Algoritmo evolutivo basado en reglas locales para resolución de objetivos sobre agentes en un entorno simulado

Evolutionary algorithm based on local rules for targets resolution over agents in a simulated environment

Arles Rodríguez Ing. & Jonatan Gómez Ph.D.
Dpto. Ingeniería de Sistemas e Industrial, Universidad Nacional de Colombia
Grupo de Investigación Alife
aerodriguezp@unal.edu.co, jgomezpe@unal.edu.co

Recibido para revisión 02 de septiembre de 2010, aceptado 03 de enero de 2011, versión final 09 de febrero de 2011

***Resumen*— Este artículo muestra la construcción y evolución de un entorno multiagente. Los agentes están ejecutando programas de movimiento buscando objetivos en un mundo que tiene objetivos aleatorios y temperaturas distribuidas en el espacio. Algunas temperaturas son letales para los agentes. Un algoritmo evolutivo evalúa y selecciona las acciones de los agentes en el mundo virtual basado en las percepciones locales de cada agente. Se definen programas básicos de movimiento derivados de un lenguaje simple para representar las direcciones de movimiento y son codificados en un genotipo binario de 9 bits (los 3 próximos movimientos). La función de evaluación está basada en interacciones locales y evalúa la proximidad de los programas de movimiento al objetivo y acciones para prevenir la muerte del agente por calor. En cada hilo de ejecución del agente, el algoritmo evolutivo se ejecuta en múltiples ocasiones hasta alcanzar el objetivo. Como resultado, los agentes alcanzan el objetivo rápidamente y al mismo tiempo evitan puntos de temperatura letal.**

***Palabras Clave*— Algoritmos evolutivos, simulación por computador, Resolución de objetivos, sistemas multiagente, evolución de programas de movimiento.**

***Abstract*— This paper discusses the construction and evolution of a multiagent environment. The agents are running movement programs looking for objectives in a world that has random targets and temperatures distributed in the space. Some temperatures are lethal for agents. An evolutionary algorithm evaluates and sets the agents actions in the virtual world based on local perceptions of each agent. We define basic movement programs, which are generated from a simple language to represent directions and we encode the programs in a binary genotype of 9 bits (3 next movements). The fitness function is based on local interactions and assesses the proximity of movement programs to the goal and actions to prevent death of the agent by the heat. In each agents thread, the evolutionary algorithm is executed several times until reach the target. As result, the agents reach the target quickly and at the same time, they avoid points of lethal temperature.**

***Keywords*— Evolutionary algorithms, computer simulation, target resolution, multiagent systems, movement programs evolution.**

I. INTRODUCCIÓN

Los comportamientos cooperativos entre sistemas (entornos, mundos artificiales) de múltiples agentes son un área de investigación prominente. Como expuso Yannakakis[15], esta tarea es difícil dadas las definiciones del entorno. Un entorno multiagente es dinámico, no determinístico y los agentes no tienen conocimiento de todas las condiciones del entorno. Saito, M y Hatanaka, exponen problemas de búsqueda en los que un grupo de agentes localiza un objetivo en una posición aleatoria por un intervalo de tiempo a través de macro-ordenes y micro-ordenes mostrando toda una formulación matemática y la convergencia del planteamiento dado[11]. Polycarpou y Yang propusieron una aproximación de búsqueda cooperativa de objetivos utilizando simulaciones por computador y aprendizaje distribuido[10]. Una gran gama de trabajos en el área de inteligencia de enjambres pueden encontrarse en la literatura con aplicaciones como Optimización por Colonias de Hormigas[3,6] o inteligencia de enjambres[12]. También se presentan trabajos de búsqueda de objetivos que emplean un filtro Bayesiano para predecir funciones de densidad de probabilidad[4,5,13,14] y otras propuestas basadas en el tema de comunicación entre los agentes[1,2].

Revista Avances en Sistemas e Informática, Vol.8 No.1, marzo de 2011 - Medellín. ISSN 1657-7663

En nuestro caso definimos un mundo virtual en una forma sencilla y como tarea para los agentes la localización de objetivos aleatorios alrededor del mundo. Los agentes conocen la posición del objetivo pero no conocen el camino para alcanzarlo y el entorno posee obstáculos y puntos letales que dificultan la tarea de búsqueda. Para lograr esto se evolucionan programas de movimiento utilizando al algoritmo HAEA, propuesto por Jonatan Gómez[7]. Nuestra aproximación pretende resolver éste problema de búsqueda en una forma simple y sin una formulación matemática estricta, a través de la ejecución de pequeños algoritmos evolutivos que se ejecutan en cada hilo de ejecución de un agente.

