Una implementación paralela del algoritmo de Q-Learning basada en un esquema de comunicación con caché

Alicia Marcela Printista, Marcelo Luis Errecalde, Cecilia Inés Montoya

Proyecto UNSL Nº 338403[1]
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950 - Local 106
5700 - San Luis - Argentina
E-mail: {mprinti, merreca, cmontoya}@unsl.edu.ar
Fax: +54 652 30224

Resumen

Q-Learning es un método de Aprendizaje por Refuerzo que permite resolver *problemas de decisión secuencial* en los cuales la utilidad de una acción depende de una secuencia de decisiones y donde además existe incertidumbre en cuanto a las dinámicas del ambiente en que está situado el agente. Este marco general de trabajo ha permitido aplicar Q-Learning y otros métodos de Aprendizaje por Refuerzo a una amplia gama de problemas del mundo real de considerable complejidad, como por ejemplo navegación de robots, manufacturación industrial, juegos, control de ascensores, etc.
A pesar de las características interesantes de Q-Learning, uno de sus principales problemas es que es un método lento, ya que el agente requiere un tiempo considerable de entrenamiento para aprender una política aceptable.
A los fines de solucionar, o al menos atenuar este problema, este trabajo propone un modelo de implementación paralela de Q-Learning manteniendo una representación tabular, y utilizando un esquema de comunicación basada en caché.
Este modelo es aplicado en un problema particular, reportándose los resultados obtenidos con distintas configuraciones de procesadores y analizándose las ventajas y limitaciones actuales del enfoque.

Palabras Claves: Programación Paralela, Comunicación basada en caché. Aprendizaje por Refuerzo, Programación Dinámica Asincrónica.

[1] El grupo de investigación, dirigido por el MSc. Raúl Gallard, está soportado por la UNSL y la ANPCyT (Agencia Nacional para la promoción de la Ciencia y Tecnología).

1. Introducción

Este trabajo propone una implementación paralela del algoritmo de Q-Learning sobre una máquina masivamente paralela utilizando la librería de pasaje de mensajes Parallel Virtual Machine (PVM) con un esquema de comunicación eficiente basada en caché.
Q-Learning [21, 22] es un método de Aprendizaje por Refuerzo (AR) [4, 7, 8, 11, 15, 16] que ataca el problema de aprender a controlar agentes autónomos, mediante interacciones por prueba y error con un ambiente dinámico, el cual le provee señales de refuerzo por cada acción que realiza.
Q-Learning permite resolver *problemas de decisión secuencial* en los cuales la utilidad de una acción depende de una secuencia de decisiones y donde además existe incertidumbre en cuanto a las dinámicas del ambiente en que está situado el agente. Este marco general de trabajo ha permitido aplicar Q-Learning y otros métodos de AR a una amplia gama de problemas del mundo real de considerable complejidad, como por ejemplo navegación de robots, manufacturación industrial, juegos, control de ascensores, etc.
Q-Learning, al igual que otros métodos de AR, tiene la ventaja de estar basado en un modelo matemático formal (Markov Decision Processes) lo que permite una definición precisa de la tarea a resolver y de la forma de su solución. Entre los métodos de AR es probablemente el más simple de entender e implementar y no asume ningún tipo de conocimiento previo de las dinámicas del ambiente, lo que lo convierte en un método muy atractivo en dominios no muy conocidos por el implementador del sistema. Existe además una demostración formal de convergencia a una política óptima para el agente, asumiendo una representación tabular de la función a aprender.
A pesar de las características interesantes de Q-Learning, uno de sus principales problemas es que es un método lento, ya que el agente requiere un tiempo considerable de entrenamiento para aprender una política aceptable.
A los fines de solucionar, o al menos atenuar este problema, este trabajo propone un modelo de implementación paralela de Q-Learning manteniendo una representación tabular que garantice la convergencia al óptimo. Este modelo es aplicado en un problema particular, reportándose los resultados obtenidos con distintas configuraciones de procesadores y analizándose las ventajas y limitaciones actuales del enfoque.
El resto del trabajo está organizado de la siguiente manera: la sección 2 (Modelo matemático) presenta una formalización (MDP´s) que muestra la estructura matemática subyacente en la clase de problemas que se pueden resolver con Q-Learning. La sección 3, realiza una descripción general del tipo de algoritmos utilizados para resolver MDP´s y en particular de Q-Learning. Ambas secciones tienen un objetivo tutorial intentando hacer este trabajo lo más autocontenido posible, pudiendo ser obviadas por quienes tengan experiencia en el tema. La sección 4, presenta aspectos generales y de implementación del modelo paralelo de Q-Learning propuesto en este trabajo. En la sección 5 se describe un problema particular y los resultados experimentales obtenidos, finalizándose en la sección 6 con las conclusiones de este trabajo.

2. Modelo matemático

Q-Learning es un algoritmo de aprendizaje automático para resolver tareas que pueden ser modeladas como procesos de decisión Markovianos (o Markov Decision Processes) y que a partir de ahora se referenciarán con el término general de MDPs.
La idea general intuitiva dentro de este modelo, es tener un agente conectado a algo que denominaremos el *ambiente* vía percepción y acción. En cada paso del tiempo *t*, el agente recibe como entrada desde el ambiente una indicación del estado actual *s* del ambiente, y selecciona una acción *a* disponible en el estado *s*. La acción generada por el agente cambia el estado del ambiente, el cual responde un paso del tiempo después con una indicación del nuevo estado del ambiente y una señal de recompensa inmediata *r*, por la acción tomada previamente.
Esta interacción se puede visualizar gráficamente en la Figura 1:

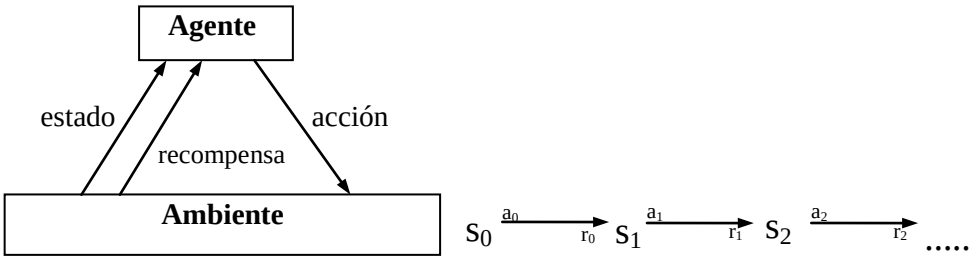

Figura 1

Si los objetivos del agente están definidos por la función de recompensa inmediata, la tarea del agente se reduce a encontrar un comportamiento que le permita decidir en cada estado, que acciones tomar para maximizar los valores de la señal de recompensa acumulados a lo largo del tiempo.
Cuando las respuestas del ambiente a una acción del agente, dependen sólo del estado en que se encontraba y de la acción tomada, sin influencias de los estados del ambiente y las acciones del agente previas, el ambiente y la tarea en su conjunto se dice que satisfacen la *propiedad Markov*, y pueden ser definidas formalmente como un MDP que consiste de:

- un conjunto de estados posibles *S*
- un conjunto de acciones posibles *A*
- las dinámicas de un paso del ambiente, dadas por
    - Las *probabilidades de transición P(s,a,s')*
      $P(s,a,s') = \Pr\{ s_{t+1} = s' \mid s_t = s, a_t = a\}$
    - Las *recompensas esperadas R(s,a,s')*
      $R(s,a,s') = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$ [2]

definidos para todo $s, s' \in S$ y $a \in A(s)$, donde $A(s)$ es el conjunto de acciones válidas en s

y la *solución* de este MDP se reduce a encontrar una política

$$\pi^* : S \rightarrow A$$

que en cada paso del tiempo t, elige la acción a_t que maximiza la suma descontada de las recompensas acumuladas en el futuro, denominada *retorno descontado* R_t, y definido formalmente como

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots\ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

donde $0 <= \gamma <= 1$, recibe el nombre de *factor de descuento*.

3. Técnicas para la resolución de MDP´s. Q-Learning

Antes de comenzar a hablar del algoritmo de Q-Learning, cuya versión paralela es el sujeto de las próximas secciones, es necesario entender algunos conceptos generales comunes a las mayoría de las técnicas utilizadas para resolver MDP´s.

[2] En este caso, y en general a lo largo del informe se usa la terminología y notación utilizada en [16]. Para el caso de esta definición particular, otros autores consideran a R como una función de 2 variables: $R(s,a) = \Sigma_{s'} P(s,a,s') R(s,a,s')$

La solución de un MDP, se basa normalmente en la estimación de *funciones de valor óptimas*, definidas sobre el conjunto de estados o de acciones. Intuitivamente, una función de valor indica *cuán bueno* es para el agente estar en un estado dado, o realizar una acción particular en un estado dado. La noción de *cuán bueno* estará dada obviamente por las recompensas esperadas en el futuro (retorno esperado) y dependerá de la política π con que se maneje el agente.
Dado que el objetivo es encontrar una política óptima π^*, los métodos para resolver MDP´s se concentran en aprender la *función de valor-estado óptima* (denotada V^*) o la *función de valor-acción óptima* (denotada Q^*) y definidas formalmente como:

$$V^*(s) = \max_\pi E_\pi \{ R_t \mid s_t = s\} = \max_\pi E_\pi \{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\}$$

$$Q^*(s,a) = \max_\pi E_\pi \{ R_t \mid s_t = s, a_t = a \} = \max_\pi E_\pi \{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \}$$

Básicamente, V^* indica para cada estado cual es el retorno esperado máximo que se puede obtener desde ese estado, sobre todas las políticas posibles, y las políticas que obtienen este máximo se denominan *políticas óptimas*.
Q^* por su parte, da el retorno esperado de tomar la acción *a* en el estado *s* y en lo sucesivo seguir una política óptima.
Es importante destacar que si bien pueden haber mas de una política óptima (se referenciará a todas ellas como π^*) todas las políticas óptimas comparten las mismas funciones de valor.
En general, una vez obtenidas V^* o Q^* es fácil determinar una política óptima haciendo

$$\pi^*(s) = \max_{a \in A(s)} \sum_{s'} P(s,a,s') [R(s,a,s') + \gamma V^*(s')]$$

o bien

$$\pi^*(s) = \max_{a \in A(s)} Q^*(s,a)$$

La diferencia principal entre aprender V^* o Q^*, reside en que si se conoce Q^* se puede determinar en forma directa cual es la acción óptima, mientras que para poder usar V^* para elegir la acción óptima, se deberá conocer además las dinámicas de un paso del ambiente.
Si bien existen una gran variedad de métodos para resolver MDP´s en base a la estimación de funciones de valor, ellos difieren básicamente en alguno de los siguientes puntos:

- Aprenden V^* o Q^*
- Cuentan o no con información sobre las dinámicas de un paso del ambiente (MDP´s con información completa vs MDP´s con información incompleta)
- La política óptima se obtiene antes de ser usada en el agente a ser controlado (métodos off-line), o la política óptima es aprendida *durante* la fase de ejecución del agente, utilizando información que el agente acumula a medida que interactúa con el ambiente (métodos on-line).
- La actualización de las funciones de valor utilizan la información de valor de *todos* los estados en forma simultánea (versión síncrona), o la actualización de las funciones de valor requieren información de sólo un subconjunto de estados, en instantes de tiempo posiblemente distintos (versión asíncrona).

El principal aporte histórico para la resolución de MDP´s, surgió del área de teoría de control óptimo y en particular de los métodos de Programación Dinámica (PD).
El término PD, refiere a una colección de algoritmos que sucesivamente aproximan funciones de valor óptimas, aprovechando las relaciones de recurrencia de las funciones de valor. Estos métodos han sido tradicionalmente utilizados off-line, y en general se han concentrado sobre la aproximación de V^*.

Los métodos de PD, asumen el conocimiento de un modelo preciso y completo del ambiente (las dinámicas de un paso) y en general trabajan en forma síncrona sobre *todo* el espacio de estados, lo que los torna inadecuados para problemas grandes.
Este último inconveniente dió lugar al surgimiento de una serie de métodos denominados de *Programación Dinámica asíncrona*, donde la idea es actualizar las funciones de valor sin necesidad de hacer un barrido exhaustivo y sistemático de todo el espacio de estados, sino concentrarse sobre subconjuntos del mismo.
Los métodos de PD asíncrona, se basan esencialmente en la aproximación de V^* en forma off-line, asumiendo al igual que PD clásica, el conocimiento de las dinámicas de un paso del ambiente.
Los principios de PD asíncrona son importantes para este trabajo, ya que es precisamente la naturaleza asíncrona de las actualizaciones de las funciones de valor lo que lo torna adecuado para el procesamiento paralelo, como es propuesto en [2, 3]. Si bien PD asíncrona trabaja en forma off-line, recientemente ha sido propuesta una versión on-line en [1] con el nombre de Real Time Dynamic Programming (RTDP).
Cuando no se dispone de las dinámicas de un paso del ambiente, el problema a resolver se conoce con el nombre de *MDP con información incompleta*, y los métodos para su solución se conocen en la terminología de control como *métodos de control adaptativos*.
Estos métodos, en aquellos casos en que aprenden V^*, deben *estimar* además un modelo de las dinámicas de un paso del ambiente para poder construir las políticas óptimas, y se denominan con el término general de *métodos adaptativos indirectos.*
Como ejemplo de este tipo de métodos podemos mencionar al método on-line propuesto en [1] bajo el nombre de *RTDP adaptativo*.
Cuando el método adaptativo, construye las políticas sin usar un modelo del sistema explícito, se denomina un *método directo* o *model free.*
Estos métodos usualmente aproximan Q^*, lo que les permite construir la polítca óptima directamente sin necesidad de requerir las dinámicas de un paso del ambiente.
Como ejemplo sobresaliente de este tipo de métodos se tiene el algoritmo de Q-Learning, el cual es el tema principal de esta sección.

3.1. Q-Learning

Q-Learning es un método propuesto por Watkins [21, 22] para resolver MDP's con información incompleta.
Desde el punto de vista de teoría de control es un método directo adaptativo, y como su nombre lo indica se basa en el aprendizaje de Q^*, utilizando para ello el siguiente algoritmo [16]:

```
1  Inicializar (con 0's o valores aleatorios) Q(s,a) para todo s ∈ S y para todo a ∈ A(s)
2  Repetir (por cada episodio)
3      Inicializar s
4      Repetir (por cada paso del episodio):
5          Elegir a desde s usando una política derivada desde Q (p. ej. ∈-greedy)
6          Ejecutar acción a, observar estado resultante s' y reward r.
7          Q(s,a) ← Q(s,a) + α[r + γ max_a' Q(s',a') - Q(s,a)]
8          s ← s';
9      hasta que s es terminal
```

Figura 2

A medida que el agente se mueve hacia adelante desde un viejo estado a uno nuevo, Q-Learning propaga las estimaciones de Q *hacia atrás* desde el nuevo estado al viejo.
Si bien el ciclo de Q-Learning se realiza infinitamente en teoría, en la práctica el aprendizaje se realiza por *episodios* (o *trials*), donde cada episodio comienza en un cierto estado inicial hasta alcanzar alguna condición definida por el diseñador del sistema de learning (como por ejemplo: llegar al estado objetivo, alcanzar un estado absorvente, superar un número máximo de iteraciones, etc.).
El factor α, presente en la fórmula para la actualización de los valores de Q, se denomina *parámetro del*

tamaño del paso y sus valores usualmente varían entre 0 y 1. Este parámetro suele ser decrementado de paso en paso o mantenido constante, dependiendo de si se está trabajando en ambientes estacionarios o no.
Bajo las condiciones de que la política para seleccionar las acciones garantice que todas las acciones sean ejecutadas en cada estado un número infinito de veces, y que α sea decrementado apropiadamente, este algoritmo ha demostrado que sus estimaciones de Q^* convergen con probabilidad 1 a Q^*.
Un comentario especial del algoritmo de Q-Learning merece el paso en el cual el agente elige una acción *a*, en un estado *s* (Ver línea 5 de la Figura 2). Si la estimación de Q^* mantenida en la tabla Q, fuera exactamente Q^*, lo mejor que podría hacer el agente sería comportarse en forma greedy eligiendo aquella acción con máximo valor de Q para ese estado. En este caso, se dice que el agente está *explotando* la información que tiene sobre las funciones de valor. Lamentablemente esta es una situación no realista, ya que la tabla Q es una estimación de Q^*, y esta estimación será precisa en la medida que el agente *explore* nuevos estados y acciones y no sólo los sugeridos por los valores corrientes de Q.
Dado que el agente es responsable tanto de *explotar* la información disponible, como de *explorar* nuevos estados y acciones para mejorar sus estimaciones de las funciones de valor, este dilema es conocido como el dilema exploración/explotación y es común a Q-Learning y a la mayoría de los métodos adaptativos.
La naturaleza estocástica del ambiente, con posibles máximos locales y en el peor de los casos ambientes que cambian a lo largo del tiempo, hacen imprescindible un mínimo de exploración, que seleccione acciones que no son sugeridas por la estrategia greedy.
Si bien se han propuesto innumerables y sofisticadas estrategias para manejar este problema [19, 20], esta discusión escapa de los alcances de este trabajo, y utilizaremos para nuestros experimentos una estrategia muy simple denominada *$\in$-greedy*. Esta estrategia de exploración se comporta de manera greedy por default pero, con probabilidad $\in$, elige una acción random (con $\in$ entre 0 y 1). Si bien es común elegir valores fijos para $\in$ (como por ejemplo 0.1 para lograr aproximadamente un 10% de exploración), también se suele comenzar con valores más altos de $\in$ los cuales se reducen con el paso del tiempo, favoreciendo de esta manera la exploración durante las etapas tempranas del aprendizaje y la explotación en las etapas posteriores.

4. Q-Learning paralelo

Uno de los principales problemas de Q-Learning es que el agente requiere un gran número de episodios de entrenamiento para aprender una función de valor aceptable [4].
Existen actualmente dos enfoques principales para la aceleración del proceso de aprendizaje: 1) permitir la incorporación de información provista por un observador externo [9, 10, 13] y 2) integrar learning con planning [9, 12, 13, 17, 18].
El primero consiste en posibilitar que un experto u observador externo pueda incorporar "consejos" que le sirvan al agente para aprender ciertos aspectos complejos del ambiente en forma más eficiente.
En el segundo caso, el agente aprende un modelo del ambiente en forma simultánea al aprendizaje de la política, lo que permite hacer un uso más intensivo de una cantidad limitada de experiencia.
En este trabajo se propone un enfoque alternativo para la aceleración del proceso de aprendizaje, basado en la implementación paralela del algoritmo de Q-Learning
La motivación principal en el desarrollo de implementaciones paralelas ha sido la necesidad de aumentar la velocidad computacional de una tarea distribuyendo el trabajo de un algoritmo entre procesadores que tienen la capacidad de trabajar en paralelo. Sin embargo, otro motivo más reciente sugiere que, en algunos casos, el procesamiento paralelo mejora la calidad de las soluciones obtenidas, ya que permite descubrir relaciones internas del problema que un enfoque secuencial convencional no es capaz de explorar.
La base de toda programación paralela, de una forma u otra, es la estrategia de particionado, la cual divide el problema en partes y una vez que estas partes se hayan completado, sus resultados deben ser combinados para obtener el resultado deseado.
La técnica de Particionado puede ser aplicada a las funciones de un programa (descomposición funcional). En este enfoque, el hecho de dividir un programa originalmente secuencial en unidades paralelas independientes y asicrónicas, involucra identificar dependencias inter-módulos para determinar donde la implementación paralela es segura [6].
Otra forma de particionado, es el aplicado a los datos del programa (descomposición de dominio). Esta

estrategia de particionado fue la que se utilizó para obtener una versión paralela del algoritmo Q-Learning, manteniendo dos o más procesos idénticos operando sobre diferentes fracciones de la tabla Q. Esto significa que cada proceso se concentra en la tarea de aprendizaje sobre sólo un subconjunto de estados y sus acciones asociadas. Sin embargo alguna interacción es necesaria; en algún instante de su aprendizaje los procesos pueden requerir información de estados no locales que pertenezcan a un subconjunto de estados vecino.
Dependiendo del esquema específico de particionado de datos y del número de particiones, esta interacción podría aumentar considerablemente la comunicación entre procesos vecinos ya que éstos procesos frecuentemente permanecerían esperándose unos por otros. Esto incluye fuertes requerimientos de sincronización. Lo anterior sugiere que un único proceso de propósito específico ejecute estás operaciones. Con este enfoque, una operación de sincronización sólo es necesaria para actualizar y/o requerir información de la tabla Q con un procesador *master* global. Por lo anterior, la estructura resultante del algoritmo es *Master/Slave* [5, 23]
Dentro de esta organización, el master estará encargado de decidir sobre que porciones de la tabla Q trabajará cada slave, de difundir esta información entre los procesos slaves, y posteriormente de mantener una versión actualizada de la tabla Q completa. Por otro lado esta tabla será consultada y actualizada por los procesos slaves mediante los siguientes requerimientos al master:

- *req_msg*: Dado que el algoritmo que ejecuta cada proceso slave es el mismo que el presentado en la Figura 2, un proceso slave necesitará comunicarse con el master cuando para la actualización de su tabla Q requiera el máximo valor de Q de un estado procesado en otro slave (Ver línea 7 de la Figura 2); *req_msg* cumple este rol y recibe como respuesta desde el master la información requerida. En la versión paralela de Q-Learning esta situación es considerada como otra condición de fin de episodio, además de las dos utilizadas en el secuencial (alcanzar un estado objetivo o superar un número máximo de pasos).

  Sin bien en la implementación original de Q-Learning Paralelo se enviaba un mensaje *req_msg* cada vez que este tipo de información era requerida, pudo observarse que esto implicaba un alto overhead de comunicación. Para atenuar este problema, se implementó una cache en la que se mantenía el último valor provisto por el master, el cual sólo era actualizado si el valor máximo de requerimientos alcanzaba un valor especificado por el parámetro *cache_size*.
  Si un valor demasiado alto fuese elegido para *cache_size* entonces los procesadores ejecutarían con muy poca comunicación entre ellos, pero la performance del algoritmo de aprendizaje sería relativamente baja, ya que la mayor parte del tiempo los procesadores estarían trabajando sobre valores desactualizados de sus estados vecinos. Si por el contrario, un valor demasiado bajo fuese elegido, la performance del algoritmo, en cuanto a calidad de resultados, estaría cerca del Q-Learning secuencial, pero el *overhead* de comunicación sería considerablemente elevado ya que tomaría lugar una gran cantidad de intercambio de mensajes a fin de que cada slave trabaje con información actualizada de estados sobre los que otros procesadores están trabajando.

- *inf_msg:* Cada slave procesa secuencialmente sobre su conjunto local de estados. Con el objetivo de mantener la tabla Q global actualizada en el master, los procesos slaves le envían periódicamente su partición de la tabla Q. Con el fin de reducir el número de *inf_msg* realizados, este esquema fue posteriormente mejorado, condicionando el envío a la existencia de algún cambio en la tabla Q local de cada slave.

El programa fue ejecutado en una Máquina Paralela llamada **PowerMouse de Parsytec**, el cual es un sistema "*host-based*". El Sistema operativo de la máquina host, SUN-SPARC, es Solaris con el ambiente PARIX [14].
PARIX (PARallel extensions to unIX, extensiones paralelas para UNIX) es el sistema operativo de las computadoras paralelas de Parsytec el cual proporciona un ambiente adecuado para la programación con pasaje de mensajes y de tipo SPMD (Simple Programa Múltiples Datos). PARIX está provisto del sistema

PowerPVM el cual realiza la distribución de los procesos en los distintos procesadores.
En el modelo de programación de PARIX la comunicación está definida entre los procesos o threads. En el presente trabajo no fue necesario igualar el modelo de comunicación de la implementación a la configuración física de esta máquina, ya que todo esto está cubierto por el nanoKernel de PARIX.

5. Resultados experimentales

En esta sección se realiza un análisis comparativo de los resultados obtenidos entre el enfoque secuencial tradicional y la implementación paralela.
Se implementó además el método, Value Iteration, del área de Programación Dinámica, el cual fue utilizado como punto de referencia para realizar los tests de convergencia a una política óptima de los métodos restantes.

5.1. Descripción del problema seleccionado

El problema seleccionado para realizar el análisis de performance, corresponde a uno de los problemas de laberinto típicos usados en este área.
En este caso se utilizó el laberinto de 135 celdas mostrado en la Figura 3. Cada celda del laberinto corresponde a un estado del ambiente. En cada estado se pueden realizar cuatro acciones: RIGHT, LEFT, UP, DOWN, las cuales causan que el agente se mueva en forma determinística a la celda vecina correspondiente. Las acciones bloqueadas (por una barrera o por los límites del laberinto) no mueven al agente, dejándolo en la misma ubicación. Todas las transiciones tienen una recompensa de 0, excepto las que conducen a alguno de los dos estados terminales - indicados con **G** en la Figura 3- las cuales dan 100 unidades de recompensa. El factor de descuento γ es 0.95. Los episodios comienzan siempre desde un estado no terminal elegido en forma aleatoria, repitiéndose este proceso sobre el final de cada episodio.

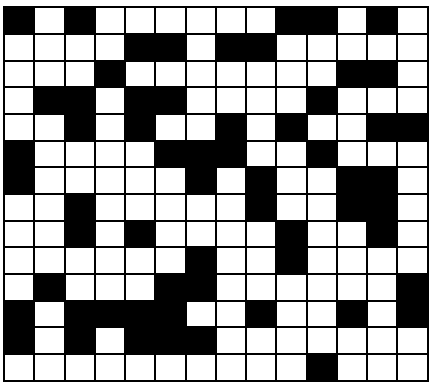

Figura 3

En los experimentos realizados, el fin de episodio está determinado por el ingreso a un estado terminal o cuando se ha superado un número máximo de pasos en el episodio, valor al que se denominará MPE.
La tarea del agente consiste en maximizar la suma de recompensas descontadas acumuladas sobre el tiempo, lo que en este caso es equivalente a llegar al estado terminal **G** (con recompensa 100) más cercano, partiendo desde cualquier estado arbitrario del laberinto y realizando el mínimo número de pasos posibles.
Para este problema se obtuvieron (usando el método de Programación Dinámica, Value Iteration) los siguientes valores correspondientes a la política óptima:

- Promedio de los valores de V^* de todos los estados no terminales = 58.3419.
- Promedio del mínimo número de pasos requeridos por el agente para viajar desde cualquier estado no terminal al estado **G** más cercano = 12.2556.

5.2. Análisis de los resultados

Los parámetros utilizados tanto en la versión secuencial como paralela de Q-Learning, fueron los siguientes: tamaño de paso α = 1 y número máximo de pasos por episodio MPE = 10000. Todos los algoritmos utilizaron la política de exploración $\in$-greedy con $\in$ = 0.1.
Si bien en teoría, la convergencia de Q-Learning a los valores óptimos de Q, es garantizada si los estados y acciones son probados infinitas veces, a los fines prácticos, se eligió un máximo de 32000 episodios para testear la convergencia a la política óptima.
Es importante recalcar, que en todos los casos los parámetros fueron seleccionados tomando en cuenta los mejores resultados obtenidos con la versión secuencial de Q-Learning.
Para las versiones paralelas de Q-learning, se agregó otra situación de fin de episodio, que se produce cuando un proceso slave requiere información de un estado procesado por otro slave.
Los trabajos que comparan algoritmos del tipo de Q-Learning y otros en los cuales la experiencia se divide en episodios, normalmente toman como referencia los resultados obtenidos en cada uno de los episodios.
Este tipo de comparación no es posible en este caso, ya que en la versión paralela no se tiene el concepto de un único fin de episodio, sino que se pueden producir múltiples fin de episodios en los distintos slaves.
A los fines de poder comparar los resultados, se tomó como referencia una medida de tiempo a la que se denominó *época*, cuyo valor es de 8 ms. Cada vez que transcurrió el intervalo de tiempo correspondiente a una época, se registró el estado actual de aprendizaje de la política óptima, ejecutando el agente en forma greedy y tomando como estados iniciales cada uno de los estados no terminales.
La versión paralela de Q-Learning, se ejecutó utilizando un número variable de procesadores, *np*, que en particular tomó los valores 2, 4 y 8 respectivamente. En todos los casos, el conjunto total de estados fue particionado de manera tal de lograr un número equivalente de estados en cada procesador. Las particiones fueron realizadas sobre la base de la numeración consecutiva que se asignó a cada uno de los estados, lo que derivó en una partición de tipo horizontal de los estados del laberinto.
Otro aspecto que fue analizado en el algoritmo paralelo fue la influencia de tamaño de la caché, determinado por el parámetro *cache_size*. Para los distintos experimentos, variando el número de procesadores, se tomaron como valores de *cache_size* 1, 2, 4 y 8.
En todas las comparaciones realizadas, se utilizaron los resultados promediados de diez repeticiones del experimento.

La Figura 4 muestra el tiempo promedio insumido por las versiones secuenciales y paralelas, en todas aquellas ejecuciones en que se convergió exactamente a la política óptima. Estos resultados son los mejores obtenidos con los distintos tamaños de cache (para *np*>1) y los cuales resultaron ser: *cache_size* =8 para *np*=2*; cache_size* =4 para *np* =4 y *cache_size* =8 para *np* =8.

np	1	2	4	8
tiempo	4.326607	3.40059	1.85736	2.659
aceleración	----------	1.27	2.32	1.62

Figura 4

Como se puede observar en la Figura 4, para este problema particular, el algoritmo paralelo mantiene una aceleración creciente hasta *np*=4. Sin embargo, en este problema pequeño, un aumento mayor en *np* no produce un incremento en la aceleración ya que la cantidad de carga asignada a cada procesador es baja comparada con el overhead de comunicación involucrado.

Si bien los resultados anteriores, son claros indicadores de la performance final de la versión secuencial versus la versión paralela, es interesante analizar el porqué de estos resultados, observando sus comportamientos durante el proceso de aprendizaje. Para ello se muestra en la Figura 5 el porcentaje de estados óptimos aprendidos en distintos instantes de tiempo (cada 50 épocas).

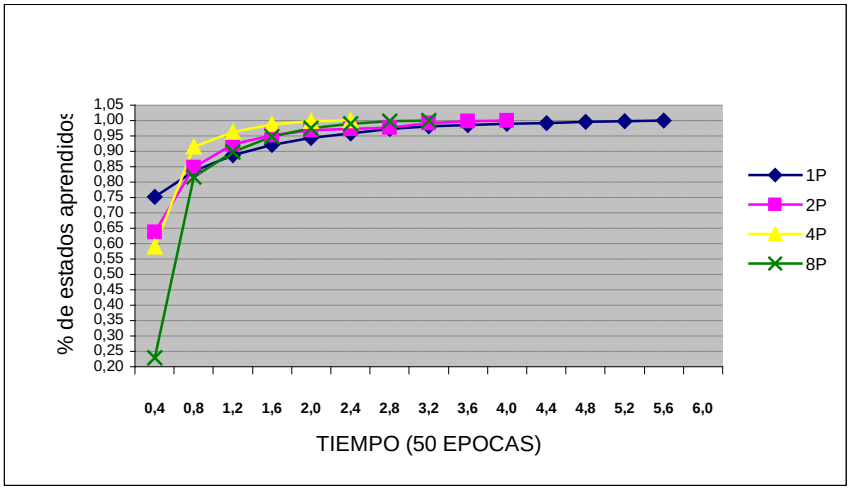

Figura 5

En esta figura se puede visualizar un porcentaje global del grado de aprendizaje, que consiste en medir el porcentaje de estados en los cuales todas las acciones sugeridas por la tabla Q, son coincidentes con las de la política óptima. Para cada configuración de procesadores particular, se grafica hasta el momento en que todas las corridas utilizadas para promediar los resultados, convergieron a la política óptima. Se puede observar que en las primeras épocas este porcentaje es inversamente proporcional a *np*. Esto no es extraño si se considera que la información en Q-Learning es propagada en forma *backward* desde los estados objetivos, por lo que en las primeras épocas, los procesos slaves que trabajan sobre los estados intermedios del laberinto, todavía no han recibido información relevante desde los estados vecinos. Sin embargo, una vez que esta información es disponible para todos los procesadores, la convergencia exacta a la política óptima se realiza en forma mucho más rápida que el secuencial, el cual demorará un tiempo considerablemente mayor que todas las configuraciones paralelas de Q-Learning.

Un último aspecto que merece ser considerado es el relativo al grado de certeza de convergencia al óptimo, dado un número máximo de episodios para ejecutar el algoritmo. Para este ejemplo particular, se tomó un máximo de 32000 episodios y en la fila 2 de la Figura 4, se pudo observar los tiempos de convergencia en aquellos casos que con esta cantidad de episodios, se convergió exactamente a la política óptima. Estos resultados, sin embargo, no dicen nada respecto al porcentaje de corridas en los que se convergió exactamente a la política óptima.

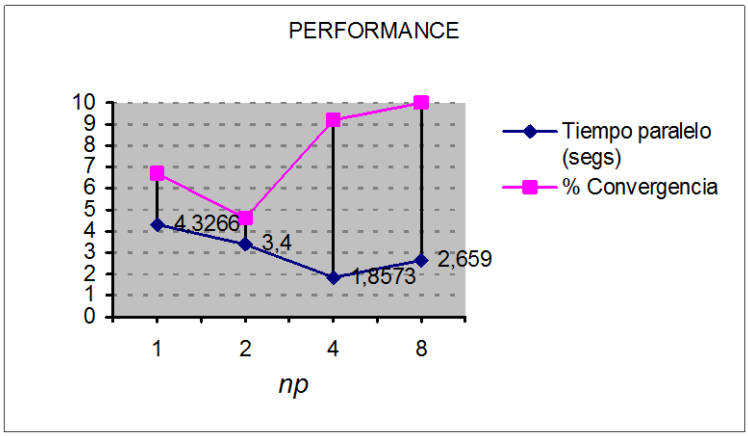

Figura 6

A nuestro criterio, un análisis mas realista de la performance de los algoritmos debe considerar este aspecto, por lo que en la Figura 6, se muestra demás de los tiempos de convergencia a la política óptima, el porcentaje de corridas (multiplicado por 10) en las que se convergió exactamente al óptimo.
Como se puede observar, la versión paralela con *np*=4, no sólo exhibe un buen comportamiento en cuanto a tiempo, sino que además converge en más del 90% de los casos, dado el límite de 32000 episodios. Sin embargo cuando *np*=8, si bien no se mejora el tiempo obtenido con 4 procesadores, ésta converge en el 100% de los casos. Esto está indicando que un incremento en el número de procesadores puede no mejorar el tiempo del algoritmo pero puede asegurar un mayor porcentaje del número de convergencias, si existen restricciones en el número de episodios permitidos.

6. Conclusiones

En este trabajo, se presentó una implementación paralela del algoritmo de Q-Learning, basada en un esquema de comunicación con caché.
Los resultados obtenidos sugieren la factibilidad del enfoque propuesto no sólo en cuanto a la aceleración obtenida en relación a la versión secuencial, sino también en el aspecto de garantía de convergencia a la política óptima cuando existen restricciones en el número de episodios para correr los algoritmos.
Si bien el problema seleccionado fue pequeño (135 estados) se considera que las aceleraciones alcanzadas fueron significativas y que tendrán un impacto práctico considerable cuando este enfoque sea aplicado a problemas reales de decisión secuencial que involucren miles o millones de estados.
Finalmente, y en relación a los aspectos de implementación de la versión paralela, se pudo observar que la estrategia de descomposición de dominios utilizada facilitó la tarea de programación, y que potencialmente sería adecuada para escalar el enfoque propuesto al tratamiento de problemas mayores.

7. Referencias

[1] A. G. Barto, S. J. Bradtke y S. P. Singh. "Learning to act using real-time dynamic programming". *Artificial Intelligence*, 72: 81-138, 1995.
[2] D. P. Bertsekas. "Distributed dynamic programming". *IEEE Transactions on Automatic Control*, 27: 610-616, 1982.
[3] D. P. Bertsekas y J. N. Tsitsiklis. "Parallel and Distributed Computation: Numerical Methods". Prentice Hall, Englewood Cliffs, NJ, 1989.
[4] M.Errecalde, M. Crespo y C. Montoya. "Aprendizaje por Refuerzo: Un estudio comparativo de de sus principales métodos". Proc. del II Encuentro Nacional de Computación (ENC´99). Sociedad Mexicana de Ciencia de la Computación. México, 1999.
[5] I. Foster "Designing and Building Parallel Programs". Addison Wesley – 1995.
[6] R. Guerrero, F. Piccoli y M. Printista. " Parallelism and Granularity in an Echo Elimination System". Proceedings of 12th. International Conference on Control Systems and Computer Science. Vol. II – pags. 232-237, Bucharest, Romania, 1999.
[7] L. P.Kaelbling, M. Littman y A. Moore. "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research 4 (1996) - 237-285 - Mayo 1996.
[8] L.P.Kaelbling. Learning in Embedded Systems. MIT Press. Cambridge, MA, 1993.
[9] L. - J.Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". Machine Learning - Volumen 8 - Número 3/4 - Mayo 1992.
[10] R.Maclin y J. W. Shavlik. "Creating Advice-Taking Reinforcement Learners". Machine Learning - Volumen 22 - Págs. 251 - 282. 1996.
[11] T.Mitchell. "Machine Learning". Capítulo 13. (Versión preliminar).
[12] A.Moore y C. Atkeson. "Prioritized Sweeping: Reinforcement Learning with Less Data and Less Time ". Machine Learning - Volumen 13 - Número 1 - Octubre 1993.
[13] J.Peng y R. J. Williams. "Efficient learning and planning within the Dyna framework". Adaptative

Behavior, 1(4), Págs. 437-454, 1993.
[14] Power Mouse.in details "http://www.parsytec.de/top/products/pm-detail.htm
[15] S.Russell y P. Norvig. "Artificial Intelligence. A modern Approach".Prentice –Hall –1995.
[16] R.Sutton y A. Barto. " Reinforcement Learning: an introduction". The MIT Press, 1998.
[17] R.Sutton. "Dyna. an Integrated Architecture for Learning, Planning, and Reacting" Working Notes of the AAAI Spring Symposium, pp.151-155, 1991.
[18] R.Sutton. "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". – Proceedings of the Seventh Int. Conf. On Machine Learning, pp. 216-224, Morgan Kaufmann, 1990.
[19] S.Thrun y K. Moller. "Active exploration in dynamic environments". Advances in Neural Information Proccesing Systems, 4, pags. 531 - 538. San Mateo, CA, Morgan Kaufmann, 1992.
[20] S.Thrun. "The role of Exploration in Learning Control". Handbook in Intelligent Control: Neural, Fuzzy, and Adaptative Approaches, White, D. A., & Sofge, D. A. (Eds.).
[21] C. J. C. H. Watkins. "Learning from Delayed Rewards". PhD thesis, Cambridge University, 1989.
[22] C. J. C. H. Watkins y P. Dayan. "Q-Learning". *Machine Learning*, 8: 279-292, 1992.
[23] B. Wilkinson y M. Allen "Parallel Programming: Techniques and Application using Networked Workstations and Parallel Computers". Prentice Hall. – 1999.