Procesos de decisión de Markov y microescenarios para navegación y evasión de colisiones para multitudes

Sergio Ruiz[1], Benjamín Hernández[2]

[1] Tecnológico de Monterrey,
Campus Ciudad de México, México

[2] Barcelona Supercomputing Center,
Barcelona, España

sergio.ruiz.loza@itesm.mx, benjamin.hernandez@bsc.es

Resumen. La simulación en tiempo real de multitudes ha sido relevante en aplicaciones relacionadas a la planificación urbana, evacuaciones, entrenamiento para el control de multitudes y entretenimiento. Problemas fundamentales como la navegación de agentes y evasión de colisiones deben ser resueltos eficientemente para obtener una simulación interactiva. Este trabajo propone un método integrado que hace uso de Procesos de Decisión de Markov para el cálculo de trayectorias óptimas que los agentes deberán seguir y de Microescenarios, técnica geométrica basada en curvas Bézier, que permite la evasión de colisiones entre los agentes y objetos dinámicos. Se reportan resultados usando estas técnicas para la simulación interactiva de multitudes en ambientes virtuales.

Palabras clave: Procesos de decisión de Markov, simulación de multitudes.

1. Introducción

Los seres humanos constituyen una de las especies más complejas en términos físicos y conductuales [12,33], en consecuencia la simulación de multitudes representa un reto considerable, y aunque los primeros métodos[3] para lograr dicha simulación surgieron a mediados de los noventa [33], la alta demanda de recursos computacionales necesarios para la simulación de multitudes en tiempo real, sigue motivando el desarrollo de algoritmos novedosos de alto rendimiento.

Por otro lado, la imposibilidad de aplicar el método científico a ciertos subconjuntos del fenómeno de las multitudes, aunado a los costos prohibitivos de logística y económicos relacionados a la dirección de eventos experimentales, hace inminente el uso de una alternativa virtual: una simulación.

La componente principal de toda simulación de multitudes en tiempo real es la síntesis de comportamiento; para que una simulación de multitudes sea precisa es necesario que reproduzca comportamientos humanos individuales y grupales, además los algoritmos que sintetizan estos comportamientos deben

[3] Mientras que el primer análisis conductual realizado por Le Bon [14] data de 1896.

estar optimizados de tal forma que funcionen en tiempo real. Existen dos problemas fundamentales a resolver para reproducir comportamientos individuales y grupales de forma exitosa: evasión de colisiones y navegación. Este trabajo presenta tres contribuciones: primero, el uso de Procesos de decisión de Markov (MDP, por sus siglas en inglés) para la navegación de multitudes en un ambiente virtual, segundo el uso de microescenarios, una técnica geométrica basada en curvas Bézier, para la evasión de colisiones y finalmente se presentan resultados de rendimiento de los algoritmos que han sido desarrollados usando programación paralela en el procesador gráfico (GPU por sus siglas en inglés) para obtener rendimiento en tiempo real.

2. Trabajo previo

2.1. Evasión de colisiones

La evasión de colisiones se refiere a los movimientos anticipados de un agente para evitar colisionar con otros agentes presentes. Los métodos principales para la evasión de colisiones son bandada [25], fuerzas sociales [10] y obstáculos con velocidades recíprocas [4] con sus respectivas extensiones.

Bajo condiciones no óptimas, conforme el número de agentes (denotado por N) crece, la complejidad del algoritmo es $O(N^2)$, es decir, la posición de cada agente debe compararse con cada una de las demás. Los trabajos que se citan a continuación abordan el problema de reducir esta complejidad mediante tres enfoques, según Azma et al.[21]: el primero es adaptar la técnica al modelo de programación del GPU, el segundo enfoque es reducir el orden de complejidad del algoritmo de evasión de colisiones y finamente el tercero es adoptar una técnica de aprendizaje máquina.

Para reducir la complejidad de los algoritmos de evasión de colisiones, Bonner y Kelley [7] proponen un método jerárquico basado en aproximaciones sucesivas esféricas (SSA por sus siglas en inglés), en el cual una jerarquía de reglas es usada heurísticamente para obtener rutas libres de colisiones en un ambiente estructurado. Por otro lado, Li et al. [16] proponen la transformación del problema de evasión de colisiones de 3D a 2D mediante la proyección de secciones de la escena a distancias predefinidas, para después hacer pruebas de colisiones en cada una de las sub-secciones. En este sentido, Bing et al. [5] propone un algoritmo en GPU basado en la división del espacio y utiliza elipses acotantes para reducir la complejidad de las pruebas de colisión. En otro método para reducir la complejidad inherente en los algoritmos de evasión de colisiones, Reynolds [25,24] implementa una cuadrícula que particiona el espacio navegable, de esta forma reduce el espacio de búsqueda de colisiones a nivel cuadrícula y lo simplifica aun más definiendo un radio de búsqueda por cada agente.

Aunque no es adecuada para multitudes a gran escala, debido a sus altos requerimientos computacionales y su enfoque en evasión de colisiones del tipo vehículo-peaton, Fernández et al. [19] proponen una base de conocimiento (lo cual implica una etapa de entrenamiento y depuración), una etapa de percepción, planeación y ejecución de una arquitectura de control para la evasión de

colisiones. Otra contribución relevante en el área de Aprendizaje Máquina es de Li et al. [15]. Los autores proponen una base dinámica de reglas, un motor de inferencia y una base de conocimiento para la evasión de colisiones entre agentes que representan embarcaciones.

Van den Berg et al. [3] introducen el concepto de "evasión de colisiones óptimas y reciprocas"(ORCA por sus siglas en inglés), extendiendo su método "obstáculos con velocidades recíprocas"(RVO por sus siglas en inglés) [4]. Mientras que Guy et al. [9] proponen una implementación paralela en CPU del método RVO original.

2.2. Navegación

La navegación es la habilidad de los agentes para evitar colisiones efectivamente contra los objetos de una escena mientras se dirigen hacia una meta. Estos obstáculos pueden ser dinámicos o estáticos, pero el criterio que los califica como tales es el hecho de que no evitan colisiones, o en otras palabras, no presentan comportamiento reactivo.

Los algoritmos clásicos de planeación de rutas como A^* o Dijkstra han sido utilizados intensivamente en video juegos cuando los obstáculos son fijos y el número de agentes permite al CPU efectuar la simulación en tiempo real. Se han propuesto alternativas a A^* para reducir el tiempo de planeación de la ruta mediante la búsqueda incremental y heurística [31,13], o mediante la búsqueda rápida de una solución subóptima usando restricciones imprecisas al inicio y conforme avanza el tiempo, se ajustan estas restricciones para encontrar una ruta probablemente óptima para el periodo de tiempo dado [17,18].

Para simulaciones complejas en ambientes dinámicos, varios métodos modelan y resuelven la navegación de multitudes: partición basada en celdas, mallas de navegación o áreas navegables.

Los métodos de partición consisten en dividir el espacio en celdas y mediante el uso diferentes reglas o modelos matemáticos, los agentes pueden encontrar su ruta hacia una meta en la presencia de obstáculos. Además de A^*, otro ejemplo lo encontramos en los autómatas celulares [6,35,36]. Los autómatas celulares resuelven el problema de evasión de colisiones y navegación en el mismo algoritmo, sin embargo carecen de control y separación de los individuos pues todos deben seguir el flujo y dirección mayoritarios hacia una meta. Sarmady et al. [29] proponen el uso de un autómata celular con una partición fina para mejorar el realismo, sin embargo rutas poco realistas siguen estando presentes.

Otra alternativa a la navegaciíon es la definición de áreas navegables como alternativa a las particiones estructuradas basadas en celdas. Pettré et al. [22] usa diagramas de Voronoi para descomponer las áreas navegables en celdas cilíndricas empalmadas. Dichas celdas generan un grafo de navegación que permite la navegación de peatones, sin embargo no soporta escenarios dinámicos. Otra alternativa a las celdas cilíndricas es el uso de mapas para codificar áreas navegables [26], mientras que Treuille et al. [34] extienden el uso de imágenes para codificar información de planeación y comportamiento.

3. Procesos de decisión de Markov para navegación

Batty [2] define la navegación como un producto de la aleatoriedad, geometría, intenciones económicas y preferencias sociales, aun así concuerda en que la navegación humana puede ser modelada con un MDP y una componente aleatoria, además las restricciones geométricas del ambiente otorgan estructura a los movimientos. El uso de MDPs para calcular trayectorias libres de colisiones no es un concepto nuevo, se ha usado esencialmente en robótica donde el problema involucra un sólo agente o robot moviéndose en un ambiente total o parcialmente observable [8,20,32]. Sin embargo para el caso de la simulación de multitudes en tiempo real no hay mucho trabajo desarrollado. Banerjee et al. [1] ha propuesto un método de navegación de multitudes usando procesos de decisión semi-markovianos; en contraste nuestra solución propone:

- Un método de una sola capa, de esta forma se reduce el uso de memoria.
- Propone un enfoque reactivo para cambios inesperados en el ambiente, dicho enfoque simplifica las estructuras involucradas en el cálculo de adaptación de rutas y mejora la velocidad de cálculo.
- Nuestro método no está limitado por el número de objetos dinámicos presentes en la simulación, pues no agrega capas por cada obstáculo presente.
- Nuestro método calcula el MDP y la evasión de colisiones usando cómputo en el procesador gráfico (GPU por sus siglas en inglés).
- Estas mejoras permiten el uso del algoritmo en ambientes virtuales habitados con personajes detallados.

3.1. Modelado del problema

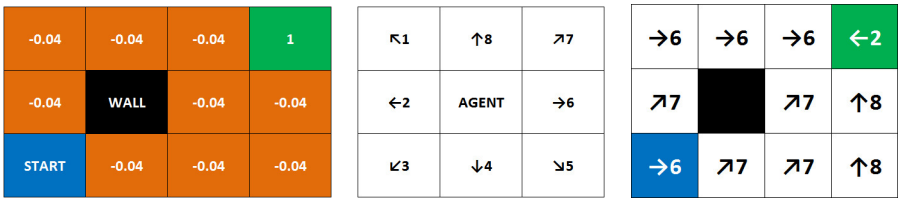
Un MDP es una tupla $M =< S, A, T, R >$, donde S es un conjunto finito de estados, A es un conjunto finito de acciones, T es una función de transición $T(s, a, s')$ y R es la función de recompensa $R(s)$. Una solución que indica lo que un agente debe hacer dado un estado es una política $\Pi(s)$.