II. DESCRIPCIÓN DE LOS AGENTES

Un agente es definido como un objeto que ocupa un lugar en el mundo y posee una velocidad y una representación gráfica. Los agentes tienen la misión de alcanzar un objetivo que es definido como una posición en el mundo contando con cinco posibles movimientos: no hacer nada (0), abajo (1), izquierda (2), derecha (3) y arriba (4). Dichos agentes, conocen la posición de los objetivos pero no saben cómo alcanzarlos. Los agentes además tienen un sensor que obtiene la temperatura en sus respectivas vecindades.

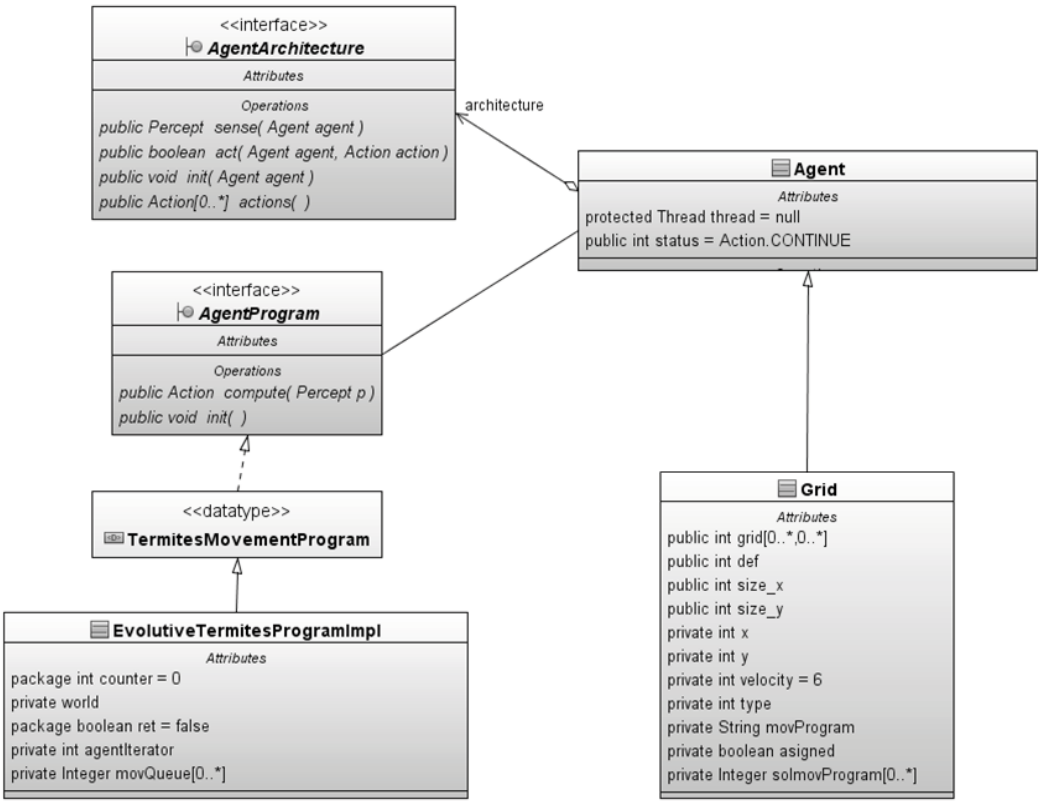


Figura 1. Diagrama de clases de la librería de agentes

Cada agente tiene su propio hilo de ejecución, en el cual se ejecutan programas de movimiento derivados de un algoritmo evolutivo sobre el conjunto de instrucciones básicas. Los agentes tienen percepciones sobre su vecindad para realizar acciones en el mundo virtual. Éste diseño permite ejecutar algoritmos evolutivos para generar instrucciones de movimiento que se escriben en una cola de instrucciones. El diseño de agentes es propuesto e implementado en una librería de J. Gómez (Figura 1).

III. DISEÑO DEL MUNDO

El mundo es un espacio toroidal de dos dimensiones con un tamaño definido (ancho y largo), un vector de agentes y otro de objetivos. El mundo controla las colisiones impidiendo que dos agentes o elementos puedan estar en la misma posición. Adicionalmente, invoca el método de operación de cada hilo de ejecución por agente y actualiza la posición de los agentes en el mundo.

Las condiciones físicas del mundo se establecen en un arreglo de memoria de estados. El arreglo de memoria de estados define temperaturas utilizando un generador gaussiano de números aleatorios con una media de 0.5 y una desviación estándar de 0.2. Un agente muere si se encuentra sobre una posición con una temperatura mayor a 0.7. La tabla 1 muestra un resumen de los componentes del mundo.

Tabla 1. Componentes del mundo

Componente	Descripción
Vector de Agentes	Estructura de datos que almacena agentes.
Vector de Objetivos	Contiene las localizaciones de los objetivos.
Arreglo de memoria de estados	Agrupa valores de estado para el mundo. Es útil para definir estado de los agentes en el mundo. Utilizado para controlar colisiones, y almacenar el estado de las condiciones del mundo como temperatura.

