PROPAGACIÓN ACOTADA EN BUSQUEDA HEURÍSTICA EN TIEMPO REAL EN ENTORNOS INICIALMENTE DESCONOCIDOS

por

Carlos Hernández Ulloa

Tesis propuesta para el doctorado en

Ciencias de la Computación e
Inteligencia Artificial

Escuela Técnica Superior de Ingeniería
Universidad Autónoma de Barcelona

Supervisor: Dr. Pedro Meseguer

Trabajo realizado en:

Instituto de Investigación en Inteligencia Artificial
Consejo Superior de Investigaciones Científicas
Bellaterra, Barcelona, España

Con el apoyo de:

Comisión Nacional de Investigación en Ciencia y Tecnología,
Gobierno de Chile
Universidad Católica de la Ssma. Concepción, Chile

Diciembre 2007

A mis padres que me permitieron intuir que no hay camino,
que se hace al andar.

AGRADECIMIENTOS

Quiero dar las gracias a mi supervisor Pedro Meseguer. Esta tesis se debe en gran parte a su apoyo y colaboración en las distintas etapas de mi trabajo de investigación. También quiero agradecer a la dirección, al personal administrativo y toda la gente del IIIA que han sido parte de mi vida en este período. Les agradezco su conversación, compañerismo y prestancia. En especial a Pablo Noriega, Juan Antonio Rodríguez, Dani Polak, Santi Ontañón, Martí Sánchez, Eloy Puertas, Carles Sierra y Mario Gomez. Agradezco a Vadin Bulitko de la Universidad de Alberta por facilitarnos los mapas de juegos para ordenador que usamos en el capítulo de aplicaciones y a Richard Korf por enviarnos la tesis de master de P. Thorpe.

Mi familia siempre me ha brindado apoyo en mis proyectos. Les agradezco ser parte de este. Gracias a Enedita mi madre, Carlos mi padre, Danilo, Alvaro y Cristian mis hermanos y a mi querida mujer Marcela. Agradezco a las personas que colaboraron en mi formación y me ayudaron a llegar a esta etapa. Gracias a Andreas Polymeris mi amigo y profesor en la Universidad de Concepción por su guía y conversación y a John Atkinson por introducirme en la Inteligencia Artificial.

Finalmente quiero agradecer a mis amigos en Barcelona, les doy las gracias por hacer mi estancia interesante y agradable. Con ellos compartí a fuerza vino y miel el aullido de esta bella ciudad. Mención para Carlo, Juan Pablo, Alvaro, Mark, Alejandra, Angel, Jordi Anarco, Viviana, Lucho del Cheroga y los sobrevivientes del bar Giner.

RESUMEN

Los problemas de búsqueda en donde el agente tiene un tiempo limitado para calcular una solución en un entorno inicialmente desconocido, no pueden ser abordados por mecanismo tradicionales de búsqueda heurística. Algunos problemas de este tipo son la planificación de rutas en: robots autónomos, personajes de juegos en tiempo real para ordenador y paquetes de información en redes de sensores.

Esta tesis esta dedicada a resolver este tipo de problemas, mediante algoritmos de búsqueda heurística en tiempo real. Presentamos varios mecanismos genéricos que aplicados a algoritmos existentes, generan nuevos algoritmos. Los nuevos algoritmos que desarrollamos mejoran el rendimiento de las aproximaciones existentes. Estos se evalúan considerando varias medidas de desempeño, las más importantes son: el coste de la solución, el tiempo total de búsqueda y el tiempo por episodio de planificación. Las principales ideas que hemos desarrollado en la tesis son:

Propagación acotada de cambios heurísticos con iteración de valores: Presentamos un mecanismo de aprendizaje de heurísticas basado en el método de iteración de valores de programación dinámica.

Propagación acotada de cambios heurísticos sobre un espacio local: Presentamos un mecanismo de aprendizaje de heurísticas que mejora el mecanismo basado en iteración de valores.

Combinación entre anticipación y propagación acotada: Presentamos un mecanismo que permite combinar anticipación con propagación acotada sobre un espacio local. Esta combinación mejora el coste de la solución aumentando el tiempo de planificación por paso.

Anticipación y movimientos del agente: Analizamos distintas estrategias de movimiento del agente. Cada estrategia tiene un rendimiento distinto, el uso de una u otra dependerá de la medida de desempeño que más importe al usuario.

Aplicaciones: Implementamos nuestros algoritmos en mapas extraídos de juegos en tiempo real comerciales para ordenador. El desempeño de los algoritmos es mejor que el desempeño de las principales aproximaciones de la comunidad de búsqueda heurística en tiempo real.

Todos los algoritmos propuestos en la tesis han sido implementados. Se presentan resultados formales y experimentales.

TABLA DE CONTENIDO

Capítulo uno

1 INTRODUCCIÓN

Esta tesis trata sobre búsqueda de rutas para un agente. La búsqueda está sujeta a restricciones en el tiempo de planificación de las acciones que debe ejecutar el agente, en un entorno que inicialmente desconoce. Esta problemática se aborda con algoritmos de búsqueda heurística en tiempo real [Korf, 1990]. La búsqueda heurística en tiempo real se caracteriza porque el agente planifica una solución parcial durante un tiempo limitado y luego ejecuta acciones. Este proceso se repite hasta solucionar el problema. En cada iteración planificación/ejecución de acciones, hay tres procesos comunes a la mayoría de los métodos: la anticipación, el aprendizaje de heurísticas (que se realizan durante la planificación de la solución parcial) y el movimiento del agente (o ejecución de acciones).

La contribución principal de este trabajo es la inclusión del mecanismo de propagación acotada de valores heurísticos. Este mecanismo mejora el aprendizaje de heurísticas que realiza el agente en cada episodio de planificación. Se presentan dos versiones de propagación acotada y se implementan sobre los algoritmos más representativos. El resultado de esta implementación, es una notable mejora en la calidad de la solución acompañado de un aumento moderado en el tiempo por episodio de planificación.

Otra novedad de este trabajo es la combinación del mecanismo de propagación acotada con un mecanismo de anticipación acotada. Esta combinación permite obtener soluciones de mejor calidad, pero con un aumento importante del tiempo por episodio de planificación. De esta combinación hemos concluido, a partir de los resultados experimentales, que es preferible usar una mayor cantidad de recursos en propagar en lugar de anticipar, para obtener buenas soluciones manteniendo un tiempo de planificación moderado.

Finalmente implementamos los algoritmos presentados en una aplicación real. La implementación se hace sobre mapas extraídos de juegos en tiempo real comerciales para ordenador: "Baldur's Gate" [BioWare Corp., 1998] y "Warcraft III: Reign of chaos" [Blizzard Entertainment., 2002]. Como se puede observar en los resultados experimentales, las soluciones que obtenemos son de mejor calidad que las obtenidas por las aproximaciones más relevantes del estado del arte.

1.1 Motivación

Los métodos de búsqueda heurística clásica [Pearl, 1984] permiten resolver problemas búsqueda de rutas para un agente. Estos métodos asumen que el entorno es conocido y se cuenta con el tiempo suficiente para calcular una solución completa. En muchos problemas reales de búsqueda de rutas estas suposiciones no se cumplen. Por ejemplo, la búsqueda que realiza un agente o robot en un entorno que desconoce a priori [Koenig et al. 2003], los personajes de video juegos para ordenador [Bulitko y Lee, 2006] y los paquetes de información en redes ad-hoc [Shang et al. 2003]. La motivación de esta tesis es la resolución de problemas reales de búsqueda de rutas, que no se pueden resolver con los métodos clásicos. En concreto la resolución de problemas en los que la búsqueda esta sujeta a restricciones de tiempo, en un entorno inicialmente desconocido por el agente. Para esto se proponen un conjunto de algoritmos de búsqueda heurística que se presentan a largo de este trabajo.

1.2 Orientación y alcance

Los límites de este trabajo están sujetos a las siguientes decisiones:

- Aproximación genérica. El trabajo presentado en esta tesis es genérico, en el sentido de que no se presupone ninguna característica específica de los problemas considerados. Es aplicable a cualquier problema resoluble mediante búsqueda heurística en tiempo real.

- Búsqueda heurística en tiempo real. Todos los algoritmos que se presentan son de búsqueda heurística en tiempo real para un agente. Nuestro trabajo se basa en los algoritmos LRTA*, RTA* [Korf, 1990] y HLRTA* [Thorpe, 1994]. Se asume que se conoce el efecto de las acciones [Russell y Norvig, 2003]. La aplicación de los métodos propuestos en entornos no-deterministas se considera como investigación futura.

- Tiempo de planificación limitado. El tiempo de planificación del agente es limitado. Para limitar el tiempo de planificación, generalmente se usa un parámetro para controlar la cantidad de anticipación y otro para controlar la cantidad de propagación. En algunas de nuestras aproximaciones sólo controlamos la propagación porque usamos la cantidad mínima de anticipación.

- Evaluación empírica. Este trabajo tiene una orientación práctica. Las contribuciones están sustentadas principalmente por evaluaciones empíricas. En nuestros experimentos usamos escenarios de prueba ampliamente utilizados por la comunidad que trabaja en búsqueda heurística en tiempo real. Comparamos nuestras contribuciones con otras de la comunidad.

- Aplicaciones reales. Nuestro trabajo esta motivado por aplicaciones reales. El capítulo 8 presentamos una aplicación en mapas de juegos de ordenador. Como trabajo futuro se pueden aplicar los métodos propuestos a otros problemas reales de búsqueda.

1.3 Contribuciones

Las contribuciones principales de esta tesis son:

- *Propagación acotada de cambios heurísticos con iteración de valores*. Presentamos la estrategia de propagación acotada del cambio en la heurística del estado

actual, mediante un procedimiento de iteración de valores. Con la estrategia se realiza un aprendizaje de heurísticas más intensivo en cada episodio de planificación, permitiendo obtener soluciones de mejor calidad en los algoritmos en que se aplica.

- *Soportes de una estimación heurística.* Los soportes permiten mejorar el rendimiento de la propagación con iteración de valores.

  Estas dos contribuciones fueron publicadas en:

  - Carlos Hernández, Pedro Meseguer. LRTA*(*k*). **In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)**. pp. 1238 – 1243, 2005.

  - Carlos Hernández, Pedro Meseguer. Propagating updates in real-time search: HLRTA*(*k*). Current topics in artificial intelligence. In Proceedings of 11th Conference of the Spanish Association for Artificial Intelligence, (CAEPIA 2005). **Lecture notes in computer science**, Vol. 4177, pp. 379 - 388. Springer. 2006.

- *Versiones del mecanismo de propagación acotada.* Presentamos distintas versiones del mecanismo de propagación, que dependiendo del problema mejoran los resultados originales. Estas ideas fueron publicadas en:

  - Carlos Hernández, Pedro Meseguer. LRTA*(*k*). **Workshop on Planning and Learning in a Priori Unknown or Dynamic Domain at the 19th International Joint Conference on Artificial Intelligence (IJCAI-2005)**. Vadim Bulitko, Sven Koenig eds., pp. 69 – 75. 2005.

- *Propagación acotada sobre un espacio local.* La propagación acotada con iteración de valores tiene algunos puntos débiles que se resuelven total o parcialmente con el mecanismo de propagación acotada sobre un espacio

local. Con el nuevo mecanismo de propagación se mejora el rendimiento obtenido por el mecanismo con iteración de valores. Esta contribución fue publicada en:

- Carlos Hernández, Pedro Meseguer. Improving LRTA*(*k*). **In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)**. pp. 2312– 2317, 2007.

- *Anticipación*. Presentamos un mecanismo de anticipación acotada. El mecanismo esta limitado por la calidad de la heurística y un parámetro dado por el usuario.

- *Movimientos en el espacio local de búsqueda.* Presentamos una manera de decidir entre ejecutar uno o varios movimientos en el espacio local de búsqueda, dependiendo de la calidad de la heurística.

- *Aplicación*. Presentamos una aplicación de los mecanismos propuestos en mapas de juegos de tiempo real para ordenador. Veremos que nuestras aproximaciones mejoran a las más representativas del estado del arte.

  Estas tres contribuciones han sido total o parcialmente tratados en las publicaciones:

  - Carlos Hernández, Pedro Meseguer. Improving Real-Time Heuristic Search on Initially Unknown Maps. **Workshop on Planning in Games at the International Conference on Automated Planning & Scheduling (ICAPS-07)**. 2007.

  - Carlos Hernández, Pedro Meseguer. Improving HLRTA*(*k*). In Proceedings of 12th Conference of the Spanish Association for Artificial Intelligence, (CAEPIA 2007). **Lecture notes in computer science**. 2007.

- *Generalización de la propagación acotada sobre un espacio local.* Presentamos un mecanismo que generaliza el mecanismo de propagación acotada sobre un espacio local. El mecanismo permite iniciar la propagación desde varios estados con heurística inexacta. Al combinar este mecanismo con anticipación se puede realizar un mayor aprendizaje de heurísticas por episodio de planificación.

1.4 Estructura de la tesis

Este documento esta divido en cuatro partes. La parte II contiene las contribuciones de nuestro trabajo.

- *Parte I Preliminares.* Esta parte incluye los capítulos 2 y 3. En el capítulo 2 se describe el problema a resolver en detalle, los escenarios de prueba y las medidas de desempeño utilizadas en los resultados experimentales. El capitulo 3 describimos búsqueda heurística en tiempo real y presentamos un estado del arte. En estos capítulos realizamos las definiciones que se utilizan en los capítulos siguientes.

- *Parte II Contribuciones.* Esta parte comprende los capítulos 4, 5, 6 y 7. En el capitulo 4 se presenta el mecanismo de propagación acotada con iteración de valores. En el capitulo 5 se presenta el mecanismo de propagación acotada sobre un espacio local. En el capitulo 6 se presentan los mecanismos de anticipación, propagación acotada sobre un espacio local generalizada y movimientos en el espacio local de búsqueda. Las contribuciones son implementadas sobre el algoritmo LRTA* [Korf. 1990]. El capitulo 7 contiene la implementación de las contribuciones presentadas en los capítulos 4, 5 y 6 sobre HLTRA* [Thorpe, 1994].

- *Parte III Aplicaciones.* Esta parte es el capítulo 8. En este capítulo implementamos los algoritmos presentados en los capítulos anteriores en mapas de juegos de tiempo real para ordenador. Comparamos nuestras aproximaciones con las más relevantes del estado del arte.

- *Parte V Conclusiones*. Esta parte contiene el capítulo final. En el capítulo 9 presentamos las conclusiones y el trabajo futuro.

Capítulo dos

2 PROBLEMA Y ESCENARIOS DE PRUEBA

En este capítulo describimos el problema que motiva los métodos búsqueda heurística en tiempo real que se presentan a lo largo de la tesis. Después, presentamos los escenarios de prueba usados para obtener resultados experimentales. Por último, exponemos las medidas de desempeño utilizadas en la evaluación de las distintas aproximaciones.

2.1 Descripción del problema

La búsqueda heurística es un mecanismo de resolución de problemas en inteligencia artificial [Nilsson, 1971]. Uno de los problemas que le concierne es la búsqueda de rutas para un agente. La búsqueda de rutas se ejecuta sobre el espacio de estados de un problema. Un estado es una configuración determinada del problema. Es posible obtener los estados sucesores de un estado cualquiera, aplicando las acciones legales (u operadores). La ejecución de cada acción legal tiene un coste asociado. Se asume que el problema es determinista, por tanto se conoce el efecto de las acciones [Russell y Norvig, 2003]. El espacio de estados tiene la forma de un grafo. Los nodos del grafo corresponden a los estados y los arcos a los operadores. Los arcos pueden ser dirigidos o no dirigidos, dependiendo de si el operador correspondiente es reversible o no. Por ejemplo en la Figura 1, el problema es un laberinto. En el laberinto las celdas son los estados. Las celdas de color negro están bloqueadas. Las acciones legales son los movimientos al norte, sur, este u oeste desde una celda a otra, por tanto cada estado tiene como máximo 4 sucesores. En (ii), podemos observar el grafo que representa el espacio de estados del problema. En el grafo hay algunos estados distinguidos, que son el estado inicial y el estado (o conjunto de estados) objetivo. En (i), se puede apreciar el estado inicial y objetivo. Otro estado que se distingue, es el *estado* (o posición) *actual* del agente, mientras recorre el espacio de estados. Un *ejemplar de un problema* consiste en un espacio de estados más un estado inicial y

objetivo(s) concretos. Una *solución* es la secuencia de acciones que llevan desde el estado inicial al estado objetivo. El coste de una solución es la suma del coste de la acciones. En la Figura 1(iii), la flecha representa una solución al ejemplar del laberinto.

En este trabajo nos centramos en resolver el problema de búsqueda de rutas para un agente cuando el entorno es inicialmente desconocido y cuando el agente tiene un tiempo limitado para planificar las acciones que ejecuta en el entorno. A continuación describimos estas dos problemáticas adicionales a la búsqueda de rutas.

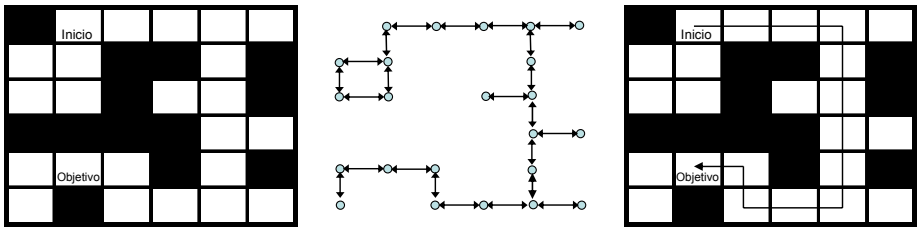


Figura 1. En (i) se puede apreciar el ejemplar de un laberinto con su estado inicial y objetivo. En (ii) el grafo que representa el espacio de estados. En (iii) una solución al ejemplar del problema.

2.1.1 Entorno inicialmente desconocido

En búsqueda de rutas desde un estado inicial a un estado objetivo, en un entorno inicialmente desconocido, el agente sólo puede censar los estados alrededor del estado actual dentro de su *rango de visibilidad*. El rango de visibilidad esta limitado por el alcance de los sensores del agente. Además, el agente puede recordar la porción del entorno o espacio de estados que ha visitado previamente [Koenig, 2004][Bulitko y Lee, 2006]. Un ejemplo de esta situación la podemos ver en la cuadrícula 4-conectada de la Figura 2. Supongamos que el rango de visibilidad tiene profundidad uno, es decir el agente sólo puede censar los estados vecinos inmediatos al estado actual. En (i) podemos ver el mapa exacto. En (ii) el agente, representado por el círculo negro esta en el estado inicial. Este sólo puede censar los estados vecinos. En

(iii) después de algunos movimientos, el agente aumenta su conocimiento del entorno.

En nuestro problema de búsqueda un estado puede encontrarse *bloqueado* cuando no puede ser visitado por el agente (por ejemplo los estados bloqueados del laberinto en la Figura 1), o *libre* cuando se puede visitar. Para hacer el problema de búsqueda de rutas viable en entornos desconocidos, se usa el *supuesto de espacio libre*: si un estado no esta en el rango de visibilidad y no hay evidencia que el estado este bloqueado, se asume que esta libre [Koenig et al. 2003].

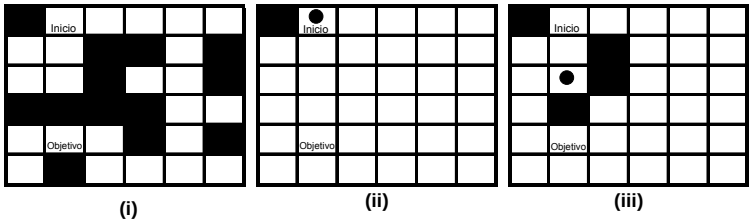


Figura 2. Ejemplo de búsqueda de rutas en un entorno inicialmente desconocido en una cuadrícula 4-conectada. (i) el mapa exacto, (ii) el agente, que se representa con un círculo negro, esta el estado inicial, (iii) después de algunos pasos, el agente aumenta su conocimiento del mapa.

2.1.2 Tiempo limitado de planificación

El agente no cuenta con un tiempo ilimitado para planificar los movimientos que debe hacer en su entorno. Este debe moverse en una cantidad de tiempo limitada. Un ejemplo es la búsqueda de rutas en personajes de juegos de ordenador en tiempo real. En los juegos, un personaje debe moverse rápidamente desde su posición actual en el terreno, hasta una posición indicada por un clic de ratón hecho por el usuario. La planificación debe ser rápida porque se comparte el procesador con varias tareas y personajes del juego, y el usuario requiere respuestas en tiempo real [Bulitko y Lee, 2006] [Koenig y Likhachev, 2006].

2.2 Escenarios de prueba.

Los escenarios de prueba que se utilizan para obtener resultados experimentales y evaluar las distintas aproximaciones que se presentan en este trabajo, son tres tipos de cuadrículas 4-conectadas:

- **Cua-35**: cuadrículas de tamaño 301x301, con un 35% de obstáculos ubicados de manera aleatoria.

- **Lab-acic**: laberintos acíclicos de tamaño 181x181. La estructura de los corredores fue generada con la estrategia de búsqueda primero en profundidad.

- **Lab-cic**: laberintos cíclicos de tamaño 181x181. La estructura de los corredores fue generada con la estrategia de búsqueda primero en profundidad y luego se eliminaron 70 paredes de manera aleatoria.

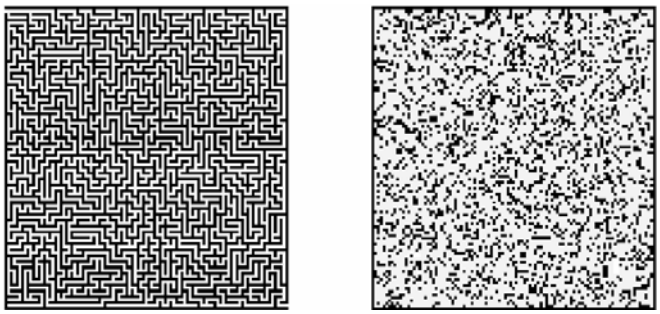


Figura 3. Ejemplo de los escenarios de prueba usados en esta tesis.

En la Figura 3, se pueden apreciar los tipos de escenarios de prueba utilizados. Estos tipos de escenario, han sido utilizados en muchos trabajos relevantes en búsqueda heurística en tiempo real [Ishida y Korf, 1991] [Shimbo e Ishida, 2003] [Koenig, 2004] [Hernández y Meseguer, 2005a] [Koenig y Likhachev, 2006] [Hernández y Meseguer, 2007a].

En los escenarios el agente se puede mover en cuatro direcciones: norte, oeste, sur y este. Todas las acciones tienen coste 1. El radio de visibilidad del agente es 1, es decir sólo puede censar el estado actual y los sucesores

inmediatos. La heurística inicial entre dos estados es la distancia Manhattan. Es importante mencionar que en Cua-35 la heurística inicial tiende a ser ligeramente errónea y en Lab-acic es muy errónea. Lab-cic es un caso intermedio. Todos los resultados que se presentan son promedios sobre 1500 ejemplares distintos de cada escenario. Los estados inicial y objetivo fueron elegidos de manera aleatoria, asegurando que exista una ruta entre ellos.

En el capítulo 8 se implementan los métodos de búsqueda de rutas presentados en este trabajo, en mapas de juegos de ordenador en tiempo real.

2.3 Medidas de desempeño

Antes de presentar las medidas de desempeño, definiremos algunos términos que se usan a lo largo de este trabajo.

- Ejecución. Una ejecución es encontrar una solución a un ejemplar de un problema.
- Solución óptima. Es la solución de menor coste.
- Primera ejecución. Es la primera vez que se resuelve el ejemplar de un problema.
- Convergencia. Es el proceso de obtener la solución óptima de un ejemplar de un problema. Esto requiere varias ejecuciones sobre el mismo ejemplar, hasta converger a la solución óptima.

El desempeño de los métodos se evalúa en la primera ejecución y en convergencia. A continuación presentamos las medidas utilizadas en esta tesis.

Las medidas usadas en la primera ejecución son las siguientes.

- El coste de la solución: es igual al número de acciones realizadas en la solución, ya que todas las acciones tienen coste 1.
- El tiempo total de búsqueda: es el tiempo requerido en la obtención de la solución.

- La memoria: es la cantidad de estados que cambian su heurística mientras se realiza la búsqueda.
- El tiempo por episodio de planificación: es el tiempo promedio en que se realiza cada episodio de planificación en una solución. Cuando el agente realiza un paso (o acción) por episodio de planificación, el tiempo por episodio de planificación es lo mismo que el tiempo por paso. Se calcula dividiendo el tiempo total de búsqueda por el número de episodios de planificación (o pasos si se realiza una acción por episodio de planificación) que se realizan en una ejecución.

En convergencia las medidas de desempeño son las siguientes:

- El coste de la solución óptima: es el número de acciones realizadas para obtener la solución óptima.
- El tiempo total de convergencia: es el tiempo requerido para obtener la solución óptima.
- El número de ejecuciones: es la cantidad de ejecuciones que se hicieron sobre el mismo ejemplar para converger a una solución óptima.
- La memoria: es la cantidad de estados que cambian su heurística en el proceso de convergencia.
- El tiempo por episodio de planificación: es el tiempo promedio en que se realiza cada episodio de planificación en convergencia. Se calcula dividiendo el tiempo total de convergencia por el número de episodios de planificación (o pasos si se realiza una acción por episodio de planificación) que se realizan en convergencia.

En algunos casos se evalúan otras medidas de desempeño. Estas medidas se explicarán oportunamente antes de usarlas.

Capítulo tres

3 PRELIMINARES

En este capítulo, presentamos las características principales de la búsqueda en tiempo real, en contraste con la búsqueda heurística clásica. Vemos por qué es adecuada al tipo de problema que hemos definido en el capítulo anterior y presentamos los conceptos principales. Después, vemos algunas definiciones que se usarán a lo largo de este documento. Finalmente, presentamos un estado del arte.

3.1 Búsqueda heurística en tiempo real.

En el marco de este trabajo, se distinguen dos tipos de búsqueda heurística, la búsqueda tradicional o fuera de línea y la búsqueda en línea [Koenig, 2001a]. La búsqueda en línea es el tipo de búsqueda que nos permite resolver la clase de problemas que nos interesa. Con el objetivo de contrastar los esquemas de funcionamiento, explicamos a continuación ambos tipos de búsqueda.

3.1.1 Búsqueda tradicional o fuera de línea

Los métodos clásicos de búsqueda heurística siguen el siguiente esquema: Primero, calculan la solución de un ejemplar de un problema, mediante búsqueda en el espacio de estados. Segundo, se ejecuta la solución realizando las acciones o secuencia de operaciones en el entorno. Este esquema, a menudo considerado de forma implícita, supone que ejecutar la solución en el mundo real es un proceso directo y no presenta ninguna dificultad. La parte difícil es calcular la solución. También se supone que el problema no esta sujeto a restricciones computacionales, de información o tiempo. En la Figura 4 se observa el esquema de funcionamiento: primero se realiza una búsqueda completa en el espacio de estados y luego se ejecuta una secuencia de acciones u operaciones en el entorno.

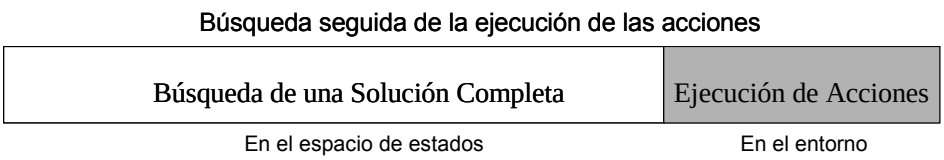


Figura 4. Esquema de funcionamiento de los métodos búsqueda tradicional o fuera de línea.

Un ejemplo típico de aplicación de los métodos fuera de línea es la búsqueda de rutas. La búsqueda de rutas consiste en encontrar una ruta en un grafo desde el nodo inicial hasta el nodo objetivo [Korf, 1999]. Algunos problemas clásicos de búsqueda de rutas en la literatura de inteligencia artificial son: 8-puzzle, el problema del vendedor viajero, navegación de vehículos y cableado de circuitos VLSI. A* [Hart, Nilsson and Raphel, 1968] y IDA* [Korf, 1985] son dos de los algoritmos más populares de búsqueda fuera de línea.

Como se mencionó, en una búsqueda fuera de línea se supone que los problemas no están sujetos a restricciones de información o tiempo. Por tanto, no es un esquema adecuado para resolver la clase de problemas presentados en esta tesis. La búsqueda tradicional es adecuada para problemas totalmente controlables en donde no se necesita ejecutar la solución o bien la ejecución es directa.

3.1.2 Búsqueda en línea

A este tipo de búsqueda se le llama "en línea" porque intercala entre búsqueda en el espacio de estados y ejecución de acciones en el entorno. A continuación presentamos dos tipos de algoritmos de búsqueda en línea.

3.1.2.1 Búsqueda heurística incremental

Los métodos de búsqueda incremental como dynamic A* (D*) [Stenz, 1995] y D* Lite [Koenig y Likhachev, 2002] son algoritmos en línea que pueden trabajar en entornos inicialmente desconocidos. Estos algoritmos calculan una solución inicial completa y luego la ejecutan. Si el agente encuentra un obstáculo mientras ejecuta la solución, repara la heurística en la porción del espacio de estados que se ve afectada por el descubrimiento y calcula una

nueva ruta a seguir, a partir de los nuevos valores heurísticos. Estos métodos trabajan bien cuando los movimientos del agente son lentos respecto a velocidad de planificación [Koenig, 2004] o cuando no existen restricciones en el tiempo de búsqueda por episodio de planificación. Si un agente necesita realizar un primer movimiento rápido hacia su objetivo, como sucede en algunos juegos comerciales de ordenador [Bulitko y Lee, 2006], puede suceder que el agente no cuente con el tiempo suficiente para calcular una solución inicial completa. Debido a esto los métodos de búsqueda incremental no son adecuados para el tipo de problemas abordado en este trabajo.

3.1.2.2 Búsqueda heurística en tiempo real

En búsqueda heurística en tiempo real (BHTR), el calificativo "tiempo real" se debe a que cada acción individual ha de ser ejecutadas en un tiempo limitado. Generalmente, se requiere sacrificar la optimalidad de la solución. En la Figura 5 se observa el esquema de funcionamiento de los métodos: primero se realiza una búsqueda durante un tiempo limitado, y luego se ejecutan acciones en el entorno. Se intercala entre estos dos episodios hasta solucionar el problema.

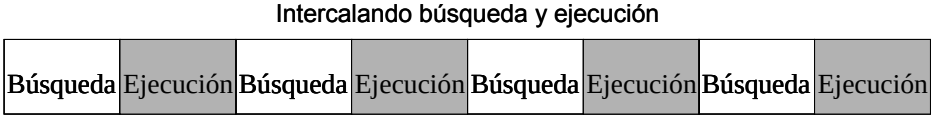


Figura 5. Esquema de funcionamiento de los métodos búsqueda heurística en tiempo real.

La búsqueda (antes de cada episodio de ejecución de acciones), usa una *anticipación* limitada. La anticipación, permite explorar hasta cierta profundidad el espacio de estados alrededor del estado actual. El agente anticipa, para tomar una buena decisión (en términos de coste), sobre qué acciones debería ejecutar. Existen distintas estrategias de anticipación. En el capítulo 6, veremos las más relevantes. Denominaremos *espacio local de búsqueda*, al

conjunto de estados cuyos valores heurísticos son accedidos durante la anticipación, antes de cada episodio de ejecución de acciones.

Otra característica importante en BHTR es el *aprendizaje* de heurísticas. El aprendizaje fue introducido inicialmente para evitar ciclos infinitos [Korf, 1990]. El aprendizaje consiste en *actualizar* la información heurística de algunos estados, a partir de la información de otros. Típicamente, el aprendizaje se realiza antes del episodio de ejecución de acciones y esta acotado por algún parámetro. Existen distintas estrategias de aprendizaje. A lo largo de este documento veremos varias (incluyendo las contribuciones principales de esta tesis). Denominaremos *espacio local de aprendizaje*, a todos los estados que pueden cambiar su heurística, antes de cada episodio de ejecución de acciones.

En búsqueda tradicional se busca en el espacio de estados y se ejecutan acciones. La búsqueda heurística en tiempo real incluye el aprendizaje. En BHTR se realiza una búsqueda con anticipación limitada y un aprendizaje limitado a cierto número de estados. La duración de ambos procesos es controlada con un parámetro. Denominaremos un *episodio de planificación*, a la búsqueda y aprendizaje que realiza el agente, antes de cada episodio de ejecución de acciones en el entorno.

El intercalar entre episodios de planificación (durante un tiempo limitado) y ejecución de acciones, permiten a este tipo de métodos resolver problemas sujetos a limitaciones de información o tiempo. Por tanto, son adecuados para búsqueda de rutas en entornos inicialmente desconocidos, cuando el agente tiene un tiempo limitado para planificar sus movimientos.

Existe cierta discusión sobre la cantidad de acciones que el agente debe realizar después de cada episodio de planificación. En el capítulo 6, discutimos diferentes alternativas.

Algunas aplicaciones recientes en búsqueda heurística en tiempo real son: ruteo en redes ad-hoc [Shang et al. 2003], planificación de ruta para múltiples objetivos desde un avión [Howlett, 2002], ruteo en redes de sensores [Zhang y Fromherz, 2004], planificación de ruta en juegos de ordenador en tiempo real [Goldenber et al. 2003] [Bulitko y Lee, 2006] y navegación en robots autónomos [Koenig, 2001b].

3.2 Espacio de estados.

El espacio de estados del problema esta definido por la tupla (X, A, c, s_0, G), donde (X, A) es un grafo finito, $c : A \rightarrow (0, \infty)$ es una función de coste que asocia a cada arco un coste finito positivo, s_0 es el estado inicial, $G \subset X$ es el conjunto de estados objetivos. X es un conjunto finito de estados, y $A \subset X \times X \setminus \{(x, x) | x \in X\}$ es un conjunto finito de arcos. Cada arco (v, w) representa una acción, cuya ejecución causa que el agente se mueva de v a w, se excluye las acciones que no cambian de estado. El espacio de estados es no dirigido: para cualquier acción $(x, y) \in A$ existe su inversa $(y, x) \in A$ con el mismo costo $c(x, y) = c(y, x)$. Una ruta $(x_0, x_1, x_2,....)$ es una secuencia de estados en que cada par $(x_i, x_{i+1}) \in A$. $k(n, m)$ es el coste de una ruta entre los estados n y m, se calcula sumando el coste de las acciones en la ruta. $Succ(x) = \{y | (y, x) \in A\}$ corresponde a los sucesores de un estado x. Una función heurística $h : X \rightarrow [0, \infty)$ asocia a cada estado x una aproximación $h(x)$ del coste de la ruta entre x y el estado objetivo, donde $h(s_g) = 0$, $s_g \in G$. El coste óptimo $h^*(x)$ es el coste de la ruta entre x y el estado objetivo con valor mínimo. Las función h es admisible si y solo si $\forall\ x \in X$, $h(x) < h^*(x)$. Las función h es consistente si y solo si $0 \leq h(x) \leq c(x, w) + h(w)\ \forall\ w \in Succ(x)$. Una ruta $(x_0, x_1, x_2,....x_n)$ con $h(x_i) = h^*(x_i)\ \forall\ i \in [0, n]$, es una ruta óptima. El valor de la función h_0 para un estado x corresponde a la estimación heurística inicial de $h(x)$.

3.3 Algoritmos de búsqueda heurística en tiempo real

En esta sección se presentan los algoritmos más relevantes en búsqueda heurística en tiempo real. Los algoritmos que se usan en esta tesis se describen en detalle.

*3.3.1 Real-Time A**

Real-Time A* (RTA*) [Korf, 1990], es probablemente el primer algoritmo propuesto de búsqueda heurística en tiempo real. Su funcionamiento es el siguiente: si x es el estado actual, en cada paso RTA* actualiza $h(x)$ con el segundo valor mínimo de $c(x,z) + h(z)$, $z \in Succ(x)$ y selecciona el estado y, que minimiza la expresión $c(x,z) + h(z)$, $z \in Succ(x)$, como próximo estado actual. Por ejemplo, supongamos la situación de la Figura 6 (i), los números son las estimaciones heurísticas admisibles de cada estado, el estado con $h = 0$ es el estado objetivo, moverse de un estado a otro cuesta 1 y el estado actual es donde se encuentra el agente. RTA* actualiza la heurística del estado actual x con $h(x) = \text{secondmin}_{z \in Succ(x)} [c(x,z) + h(z)] = 1 + 3 = 4$, y selecciona como siguiente estado a $y = \text{argmin}_{z \in Succ(x)} [c(x,z) + h(z)]$, el valor que minimiza la expresión es $1 + 2 = 3$. En la (ii) se aprecia la actualización y movimiento del agente. El nuevo valor sobrevalora la estimación heurística, ya que la distancia real desde el estado es 3 y no 4.

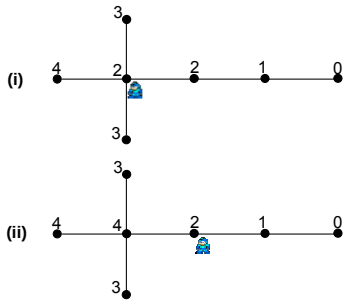


Figura 6. Ejemplo de actualización de la heurística del estado actual con RTA*. Se puede apreciar que RTA* provoca la pérdida de admisibilidad en la heurística.

Si se guardan los valores de h para otras ejecuciones, puede suceder que el nuevo h sea no admisible en algunos estados. Debido a esto RTA* no garantiza la convergencia a rutas óptimas ya que puede sobrevalorar el coste óptimo [Korf, 1990].

En un espacio de estados finito con costes de arcos positivo, valores heurísticos positivos y en donde se puede alcanzar el estado objetivo desde cualquier estado, RTA* es correcto y completo (Teorema 1 y 2 en [Korf, 1990])

*3.3.2 Learning Real-Time A**

Learning Real-Time A* (LRTA*) [Korf, 1990], es uno de los algoritmos más populares en búsqueda heurística en tiempo real. Se diferencia de RTA* en el mecanismo de actualización de la estimación heurística del estado actual. LRTA* actualiza $h(x)$ con el primer valor mínimo de $c(x,z) + h(z)$, $z \in Succ(x)$, en vez del segundo. Si la heurística inicial es admisible, el algoritmo mantiene la admisibilidad y por tanto converge a soluciones óptimas sobre ejecuciones sucesivas sobre un mismo ejemplar de un problema.

```
procedure LRTA*(X, A, c, s0,G)
   for each x ∈ X do h(x) ← h0(x);
   repeat
      LRTA*-trial(X, A, c, s0,G);
   until h does not change;

procedure LRTA*-trial(X, A, c, s0,G)
   x ← s0;
   while x ∉ G do
      dummy ← LookaheadUpdate1(x);
      y ← argmin_{w∈Succ(x)} [c(x,w) + h(w)]; (Break ties randomly)
      execute(a ∈ A such that a = (x, y));
      x ← y;

function LookaheadUpdate1(x): boolean;
   y ← argmin_{w∈Succ(x)}[c(x,w) + h(w)];
   if h(x) < c(x,y) + h(y) then
      h(x) ← c(x,y) + h(y); return true;
   else return false;
```

Figura 7. Algoritmo LRTA*.

El algoritmo intenta actualizar la estimación heurística del estado actual en cada paso. Los valores de h aprendidos en una ejecución se usan en la ejecución siguiente. Con este mecanismo el algoritmo converge a rutas óptimas.

El algoritmo se muestra en la Figura 7. El procedimiento LRTA* inicializa la estimación heurística de todos los estados usando la función h_0, y resuelve repetidas veces el ejemplar del problema mediante el procedimiento LRTA*-trial. Los valores de h después de la resolución i se toman como valores iniciales para la resolución $i + 1$. El proceso se repite hasta convergencia, es decir hasta que h no cambie. Si h no cambia en una ejecución significa que se encontró una ruta óptima. El procedimiento LRTA*-trial inicializa el estado actual x con el estado inicial s_0. Después, se realiza el siguiente ciclo hasta que se encuentra el estado objetivo. Primero, la función LookaheadUpdate1 es ejecutada ignorando su resultado. Segundo, el estado y que pertenece a los $Succ(x)$ con el valor mínimo de $c(x, z) + h(z)$, $z \in Succ(x)$ es seleccionado como

próximo estado, los empates se resuelven de manera aleatoria. Tercero, se ejecuta la acción de pasar desde el estado x al estado y. Después de esto, y es el nuevo estado actual y el ciclo continua iterando.

La función LookaheadUpdate1 realiza una anticipación de profundidad 1 desde x, después actualiza $h(x)$ siempre y cuando sea menor que el mínimo valor de $c(x, z) + h(z)$, $z \in Succ(x)$. Si $h(x)$ cambia la función retorna verdadero, de lo contrario retorna falso.

LRTA* es completo en un espacio de estados finito, con arcos de costes positivos y estimaciones heurísticas finitas, en donde desde cualquier estado existe una ruta al estado objetivo (Teorema 1, [Korf, 1990]). Adicionalmente, si las estimaciones de h_0 son admisibles, en repetidas ejecuciones sobre un mismo ejemplar de un problema, las estimaciones heurísticas convergen a sus valores exactos a través de todas las rutas óptimas (Teorema 3, [Korf, 1990]).

*3.3.3 Hibrid Learning Real-Time A**

Hibrid Learning Real-Time A* (HLRTA*) [Thorpe, 1994], es un algoritmo que combina características de LRTA* y RTA*. El mecanismo de actualización de RTA* es más agresivo que el de LRTA*: si x es el estado actual, RTA* usa el valor del segundo mínimo, que siempre será mayor o igual que el primer mínimo de $c(x, z) + h(z)$, $z \in Succ(x)$. El mecanismo de actualización de RTA* puede ocasionar que h se vuelva no admisible. La motivación de HLRTA* es mejorar la capacidad de aprendizaje de LRTA*, usando el segundo mínimo en la actualización, pero manteniendo la admisibilidad de la heurística.

HLRTA* usa dos estimaciones heurísticas por cada estado, h_1 y h_2, que corresponden a las estimaciones heurísticas de LRTA* y RTA* respectivamente. Además, si x es el estado actual, mantiene en $d(x)$ la dirección del movimiento desde x, es decir el estado que el agente selecciona como estado siguiente. El punto interesante de este algoritmo es que cuando

el agente pasa por x y luego regresa a x desde $d(x)$ (esto es, cuando el agente vuelve al estado x por el mismo arco que usó para dejarlo), entonces la estimación h_2 es admisible y puede ser usada en lugar de h_1 [Thorpe, 1994]. HLRTA* siempre usa heurísticas admisibles, por tanto HLRTA* converge a rutas óptimas de la misma manera que lo hace LRTA*.

En la Figura 8, se puede observar el algoritmo HLRTA* con una anticipación de profundidad uno. El procedimiento HLRTA*, inicializa las estimaciones heurísticas h_1 y h_2 de todos los estados con h_0. También se inicializa $d(x)$ con *null*. Luego se repite la ejecución de HLRTA*-Trial hasta lograr la convergencia (hasta que h_1 no cambie). Se logra la convergencia cuando se ha encontrado una ruta óptima mediante ejecuciones sucesivas de procedimiento HLRTA*-trial sobre el mismo ejemplar de un problema. HLRTA*-trial inicializa el estado actual x con el estado inicial s_0, y ejecuta el siguiente ciclo hasta encontrar el estado objetivo. Primero, se realiza una anticipación de profundidad uno y se actualizan las estimaciones heurísticas (mediante la llamada a la función HLRTA-LookaheadUpdate1). Segundo, se selecciona el estado y entre los $Succ(x)$ con menor valor $c(x, z) + H(z)$, $z \in Succ(x)$ como próximo estado actual (los empates se solucionan aleatoriamente). Tercero, se ejecuta la acción de pasar desde el estado x al estado y. Cuarto, se guarda en $d(x)$ el estado seleccionado como próximo estado actual. En este punto, y es el nuevo estado actual y se continua con la ejecución del ciclo.

La función H implementa el uso de las dos heurísticas. Si la función se llama con los argumentos (estados) (x, z), retorna en valor máximo entre $h_2(z)$, $h_1(z)$ si $d(z) = x$, o $h_1(z)$ en otro caso.

La función HLRTA-LookaheadUpdate1 realiza la anticipación con profundidad uno desde el estado x. Se calcula el estado y con menor valor $c(x, v) + H(v, x)$, $v \in Succ(x)$ y el estado z con segundo menor valor de la misma expresión. Luego, se actualizan los valores h_2 y h_1 de x si son menores que $c(x,$

z) + $H(z, x)$ y $c(x, y)$ + $H(y, x)$ respectivamente. Si $h_1(x)$ cambia, la función retorna verdadero, de lo contrario retorna falso.

En un espacio de estados finito, con arcos de costes positivos y estimaciones heurísticas finitas, en donde desde cualquier estado existe una ruta al estado objetivo, HLRTA* es completo. Adicionalmente, si las estimaciones de h_0 son admisibles, en repetidas ejecuciones sobre el mismo ejemplar de un problema, las estimaciones heurísticas convergen a sus valores exactos a través de todas las rutas óptimas [Thorpe, 1994].

```
procedure HLRTA*(X, A, c, s0,G)
  for each x ∈ X do h1(x) ← h0(x); h2(x) ← h0(x); d(x) ← null;
  repeat
    HLRTA*-trial(X, A, c, s0,G);
    until h1 do not change;

procedure HLRTA*-trial(X, A, c, s0,G)
  x ← s0;
  while x ∉ G do
    dummy ← HLRTA-LookaheadUpdate1(x);
    y ← argmin_{v∈Succ(x)} [c(x,v) + H(v,x)]; (Break ties randomly)
    execute(a ∈ A such that a = (x, y));
    d(x) ← y;
    x ← y;

function HLRTA-LookaheadUpdate1(x): boolean;
  y ← argmin_{v∈Succ(x)}[c(x,v) + H(v,x)];
  z ← arg2ndmin_{v∈Succ(x)}[c(x,v) + H(v,x)];
  if h2(x) < c(x, z) + H(z,x) then h2(x) ← c(x, z) + H(z,x);
  if h1(x) < c(x, y) + H(y,x) then h1(x) ← c(x, y) + H(y,x); return true;
  else return false;

function H(v, from): real;
if d(v) = from then return max{h1(v),h2(v)}; else return h1(v);
```

Figura 8. Algoritmo HLRTA*.

3.3.4 Rendimiento de los algoritmos.

En la primera ejecución, RTA* exhibe el mejor desempeño seguido por HLRTA* y LRTA*. En convergencia hacia soluciones óptimas LRTA*

exhibe el mejor desempeño que HLRTA*. [Thope, 1994], [Furcy y Koenig, 2000], [Shimbo e Ishida, 2003] y [Hernández y Meseguer, 2005c].

3.3.5 Otras aproximaciones

FALCONS (FAst Learning and CONverging Search) [Furcy y Koenig, 2000], es un algoritmo pensado para dominios dirigidos. El algoritmo introduce dos novedades: primero, el uso la función heurística $g: X \rightarrow (0, \infty)$ que asocia a cada estado x una estimación del coste de la ruta entre el estado inicial y x. Segundo, de manera similar a A*, FALCONS usa la estimación heurística $g(x) + h(x)$ para seleccionar el próximo estado que visitará el agente. En el trabajo [Shimbo e Ishida, 2003], se crítica la inestabilidad en el proceso de convergencia a soluciones óptimas en LRTA*. Experimentalmente se observa que el proceso de convergencia en LRTA* es no monótono: una ejecución puede requerir más pasos que la ejecución anterior, sobre el mismo ejemplar de problema. En este trabajo se presentan dos algoritmos para tratar con este inconveniente: ε-search, un algoritmo permite encontrar soluciones subóptimas o ε-óptimas, y δ-search, un algoritmo que balancea la relación entre exploración y explotación del espacio de búsqueda. Otra aproximación más reciente en este contexto, que incluye anticipación mediante la estrategia Maxmin, es γ-Trap [Bulitko, 2004]. Este algoritmo permite encontrar soluciones subóptimas o γ-óptimas.

Otras aproximaciones incluyen el uso de multiples agentes. En Multi-Agent RTA* (MARTA*) [Knight, 1993], varios agentes RTA* buscan de manera concurrente un objetivo, mejorando la calidad de la solución debido a la exploración intensiva del espacio de estados. MARTA* usa un rudimentario esquema de comunicación y cooperación. Los agentes se comunican y cooperan, compartiendo la tabla de valores heurísticos. De esta forma un agente se beneficia de la experiencia de otros. Por otro lado, cada agente elige sus movimientos independientemente y no los coordina con los demás. Otros trabajos con múltiples agentes son: [Yokoo y Kitamura, 1996], en este trabajo se introduce un mecanismo de cooperación entre agentes, basado en el

mecanismo de selección usado en algoritmos genéticos. En [Kitamura et al. 1996], se introducen las estrategias organizacionales de repulsión y atracción para coordinar a los agentes. En [Cakir y Polat, 2002], se usa un mecanismo de coordinación basado en los movimientos de cada agente. Para explorar el espacio de estados, los agentes eligen visitar estados que otros no han visitado.

La búsqueda en tiempo real con objetivos móviles fue presentada inicialmente en [Ishida y Korf, 1991]. Estudios posteriores son [Ishida, 1992], [Ishida y Korf, 1995] y [Koenig et al. 2007]. Otros trabajos con múltiples agentes y objetivos móviles son [Goldenberg et al. 2003] y [Ramachandra et al. 2003].

Las estrategias de búsqueda en tiempo real que utilizan un parámetro para controlar la anticipación, se expondrán en el capítulo 6.

Capítulo cuatro

4 PROPAGACIÓN ACOTADA CON ITERACIÓN DE VALORES

A medida que la búsqueda progresa, se actualizan los valores heurísticos de los estados que el agente visita. Es posible mejorar la estimación heurística de un estado mediante la actualización de su valor a partir de la información obtenida en la anticipación, tal como lo hace LRTA*. Esta es una estrategia general de aprendizaje en los algoritmos de búsqueda heurística en tiempo real (ver Capítulo 3). En la mayoría de los algoritmos, la actualización se limita al estado actual.

En este capítulo, introducimos la idea propagación acotada con iteración de valores. Este mecanismo permite realizar varias actualizaciones por episodio de planificación. La propagación acotada con iteración de valores es una variación del método de iteración de valores en programación dinámica [Pemberton y Korf, 1992]. Algunos mecanismos de actualización de heurísticas similares, son utilizados en [Barto et al. 1995], [Bonnet y Geffner, 2006] y [Rayner et al. 2007]. Después de explicar la contribución principal de este capítulo, se expone el concepto de soporte de una estimación heurística. Los soportes mejoran la eficiencia de la propagación. Luego, se presenta el algoritmo LRTA*(k). El algoritmo es una combinación del mecanismo de propagación acotada con LRTA*. Finalmente, se presentan resultados experimentales en los distintos escenarios de pruebas, en donde se podrá apreciar las ventajas de esta combinación.

4.1 Propagación acotada

Nuestra contribución se basa en la siguiente idea. Supongamos que después de la anticipación, el estado actual x cambia el valor de su estimación heurística (se actualiza), para que se satisfaga la igualdad $h(x) = \min_{y \in Sua(x)}[c(x, y) + h(y)]$ [Korf, 1990]. Este cambio puede generar otros cambios en la estimación heurística de los estados sucesores de x, por lo que dichos estados se deben revisar. Supongamos que y es uno de esos sucesores, y que $h(y)$

cambia en el proceso de revisión. Como efecto de ese cambio, los estados sucesores de *y* también pueden cambiar el valor de su estimación heurística, por tanto también han de revisarse. Si hay otros cambios, estos a su vez pueden generar otros cambios en sus sucesores, y así sucesivamente. Este proceso es la propagación de los cambios en las estimaciones heurísticas a partir del cambio o actualización de la heurística del estado actual. El proceso itera hasta que no haya más cambios, o termina cuando haya alcanzado un límite de recursos (tiempo, número de actualizaciones, etc.). En nuestra propuesta, la propagación se realiza hasta reconsiderar *k* estados. Llamamos a esta idea *propagación acotada*, porque el cambio en la heurística del estado actual, es propagado hasta un límite de *k* estados o hasta que no se puedan realizar más cambios, en cada episodio de planificación del agente [Hernández y Meseguer, 2005a].

4.1.1 Propagación no acotada

Quizá la forma más sencilla de presentar la propagación de cambios en valores heurísticos sea explicando primero la *propagación no acotada*. En la propagación no acotada, el proceso de propagación se realiza hasta que no haya cambios en la heurística, sin límite en el número de estados a actualizar, y el tiempo o recursos a usar. La propagación no acotada, no satisface la condición de realizar acciones en un tiempo limitado, una propiedad fundamental en búsqueda en tiempo real. Un algoritmo que implementa la idea de propagación no acotada es LCM [Pemberton y Korf, 1992]. Nuestra idea de propagación acotada (que surge de manera independiente) es similar a la idea de LCM, limitando el número de estados a reconsiderar.

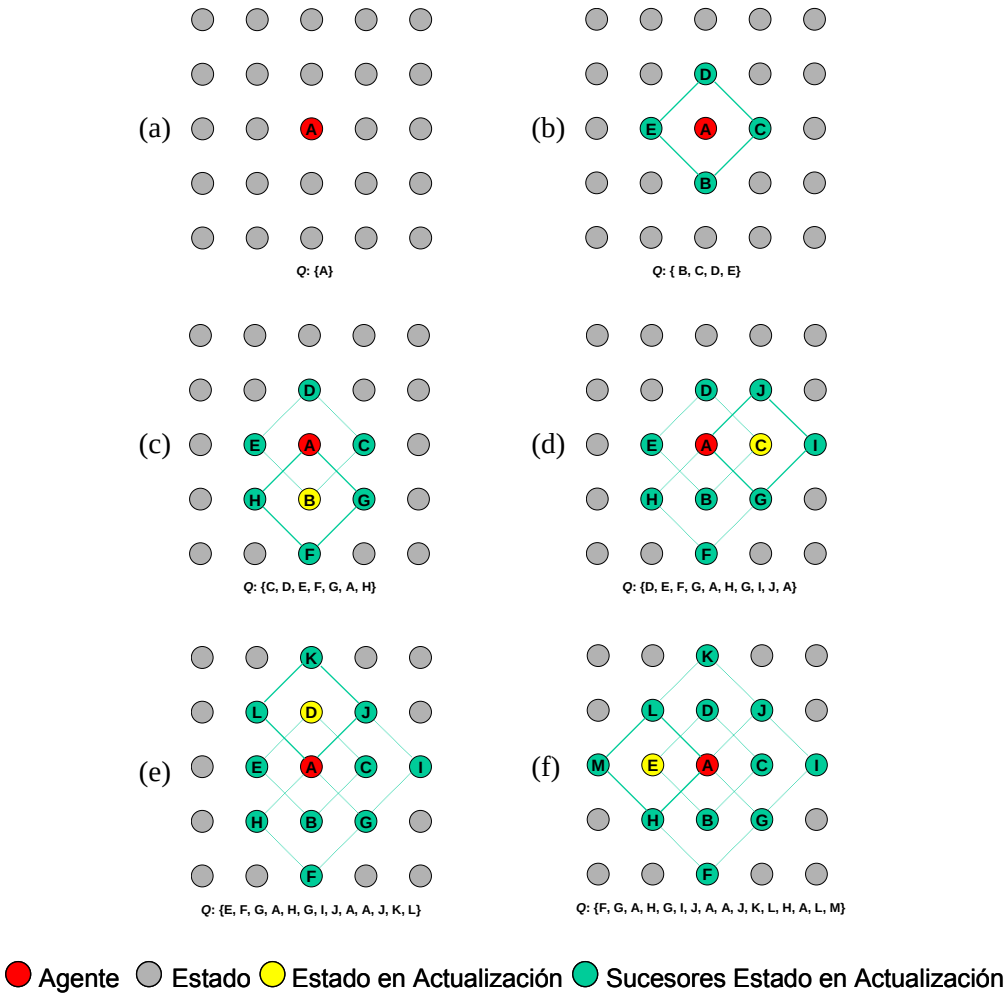


Figura 9. Funcionamiento de la propagación no acotada.

Explicaremos la propagación no acotada con el ejemplo de la Figura 9. La parte (a) de la figura muestra el espacio de estados y la *cola de propagación* Q, que mantiene una secuencia de estados candidatos a ser actualizados. Si un estado en Q cambia, se insertan sus sucesores al final de la cola. El orden en que se insertan los estados es aleatorio, en el ejemplo usaremos sur-este-norte-oeste. Los elementos de Q, se revisan en orden FIFO. El primer elemento que se inserta en Q es el estado actual. En (a) se inserta el estado actual en Q. En (b) se extrae A, el primer elemento de Q y se revisa $h(A)$. Supongamos que después de la anticipación $h(A)$ cambia, si esto sucede se agregan sus sucesores al final de la cola Q como se ve en la figura. En (c) se extrae B, el

primer elemento de Q y se revisa $h(B)$. $h(B)$ cambia, por tanto insertamos sus sucesores en Q. En (d) se extrae C, el primer elemento de Q y se revisa $h(C)$. $h(C)$ cambia, por tanto se insertan sus sucesores en la cola. En (e) se extrae D, el primer elemento de Q y se revisa $h(D)$. Como $h(D)$ cambia insertamos sus sucesores al final de la cola. En (f) se extrae E y se revisa $h(E)$, como $h(E)$ cambia se insertan sus sucesores al final de la cola de propagación. La propagación continuaría revisando el siguiente elemento en Q, y después el siguiente, etc. El cambio en la estimación heurística del estado actual inicia el proceso de revisión de sus sucesores, si uno de sus sucesores cambia su heurística, se revisan los sucesores de este sucesor y así sucesivamente hasta que no haya más cambios o se haya revisado todos los estados en Q.

4.1.2 Propagación acotada

La propagación acotada consiste en limitar el número elementos que entran en la cola de propagación con un parámetro k. Si fijamos $k = 9$, se puede agregar un máximo de 9 estados en Q (Figura 10). En el ejemplo de la Figura 9, después de agregar los sucesores de B en (c), no se podría agregar nada más. A partir de esta situación, sólo se verifica si las estimaciones heurística de los elementos en Q que faltan por revisar cambian o no, sin agregar estados a la cola.

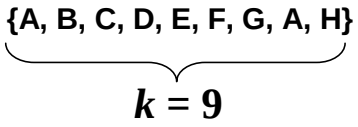


Figura 10. Los 9 estados que entran a la cola de propagación.

En nuestra propuesta se usa anticipación de profundidad uno, por tanto el espacio local de búsqueda lo constituye el estado actual y sus sucesores inmediatos. El espacio local de aprendizaje esta conformado por todos los estados que puede cambiar su heurística en la actualización, es decir por todos los estados que entran en Q.

4.1.3 Versiones de propagación acotada

La propagación se puede hacer sobre cualquier estado del espacio de estados o sobre estados que han sido previamente visitados. Por tanto, tenemos dos versiones de propagación:

1. Propagación sobre cualquier estado del espacio.

2. Propagación sobre estados visitados.

Veremos con un ejemplo las diferencias entre las dos versiones de propagación. Primero veremos la propagación sobre cualquier estado del espacio y después la propagación sobre estados visitados.

En la Figura 11 (A), podemos ver una cuadrícula de 3 x 3 en donde *g* es el estado inicial, *i* el estado objetivo. La estimación heurística inicial es la distancia "Manhattan". Los movimientos permitidos son norte, sur, este y oeste (B), todas las acciones cuestan 1. En (C) se representa la situación inicial, el agente esta en *g* y *Q* se inicializa con el estado actual. En el ejemplo, sólo se usará la porción del espacio de estados donde tienen lugar las actualizaciones. En (D), podemos observar como trabaja la propagación sobre cualquier estado. En (i) se revisa *g*, *h*(*g*) cambia, por lo tanto agregamos *d* a *Q*. En (ii) se revisa *d*, *h*(*d*) cambia, agregamos los sucesores *g* y *a* a *Q*. En (iii) se revisa *g*, *h*(*g*) cambia, nuevamente agregamos *d* a *Q*. En (iv) se revisa *a*, *h*(*a*) no cambia, por tanto no se agrega nada a la cola. En (v) se revisa *d*, *h*(*d*) no cambia, no se agrega nada. Como *d* es el último elemento en *Q*, se termina el proceso de propagación, después el agente se mueve a *d* y se inicializa *Q* con este estado. En (E) vemos como trabaja la propagación sobre estados visitados. En (i) se revisa *g*, *h*(*g*) cambia, *d* es un sucesor de *g*, como no ha sido visitado, no se agrega. En (ii) el agente se mueve y revisa *d*, *h*(*d*) cambia, por tanto se agrega *g* a *Q*, ya que es sucesor de *d* y ha sido visitado, *a* no se agrega. En (iii) se revisa *g*, *h*(*g*) cambia, como *d* es sucesor y ha sido visitado se agrega a *Q*. En (iv) se revisa *d*, *h*(*d*) no cambia, no se agrega nada. Después el agente se mueve a *a* y se inicializa *Q* con el estado actual.

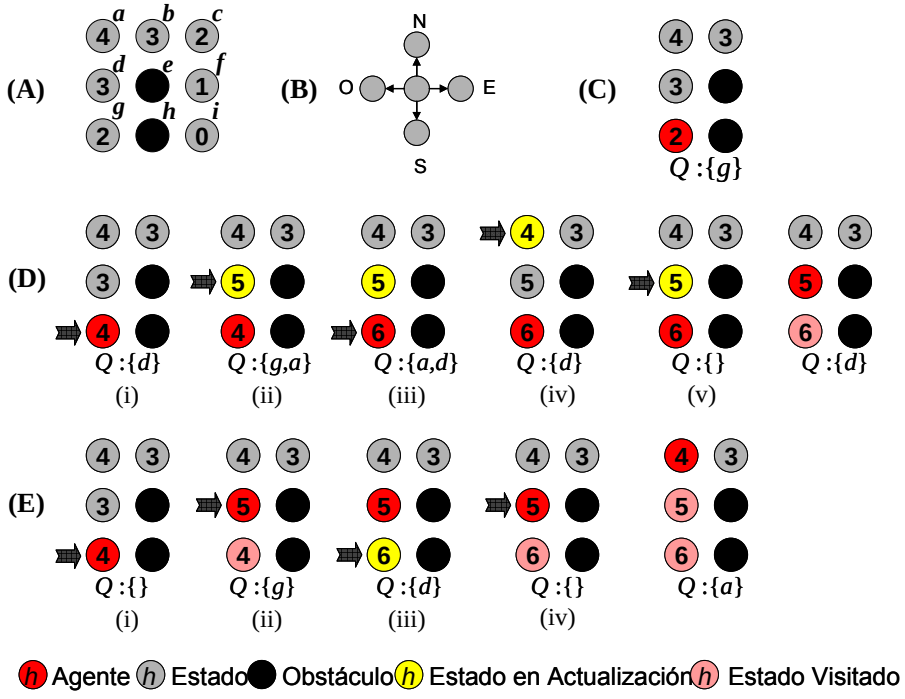


Figura 11. Ejemplo de propagación sobre estados visitados y sobre cualquier estado del espacio de estados. En (A), (B) y (C) se aprecian las condiciones iniciales. En (D) la propagación sobre cualquier estado del espacio de estados. En (E) la propagación sobre estados visitados.

Con la propagación sobre cualquier estado del espacio (D), el agente se mueve al estado *d* después de censar 5 estados y realizar 3 actualizaciones. Con la propagación sobre estados visitados (E), se hace un manejo más eficiente de *Q* (ya que se insertan menos estados) y el agente realiza menos trabajo antes de cada acción. Este se mueve inmediatamente a *d* después de actualizar *g*, luego censa 3 estados y realiza 2 actualizaciones antes de moverse a *a*.

En [Hernández and Meseguer, 2005b] se compara experimentalmente las versiones de propagación. Los resultados muestran que dependiendo del problema y la calidad de la heurística inicial, un tipo de propagación puede obtener mejores resultados que la otra. En la sección de resultados experimentales se discutirá sobre las ventajas de usar una u otra versión de propagación.

Las dos versiones de propagación acotada mejoran la calidad de la información heurística manteniendo la admisibilidad. En consecuencia, el agente encuentra mejores estimaciones heurísticas en los estados a medida que la búsqueda progresa, lo que ayuda a solucionar los problemas más rápido. Sin embargo, se requiere de fases de planificación más largas, ya que actualizar hasta k estados es computacionalmente más caro que actualizar un estado (como en el caso de LRTA*). A pesar de esto, los beneficios que se obtienen con el uso de propagación acotada son importantes y el tiempo extra requerido para planificar es moderado.

La calidad de los resultados dependen del problema y del parámetro k. A mayor k se obtienen mayores beneficios con el coste de un mayor tiempo de planificación por paso. Algo importante de mencionar es que el efecto de la propagación acotada esta restringido a un número máximo de actualizaciones que se pueden realizar por paso. Este número máximo depende de la calidad de la heurística. Si la heurística es mala y se usa por ejemplo un $k = 3$, es muy probable que en cada paso, el agente realice 3 actualizaciones. Si la heurística es buena, puede que se hagan menos de 3 actualizaciones. En el caso extremo en que la heurística corresponda a los valores exactos, k no tiene ningún efecto, ya que el número máximo de actualizaciones que se pueden realizar es cero.

4.2 Soportes

La propagación acotada de los cambios heurísticos puede ser fácilmente mejorada, usando la noción de *soporte* de la estimación heurística de un estado. Supongamos que x es el estado actual y que $h(x)$ es igual a $c(x, w) + h(w)$, en donde $w = \text{argmin}_{v \in Suc(x)}[c(x, v) + h(v)]$. Si en algún paso posterior $h(w)$ cambia, entonces $h(x)$ puede cambiar, por tanto la heurística de x debe ser revisada. Por otro lado, si w no es el estado con mínima estimación heurística entre los sucesores, el cambio en $h(w)$ no afectará a la heurística de x. Todos los estados sucesores un estado x con valor heurístico igual a $\min_{v \in Suc(x)}[c(x, v) + h(v)]$ se denominan soportes de $h(x)$. Los soportes se pueden usar para determinar

cuando el cambio en $h(w)$, donde $w \in Succ(x)$, produce un cambio en $h(x)$. $h(x)$ cambiará sí sólo si existe un único sucesor w con mínima estimación heurística y ese sucesor cambia su estimación heurística. En la Figura 12, podemos ver un ejemplo. Las acciones cuestan uno y los movimientos que puede hacer el agente son norte-sur-este-oeste. En el ejemplo los 2 estados con $h = 5$, soportan el valor heurístico del estado con $h = 6$ y este soporta el valor heurístico de los estados con $h = 7$. Supongamos que uno de los estados con $h = 5$ aumenta su estimación heurística. En esta situación, el estado con $h = 6$ no cambiará su valor porque existe otro estado con $h = 5$ que lo soporta. Ahora, si el otro estado con $h = 5$, el único soporte del estado con $h = 6$, también cambia, entonces el estado con $h = 6$ cambiará su estimación. Debido a este último cambió, la heurística de los estado con $h = 7$ también cambiarán, porque el estado con $h = 6$ cambió y este era su único soporte. Formalmente, un estado z es soporte de $h(x)$, lo que se escribe $z = supp(x)$, si y sólo si $z = \text{argmin}_{v \in Succ(x)}[c(x, v) + h(v)]$. $SUPP(x)$ es el conjunto de estados soportes de $h(x)$.

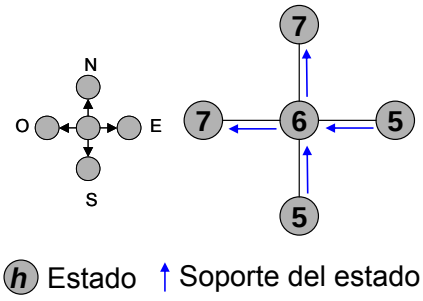


Figura 12. Ejemplo de soportes.

Los soportes se usan para evitar chequear estados que no cambiarán en la propagación. Explicaremos la asignación y uso de los soportes con un ejemplo. Supongamos que las condiciones y la situación inicial en la Figura 13 son las mismas que en la Figura 11 y que la propagación sólo se hace sobre estados visitados.

Asignación: En (i), se revisa g y $h(g)$ cambia. Es fácil ver que después de actualizar $h(g)$, $h(g)$ se justifica por la estimación heurística del estado d, ya que $d = \text{argmin}_{v \in Succ(g)}[c(g, v) + h(g)]$. Por definición d pasa a ser el único soporte de g, es decir, $d = supp(g)$ y $SUPP(x) = \{d\}$.

Uso: Los soportes se implementan agregando la siguiente condición en la inserción de estados en la cola de propagación: Si un estado $v \in Q$ cambia, se agregará un sucesor w de v a Q, sólo si v es su único soporte (esto es si $v = supp(w)$ y $|SUPP(w)| = 1$). En la Figura 13 (ii) se revisa d, $h(d)$ cambia y se determina que a y g son soportes de $h(d)$ ($Supp(d) = \{a, g\}$). Debido a que d es el único soporte de g ($SUPP(g) = \{d\}$), y que g ha sido visitado, entonces se puede insertar el estado g en Q. Podemos estar seguros que $h(g)$ cambiará al propagar el cambio de $h(d)$, ya que su estimación heurística sólo se soporta por el estado d ($d = supp(g)$). En (iii) se revisa g, $h(g)$ cambia y se determina nuevamente que d es el único soporte de $h(g)$ ($d = supp(g)$, $SUPP(g) = \{d\}$). El estado d es sucesor de g, ha sido visitado y es soportado por g. Como g no es el único soporte de $h(d)$, d no se agrega en la cola, ya que podemos estar seguros que el cambio en $h(g)$ no va afectar a $h(d)$. Antes de continuar, se actualiza el conjunto de soportes de d, $Supp(d) = \{a, g\} - \{g\} = \{a\}$.

Si comparamos la propagación en la Figura 11 con la de la Figura 13, vemos que mediante el uso de soportes el agente realiza menos trabajo antes de ejecutar cada movimiento, este se mueve inmediatamente a d después de actualizar g. Luego censa y actualiza 2 estados y se mueve hacia el estado a. Además, se hace un manejo más eficiente de la cola de propagación, ya que se insertan menos estados.

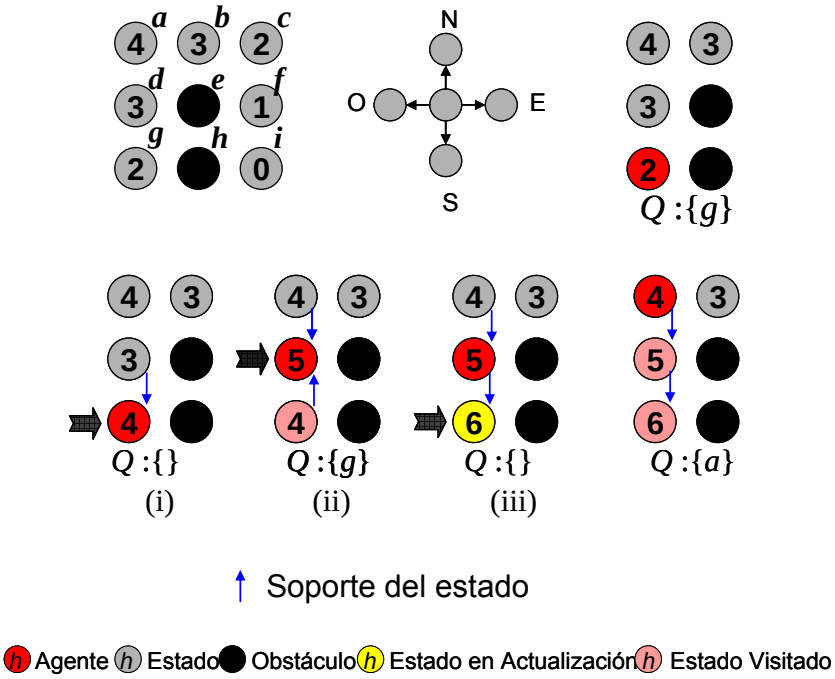


Figura 13. Ejemplo de propagación limitada a los estados visitados y que cumplen la condición del soporte.

LRTA*(*k*) es un algoritmo basado en LRTA*, en el que se implementa la idea de propagación acotada con iteración de valores. El algoritmo se explica en la siguiente sección.

4.3 LRTA*(*k*)

LRTA*(*k*) es un algoritmo que combina la propagación acotada con LRTA*, de hecho LRTA* es un caso particular de LRTA*(*k*) con $k = 1$.Tal como LRTA*, LRTA*(*k*) guarda la información heurística entre ejecuciones. Además, almacena los soportes en una tabla. La tabla de soportes se inicializa antes de la primera ejecución. Los soportes también se guardan entre ejecuciones ya que los asignados en la ejecución *i*, continúan siendo válidos en la ejecución $i + 1$.

```
procedure LRTA*(k) (X, A, c, s0,G, k)
1 for each x ∈ X do h(x) ← h0(x); SUPP(x) ← ∅;
2 repeat
3   LRTA*(k)-Trial(X, A, c, s0, G, k);
4 until h does not change;

procedure LRTA*(k)-Trial(X, A, c, s0,G, k)
1 x ← s0;
2 while x ∉ G do
3   LookaheadUpdateK(x, k);
4   y ← argmin_{w∈Succ(x)} [c(x,w) + h(w)];(Break ties randomly)
5   x ← y;

procedure LookaheadUpdateK(x, k)
1 Q ← ⟨x⟩; cont ← k – 1;
2 while Q ≠ ∅ do
3   v ← extract-first(Q);
4   if LookaheadUpdate1(v) then
5     for each w ∈ Succ(v) do
6       if v ∈ SUPP(w) then
7         if cont > 0 ∧ |SUPP(w)| = 1 then
8           Q ← add-last(Q,w);
9           cont ← cont – 1;
10        SUPP(w) ← SUPP(w) – {v};

function LookaheadUpdate1(x): boolean;
1 y ← argmin_{w∈Succ(x)}[c(x,w) + h(w)];
2 for each w ∈ Succ(x) do
3   if c(x,w) + h(w) = c(x,y) + h(y) then
4     SUPP(x) ← SUPP(x) ∪ {w};
5 if h(x) < c(x,y) + h(y) then
6   h(x) ← c(x,y) + h(y); return true;
7 else return false;
```

Figura 14. Algoritmo LRTA*(*k*).

El algoritmo LRTA*(*k*) aparece en la Figura 14. Lo explicaremos comparando con el algoritmo LRTA* (Figura 7). El procedimiento LRTA*(k) inicializa la heurística de cada estado del espacio de estados, su conjunto de soportes con *vacío* (línea 1) y ejecuta el procedimiento LRTA*(k)-Trial hasta que se produzca la convergencia (cuando *h* no cambia) (líneas 2, 3 y 4). El procedimiento LRTA*(k)-Trial difiere del procedimento LRTA*-Trial en que ejecuta el procedimiento LookaheadUpdateK (que realiza la propagación acotada) en vez del procedmiento LookaheadUpdate1 (línea

3). El procedimento LookaheadUpdateK realiza la propagación acotada usando la cola de propagación Q (la secuencia de estados candidatos a ser actualizados), de la siguiente manera: Q se inicializa con el estado actual. Entran como máximo k estados en Q, esto es controlado por el contador *cont*, que se inicializa a $k - 1$ (línea 1). Luego, se realiza el siguiente ciclo (línea 2) hasta que Q no contenga estados. Se extrae el primer estado v de Q (línea 3). Con este estado se realiza la anticipación y actualización si corresponde. Si $h(v)$ cambia (si LookaheadUpdate1 retorna verdadero), el cambio se propaga sobre sus sucesores así (línea 5): un sucesor w de v entra en la última posición de Q si,

i. v es soporte de w (línea 6),

ii. hay espacio en Q (línea 7), y

iii. el número de soportes de w es 1(línea 7).

Cuando se satisfacen estas condiciones, nos aseguramos que los estados que entran en Q cambiarán su estimación heurística. Luego, se agrega w al final de Q (línea 8) y se reduce el contador en uno (línea 9). En este punto, si v es soporte de w, se quita del conjunto de soportes de w, porque $h(v)$ cambió y por tanto no cumple la condición de soporte para w (línea 10). Si $h(v)$ no cambia, el ciclo itera para procesar el siguiente estado en Q. La función LookaheadUpdate1 anticipa desde x con profundidad 1 (línea 1), determina y asigna los soportes de x (línea 2, 3 y 4), y actualiza $h(x)$ siempre y cuando x cumpla la *condición de actualización*, es decir cuando $h(x)$ es menor que $\min_{v \in Succ(x)}[c(x, v) + h(v)]$. Si $h(x)$ cambia, la función retorna verdadero, de lo contrario retorna falso (línea 5, 6 y 7). Es importante notar que sólo pueden entrar en Q, el estado actual y aquellos que tengan soporte, y que sólo los estados visitados tienen soportes. Dado que los soportes son asignados cuando se verifica si la heurística de un estado cambia (con la función LookaheadUpdate1) y que para verificar la heurística de un estado este

debe ser visitado con anterioridad (debe tener soporte o ser el estado actual), la condición de propagar sólo sobre estados visitados se satisface con la condición de propagar sólo hacia estados con soporte. Cuando se desee propagar sobre cualquier estado, se debe relajar las condiciones de inserción de estados en Q: un estado entrará en Q si cumplen las condiciones i, ii y iii descritas en el procedimiento LookaheadUpdateK o si $h(v) > h(w)$. Con esta modificación, no se puede asegurar que los estados que entran en Q cambiarán su estimación heurística.

Después de la propagación, LRTA*(k), al igual que LRTA*, selecciona el estado con menor $c(x, w) + h(w)$ como próximo estado. Los empates se resuelven de manera aleatoria. Un efecto importante de la propagación acotada es que el estado que selecciona LRTA*(k) como próximo estado a visitar, puede ser distinto al que selecciona LRTA*. Por ejemplo, en la parte superior de la Figura 15, vemos un espacio de estados, las estimaciones heurísticas y el estado actual del agente. Los movimientos permitidos son norte, sur, este y oeste. Todas las acciones cuestan 1. Veamos el funcionamiento de LRTA* y LRTA*(k). En la planificación LRTA* realiza una actualización, la del estado actual. LRTA*($k = 2$) hace dos actualizaciones, la del estado actual y la del estado al este (como resultado de la propagación). Con LRTA* el agente se mueve al este, en cambio con LRTA*(k) el agente se mueve al sur. En general LRTA*(k) permite seleccionar mejores acciones que LRTA*. Esto sucede porque con LRTA*(k) el agente tiene mejor información heurística que con LRTA* en el momento de seleccionar la acción.

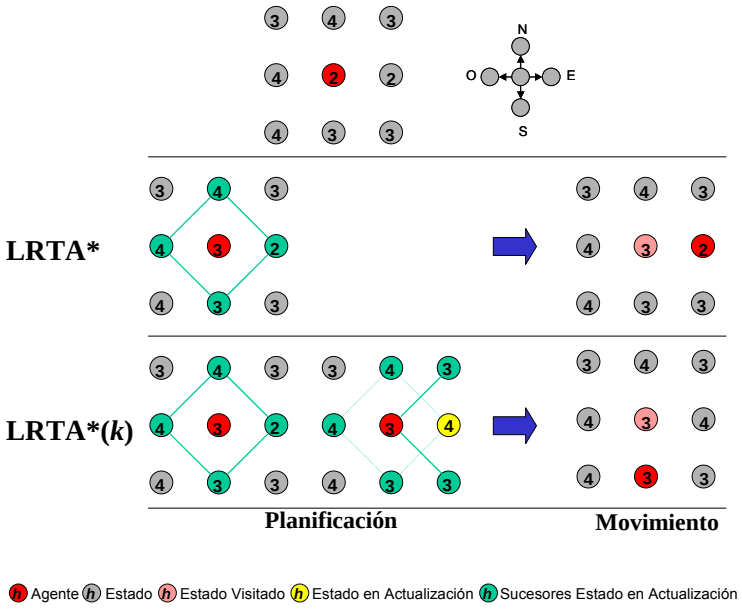


Figura 15. En la misma situación inicial de un paso en concreto, el próximo estado en LRTA* y LRTA*(*k*) pueden ser distintos. En el ejemplo LRTA* selecciona el estado al este y LRTA*(*k*) el estado al sur.

4.3.1 Prueba formal

Para probar que LRTA*(*k*) es un algoritmo que converge a soluciones óptimas basta con probar que los valores heurísticos se mantienen admisibles después de la propagación acotada. Esto se garantiza con el siguiente lema.

Lema 1 (Lema 2 de [Edelkamp and Eckele, 1997]). Sea $x \in X - G$ y h admisible. Actualizar $h(x)$ con $\max[h(x), \min_{v \in Suc(x)}(c(x, v) + h(v))]$ implica que $h(x) \leq h^*(x)$.

Dem. Si $h(x) \geq \min_{v \in Suc(x)}(c(x, v) + h(v))$ no hay nada que probar. Sino, existe una ruta óptima desde x al estado objetivo que pasa por un sucesor w. Entonces $h^*(x) = c(x, w) + h^*(w) \geq c(x, w) + h(w)$. En particular, esto es verdad para el mínimo $c(x, v) + h(v)$ entre los sucesores de x, esto es, $h^*(x) \geq \min_{v \in Suc(x)}(c(x, v) + h(v))$. Por tanto, $h(x) \leq h^*(x)$.

A partir de este resultado se garantiza la convergencia a rutas óptimas de LRTA*(*k*) en los mismos términos que en LRTA*, porque el Teorema 3 de

[Korf, 1990] es también valido para LRTA*(k). LRTA*(k) hereda las buenas propiedades de LRTA*.

4.4 Resultados experimentales

En esta sección se evalúa LRTA*(k) para varios valores de k, en Cua-35, Lab-acic y Lab-cic. Se presentan resultados para: la primera ejecución, convergencia a soluciones óptimas y estabilidad del proceso de convergencia. En cada uno de los escenarios de experimentación, se analiza la versión con propagación sobre estados visitados (versión 1) y sobre cualquier estado (versión 2). Los resultados para los distintos k se presentan en términos de porcentaje respecto a los resultados obtenidos con LRTA* (LRTA*($k = 1$)). El tiempo total de búsqueda (T) está el milisegundos, el coste (C) en cantidad de acciones, la memoria (M) en cantidad de estados que han aumentado su estimación heurística y el tiempo por paso en milisegundos. Los resultados para convergencia incluyen el número de ejecuciones para converger a una solución óptima (E), y el tiempo total de convergencia a una solución óptima en lugar del tiempo total de búsqueda (T). La fila 100% de las tablas que se presentan en esta sección contienen los resultados para LRTA*.

4.4.1 Primera ejecución

La Tabla 1 contiene resultados para la primera ejecución en Cua-35, Lab-acic y Lab-cic. Vemos cada una de las medidas de desempeño:

- Tiempo total de búsqueda: podemos observar que es menor al obtenido con LRTA* en todos los valores de k (excepto $k = \infty$). El tiempo primero disminuye y luego aumenta a medida que k crece en la versión 1 y 2. Esto sucede porque con valores pequeños de k, el número de acciones disminuye drásticamente (por ejemplo, de un 100% a un 20% en Lab-acic versión 1 con $k = 5$). Es decir, con un poco más de esfuerzo invertido en la actualización de heurísticas, se obtienen disminuciones importantes en coste, lo que se refleja en la disminución del tiempo total de búsqueda. Al seguir aumentando el

valor de k, se realiza un esfuerzo mayor en actualizar heurísticas, con una disminución pequeña en el número de acciones. Por tanto, a medida que el esfuerzo en actualizar es mayor que el beneficio en coste, aumenta el tiempo de búsqueda. En cualquier caso, el tiempo de búsqueda para los valores de k probados, mayores que 1 y menores que infinito, es siempre menor que el de LRTA*(k = 1). También podemos observar que el tiempo total obtenido con la versión 1 es siempre menor o igual al obtenido con la versión 2 (excepto con k > 20 en Lab-cic). Este resultado esta relacionado con el esfuerzo que realiza una versión u otra. El esfuerzo lo podemos calcular multiplicando las actualizaciones realizadas (*ar*) por el coste de realizar una actualización sumado a las actualizaciones que se intentan pero no se hacen (*an*) por el coste de intentar una actualización (notar que *ar* + *an* es el número de llamadas a la función LookaheadUpdate1). En la Figura 16 (se omite $k = \infty$ por razones de escala) podemos ver que en Cua-35 el número de actualizaciones realizadas y no realizadas es mayor en la versión 2, lo cual justifica que el tiempo en la versión 2 sea mayor que en la versión 1. En Lab-acic el número de actualizaciones realizadas y no realizadas es mayor en la versión 2, salvo para k = 160 en donde la versión 1 tiene un valor levemente mayor de actualizaciones realizadas, pero menor en actualizaciones no realizadas. Esto justifica el tiempo total, mayor para versión 2 con $1 < k < 160$, e igual entre versiones con k = 160. En Lab-cic la versión 2 realiza más esfuerzo con $k \leq 20$ y la versión 1 con valores de k mayores que 20. Esto explica que el tiempo total de búsqueda primero sea mayor en la versión 2 y luego en la versión 1.

- Coste: en ambas versiones se observa una mejora importante respecto a LRTA*. El coste disminuye a medida que k crece. Esto nos indica que invertir tiempo en realizar una mayor cantidad de actualizaciones por episodio de planificación, tiene un efecto positivo en la calidad de

la solución. Podemos observar que para un mismo k, dependiendo del problema, una versión es mejor que otra. En Cua-35 se obtiene un menor coste con la versión 1 con k pequeño ($k < 40$), cuando el k es grande la versión 2 es mejor. Algo similar sucede en Lab-cic, con $k <$ 80 es mejor la versión 1 y con $k < 80$ la versión 2. En Lab-acic, la versión 1 obtiene menor coste (excepto con $k = \infty$). La diferencia entre las versiones se puede analizar considerando la manera en que el agente recorre el espacio de estados. Esto se puede ver contando los estados que realmente visita el agente en una ejecución y cuantas veces se vuelven a visitar[1]. Si el agente pasa n veces por un estado en una ejecución, diremos que el estado fue visitado una vez y revisitado n - 1 veces. La suma entre la cantidad de estados visitados y la cantidad de veces en que los estados se visitan nuevamente en una ejecución corresponde al coste de la solución. En la Figura 17 podemos ver la cantidad de estados visitados, la cantidad de revisitas a los estados y el coste (que es la suma entre la cantidad de visitados y las revisitas) en Cua-35, Lab-acic y Lab-cic. Podemos apreciar que la cantidad de estados visitados varía poco y que la cantidad de veces que los estados se revisitan disminuye considerablemente cuando k crece en las dos versiones. A medida que k aumenta su valor, la diferencia entre los estados visitados por versión 1 y 2 no varía mucho. En cambio la diferencia entre la cantidad de estados revisitados es grande con k pequeño, para luego disminuir a medida que k aumenta hasta que ya no hay diferencia. Esto justifica que la diferencia en coste entre las versiones disminuya a medida que k crece. En Cua-35 la suma entre la cantidad de estados visitados y la cantidad de revisitas en la versión 1 es menor que en la versión 2 con $k \leq 20$ y menor con la versión 2 para los otros valores de k. Esto explica la diferencia en coste entre las versiones en Cua-35. En los

[1] Aquí estamos contando los estados visitados en la trayectoria del agente. No confundir con los estados visitados durante el proceso de búsqueda.

otros escenarios la diferencia en coste entre las versiones también se explica por la diferencia entre la suma de los estados visitados con la cantidad de revisitas.

	Cua-35 versión 1				Cua-35 versión 2			
k	*T* %	*C* %	*M* %	*T/P* %	*T* %	*C* %	*M* %	*T/P* %
100%	4,1	4127	498	0,0010	4,1	4127	498	0,0010
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%
5	73%	43%	105%	170%	81%	46%	106%	174%
10	71%	35%	111%	204%	79%	36%	117%	219%
20	72%	30%	114%	239%	82%	30%	134%	274%
40	75%	28%	116%	270%	85%	25%	152%	343%
80	79%	27%	118%	298%	95%	23%	176%	421%
160	83%	26%	117%	319%	106%	21%	202%	501%
∞	98%	26%	120%	377%	240%	20%	646%	1197%
	Lab-acic versión 1				Lab-acic versión 2			
100%	344,5	397433	5417	0,0009	344,5	397433	5417	0,0009
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%
5	39%	22%	94%	181%	73%	41%	105%	179%
10	38%	16%	100%	241%	64%	27%	110%	235%
20	38%	11%	102%	336%	61%	19%	120%	317%
40	45%	10%	103%	473%	56%	13%	123%	439%
80	52%	8%	103%	652%	55%	9%	122%	607%
160	61%	7,3%	104%	840%	61%	7,4%	122%	819%
∞	434%	2,4%	104%	18344%	453%	2,3%	132%	19352%
	Lab-cic versión 1				Lab-cic versión 2			
100%	102,6	116832	2761	0,0009	102,6	116832	2761	0,0009
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%
5	52%	27%	117%	191%	72%	41%	108%	179%
10	52%	20%	130%	257%	65%	28%	121%	234%
20	56%	16%	137%	360%	59%	19%	133%	317%
40	65%	13%	138%	501%	60%	14%	148%	438%
80	78%	11,7%	144%	666%	69%	11,5%	157%	595%
160	88%	10,5%	146%	844%	79%	10,0%	164%	794%
∞	283%	5,6%	148%	5034%	285%	5,1%	190%	5551%

Tabla 1. Resultados en la primera ejecución en Cua-35, Lab-acic y Lab-cic para la versión 1 y 2 de LRTA*(*k*).

- Memoria: el consumo es mayor a medida que *k* aumenta porque se actualiza una mayor cantidad de estados (sólo Lab-acic con *k* = 5 y la versión 1 el consumo de memoria disminuye levemente). Se puede observar que la versión 2 tiene un mayor consumo para todos los valores de *k* probados (excepto con *k* = 5, 10 y 20 en Lab-cic). Esta diferencia es notoria con valores grandes de *k*, con valores pequeños el consumo es similar. Esto se debe a que la versión 1 actualiza sólo los estados visitados, y las actualizaciones en la versión 2 se pueden propagar a sobre cualquier estado.

- Tiempo por paso: podemos ver que aumenta a medida que *k* crece. Esto sucede porque se intenta hacer *k* actualizaciones en vez de una,

en cada episodio de planificación. El esfuerzo en cada episodio de planificación lo podemos calcular multiplicando las actualizaciones realizadas por el coste de realizar una actualización sumando a las actualizaciones que se intentan pero no se hacen por el coste de intentar una actualización. El tiempo por paso es mayor en la versión que realiza un mayor esfuerzo por episodio de planificación para cierto valor de k. En la Figura 18 vemos las gráficas de las actualizaciones realizadas y no realizadas por paso en Cua-35, Lab-acic y Lab-cic para distintos k (no presentamos $k = \infty$ por razones de escala) con ambas versiones. Estas gráficas nos permiten apreciar el esfuerzo por paso, que se corresponde con el tiempo por paso en cada escenario y versión. En Cua-35 la cantidad de actualizaciones realizadas y no realizadas por paso son mayores en la versión 2, lo cual justifica que el tiempo por paso en la versión 2 sea mayor que en la versión 1. En Lab-acic y Lab-cic vemos que las actualizaciones no realizadas son similares en ambas versiones, y que en la versión 1 las actualizaciones realizadas superan en número a las realizadas por la versión 2 en los dos escenarios. Esto explica por que el tiempo por paso con la versión 1 es mayor que con la versión 2 en Lab-acic y Lab-cic. Las actualizaciones realizadas tienen una mayor influencia en el esfuerzo por paso, lo que justifica que con $k = \infty$, el tiempo por paso siempre es mayor en la versión 2 que en la versión 1. La versión 2 puede actualizar hacia cualquier estado y por tanto realiza un mayor número de actualizaciones por episodio de planificación en los tres escenarios.

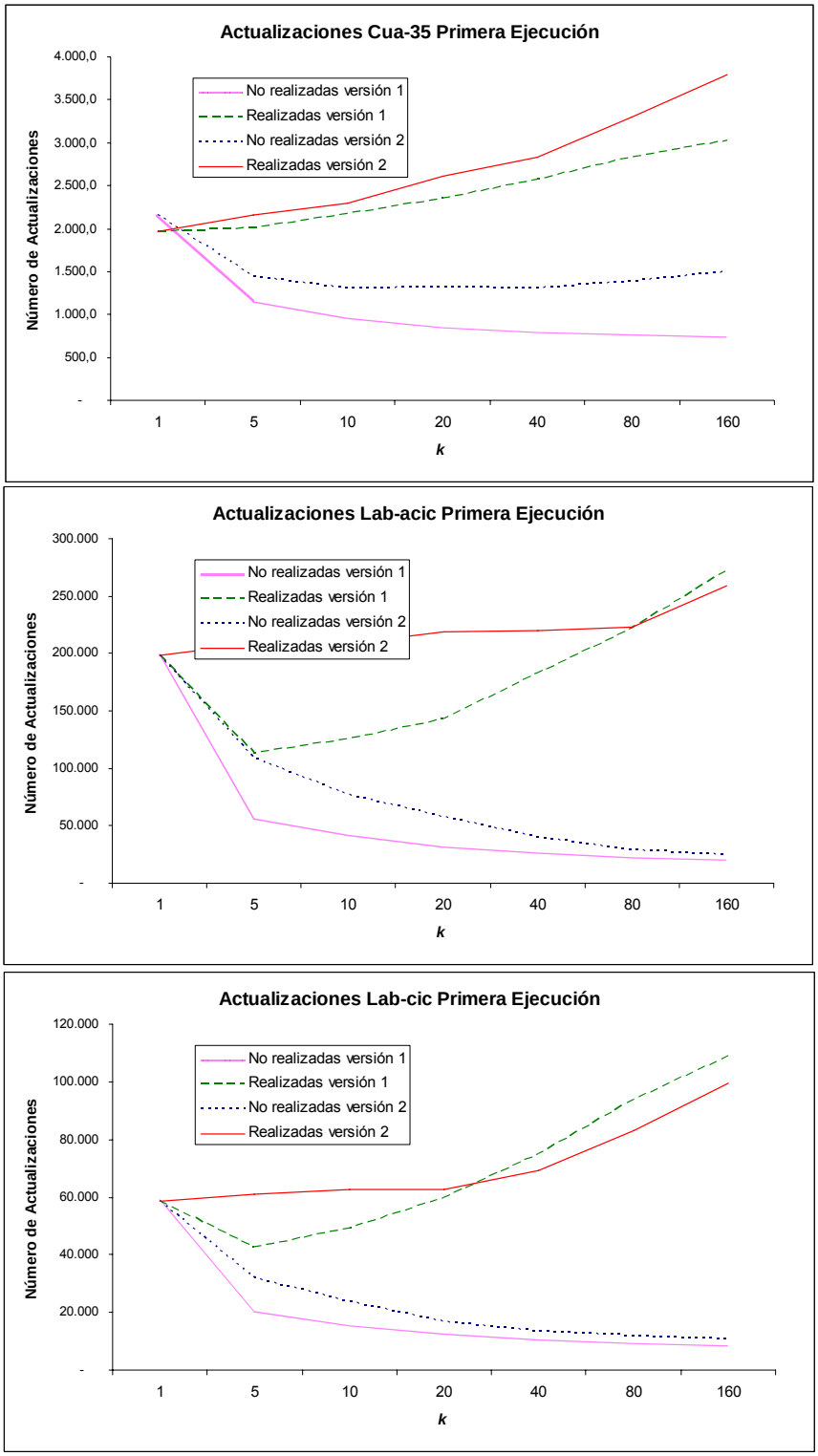


Figura 16. Resultados en Cua-35, Lab-acic y Lab-cic. Presentamos las actualizaciones realizadas y no realizadas en la primera ejecución con la versión 1 y 2 de LRTA*(*k*).

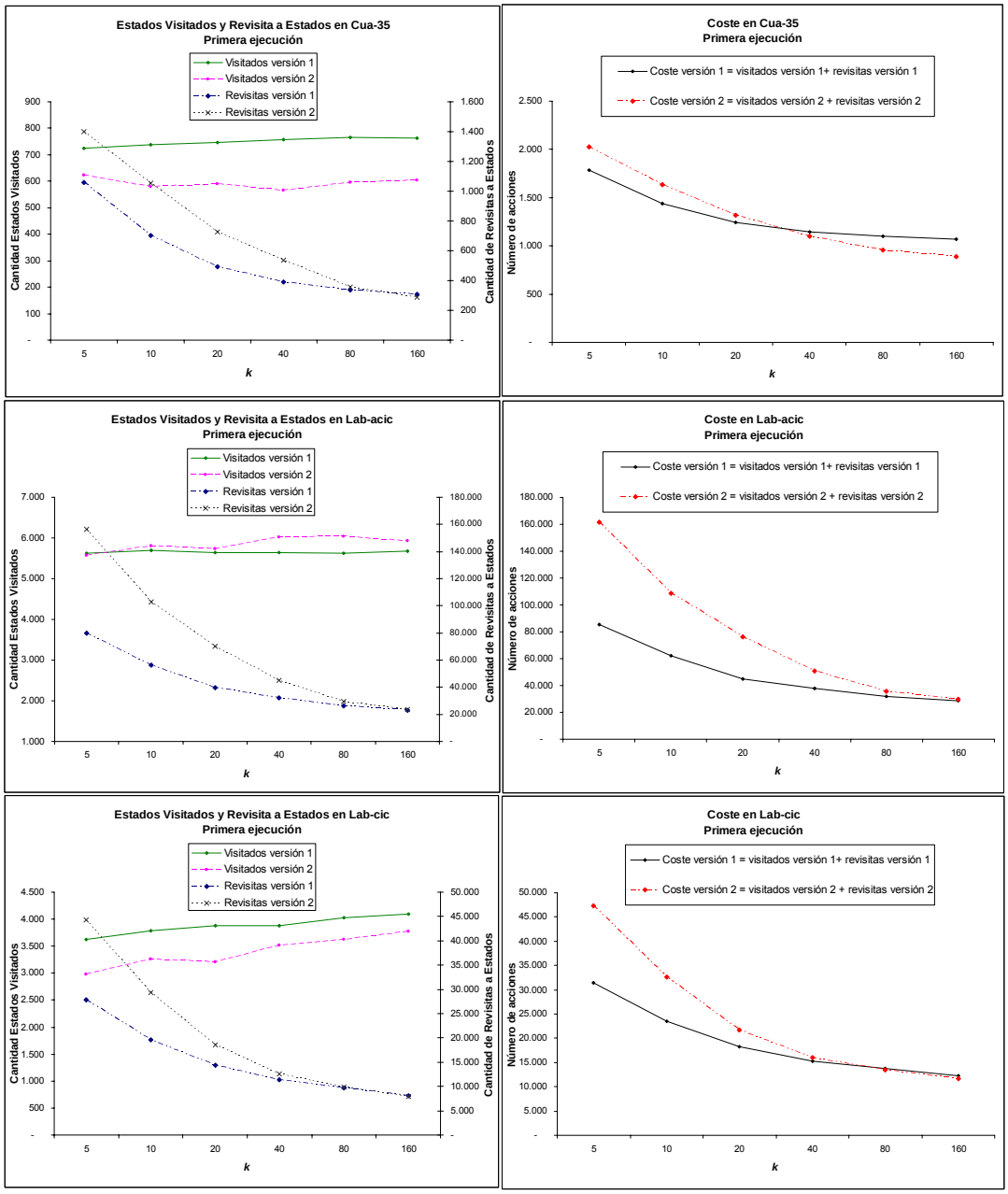


Figura 17. Resultados en Cua-35, Lab-acic y Lab-cic. Presentamos la cantidad de estados visitados, la cantidad de revisitas a los estado ya visitados y el coste en primera ejecución con la versión 1 y 2 de LRTA*(*k*).

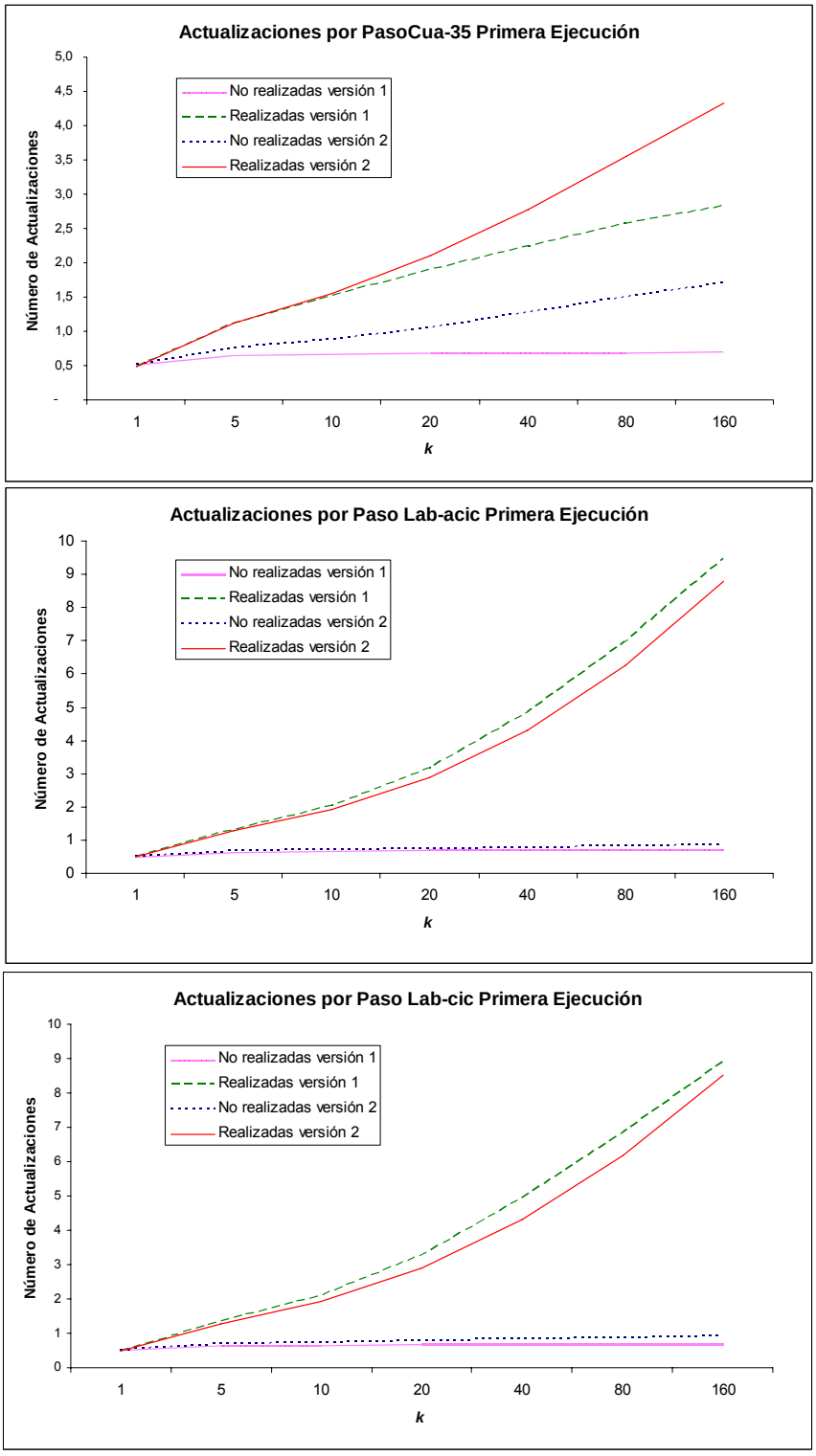


Figura 18. Resultados en Cua-35, Lab-acic y Lab-cic. Presentamos las actualizaciones realizadas y no realizadas por paso en la primera ejecución con la versión 1 y 2 de LRTA*(*k*).

Es de destacar que la mejora en coste obtenida a medida que k crece, es proporcionalmente mayor que el empeoramiento en tiempo por paso (excepto con $k = \infty$ en tiempo por paso y $k = 160$ en Cua-35 con la versión 2). Por ejemplo, con $k = 40$ con la versión 2: en Cua-35 el coste disminuye 4 veces y el tiempo por paso aumenta 3,4 veces, en Lab-acic el coste disminuye 7,7 veces y el tiempo por paso aumenta 4,4 veces y en Lab-cic el coste disminuye 7,1 veces y el tiempo por paso aumenta 4,4 veces.

	Cua-35 versión 1					Cua-35 versión 2				
k	*T* %	*C* %	*E* %	*M* %	*T/P* %	*T* %	*C* %	*E* %	*M* %	*T/P* %
100%	552	577.694	656	12.984	0,0010	552	577.694	656	12.984	0,0010
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	65%	44%	57%	103%	148%	66%	44%	57%	102%	150%
10	54%	30%	39%	103%	181%	55%	30%	39%	106%	184%
20	47%	21%	27%	104%	223%	48%	21%	27%	111%	230%
40	43%	15%	19%	105%	279%	44%	15%	19%	116%	292%
80	40%	12%	14%	107%	346%	42%	11%	14%	123%	375%
160	39%	9,3%	10,0%	109%	427%	40%	8,5%	10,4%	130%	476%
∞	48%	5,5%	4,4%	115%	871%	53%	4,6%	4,8%	264%	1165%
	Lab-acic versión 1					Lab-acic versión 2				
100%	14.263	16.617.074	1.054	9.816	0,0009	14.263	16.617.074	1.054	9.816	0,0009
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	64%	38%	43%	106%	168%	64%	38%	43%	104%	170%
10	55%	25%	26%	107%	217%	55%	25%	26%	107%	219%
20	49%	17%	17%	107%	290%	49%	17%	17%	111%	291%
40	44%	11%	11%	107%	393%	45%	11%	11%	115%	395%
80	42%	7,9%	7,7%	107%	531%	42%	7,9%	7,8%	117%	534%
160	40%	5,7%	5,5%	107%	711%	41%	5,7%	5,5%	119%	716%
∞	59%	0,1%	0,2%	107%	41164%	59%	0,1%	0,2%	123%	43402%
	Lab-cic versión 1					Lab-cic versión 2				
100%	4.413	5.085.788	536	10.314	0,0009	4.413	5.085.788	536	10.314	0,0009
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	70%	39%	50%	105%	179%	67%	39%	50%	104%	172%
10	61%	26%	32%	106%	231%	58%	26%	32%	109%	221%
20	54%	18%	21%	106%	308%	52%	18%	21%	112%	294%
40	50%	12%	14%	106%	419%	48%	12%	14%	116%	397%
80	47%	8,4%	10%	106%	564%	45%	8,4%	10%	117%	534%
160	45%	6,1%	7,2%	105%	751%	44%	6,1%	7,3%	118%	717%
∞	59%	0,4%	0,7%	102%	13226%	59%	0,4%	0,7%	120%	14097%

Tabla 2. Resultados en convergencia a soluciones óptimas en Cua-35, Lab-acic y Lab-cic para la versión 1 y 2 de LRTA*(k).

4.4.2 Convergencia a soluciones óptimas

La Tabla 2 contiene resultados para convergencia en Cua-35, Lab-acic y Lab-cic. A continuación vemos las medidas de desempeño:

- Tiempo total de convergencia: en ambas versiones se obtiene un tiempo menor que LRTA* en todos los valores de k. El tiempo total de convergencia obtenido con la versión 1 es similar al de la 2. Ambas versiones realizan un esfuerzo (en cantidad de actualizaciones)

parecido, como se aprecia en la Figura 19. Podemos observar que el tiempo total de convergencia disminuye a medida que k crece, en la versión 1 y 2. Sólo con $k = \infty$ aumenta. Observe que el mayor número de actualizaciones se realizan en las primeras ejecuciones (como se aprecia en la Figura 20 en Cua-35), por tanto en estas ejecuciones se consume una mayor cantidad de tiempo.

- Coste: en ambas versiones el coste siempre disminuye a medida que k crece. Se obtienen resultados similares para ambas versiones, sólo en Cua-35 la versión 2 obtiene un coste algo menor para $k > 40$. A medida que se converge a la solución óptima, el espacio de estados se va conociendo más y más en cada ejecución. Esto hace que el efecto de la propagación en ambas versiones sea similar en los 3 escenarios de prueba. La diferencia entre las versiones se tiende a reducir durante el proceso de convergencia.

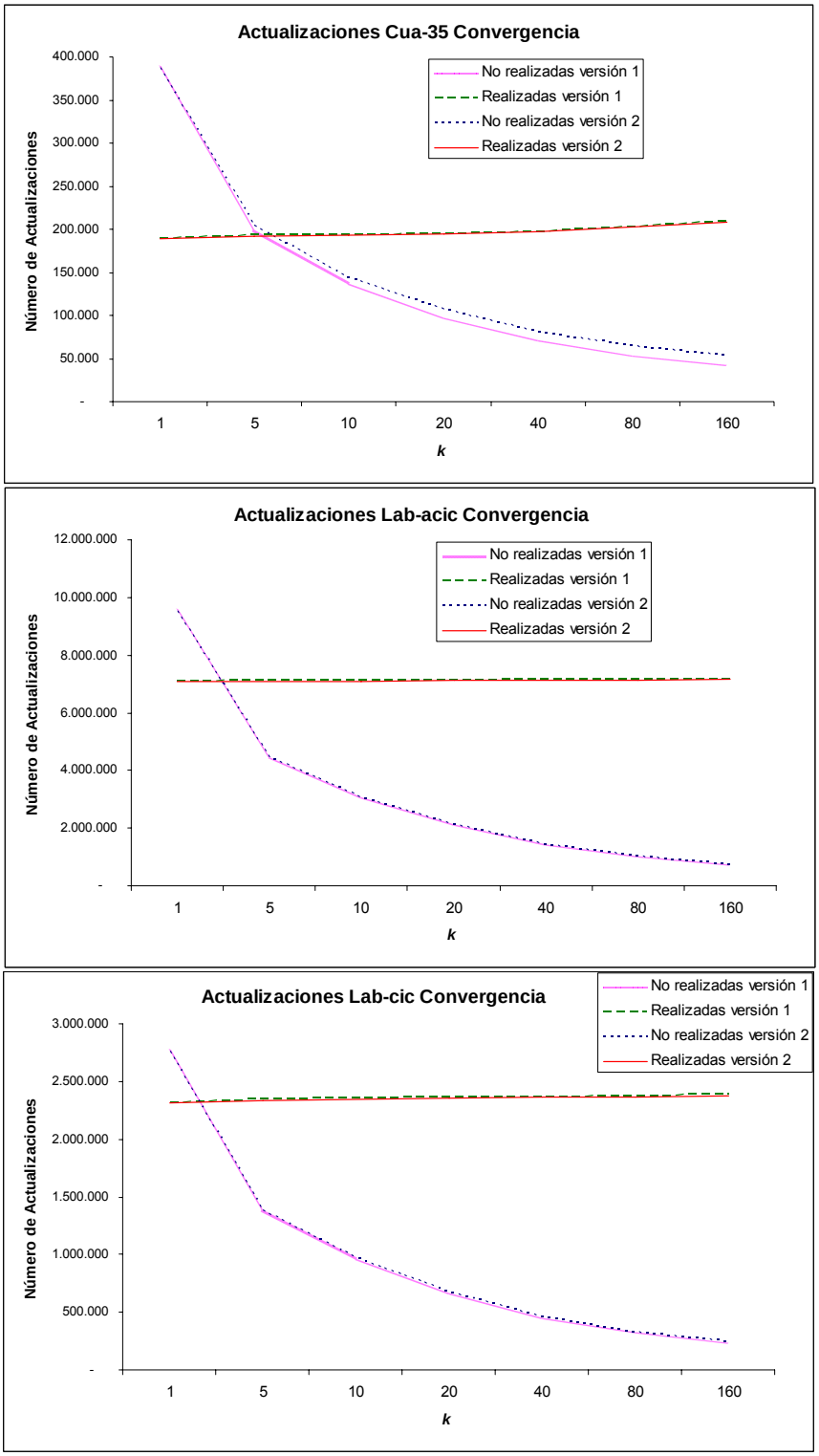


Figura 19. Resultados en Cua-35, Lab-acic y Lab-cic. Presentamos las actualizaciones realizadas y no realizadas en convergencia con la versión 1 y 2 de LRTA*(*k*).

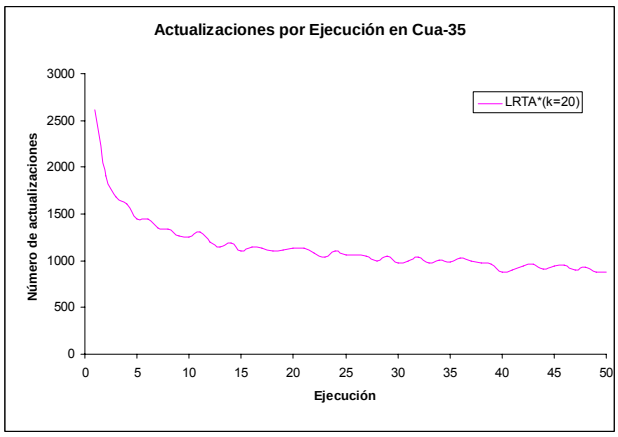


Figura 20. Resultados en Cua-35. Presentamos el número de actualizaciones promedio en las 50 primeras ejecuciones en convergencia con LRTA*(*k*).

- Ejecuciones: el número de ejecuciones necesarias para converger a una solución óptima disminuye cuando *k* crece. Este efecto positivo de la propagación, se debe a que se aprenden más rápido los valores exactos de la heurística en la ruta óptima. Para justificar esta afirmación, realizamos el siguiente experimento. En un ejemplar de laberinto acíclico calculamos la ruta óptima con A* (sólo existe una). Luego actualizamos la heurística de todos los estados en la ruta óptima y sus sucesores inmediatos con los valores exactos. En el laberinto de la Figura 21 podemos observar un ejemplo. En estado inicial es *c1* y el objetivo *f5*.En (i) vemos la ruta óptima calculada con A* y en (ii) las heurísticas actualizadas. Si usamos como heurística inicial la heurística obtenida después de actualizar (como en (ii)) en LRTA* o LRTA*(*k*), el agente seguirá la ruta óptima. Este no sale de la ruta porque se ha formado una especie de talud con la heurística de los sucesores del camino óptimo (recordemos que un laberinto acíclico existe sólo una ruta entre dos estados). Llamamos *heurística ideal* (h_{ideal}) a la heurística en los estados en el camino óptimo y sus sucesores después de actualizar. Medimos el aprendizaje de h_{ideal} de un algoritmo en una ejecución, sumando las diferencias entre h_{ideal} y la heurística de los estados en la ruta óptima y sus sucesores inmediatos, obtenida en una ejecución con el algoritmo. Cuando esta suma es

cero, el algoritmo ha aprendido h_{ideal}. En la Figura 22 evaluamos el aprendizaje sobre ejecuciones sucesivas hasta la convergencia en un ejemplar de Lab-acic con la versión 1. En el eje horizontal de la gráfica vemos el número de ejecución y en el vertical la suma de las diferencias entre h_{ideal} y la heurística de los estados en la ruta óptima y sus sucesores inmediatos obtenidas en una ejecución de LRTA*(k). Podemos apreciar que a mayor k el aprendizaje de h_{ideal} se hace con menos ejecuciones. Esto explica la disminución en el número de ejecuciones cuando k crece.

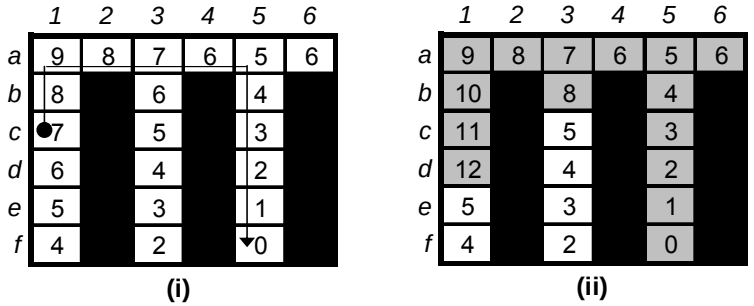


Figura 21. Ejemplo de un mecanismo para obtener la heurística ideal en los estados en la ruta óptima y sus sucesores inmediatos en un ejemplar de laberinto acíclico.

En ambas versiones el número de ejecuciones para converger es muy similar para cada valor de k. Esto nos indica que la diferencia entre las versiones se tiende a reducir en convergencia.

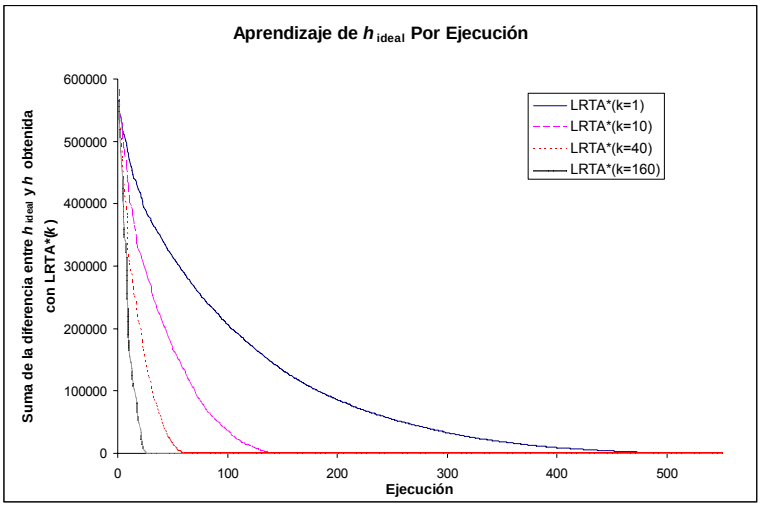


Figura 22. Resultados en un ejemplar de Lab-cic. Presentamos el aprendizaje de la heurística ideal por ejecución con la versión 1 de LRTA*(*k*).

- Memoria: la memoria consumida aumenta cuando *k* crece. La versión 2 consume más memoria que la versión 1 (excepto con $k = 5$). El análisis de este comportamiento es el mismo que en la primera ejecución.

- Tiempo por paso: aumenta a medida que *k* crece. Se pueden apreciar pequeñas diferencias entre el tiempo promedio por paso que se obtiene con la versión 1 y 2. El tiempo por paso se calcula dividiendo el tiempo total por el número de pasos realizados en convergencia. La pequeña diferencia entre las versiones para un valor de *k* se explican por las pequeñas diferencia entre el valor del dividendo o divisor.

Tal como en la primera ejecución, la mejora en coste es proporcionalmente mayor al empeoramiento en tiempo por paso. Por ejemplo, con $k = 40$ en la versión 2: en Cua-35 el coste disminuye 6,7 veces y el tiempo por paso aumenta 2,9 veces, en Lab-acic el coste disminuye 9,1 veces y el tiempo por paso aumenta 4 veces, en Lab-cic el coste disminuye 8,3 veces y el tiempo por paso aumenta 4 veces. Los resultados muestran que la propagación tiene un efecto mayor en Lab-acic. En este escenario de pruebas es donde se obtienen los mejores resultados en convergencia a soluciones óptimas.

4.4.3 Estabilidad del proceso de convergencia a soluciones óptimas

En [Shimbo e Ishida, 2003], se crítica la inestabilidad en el proceso de convergencia a soluciones óptimas en LRTA*. Experimentalmente se observa que la convergencia no es monótona: una ejecución puede requerir un mayor número de acciones o movimientos que la ejecución anterior, sobre el mismo ejemplar de problema. En la Figura 23 podemos ver el proceso de convergencia a una solución óptima en un ejemplar de Lab-acic con LRTA*.

La estabilidad del proceso de convergencia se mide con los siguientes índices:

- $IAE = \sum_{i=1}^{\infty} | c(i) - h^*(s) |$
- $ISE = \sum_{i=1}^{\infty} (c(i) - h^*(s))^2$
- $ITAE = \sum_{i=1}^{\infty} i \, | c(i) - h^*(s) |$
- $ITSE = \sum_{i=1}^{\infty} i(c(i) - h^*(s))^2$
- $SOD = \sum_{i=1}^{\infty} \max\{0, c(i+1) - c(i)\}$

$c(i)$ denota el coste de la solución en la ejecución i y $h^*(s)$ el coste de la solución óptima. IAE es el error absoluto en convergencia. ISE es la suma del cuadrado del error en convergencia. ITAE e ITSE corresponden a dos versiones con peso de IAE e ISE respectivamente, que penalizan errores sostenidos. SOD suma la diferencia del costo de dos ejecuciones sucesivas cuando la solución empeora. Si SOD es igual a 0 la convergencia es monótona ($c(i+1) \leq c(i) \ \forall i$). Menores valores en estos índices indican mejor calidad del proceso de convergencia.

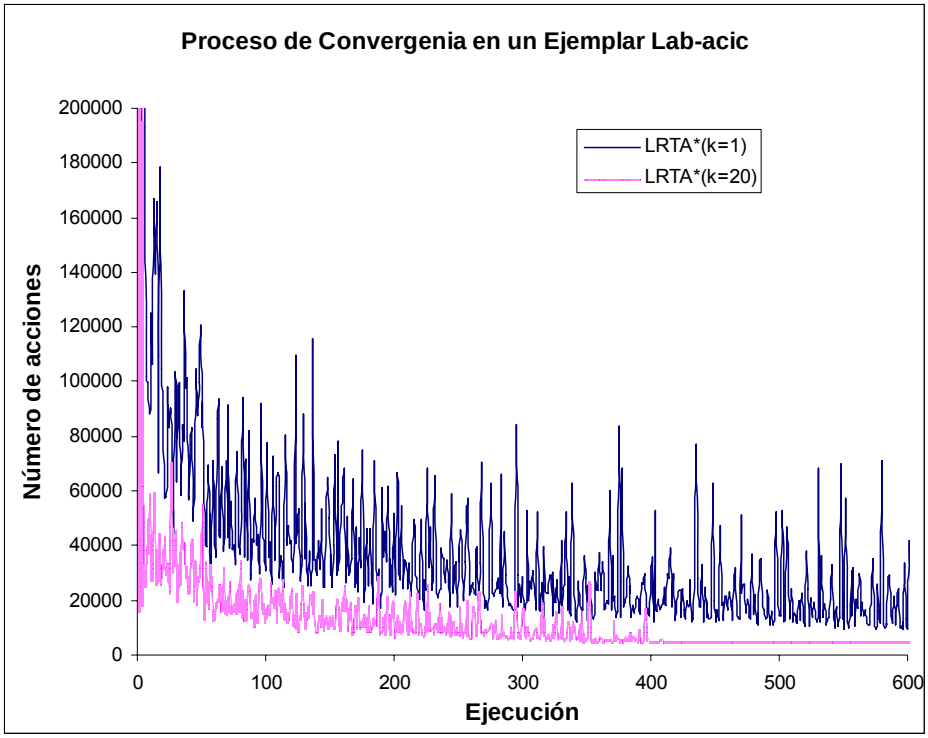


Figura 23. Proceso de convergencia a la solución óptima en un ejemplar de Lab-acic con LRTA* y LRTA*(k = 20).

En la Tabla 3 podemos apreciar los índices que miden la estabilidad del proceso de convergencia a soluciones óptimas para los tres escenarios. La tabla muestra los índices para la versión 1 (para la versión 2 el resultado es similar). Podemos observar que todos los índices disminuyen monótonamente a medida que k crece. Esto nos indica que la estabilidad del proceso de convergencia mejora cuando k crece. En la Figura 23 podemos como mejora en proceso de convergencia de LRTA*(k=1) con LRTA*(k=20) en un ejemplar de Lab-acic con la versión 1. La mejora en los índices es mayor en el escenario con peor heurística inicial. Vemos que en Lab-cic es mayor que en Lab-cic y en Lab-cic mayor que en Cua-35. Esto se explica porque el efecto de k (la propagación) es más significativo, en el escenario que necesita una mayor cantidad de actualizaciones.

Cua-35					
k	IAE	ISE	ITAE	ITSE	SOD
1 (LRTA*)	3,6E+05	8,3E+08	1,7E+08	3,6E+11	1,8E+05
5	1,3E+05	1,7E+08	3,7E+07	4,5E+10	6,6E+04
10	9,0E+04	1,0E+08	1,7E+07	1,9E+10	4,4E+04
20	6,5E+04	7,4E+07	8,3E+06	9,3E+09	3,2E+04
40	4,9E+04	5,5E+07	4,3E+06	4,8E+09	2,4E+04
80	3,9E+04	4,7E+07	2,3E+06	2,9E+09	1,9E+04
160	3,2E+04	4,4E+07	1,4E+06	1,9E+09	1,6E+04
∞	2,3E+04	6,2E+07	4,4E+05	1,2E+09	1,1E+04
Lab-acic					
k	IAE	ISE	ITAE	ITSE	SOD
1 (LRTA*)	1,3E+07	3,3E+12	3,6E+09	5,5E+14	8,3E+06
5	5,0E+06	6,0E+11	7,5E+08	6,0E+13	3,2E+06
10	3,4E+06	3,2E+11	3,4E+08	2,2E+13	2,1E+06
20	2,3E+06	1,8E+11	1,6E+08	8,4E+12	1,4E+06
40	1,5E+06	1,1E+11	6,8E+07	3,4E+12	9,0E+05
80	1,1E+06	6,8E+10	3,4E+07	1,5E+12	6,1E+05
160	7,7E+05	4,5E+10	1,8E+07	6,9E+11	4,3E+05
∞	1,7E+04	2,4E+08	2,7E+04	3,8E+08	6,7E+03
Lab-cic					
k	IAE	ISE	ITAE	ITSE	SOD
1 (LRTA*)	4,5E+06	4,7E+11	7,1E+08	4,8E+13	2,8E+06
5	1,7E+06	8,8E+10	1,6E+08	5,6E+12	1,0E+06
10	1,2E+06	5,0E+10	7,3E+07	2,2E+12	6,8E+05
20	7,7E+05	2,9E+10	3,4E+07	8,9E+11	4,4E+05
40	5,2E+05	1,8E+10	1,5E+07	3,7E+11	2,9E+05
80	3,7E+05	1,2E+10	7,6E+06	1,7E+11	2,0E+05
160	2,7E+05	7,9E+09	4,1E+06	8,5E+10	1,4E+05
∞	1,9E+04	2,0E+08	4,1E+04	4,0E+08	7,3E+03

Tabla 3. Índices de estabilidad del proceso de convergencia a soluciones óptimas en Cua-35, Lab-acic y Lab-cic para la versión 1de LRTA*(*k*).

4.4.4 Conclusiones

Las dos versiones de LRTA*(*k*) evaluadas muestran una mejora sustancial en rendimiento respecto a LRTA*, en la primera ejecución, convergencia a soluciones óptimas y estabilidad del proceso de convergencia. La mejora depende del parámetro *k*: a mayor *k* mejores resultados. La mejora en coste exige un mayor tiempo de planificación por paso. Los resultados muestran que valores pequeños de *k* provocan importantes beneficios manteniendo un tiempo por paso razonable. La diferencia en rendimiento en las versiones de LRTA*(*k*) se manifiesta principalmente en la primera ejecución. El esfuerzo que se realiza en términos de actualizaciones y la manera en que el agente recorre el espacio de estados, marcan la diferencia entre las versiones para un valor *k*. Resultados adicionales se pueden ver en [Hernandez y Meseguer, 2005a] y [Hernandez y Meseguer, 2005b].

4.5 Resumen

En este capítulo presentamos el mecanismo de propagación acotada de valores heurísticos con iteración de valores, dos versiones de propagación acotada, y el concepto de soporte una estimación heurística. LRTA*(*k*) es un algoritmo que combina la propagación acotada con LRTA*. En LRTA*(*k*) se implementan las dos versiones de propagación acotada y los soportes de una estimación heurística. Experimentalmente, las versiones de LRTA*(*k*) muestran una mejora sustancial en el rendimiento respecto a LRTA* en la primera ejecución, en convergencia soluciones óptimas y estabilidad del proceso de convergencia. La mejora depende del parámetro *k*, a mayor *k* mejores resultados. Por otro lado, a mayor *k* mayor es el tiempo por episodio de planificación. Valores de *k* pequeño implican aumentos pequeños de tiempo por episodio planificación, pero beneficios importantes en términos de coste.

Capítulo cinco

5 PROPAGACIÓN ACOTADA SOBRE UN ESPACIO LOCAL

La propagación acotada con iteración de valores es un mecanismo sencillo, que permite obtener buenos resultados cuando se combina con LRTA*. Sin embargo, esta estrategia presenta algunos puntos débiles [Hernández y Meseguer, 2007a], que pueden afectar negativamente su rendimiento. En este capítulo, describimos esos puntos débiles y proponemos un nuevo mecanismo de propagación que permite resolverlos total o parcialmente.

El contenido del capitulo es el siguiente. Primero se exponen los puntos débiles del método de propagación expuesto en el capítulo anterior. Segundo, se presenta el nuevo mecanismo de propagación y se define el espacio local de búsqueda alrededor del estado actual. Tercero, se describe un método de selección de un espacio local de aprendizaje. Cuarto, se describe como actualizar un espacio local de aprendizaje. Después, se presenta el algoritmo LRTA*$_{LS}(k)$, una combinación del nuevo mecanismo de propagación y LRTA*. Finalmente, se evalúa experimentalmente el nuevo algoritmo y se compara con LRTA*(k) en distintos escenarios de prueba.

5.1 Puntos débiles de la propagación acotada con iteración de valores

El mecanismo de actualización de la propagación acotada con iteración de valores se implementa con la cola de propagación Q. Esta cola, mantiene los estados del espacio local de aprendizaje, tal como se explica en el capítulo anterior. Hemos notado que la propagación con iteración de valores tiene tres puntos débiles, que son los seiguientes:

1. *Considerar estados que no actualizan su heurística.* Un estado x que entra en Q puede no cambiar su valor $h(x)$. Esto lleva a realizar trabajo en vano. Este punto se resuelve parcialmente con el uso de soportes.

2. *Repetición de entradas en Q*. Un estado puede entrar más de una vez en Q, por tanto se puede actualizar varias veces en un episodio de planificación, antes de obtener el valor final de la heurística. Esto nos lleva a preguntarnos: ¿es posible que un estado se actualice sólo una vez por episodio de planificación, en lugar de varias veces?

3. *Orden de entrada en Q*. El orden en que los estados entran en Q, combinado con el valor de la cota k de la propagación, puede afectar el resultado.

Por ejemplo, en la Figura 24 (*i*), *a*, *b*, *c* y *d* son estados, la columna de la izquierda (Iter.) corresponde al número de la iteración, y los números debajo de los estados son el valor heurístico que tienen en cada iteración. La acción de moverse desde un estado hacia un sucesor cuesta 1. Los sucesores son revisados en orden oeste-este. El estado actual es *d*. La iteración 0 muestra los valores heurísticos iniciales, $Q = \{d\}$. En la iteración 1, $h(d)$ cambia a 4, *d* sale de Q y se agregan sus sucesores, $Q = \{c\}$. En la iteración 2, $h(c)$ cambia a 5, *c* sale de Q y se agregan sus sucesores, $Q = \{b, d\}$. En la iteración 3, se revisa *b*. Este sale de Q, como $h(b)$ no cambia no se agregan estados a Q. En la iteración 4, $h(d)$ cambia a 6, *d* sale de Q y se agregan sus sucesores, $Q = \{c\}$. En la iteración 5, se revisa *c*. Este sale de Q, como $h(c)$ no cambia no se agregan estados a Q. Después de 5 iteraciones no se pueden hacer más cambios ($Q = \varnothing$), y el proceso se detiene. En el ejemplo podemos ver que *c* y *d*, entran dos veces en Q, *d* se actualiza dos veces y *c* una vez (punto débil 1 y 2). Si $k = 3$, el resultado depende del orden de entrada de estados en Q (punto débil 3). Si el sucesor del oeste entra primero en Q, los estados *d*, *c*, *b* entran en Q en este orden, produciendo las estimaciones heurísticas de la Figura 24 (*ii*). El agente se mueve a *c*, y luego puede visitar *d* nuevamente. Si el sucesor del este entra primero en Q, los estados entran en el orden *d*, *c*, *d*, produciendo los valores heurísticos iguales a los de la iteración 5. Por tanto el agente se mueve de *d* a *c* y luego a *b*, sin visitar *d* nuevamente.

La propagación acotada con iteración de valores sobre estados visitados, resuelve el punto débil 1 mediante el uso de soportes. Como se explica en el capítulo anterior, el uso de soportes en la propagación restringe la inserción de estados en la cola de propagación *Q*. Sólo se insertan aquellos estados que efectivamente cambiarán su estimación heurística. Cuando la propagación es sobre cualquier estado, el uso de soportes resuelve parcialmente esta desventaja.

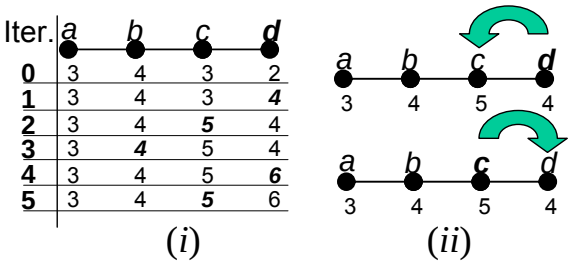


Figura 24. (*i*) Ejemplo de propagación con iteración de valores; (*ii*) si un sucesor entra en *Q*, en orden oeste-este, y $k = 3$, el estado *d* puede ser re-visitado.

5.2 Propagación acotada sobre un espacio local

Para tratar de resolver los puntos débiles, proponemos otra manera de realizar la propagación acotada de cambios heurísticos [Hernández y Meseguer, 2007a]. La diferencia principal respecto a la propagación con iteración de valores, está en la forma de seleccionar y actualizar el espacio local de aprendizaje. El nuevo método se basa en la noción de *espacio local* alrededor del estado actual [Koenig, 2004].

Formalmente, el espacio local es un par (I, F), donde $I \subset X$ es un conjunto de *estados interiores* y $F \subset X$ es un conjunto de *estados frontera*. Los conjuntos satisfacen estas condiciones: F rodea total e inmediatamente a I y $F \cap I = \varnothing$. Por ejemplo en la Figura 25, vemos una cuadrícula 4-conectada, los puntos negros son estados, las líneas continuas son conexiones (arcos) entre estados, el estado actual es *s*, el conjunto de estados interiores (I) esta dentro de la

elipse con fondo gris y el conjunto de estados frontera (F) son los estados con una marcados con una *f*.

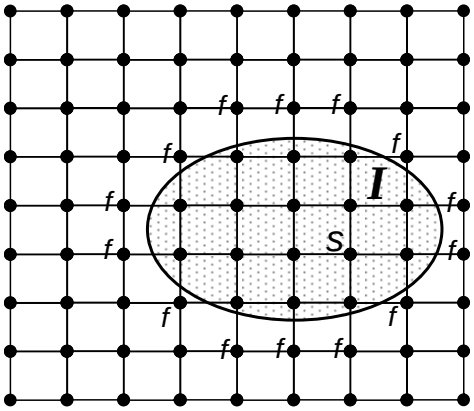


Figura 25. Ejemplo de espacio local alrededor del estado actual.

La propagación sobre un espacio local, primero selecciona el espacio local alrededor del estado actual. El tamaño del espacio local de aprendizaje esta limitado por el parámetro k ($|I| \leq k$). La idea es que los k estados que se seleccionan tengan una alta probabilidad de cambiar su heurística. Se permite seleccionar un espacio local con un máximo de k estados en interiores. Luego, se actualiza la heurística de los k estados en el conjunto de interiores a partir de la heurística de los estados en la frontera. Cada estado es actualizado sólo una vez. A continuación, se presenta el método de selección del espacio local y el método de actualización de los estados en interiores a partir de la heurística de los estados en la frontera.

5.3 Selección del espacio local de aprendizaje

Con este método, se intenta seleccionar un espacio local alrededor del estado actual, cuyos estados interiores tengan alta probabilidad de cambiar su estimación heurística. Para este fin se propone el siguiente método:

1. Inicializar, $I \leftarrow \varnothing, Q \leftarrow \{s_a \mid s_a$ es el estado actual$\}$.

2. Realizar el siguiente ciclo hasta que Q este vacío o $|I|$ sea igual a k.

   2.1. Extraer el estado v desde la primera posición de Q.

   2.2. Si v es distinto a un estado objetivo, entonces:

      2.2.1. Obtener los sucesores de v que no están en I.

      2.2.2. Si $h(v)$ tiene oportunidad de cambiar (esto es, si $h(v) < \min_{w \in \{Succ(v) - I\}} h(w) + c(v, w)$ *condición de actualización* que difiere de la condición de actualización de LRTA*(k), en que sólo se considera los sucesores que no están en interiores en el cálculo del mínimo), entonces, incluir v en I, e incluir los sucesores de v que no están en I en Q.

3. El conjunto F es el que rodea completa e inmediatamente a I.

Por ejemplo, aplicaremos la estrategia a la heurística y estado inicial de la Figura 24 (*i*). Se parte con $I = \{\varnothing\}$ y $Q = \{d\}$. Luego, se extrae d de Q y se verifica la condición de actualización sobre $h(d)$. Como $h(d) < \min_{w \in \{Succ(d) - I\}} h(w) + c(d, w)$ se agrega d a I, y los sucesores de d que no están en I a Q, $I = \{d\}$, $Q = \{c\}$. Se extrae c, el primer elemento de Q y se verifica la condición de actualización en $h(c)$. Como $h(c) < \min_{w \in \{Succ(c) - I\}} h(w) + c(c, w)$ se agrega c a I, y los sucesores de c que no están en I a Q, $I = \{d, c\}$, $Q = \{b\}$. Luego se extrae b de Q. Como $h(b) = \min_{w \in \{Succ(b) - I\}} h(w) + c(b, w)$, no se agrega nada a I o Q. Como $Q = \{\varnothing\}$, el ciclo se detiene. Finalmente, $I = \{d, c\}$ y $F = \{b\}$.

Si la heurística es inicialmente consistente, todos los estados interiores del espacio local (I, F) construido con el método anterior, cambiarán su estimación heurística, como se demuestra con el siguiente resultado.

Proposición 1. Si la heurística es inicialmente consistente, el algoritmo LRTA*($k = \infty$) con propagación sobre cualquier estado, actualizará todos los estados de I.

Dem. Por inducción en el número de estados de I. Si $|I| = 1$, entonces $\{I\} = \{x\}$, el estado actual. En este caso $h(x)$ es actualizado ya que por construcción, satisface la condición de actualización. Suponemos como cierta la proposición para $|I| = p$, y la probaremos para $|I| = p + 1$. Sea, $z \in I$, $z \neq x$, $y = \text{argmin}_{v \in Succ(z)} h(v) + c(z, v)$. Si $y \notin I$, por construcción, z satisface la condición de actualización. Si $y \in I$, sea $I' = I - \{z\}$. Dado que $|I'| = p$, cada estado en I' será actualizado, así y será actualizado, pasando de $h_{old}(y)$ a $h_{new}(y)$, $h_{old}(y) < h_{new}(y)$. Esto causa que z sea actualizado. Se consideran 2 casos:

1. Si y es $\text{argmin}_{v \in Succ(z)} c(z, v) + h(v)$. La heurística inicial es consistente, así $h(z) \leq c(z,y) + h_{old}(y) < c(z,y) + h_{new}(y)$. Luego, z satisface la condición de actualización, y $h(z)$ cambiará.

2. Si $w = \text{argmin}_{v \in Succ(z)} c(z, v) + h(v)$, $w \neq y$. Si $w \notin I'$, por construcción, z satisface la condición de actualización. Si $w \in I'$, w ha pasado de $h_{old}(w)$ a $h_{new}(w)$, $h_{old}(w) < h_{new}(w)$. Dado que y fué el mínimo inicial, sabemos que $c(z, y) + h_{old}(y) < c(z, w) + h_{new}(w)$. Así, $h(z) \leq c(z, y) + h_{old}(y) < c(z, w) + h_{new}(w)$. Luego, z satisface la condición de actualización, y $h(z)$ cambiará.

LRTA*(k=∞) actualiza todos los estados interiores porque cada sucesor de un estado que cambia entra en la cola Q, la cual no tiene limite en su tamaño (k=∞). qed.

Si la heurística es admisible pero no consistente, algunos estados en interiores pueden no cambiar con LRTA*(k=∞). Por ejemplo en la Figura 26, la heurística de lo estados a, b, c, d, e, y f, son los números sobre cada punto. El estado actual es c, y los movimientos cuestan 1. Con el método recién expuesto, se obtiene que $I = \{c, d\}$. La heurística $h(c)$ cambiará a 3, pero $h(d)$ no cambiará porque el mínimo de sus sucesores más el coste de ir a este es 4 ($h(d)$ + 1 o 3+1), valor que es menor que el h actual de c, $h(c) = 5$.

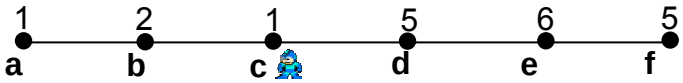


Figura 26. Ejemplo de actualización de una heurística admisible, pero no consistente.

5.4 Actualización del espacio local de aprendizaje

El método de actualización de los estados interiores es una variación del método de caminos mínimos de Dijkstra, implementado con una estrategia de búsqueda de primero el mejor. La idea es actualizar la heurística de un estado de I con la suma entre la distancia desde ese estado a un estado de la frontera y la heurística del estado de la frontera, minimizado sobre todos los estados de la frontera. Un método similar es usado en [Koenig, 2004]. Después de la actualización, la nueva heurística de un estado x que pertenece a I, es $h(x) = \min_{y \in F} k(x, y)+h(y)$. La actualización de los estados interiores a partir de la heurística de los estados en la frontera se hace como se explica a continuación.

Se realizar lo siguiente mientras I no este vacío:

1. Calcular el par de estados conectados (i, f), $i \in I, f \in F, f \in Succ(i)$, tal que $(i, f) = \text{argmin}_{v \in I, w \in F, w \in Succ(v)} c(v, w) + h(w)$.
2. Actualizar, $h(i) \leftarrow \max(h(i), c(i,f) + h(f))$.
3. Quitar i de I, agregar i a F.

Por ejemplo, aplicaremos el método de actualización a la heurística y estado inicial de la Figura 24 (i). El resultado de la actualización se muestra en la Figura 27. Tenemos como resultado del proceso de selección del espacio local que $I = \{d, c\}$ y $F = \{b\}$. En la actualización (Act.) 1 de la Figura 27, $(i, f) = (c, b)$, y $h(c)$ cambia a 5. Se extrae c de I y se agrega a F. En la actualización 2, $(i, f) = (d, c)$, y $h(d)$ cambia a 6. Se extrae d de I y se agrega a F. Como $I = \{\varnothing\}$, el ciclo termina.

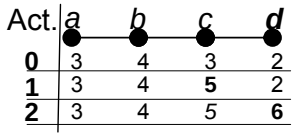


Figura 27. Ejemplo de actualización de un espacio local.

El mecanismo de actualización mantiene la admisibilidad, como se demuestra a continuación.

Proposición 2. *Si la heurística inicial es admisible, después de la actualización todos los estados interiores mantienen la admisibilidad.*

Dem. Sea $(i, f) = \text{argmin}_{v \in I,\, w \in F,\, w \in Succ(v)} c(v, w) + h(w)$. Como $i \in I$ y esta conectado al estado en la frontera f, i se actualizará (satisface la condición de actualización por construcción). Supongamos que existe una ruta óptima desde i al objetivo, que pasa por el sucesor j. Se distinguen dos casos:

1. $j \in F$. Después de actualizar,

   $h(i) = c(i, f) + h(f)$

   $\leq c(i, j) + h(j)$ (i, f) minimiza $c(v,w) + h(w)$

   $\leq c(i, j) + h^*(j)$ h es admisible

   $= h^*(i)$ porque es una ruta óptima

2. $j \notin F$. