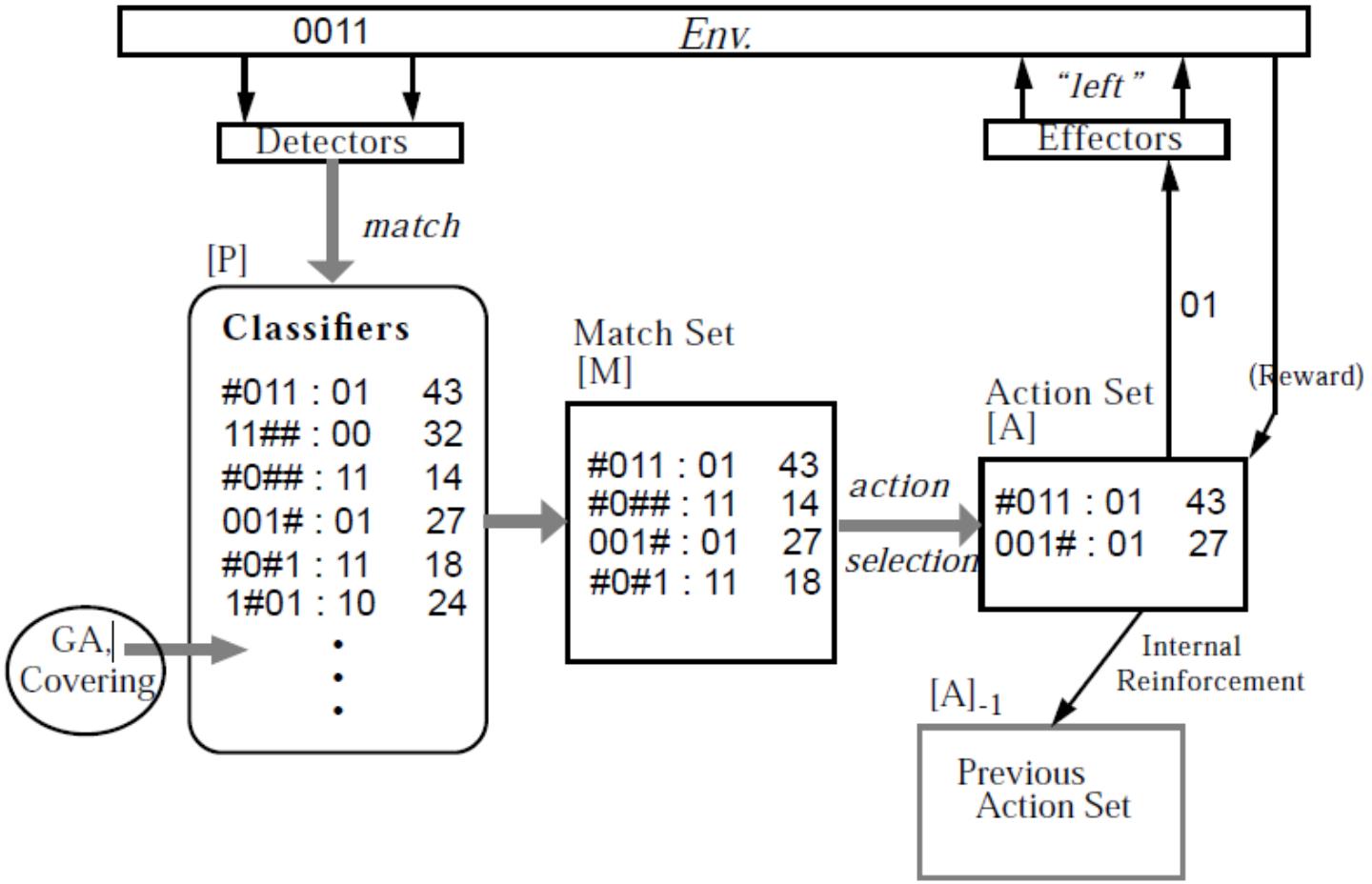
- Surge del intento de simplificar el modelo de Holland
- S. Wilson. 1994. ZCS: a zeroth-level classifier fitness. Evolutionary Computation

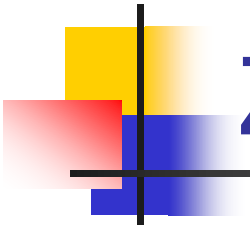


ZCS: principales novedades

- Uso de action sets en selección y refuerzo
- Refuerzo mediante implicit bucket brigade con factor de descuento
- Suprime el proceso de puja y deja de valorar la especificidad