IV. METODOLOGÍA

Se define un lenguaje básico de movimientos que genera programas de movimiento. Un programa de movimiento es un vector de posibles movimientos. Un agente en la posición (3,5) ejecutando el programa: up|up|left|left, experimentará los siguientes cambios en sus posiciones: up (3,4), up (3,3), left (2,3) left (1,3).

Los agentes son cuadros blancos y buscan por objetivos (cuadros verdes). Todos los cuadros de color rojo y negro representan temperaturas en el mundo. Un cuadro negro representa una temperatura mortal para un agente (Figura 2).

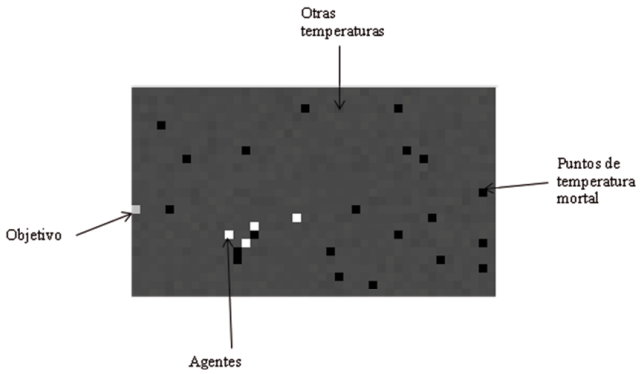


Figura 2: Interfaz del mundo virtual

4.1. Algoritmos evolutivos para la generación de movimientos

Los Algoritmos Evolutivos (EA) son técnicas de optimización basadas en los principios de la evolución natural[8]. Para evolucionar los programas de movimiento se utilizó el algoritmo HAEA propuesto por J. Gómez[7] (Algoritmo 1). HAEA adapta las probabilidades (tasas) de los operadores mientras evoluciona la solución del problema. Estas tasas dan un criterio para seleccionar únicamente un operador que es aplicado por individuo. Dichas tasas son codificadas en el individuo y son actualizadas de acuerdo al rendimiento obtenido por los descendientes (comparados con sus padres) y una tasa de aprendizaje aleatoria.

HaEA(λ, terminationCondition)

```
1. t0 = 0
2. P0 = initPopulation(λ) ,
3. while ( terminationCondition(t, Pt ) is false) do
4. Pt+1 = {}
5. for each ind ∈ Pt do
6.  rates = extract_rates( ind )
7.  δ = random(0,1) //learning rate
8.  oper = Op_Select(operators, rates)
9.  parents = ParentSelection(Pt , ind)
10.  offspring = apply(oper, parents)
11.  child = Best( offspring, ind )
12. if( fitness( child ) > fitness ( ind ) ) then
13.  rates[oper] = (1.0 + δ)*rates[oper] //reward
14. else
15.  rates[oper] = (1.0 - δ)*rates[oper] //punish
16.  normalize_rates( rates )
17.  set_rates( child, rates )
18.  Pt+1 = Pt+1 ∪ { child }
19. t = t + 1
```

Algoritmo 1: HAEA por J. Gómez

Éste algoritmo utiliza los operadores genéticos de cruce y mutación de único punto. En la mutación de único punto, un bit de la solución es seleccionado aleatoriamente (con una distribución uniforme) y es cambiado. Este operador genético siempre modifica el genotipo cambiando un sólo bit. En el cruce de único punto, un punto de corte en la solución es aleatoriamente seleccionado. Los padres son divididos en dos partes (izquierda y derecha) utilizando dicho punto de corte. La parte izquierda de un padre es combinada con la parte derecha del otro.

Para el movimiento de los agentes se decidió utilizar una aproximación basada en vida artificial en la que las interacciones locales para determinar los futuros movimientos pueden alcanzar una meta global[9]. De esta forma se tienen pequeñas ejecuciones del Algoritmo Genético por hilo de ejecución que evolucionan las tres siguientes direcciones de movimiento. Dichas direcciones se almacenan en una cola de instrucciones para cada agente.

4.2. Función de codificación para el movimiento de los agentes

Utilizamos codificación binaria para los programas de movimiento. En la codificación binaria, la solución es codificada como una cadena de símbolos binarios[8]. Cada individuo tiene 5 posibles movimientos (3 bits por movimiento). Un programa tiene 3 movimientos para un total de 9 bits como tamaño del individuo. Los movimientos están codificados como se aprecia en la tabla 2.

Tabla 2. Codificación de los posibles movimientos

Dirección	Valor	Representación Binaria
None	0	0
Down	1	1
Left	2	10
Right	3	11
Up	4	100

4.3 Función de aptitud

La función de aptitud es la parte más importante del algoritmo evolutivo pues determina cuales individuos deben continuar en la siguiente iteración. Esta función requiere las percepciones locales del agente y el programa a evaluar. Debido a la forma en la cual los agentes se están moviendo, se emplea la distancia de Manhattan para evaluar los programas. Cada programa es evaluado basado en una referencia al mundo donde se toman las percepciones locales y se asigna una solución que se aproxime en un paso o más al objetivo. Si el programa causa la muerte de un individuo, tendrá un valor de cero en su función de aptitud. Si el programa alcanza el objetivo, la función de evaluación será la suma de las dimensiones del mundo y dicho programa se asignará al agente. La función de aptitud se expone en el algoritmo 2.

Algoritmo 2. Función de aptitud

```
fitness EvaluationFunction(World, Agent)
1. fitness = 0
2. MortalTemp = 0.7
3. meanDistance = 0;
4. Program = getProgramFromGenotype()
5. PosicionInicial(x,y) = Agente.obtenerPosicionInicial
6. for each direction ∈ Program
7.   Agent.move(direction)
8. if (World.getTemperature(AgentPosX, AgentPosY) > MortalTemp) //Temperature condition
9.   return fitness
10. for each Target ∈ World.Targets
11.     manhOrigDist = Math.abs(PosicionInicial(x) - Target(x)) +
12. manhRestDist = Math.abs(AgentPosX- Target(x)) + Math.abs(AgentPosY - Target(y))
13.  if (manhRestDist == 0)
14.   Agent.setSolutionProgram(getProgramFromGenotype())
15.   fitness = World.width + World.height
16.  else if(manhOrigDist - manhRestDist) > 0)
17.   if(isShortDistance > (manhOrigDist - manhRestDist))
18.     isShortDistance = (manhOrigDist - manhRestDist)
19.     bestProgram = getProgram()
20.   if(manhOrigDist - manhRestDist) > fitness)
21.     fitness = (int) (manhOrigDist - manhRestDist)
22.     meanDistance += (manhOrigDist - manhRestDist);
23.     cont++;
24. if ( bestProgram != null ) //Some program that approaches the target
25.  meanDistance = meanDistance / cont;
26.  if (fitness <= meanDistance)
27.   Agent.setSolutionProgram(getProgramFromGenotype())
28. Agent.setPosicion(PosicionInicial(x,y))
29. return fitness
```

4.4. Condición de parada para el algoritmo genético

Para hacer que los agentes tomen una decisión rápidamente, una condición de parada para el algoritmo es definida. El algoritmo evolutivo iterará mientras un agente no tenga una solución asignada.

V. EXPERIMENTOS Y RESULTADOS

Se define un mundo virtual con temperaturas generadas aleatoriamente utilizando un generador gaussiano de números aleatorios con una media de 0.5 y una desviación estándar de 0.2. Una temperatura mayor a 0.7 es mortal para un agente. El mundo virtual tiene un área de 43x25 y el área de un agente es de 1x1 cuadros. Los experimentos se realizaron en un computador con un procesador de doble núcleo de 1.8Ghz. La aplicación es configurada para utilizar hasta 1GB de memoria. Cuando un agente alcanza el objetivo, el objetivo cambia su posición y los hilos de ejecución del agente continúan ejecutando múltiples veces el algoritmo evolutivo por agente para alcanzar el nuevo objetivo. El diseño local permite que el mundo pueda cambiar en tiempo de ejecución o que el objetivo cambie en tiempo de ejecución.

5.1. Experimentos con un agente

Se realizan algunas ejecuciones con cinco objetivos y un agente. El agente siempre esquiva los lugares con temperaturas altas y alcanza el objetivo. Cuando un agente alcanza un objetivo, se genera otro objetivo en una posición aleatoria en el mundo. Un objetivo siempre tiene una temperatura de 0.5.

5.1.1. Tiempos de respuesta para objetivos lejanos

Doscientos objetivos son alcanzados en múltiples ejecuciones, los agentes y los objetivos se colocaron en posiciones opuestas en el mundo (la distancia más lejana). Se obtuvo un tiempo medio de 270 ms con una desviación estándar de 88.71 ms. Para todas las ejecuciones el agente alcanza el objetivo (Figura 3).

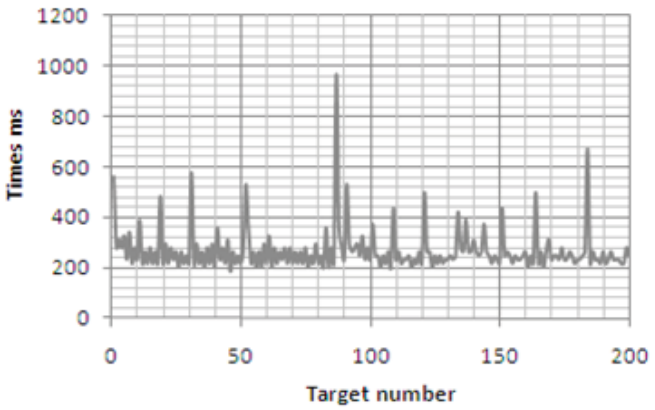


Figura 3. Resultados para un agente y un objetivo lejano en 200 ejecuciones

5.1.2. Tiempos de respuesta para objetivos cercanos

Para un objetivo cercano se definió una región de 10x6 cuadros. Se realizaron 200 ejecuciones para una porción del mundo de éste tamaño. El tiempo medio obtenido para alcanzar el objetivo es de 23.435 ms con una desviación estándar de 19.23 ms (Figura 4).