Considerando un escenario para el cual las posiciones de los obstáculos estáticos son determinadas previo a la simulación, un MDP acotado y totalmente observable puede ser evaluado para determinar las direcciones óptimas de navegación de grupos de agentes enmarcados en celdas, dado que la representación 2D del escenario está regularmente particionada en estas celdas, en este caso, el conjunto de estados finitos S del MDP. El conjunto de dichas direcciones óptimas es la política óptima (Π^*) (Fig. 1(c)).

Nuestro escenario base —arbitrario—consiste de una cuadrícula de 3x4, que representa una distribución recompensa-penalidad para un agente (Fig. 1(a)).

Se han definido ocho posibles movimientos como el conjunto de acciones que cada agente podrá realizar a cada paso de tiempo para llegar a su meta (Fig. 1(b)).

Establecer que un agente se moverá a lo largo de la mejor ruta hacia la meta, implica el uso de una función de utilidad. En este caso, $U_h([s_0, s_1, ...s_n]) = R(s_0 + \gamma R(s_1) + \gamma^2 R(s_2)) + ... + \gamma^n R(s_n))$ representa la preferencia de un agente para



(a) Recompensas $R(s)$. (b) Acciones A. (c) Política Óptima Π^*.

Fig. 1. Modelo Base de MDP.

las recompensas actuales y futuras que corresponden a los estados del conjunto $\{s_0, ..., s_n\}$, donde el factor de descuento γ es un número en el rango $[0, 1]$. Para el modelo base de la Figura 1, a la meta se le asginó una recompensa de 1.0, a la pared una penalidad de -10.0 y a la celda inicial la misma penalidad que a las otras celdas: -0.04.

La observación de este modelo revela que varias soluciones o políticas son posibles para alcanzar la celda con la recompensa máxima (1.0), la cual representa una salida y es una meta para el agente.

Una política óptima Π^*, aquella que maximiza la función de recompensa R dada una utilidad óptima esperada de acciones futuras en un horizonte infinito, será calculada en un tiempo $t > 0$, usando la Ecuación 1.

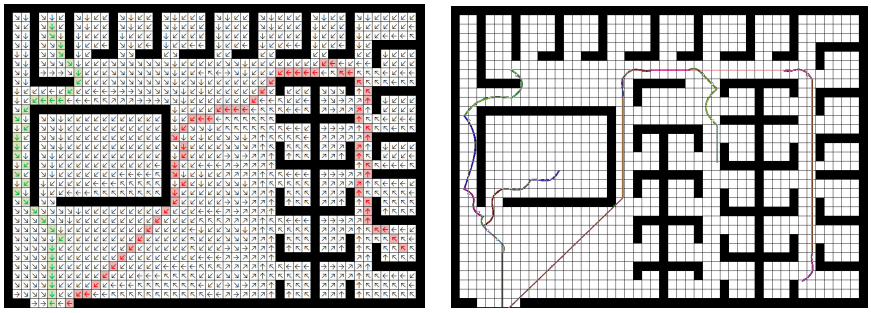
$$\begin{aligned} \Pi_t^*(s) &= \text{argmax}_a Q_t(s, a) \\ Q_t(s, a) &= R(s, a) + \gamma \sum_{j=0}^{5} T_{sj}^a V_{t-1}(j) \\ V_t(s) &= Q_t(s, \Pi^*(s)) \\ V_0(s) &= 0 \end{aligned} \tag{1}$$

Donde $Q_t(s, a)$ representa el valor de elegir la acción a—en este caso direcciones de movimiento—de la celda s; $V_t(s)$ representa el valor de la celda s en el tiempo t; $\gamma \in [0, 1]$ es el factor de descuento de la recompensa futura y T_{sj}^a es la función de transición, tomando en cuenta la probabilidad con la que un agente elige el estado j desde el estado s mediante la acción a.

La evaluación de la ecuación 1 para $t = 1$, modificará los valores de $V_0(s) = 0$ a $V_1(s), \forall s \in \{0, ..., n\}$, y cada evaluación sucesiva modificará marginalmente estos valores. Si después de la iteración x resulta que $V_{x-1}(s) = V_x(s) \forall s \in \{0, ..., n\}$ entonces la política ha convergido a su valor óptimo.

El cálculo de la política óptima (Ecuación 1) se lleva a cabo mediante la técnica iteración de valor en el GPU como se describe en un artículo previo [28].

Un escenario de prueba se muestra en la Figura 2. Note que como resultado se obtiene un arreglo de direcciones (Fig. 2(a)) que de ser usado en la etapa de visualización, los agentes se moverían siguiendo posiciones discretas de una forma poca realista. Para evitar lo anterior, se utiliza el algoritmo recursivo de Shao y Zhou [30] para obtener curvas de Bézier y suavizar las trayectorias. La Figura 2(b) muestra dos rutas suavizadas con dicha técnica.



(a) Política óptima calculada. (b) Rutas suavizadas con curvas Bézier.

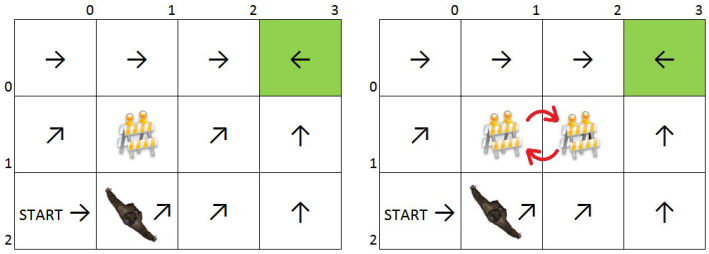
Fig. 2. Solución de un escenario de prueba.

4. Microescenarios

Una vez que a un agente se le ha asignado una ruta estática, libre de colisiones y suavizada, llegará a su meta si no se registran cambios en el escenario (Fig. 3(a)), sin embargo este caso es poco probable por dos razones:

i) La simulación de multitudes implica que los agentes compitan para ocupar el espacio navegable al momento de cada cada uno de ellos intenta llegar a su meta.
ii) No todos los escenarios son estáticos, por ejemplo, el caso de una simulación de evacuación en un terremoto.

Cuando los escenarios no son estáticos o es necesario que los agentes eviten colisiones con otros (Fig. 3(b)), proponemos el uso de micro escenarios para adaptar interactivamente las trayectorias previamente calculadas de los agentes.



(a) Escenario estático (b) Escenario dinámico.

Fig. 3. Problema de Evasión de colisiones.

Un micro escenario describe las direcciones posibles que un agente puede tomar para evitar cuerpos en movimiento (otros agentes u obstáculos). Dependiendo del número y posición de los objetos en movimiento, se puede seleccionar un micro escenario específico utilizado para recalcular en tiempo de ejecución, un segmento pequeo de la trayectoria previamente pre-calculada con el MDP. La Figura 4 muestra un microescenario de 3x3 celdas, i.e. con radio 1, los primeros dos microescenarios representan el caso cuando no hay cuerpos en movimiento cerca del agente, con dos diferentes posibilidades para una meta dada. El último microescenario, representa el caso cuando un agente es rodeado por cuerpos en movimiento.

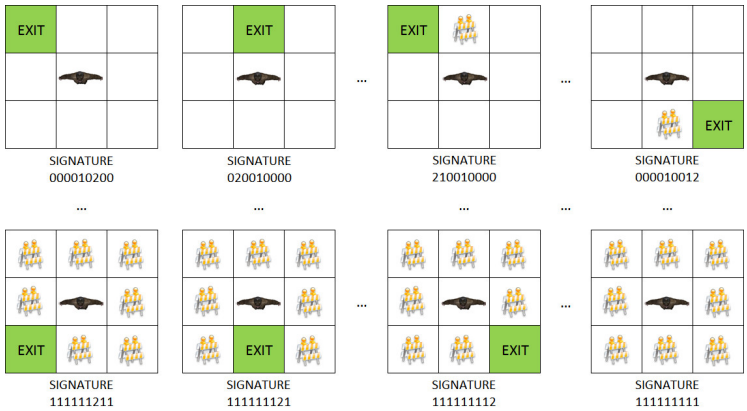


Fig. 4. Microescenarios posibles de radio 1 y sus respectivas firmas.

De esta forma el algoritmo para utilizar los microescenarios se describe a continuación:

1. En tiempo de pre procesamiento, se obtienen las trayectorias libres de colisiones suavizadas a partir del cálculo de MDP y curvas de Bézier.
2. También en etapa de pre procesamiento, se resuelve cada microescenario posible considerando que el agente está localizado en la celda del centro. Como resultado de este pre proceso se obtienen dos arreglos de información:
   i) Un arreglo único de curvas Bézier (etiquetadas como "Bézier.[en] la Figura5) para se empleado en la simulación.
   ii) Un arreglo de "firmas"(etiquetado como "Signatures.[en] la Figura 5) que relaciona cada micro escenario al conjunto de curvas Bézier ("Bézier-Signature Indirect Index.[en] la Figura 5). Cabe mencionar que para estas firmas, es indiferente si los cuerpos son obstáculos o son otros agentes.
3. En tiempo de ejecución, el algoritmo 1 es ejecutado en el GPU, entonces los agentes se desplazarán en sus trayectorias originalmente asignadas o ajustarán su ruta utilizando el microescenario si la próxima celda está ocupada.

```
for agent = 1 → N do
    bezierIndex = signature[agent]
    if bezParam[agent] == 0 then
        if occupancy[agentCell[agent]] == 1 then
            aSignature = readMicroScenario( agentCell[agent] )
            bezierIndex = BezierSignature[aSignature]
            signature[agent] = bezierIndex
        end if
    end if
    position[agent] = evalBezier( bezParam[agent], bezierIndex )
    bezParam[agent] += bezInc[agent]
    if bezParam[agent] > 1 then
        bezParam[agent] = 0
    end if
    occupancy[agentCell[agent]] = 0
    agentCell[agent] = positionToCell( position[agent] )
    occupancy[agentCell[agent]] = 1
end for
```

Algorithm 1: Algoritmo para modificar trayectorias a partir de los microescenarios.

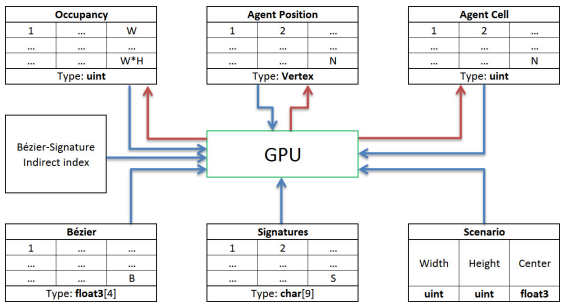


Fig. 5. Arreglos utilizados en el algoritmo propuesto.

5. Resultados

Se han diseñado tres casos para evaluar nuestro método, el primero de ellos consiste en evaluar la rapidez de cálculo de la política óptima. Se hicieron comparaciones entre la implementación de MDP en un CPU Intel móvil i7 a 1.87 GHz usando un solo hilo de ejecución y un GPU Geforce 445 móvil para el escenaro base de la Figura 2. Los resultados muestran que la política óptima se obtiene en 184 iteraciones para las cuales el GPU tarda 4.1 segundos (Tabla 1) mientras que el CPU para 10 iteraciones tarda 1500 segundos (Tabla 2) si el promedio de tiempo de cálculo se mantiene constante, la versión de CPU tardaría alrededor de 7.6 horas en obtener la política óptima.

Adicionalmente la Figura 8 muestra diferentes visualizaciones de una multitud en el escenario base. La descripción del algoritmo utilizado para visualizar la multitud puede encontrarse en [27].

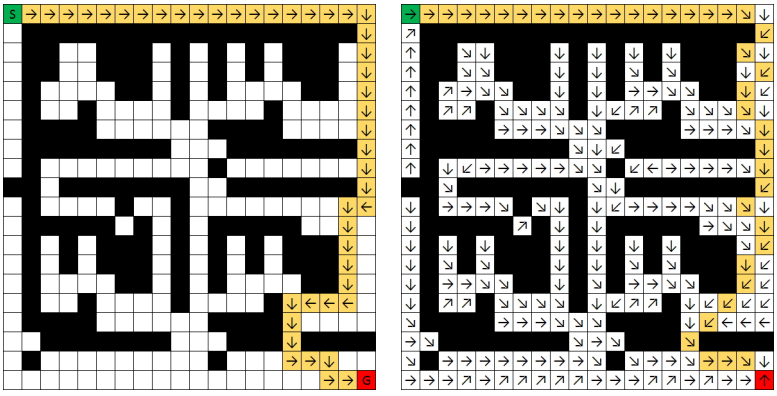
Tabla 1. Rendimiento en el GPU

GPU	NVIDIA® GeForce™ GT445M
columnas	48
filas	33
Total de celdas	1,584
Iteraciones	184
Política óptima en	4.1s

Tabla 2. Rendimiento en el CPU

CPU	Intel® Core™ i7@1.87GHz
columnas	48
filas	33
Total de celdas	1,584
Iteraciones	10
Política óptima en	1500s

Para la segunda prueba (Fig. 6) presentamos la solución de un escenario de 400 celdas (20 filas y 20 columnas) de las cuales 164 representan obstáculos, con el fin de comparar el desempeño y requerimientos de memoria de los algoritmos A^* (Figura 6(a) y Tabla 3, utilizando la implementación de Heyes [11]) y MDP (Figura 6(b) y Tabla 4) al encontrar rutas óptimas de navegación. Encontramos que el tiempo de ejecución de A^* depende del número de agentes a procesar, mientras que la solución con MDPs se realiza en tiempo constante independientemente del número de agentes. Para procesar las 236 celdas libres en el escenario de prueba, el método basado en MDPs tarda 0.02079 segundos en promedio por agente, mientras que A^* toma 0.078 segundos. Notamos también que para procesar estas 236 celdas, A^* requiere 6,580 bytes en promedio por agente para almacenar listas de nodos, mientras que el método basado en MDPs requiere 1,803 bytes en promedio por agente para almacenar Π_t^*, Q_t y V_t en vectores [28].



(a) Solución con A^*. (b) Solución con MDP.

Fig. 6. Escenario de prueba para comparar A^* y MDP.

Cuando un escenario se modela mediante un MDP para navegación, definiremos i como el número de iteraciones, a el número de acciones, c el número total de celdas y t como el número máximo de transiciones. Entonces la complejidad

Tabla 3. Desempeño de A^*.

Método	A^*
CPU	Intel i7-4800MQ@2.7GHz
Opciones	4
Salidas procesables	1
Tiempo de solución	0.078s
Tiempo 236 agentes	18.408s
Pasos de solución	46
Pasos en búsqueda	235
Agentes procesados	1
Memoria total	6.42 KB
Memoria por agente	6.42 KB

Tabla 4. Desempeño de MDP.

Método	MDP
GPU	NVIDIA GeForce GTX780M
Opciones	8
Salidas procesables	235
Tiempo de solución	4.907s
Tiempo 236 agentes	4.907s
Pasos de solución	39
Iteraciones	167
Agentes procesados	236
Memoria total	415.63 KB
Memoria por agente	1.76 KB

para estimar las rutas navegables es $O(iact)$. Como se muestra en la Figura 7, este proceso se lleva a cabo eficientemente en el GPU.

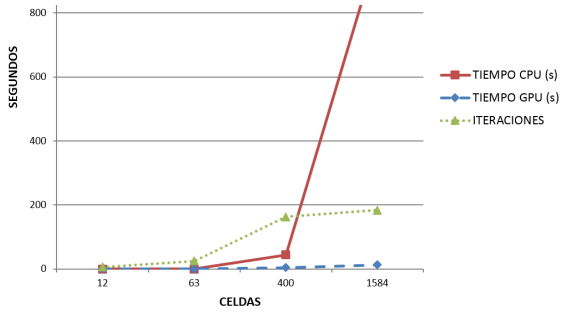


Fig. 7. Tiempo de solución de MDP para GPU y CPU.

La tercera prueba consistió en simular una multitud masiva de Lemmings y de seres humanos. Este caso fue útil para probar los microescenarios, i.e. la evasión de colisiones. De esta forma se comprobó que los agentes efectivamente evadían colisiones entre ellos, además presentaban comportamiento reactivo al momento de agregar interactivamente obstáculos. En este caso se usó un GPU Geforce GTX 560M.

Resultados numéricos de esta simulación se muestran en la figura 9(a). Cuando los personajes solamente siguen sus trayectorias, sin experimentar obstáculos inesperados, el método se ejecuta en MIN TIME (entre 1 y 2 milisegundos para una multitud de entre 128 y 8192 personajes). Por otro lado, para escenarios congestionados la simulación toma MAX TIME (de 6 a 13 milisegundos).

A partir de estas mediciones, otra ventaja de usar MDP y microescenarios es que en tiempo de ejecución, dado que la solución es independiente del tamaño y complejidad del escenario, el método es razonablemente predecible (como lo implica el número de agentes y celdas) y estable (Fig. 9(b)). A partir del rendimiento obtenido y del comportamiento de MAX TIME, se puede predecir que si se tiene memoria y poder de cómputo suficientes en un GPU, se lograrían simular

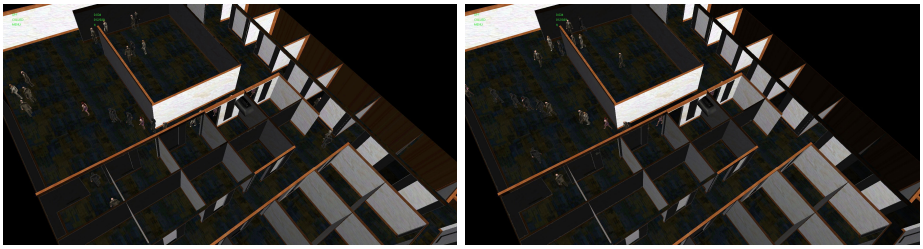
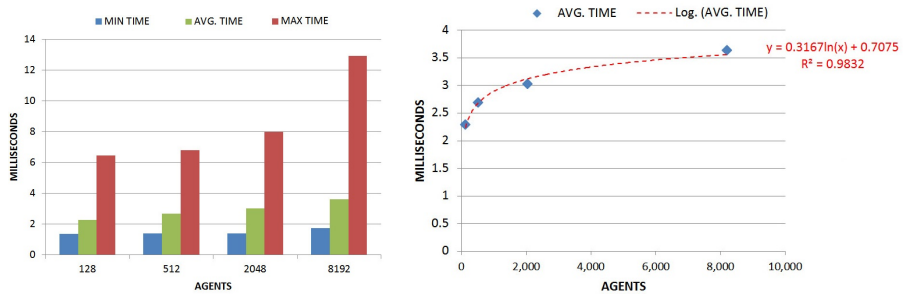


Fig. 8. Visualización de una evacuación.



(a) Tiempo de Simulación. (b) Predicción del tiempo promedio.

Fig. 9. Rendimiento general del algoritmo MDP con microescenarios.

1,000,000 de personajes en 806.37 milisegundos en escenarios congestionados, lo cual es un punto a destacar de nuestro algoritmo.

Cuando los microescenarios resueltos durante el preprocesamiento se utilizan para evadir colisiones, la complejidad del algoritmo resulta en la de la lectura de las celdas vecinas aunado a la complejidad de la búsqueda en una tabla de firmas con la correspondiente obtención de la curva Bézier para el movimiento continuo para cada agente. Esto es, para un número N de agentes: $O(10N)$, un costo lineal como confirma la Figura 9(a).

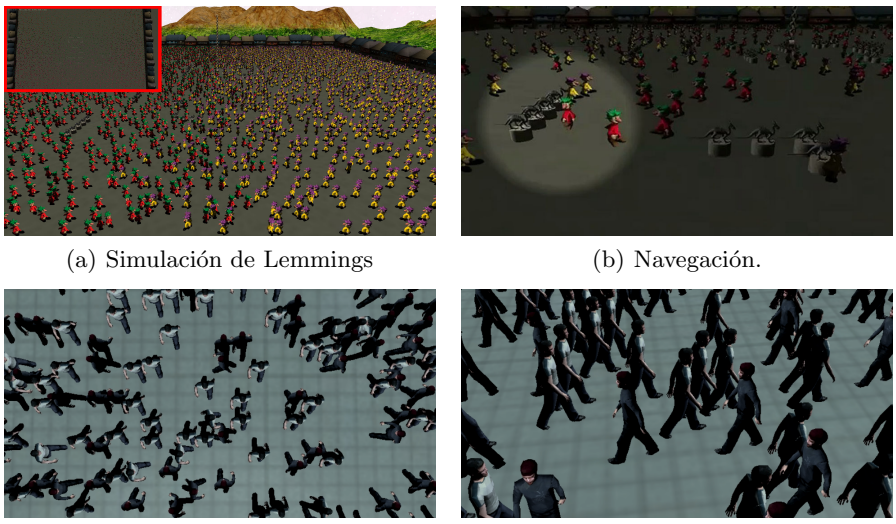
La Figura 10 muestra diferentes visualizaciones de las multitudes de Lemmings (Fig. 10(a)) y de personas (Fig. 10(c)) de este experimento. Note que en ambas multitudes los personajes cambian de dirección para evadir colisiones con obstáculos (Fig. 10(b)) o con otros agentes (Fig. 10(d)).

Finalmente invitamos al lector a que revise el video de la simulación que se encuentra en la siguiente liga: http://tinyurl.com/p3aqxm6

6. Conclusiones y trabajo futuro

El algoritmo propuesto permite adaptar un MDP no sólo para resolver el problema de navegación sino también el problema de evasión de colisiones. Además

resaltando que Reyes y Sucar [23] notaron que *"...el problema del formalismo de los MDPs es que el espacio de estados crece exponencialmente con el número de variables, y los métodos de inferencia crecen con el número de acciones, entonces, en problemas grandes, los MDPs se tornan impracticos e ineficientes..."*, nuestra implementación en el procesador gráfico o GPU permite aplicar el algoritmo en simulaciones de multitudes en tiempo real con rendimiento suficiente para llevar a cabo la visualización tridimensional de los resultados. Además, conforme se disponga de mejor hardware, problemas previamente inabordables serán reexaminados para verificar si su complejidad puede ser sujeto de análisis y solución como lo es el caso de simulación de evacuaciones masivas. Otro punto relevante que deberá explorarse es la modificación de las variables de los MDPs, una modificación correcta nos permitirá representar simulaciones más complejas. Por otro lado, es deseable desacoplar la etapa de simulación con la etapa de visualización mediante la utilización de dos GPUs, de esta forma se mejoraría aun más el rendimiento del método. Finalmente la integración y comparación del algoritmo propuesto con otros modelos conductuales permitirían otras áreas de desarrollo como por ejemplo simulaciones sociales.



(a) Simulación de Lemmings (b) Navegación.

(c) Simulación de personas. (d) Evasión de colisiones.

Fig. 10. Multitudes de Lemmings y personas.

Referencias

1. Banerjee, B., Abukmail, A., Kraemer, L.: Advancing the layered approach to agent-based crowd simulation. In: Proceedings of the 22nd Workshop on Principles of

Advanced and Distributed Simulation. pp. 185–192. IEEE, IEEE Computer Society (2008)
2. Batty, M.: Agent-based pedestrian modelling. In: Centre for Advanced Spatial Analysis. Working Paper Series, University College London (2003)
3. van den Berg, J., Guy, S.J., Lin, M.C., Manocha, D.: Reciprocal n-body collision avoidance. In: International Symposium on Robotics Research (ISRR). vol. 70, pp. 3–19. IFRR (May 2011)
4. van den Berg, J., Lin, M., Manocha, D.: Reciprocal velocity obstacles for real-time multi-agent navigation. In: International Conference on Robotics and Automation (ICRA). pp. 1928–1935. IEEE (May 2008)
5. Bing, H., Yangzihao, W., Jia, Z.: An improved method of continuous collision detection using ellipsoids. In: Computer-Aided Industrial Design & Conceptual Design. IEEE (Nov 2009)
6. Blue, V.J., Embrechts, M.J., Adler, J.L.: Cellular automata modeling of pedestrian movements. In: Systems, Man, and Cybernetics. vol. 3, pp. 2320–2323. IEEE (Oct 1997)
7. Bonner, S., Kelley, R.B.: A novel representation for planning 3-d collision-free paths. In: Systems, Man, and Cybernetics. vol. 20, pp. 1337–1351. IEEE (Nov 1990)
8. Foka, A.F., Trahanias, P.E.: Real-time hierarchical pomdps for autonomous robot navigation. Robotics and Autonomous Systems 55(7), 561–571 (2007)
9. Guy, S.J., Chhugani, J., Kim, C., Satish, N., Lin, M., Manocha, D., Dubey, P.: Clear path: highly parallel collision avoidance for multi-agent simulation. In: Symposium on Computer Animation. pp. 177–187. ACM (Aug 2009)
10. Helbing, D., Molnar, P.: Social force model for pedestrian dynamics. PHYSICAL REVIEW E 51, 4282 (1995)
11. Heyes-Jones, J.: Implementation of the a* algorithm in c++. Web Page (Mar 2014), justinhj/astar-algorithm-cpp. GitHub. Retrieved May 6, 2014, from https://github.com/justinhj/astar-algorithm-cpp
12. Initiative, T.H.O.: Brains - the smithsonian institution's human origins program (2011), http://humanorigins.si.edu/human-characteristics/brains
13. Koenig, S., Likhachev, M.: D*lite. In: Eighteenth national conference on Artificial intelligence. pp. 476–483. American Association for Artificial Intelligence, Menlo Park, CA, USA (2002)
14. Le Bon, G.: The Crowd: A Study of the Popular Mind. The Criminology Series, The Macmillan Co. (1896)
15. Li, L., Yang, S., Zhou, W., Chen, G.: Mechanism for constructing the dynamic collision avoidance knowledge-base by machine learning. In: Proceedings of the 2010 International Conference on Manufacturing Automation. pp. 279–285. ICMA '10, IEEE Computer Society, Washington, DC, USA (2010)
16. Li, X., Zhong, Z., Lu, Z.: Collision detection algorithm based on slice projection. In: International Conference on Mechatronics and Automation. IEEE (Aug 2009)
17. Likhachev, M., Ferguson, D., Gordon, G., Stentz, A.T., Thrun, S.: Anytime dynamic a*: An anytime, replanning algorithm. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS) (June 2005)
18. Likhachev, M., Gordon, G.J., Thrun, S.: Ara*: Anytime a* with provable bounds on sub-optimality. In: NIPS (2003)
19. Llorca, D.F., Milanés, V., Alonso, I.P., Gavilán, M., Daza, I.G., Pérez, J., Sotelo, M.A.: Autonomous pedestrian collision avoidance using a fuzzy steering controller. In: Intelligent Transportation Systems. IEEE (Jun 2011)

20. Mausam, Weld, D.S.: Solving concurrent markov decision processes. In: Proceedings of the 19th National Conference on Artifical Intelligence. pp. 716–722. AAAI'04, AAAI Press (2004)
21. Mustapha, N.A.B., Bade, A.B., Kari, S.: A review of collision avoidance technique for crowd simulation. In: International Conference on Information and Multimedia Technology. IEEE (2009)
22. Pettré, J.: Motion planning and autonomy for virtual humans: Part 4. case study 2. part i. design and simulation of virtual crowds. In: SIGGRAPH. ACM (2008)
23. Reyes, A., Sucar, L.: An operation auxiliary system for power plants based on decision-theoretic planning. In: International Conference on Intelligent Systems Application to Power Systems (2005)
24. Reynolds, C.: Big fast crowds on ps3. In: Sandbox Symposium. ACM (Jul 2006)
25. Reynolds, C.W.: Flocks, herds, and schools: A distributed behavioral model. In: SIGGRAPH (1987)
26. Rudomin, I., Perez, F., Millán, E.: Groups and crowds with behaviors specified in the environment. In: Intelligent Virtual Environments and Virtual Agents Proceedings. ITESM Campus Ciudad de Mexico (2004)
27. Ruiz, S., Hernández, B., Alvarado, A., Rudomín, I.: Reducing memory requirements for diverse animated crowds. In: Proceedings of Motion on Games. pp. 55:77–55:86. MIG '13, ACM, New York, NY, USA (2013)
28. Ruiz, S., Hernández, B., Rudomín, I.: A gpu implementation of markov decision process for path planning. In: The 5th International Supercomputing Conference in Mexico (ISUM 2014) (2014)
29. Sarmady, S., Haron, F., Talib, A.Z.: Simulating crowd movements using fine grid cellular automata. In: International Conference on Computer Modelling and Simulation. pp. 428–433. IEEE (2010)
30. Shao, L., Zhou, H.: Curve fitting with bézier cubics. In: Graphical Models and Image Processing. vol. 58, pp. 223–232 (1996)
31. Stentz, A.: The focussed d* algorithm for real-time replanning. In: Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2. pp. 1652–1659. IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995)
32. Sucar, L.: Parallel markov decision processes. In: Lucas, P., Gmez, J., Salmern, A. (eds.) Advances in Probabilistic Graphical Models. Studies in Fuzziness and Soft Computing, vol. 214, pp. 295–309. Springer Berlin Heidelberg (2007)
33. Thalmann, D., Musse, S.R.: Crowd Simulation. Springer (2007)
34. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. ACM Trans. Graph. 25(3), 1160–1168 (Jul 2006)
35. Zhang, S., Li, M., Li, F., Liu, A., Cai, D.: A simulation model of pedestrian flow based on geographical cellular automata. In: Geoinformatics. IEEE (Jun 2011)
36. Zhiquiang, K., Chongchong, Y., Li, T., Jingyan, W.: Simulation of evacuation based on multi-agent and cellular automaton. In: Mechatronic Science, Electric Engineering and Computer. IEEE (Aug 2011)