ZCS: esquema


0011
Env.
"left"
Detectors
Effectors
match
[P]
Classifiers
#011 : 01 43
11## : 00 32
#0## : 11 14
001# : 01 27
#0#1 : 11 18
1#01 : 10 24
Match Set
[M]
#011 : 01 43
#0## : 11 14
001# : 01 27
#0#1 : 11 18
action
selection
Action Set
[A]
#011 : 01 43
001# : 01 27
01
(Reward)
GA,
Covering
Internal
Reinforcement
[A]-1
Previous
Action Set



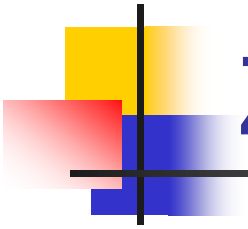
ZCS: refuerzo

1. Dado [A], se deduce una fracción β $(0 < \beta \leq 1)$ de la fuerza de cada regla y se coloca en un bucket *B*
2. Si se obtiene un refuerzo del entorno r_{imm}, la fuerza de cada regla de [A] se incrementa en $\beta\, r_{imm}/|A|$
3. Se incrementa la fuerza de los clasificadores de $[A_{-1}]$ con $\gamma B/|A_{-1}|$, donde $(0 < \gamma \leq 1)$
4. Se reemplaza $[A_{-1}]$ con [A] y se vacía *B*

$$S_{[A]} \leftarrow S_{[A]} - \beta S_{[A]} + \beta\gamma S_{[A]'}$$

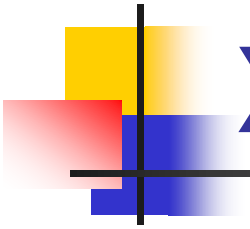
ZCS: parámetros

- **N** tamaño de la población
- $\mathbf{P_{\#}}$ Probabilidad de introducir un alelo # en la población inicial o en covering
- $\mathbf{S_0}$ Fuerza asignada a cada clasificadoren la población inicial
- β Tasa de actualización de la fuerza mediante bucket brigade
- γ Factor de descuento de la bucket brigade
- τ Tasa de penalización para clasificadores de [M] que no estén en [A]
- **χ** Probabilidad de cruce en el GA
- μ Probabilidad de mutación por alelo en GA
- ρ Promedio de clasificadores nuevos generados por el GA por unidad de tiempo
- **ϕ** Si la fuerza total de [M] es menor que ϕ veces la media de [P], usar covering



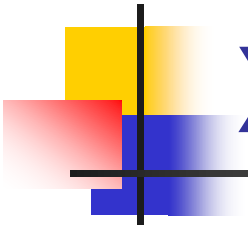
ZCS: resultados

- Buen comportamiento, pero no óptimo
- Muy sensible a los parámetros
- Técnica con vocación provisional



XCS

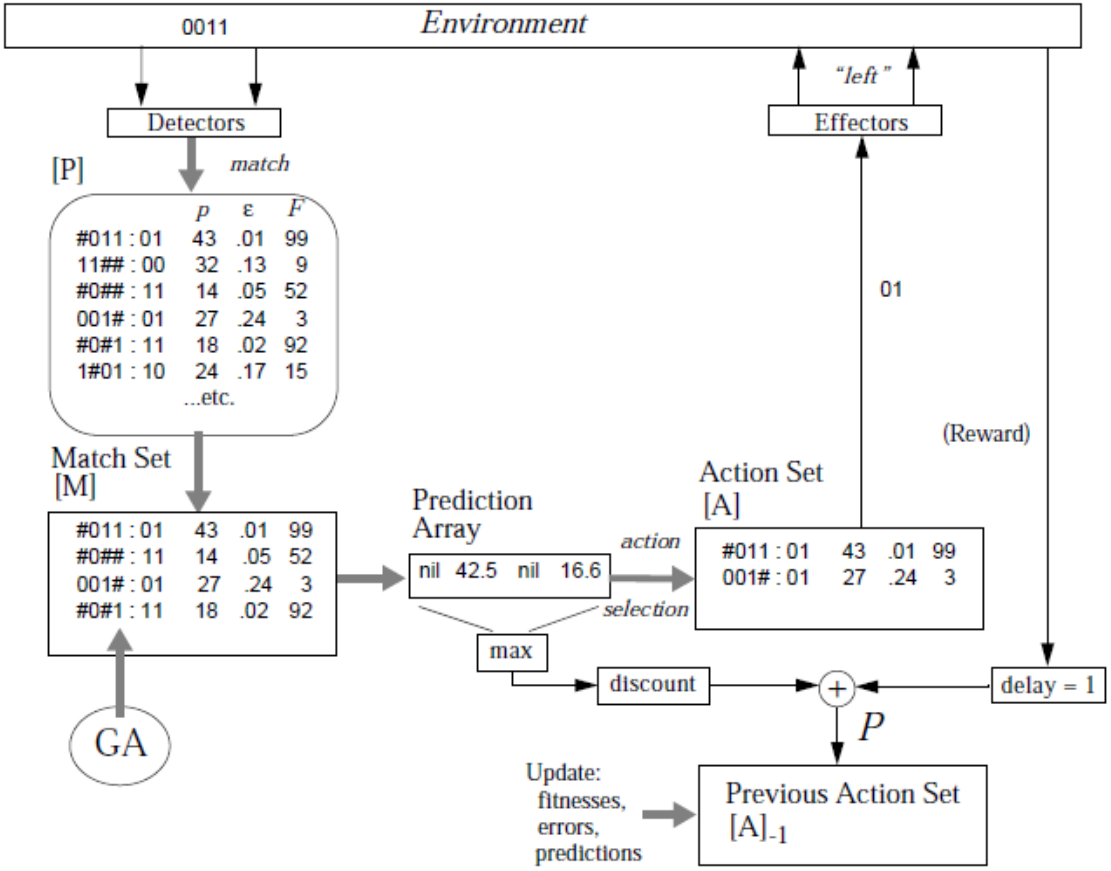
- S. Wilson. 1995. Classifier fitness based on accuracy. Evolutionary Computation



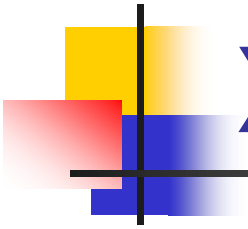
XCS: novedades principales

- El fitness de reglas pasa a fundamentarse en la precisión de las predicciones sobre la recompensa
- Cubre todo el espacio del problema (no se centra en nichos con recompensas altas)
- Reemplaza el algoritmo de bucket brigade por una forma de Q-Learning

XCS: esquema


0011
Environment
"left"
Detectors
Effectors
[P]
match
p ε F
#011 : 01 43 .01 99
11## : 00 32 .13 9
#0## : 11 14 .05 52
001# : 01 27 .24 3
#0#1 : 11 18 .02 92
1#01 : 10 24 .17 15
...etc.
01
(Reward)
Match Set
[M]
#011 : 01 43 .01 99
#0## : 11 14 .05 52
001# : 01 27 .24 3
#0#1 : 11 18 .02 92
Prediction
Array
nil 42.5 nil 16.6
action
selection
Action Set
[A]
#011 : 01 43 .01 99
001# : 01 27 .24 3
max
discount
delay = 1
P
GA
Update:
fitnesses,
errors,
predictions
Previous Action Set
[A]-1

XCS: parámetros de las reglas



- **p** Refuerzo previsto
- **ε** Error de predicción
- **F** Fitness (función inversa del error de predicción)
- **n** Numerosity
- **a** Estimación de tamaño de nicho
- Marca de tiempo de la última activación del GA

XCS: refuerzo

1. Se actualiza el error de cada regla (ε) usando la regla Delta de Widrow-Hoff

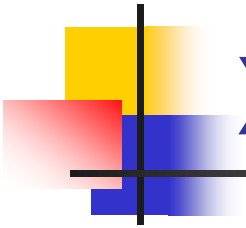
$$\varepsilon \leftarrow \varepsilon + \beta\,(\,|R - p| - \varepsilon\,)$$

2. Se ajusta el valor de la predicción

$$p \leftarrow p + \beta\,(R + \gamma\, max(P_{[A]+1}) - p)$$

3. Se calcula el valor de la precisión (*k*)

$$\begin{cases} 0.1(\varepsilon/\varepsilon_0)^{-\alpha} & \text{Si } \varepsilon > \varepsilon_0 \\ 1 & \text{Si } \varepsilon \leq \varepsilon_0 \end{cases}$$



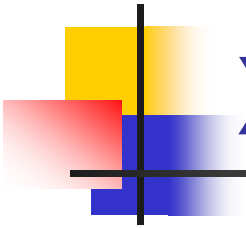
XCS: refuerzo

4. Se calcula el valor de la precisión relativa (k')

$$K' = k / \sum k$$

5. Se ajusta el fitness

$$F \leftarrow F + \beta(k' - F)$$

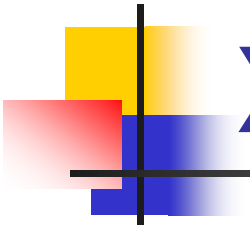


XCS: aplicación del GA

- El fitness de las reglas se fundamenta en precisión de las predicciones de los refuerzos esperados
- El sistema de eliminación de reglas depende del tamaño de los nichos
- Esto crea una base de reglas precisa y equilibrada

XCS: parámetros

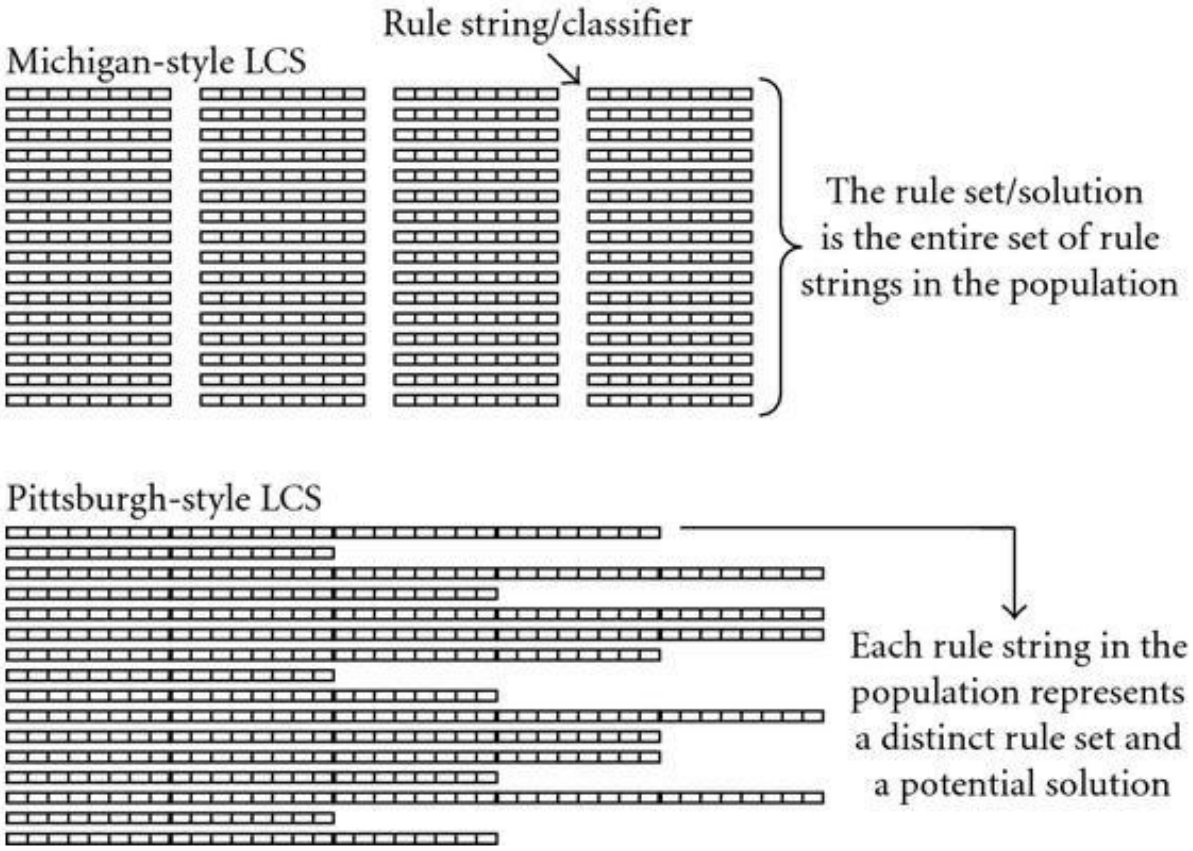
- **N** Tamaño de la población
- β Tasa de actualización para las predicciones, el error de predicción y el fitness
- γ Factor de descuento
- **Θ** Límite de iteraciones que sin uso de GA sobre un [M] que lanza su ejecución
- $\varepsilon_0, \propto$ Parámetros de la función de precisión
- Probabilidad de cruce en el GA
- μ Probabilidad de mutación por alelo en GA
- **X** Parámetro que controla la regla a reemplazar por el GA
- **ϕ** Si la fuerza total de [M] es menor que ϕ veces la media de [P], usar covering
- $\mathbf{P_\#}$ Probabilidad de introducir alelo # en la población inicial o en covering
- $\mathbf{S_0}$ Fuerza asignada a cada clasificadoren la población inicial
- τ Tasa de penalización para clasificadores de [M] que no estén en [A]
- **P-, ε-, F-** Predicción, error de predicción y fitness asignados a cada

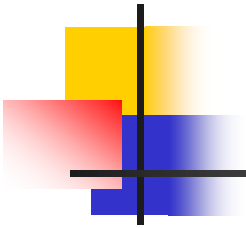


XCS: resultados

- Buenos resultados tanto en problemas teóricos como en problemas de mundo real
- Soluciones generales
- Particularmente adecuados para conjuntos grandes de datos
- Se han propuesto codificaciones que permiten usar reales
- Pueden usarse en forma de ensembles

LCS: Michigan vs Pittsburgh





LCS: conclusiones

- Área de investigación que está resurgiendo
- Gran variedad de variantes y aplicaciones
- Apropiada en minería de datos
- Buenas propiedades de modelado del espacio del problema
- Reglas inteligibles