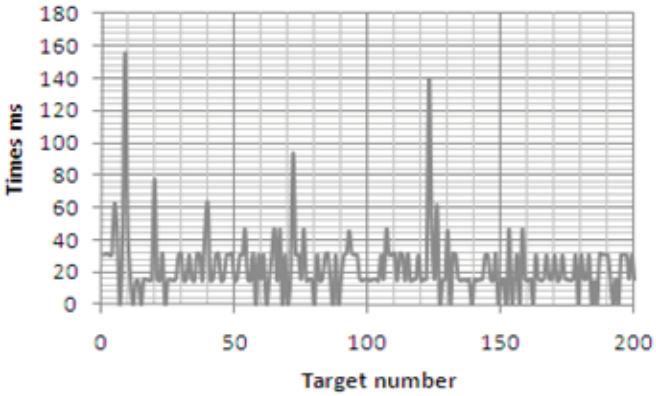


Figura 4. Tiempos de respuesta para un agente, un objetivo cercano en 200 ejecuciones

5.1.3. Tiempos de respuesta para un objetivo aleatorio en el mundo

Para doscientos objetivos localizados aleatoriamente en el mundo, se obtuvo un tiempo medio de 101.82 ms con una desviación estándar de 69.9 ms (Figura 5).

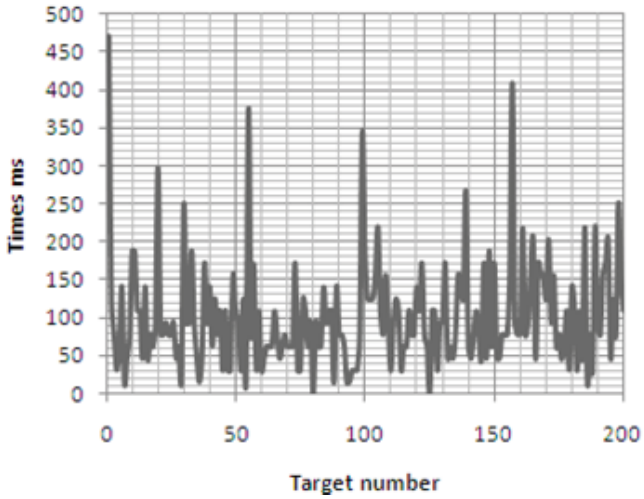


Figura 5. Tiempos de respuesta de un agente a un objetivo en posición aleatoria.

5.1.4. Tiempos de respuesta para un agente y múltiples objetivos

Para 5 objetivos y un agente se tiene un tiempo medio de 1063.7 ms, con una desviación estándar de 1167.71 ms. El agente presenta oscilaciones en direcciones opuestas cuando tiene objetivos cercanos en direcciones opuestas, pasado algún tiempo el agente alcanza los objetivos (Figura 6).

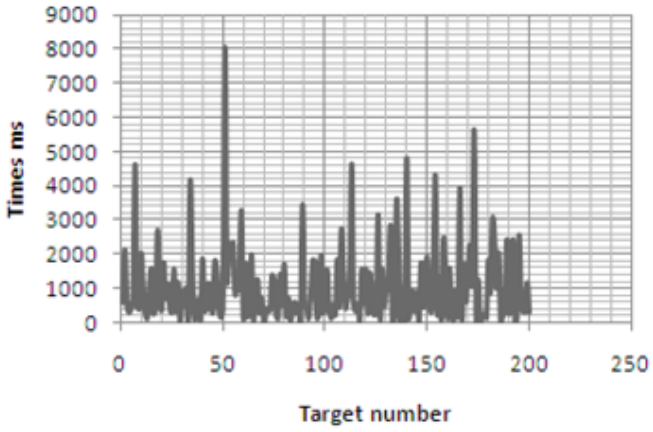


Figura 6. Tiempos de respuesta de un agente a 5 objetivos en posiciones aleatorias

5.2. Experimentos con varios agentes

Los siguientes experimentos son realizados con uno y cinco objetivos para 10 agentes. Los agentes evitan lugares con altas temperaturas exitosamente. Dado que el mecanismo de control de colisiones es controlado por el mundo ignorando el movimiento actual, en algunos casos los agentes pueden morir.

Revista Avances en Sistemas e Informática, Vol.8 No.1, marzo de 2011 - Medellín. ISSN 1657-7663

5.2.1. Tiempos de respuesta para objetivos lejanos

Doscientos objetivos son alcanzados en diferentes ejecuciones, diez agentes y un objetivo son colocados en direcciones opuestas en el mundo (distancia más lejana). Se tiene un tiempo medio en alcanzar el objetivo de 1982.45 ms, con una desviación estándar de 2115.58 ms. En todas las ejecuciones los agentes alcanzan el objetivo (Figura 7).

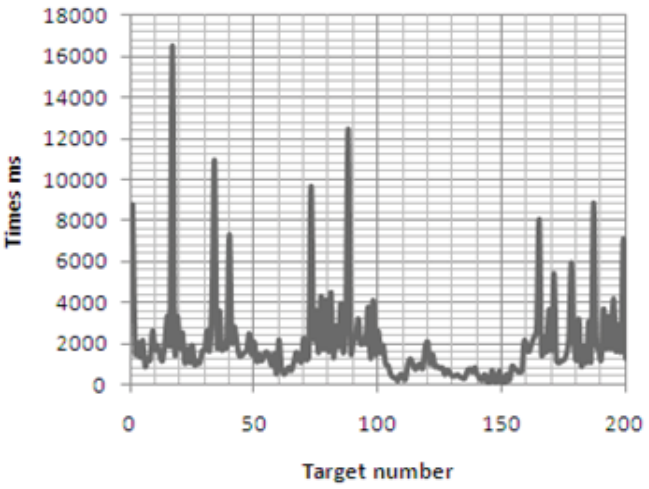


Figura 7. Tiempos de respuesta de muchos agentes a un objetivo lejano.

5.2.2. Tiempos de respuesta para un objetivo cercano

Para un objetivo cercano se definió una región de 10x6 cuadros. Se realizaron 200 ejecuciones para una porción del mundo de éste tamaño. El tiempo medio para alcanzar un objetivo es de 598.209 ms con una desviación estándar de 973.75 ms (Figura 8).

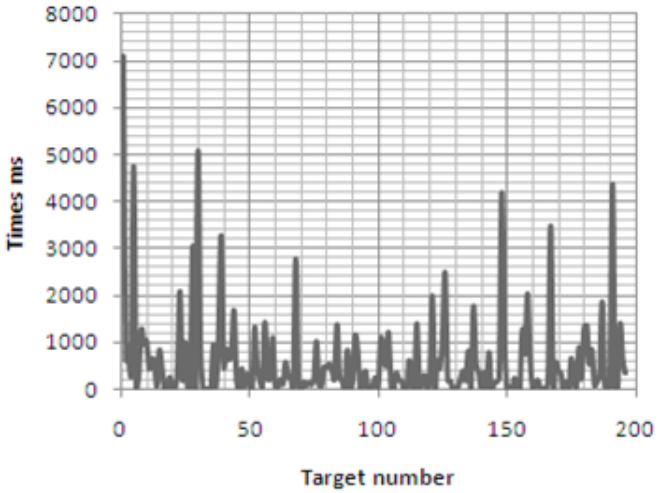


Figura 8. Tiempos de respuesta de 5 agentes a un objetivo cercano

5.2.3. Tiempos de respuesta para un punto aleatorio en el mundo

Para doscientos objetivos localizados aleatoriamente en el mundo y diez agentes, se tiene un tiempo medio de 1290.025 ms con una desviación estándar de 1812.30 ms (Figura 9).

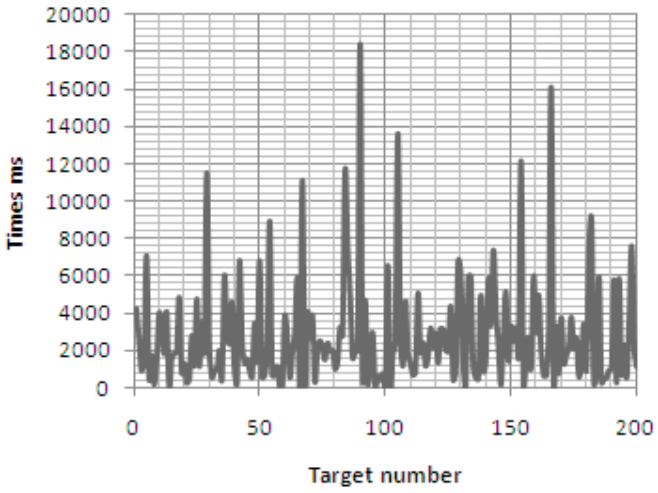


Figura 9. Tiempos de respuesta de 5 agentes a un objetivo aleatorio

5.2.4. Tiempos de respuesta para múltiples agentes en ubicaciones aleatorias

Para 5 objetivos y 10 agentes, se obtuvo un tiempo medio de 2737.825 ms con una desviación estándar de 2831.071 ms (Figura 10).

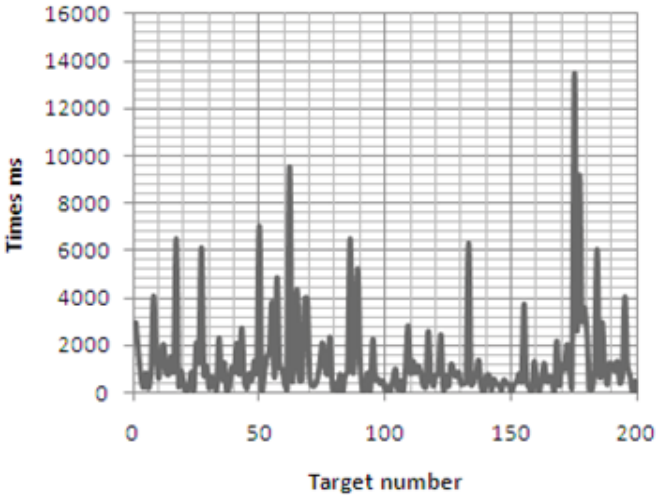


Figura 10. Tiempos de respuesta de 10 agentes a 5 objetivos

VI . CONCLUSIONES

Se ha mostrado una forma simple y eficiente de evolucionar programas de movimiento así como de diseñar un entorno simulado para los agentes. Modelando cada agente de una forma local se obtienen buenos tiempos de respuesta especialmente cuando los agentes ubican un solo objetivo. El planteamiento permite cambios dinámicos al mundo en la parte de distribución de temperaturas, tamaño del mundo y localizaciones de los objetivos con buenos resultados.

Como trabajo futuro, se tiene pensado mejorar la función de aptitud para reducir los tiempos de respuesta en múltiples objetivos. Con más de un objetivo la función de aptitud actual causa oscilaciones en los movimientos de los agentes, especialmente en mundos que tienen dos objetivos en direcciones opuestas. Sin embargo, los agentes alcanzan los objetivos exitosamente después de algún tiempo. Los tiempos de respuesta se resumen en la tabla 3.

Tabla 3. Resultados de experimentos

Número de Agentes	Experimento	#objetivos	Tiempo Medio (ms)	Desv. Estandar T. (ms)
1	Objetivos lejanos	1	270	88,71
1	Objetivos Cercanos	1	29,13	19,23
1	Objetivos en posición aleatoria	1	101,82	69,9
1	Objetivos en posición aleatoria	5	1063,7	1167,71
10	Objetivos lejanos	1	1982,45	2115,58
10	Objetivos Cercanos	1	598,209	973,75
10	Objetivos en posición aleatoria	1	1290,025	1812,3
10	Objetivos en posición aleatoria	5	2737,82	2831,071

REFERENCIAS

[1] Arkin, R. C., 1992. Cooperation without communication: Multiagent schema-based robot navigation. En: Journal of Robotic Systems, Vol. 9(3), pp. 351-364.

[2] Balch, T. and Arkin, R. C., 1994. Communication in reactive multiagent robotic systems. Autonomous Robots, Vol. 1, No. 1, pp. 27-52.

[3] Bonabeau, E., Dorigo, M. and Theraulaz G., Swarm Intelligence: From Natural to Artificial Systems. NY: Oxford Univ. Press, 1999.

[4] Bourgault, F., Furukawa, T. and Durrant-Whyte, H., 2003. Coordinated decentralized search for a lost target in a bayesian world. Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ (Oct. 2003), Vol.1, pp. 48-53.

[5] Bourgault, F., Furukawa, T. and Durrant-Whyte, H., 2004. Process model, constraints, and the coordinated search strategy. Robotics and Automation, 2004. Proceedings. IEEE ICRA '04. 2004 (Abr. 2004), Vol.5., pp. 5256-5261.

[6] Garro, B., Sossa, H. and Vazquez, R., 2006. Path planning optimization using bio-inspirited algorithms. Artificial Intelligence. MICAI '06. (Nov. 2006), pp 319-330.

[7] Gómez J. 2004. Self Adaptation of Operator Rates in Evolutionary Algorithms. (2004) In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Lecture Notes in Computer Science, (Jun. 2004) Vol. 3102, pp 1162-1173.

[8] Holland, J., 1992. Adaptation in Natural and Artificial Systems. Cambridge: MIT Press.

[9] Langton, C. G., 1989. Artificial Life: proceedings of an interdisciplinary workshop on the Synthesis and Simulation of Living Systems. Santafe Institute: Westview Press. 688 p.

[10] Polycarpou, M., Yang, Y. and Passino, K. A Cooperative Search Framework for Distributed Agents. 2001. In Proceedings of the 2001 IEEE International Symposium on Intelligent Control (ISIC '01). (Sept 2001) pp. 1-6.

[11] Saito, M., Hatanaka, T. and Fujita, M. 2010. Order formations in multi-agent search problem: A game theoretic approach. American Control Conference (ACC 2010), (jun. 2010), pp. 4768-4773.

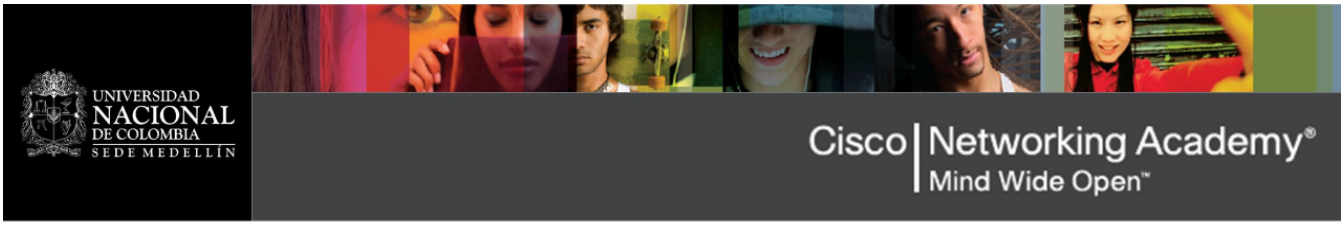
[12] Spires, S. V. and Goldsmith, S. Y. 1998. Exhaustive geographic search with mobile robots along space-filling curves. In CRW '98: Proceedings of the First International Workshop on Collective Robotics (London, UK, 1998), Springer-Verlag, pp. 1-12.

[13] Stone, L., Barlow, C. and Corwin, T., 1999. Bayesian Multiple Target Tracking. Mathematics in Science and Engineering. Boston: Artech House. 317 p.

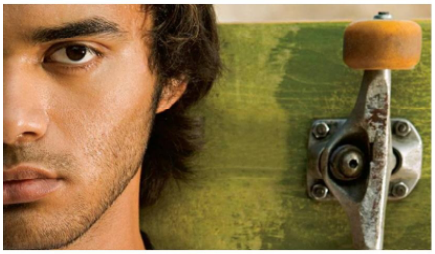
[14] Wong, E., Bourgault, F. and Furukawa, T. 2005. Multi-vehicle Bayesian search for multiple lost targets. Robotics and Automation, 2005. IEEE ICRA 2005. Proceedings. (abr. 2005), pp. 3169-3174.

[15]Yannakakis, G. N., Hallam, J. and Levine, J. 2005. Evolutionary computation variants for cooperative spatial coordination. In Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC 05) (September 2005). pp. 2715-2722.

Revista Avances en Sistemas e Informática, Vol.8 No.1, marzo de 2011 - Medellín. ISSN 1657-7663



Cisco Networking Academy es un programa ampliamente conocido de e-doing que enseña a los estudiantes las habilidades tecnológicas de Internet en una economía global. El programa proporciona contenido basado en la Web, pruebas en línea, seguimiento del desempeño de los estudiantes, laboratorios con equipos reales y con simuladores, soporte y entrenamiento por parte de los instructores, así como preparación para las certificaciones estándares de la industria.



Oferta de cursos

- Mantenimiento de PC: IT Essentials
- Redes básicas: Cisco Certified Network Associate
- Redes avanzadas: Cisco Certified Network Professional
- Seguridad en routers: CCNA Security
- Voz sobre IP
- Asterisk básico

Programación 2011

Ciclo 48: Inicia 17 de enero. Finaliza 12 de marzo
Ciclo 49: Inicia 22 de marzo. Finaliza 23 de mayo
Ciclo 50: Inicia 30 de mayo. Finaliza 29 de julio
Ciclo 51: Inicia 8 de agosto. Finaliza 3 de octubre
Ciclo 52: Inicia 10 de octubre. Finaliza 12 de diciembre

Consulte los horarios de cada nivel a través de nuestros canales informativos al pie de página

Además...

- Alquiler de laboratorios virtuales para auto-estudio o cursos empresariales
- Presentación de exámenes de certificación para multiples areas bajo el Centro Pearson VUE
- Cursos exclusivos para su empresa
- Pregunte por nuestros descuentos

CATC - Academia Regional - Academia Local
Universidad Nacional de Colombia sede Medellín
Calle 65 78-28 Bloque M1 Oficina 101. Facultad de Minas
Teléfono: +57 4 4255268 Fax: +57 4 2341002 E-mail: **catc@unal.edu.co** Web: **http://cnap.unalmed.edu.co**
Facebook: **fb.me/catcunal** Twitter: **@catcunal** Buzz: **google.com/profiles/catcunal**
Medellín, Colombia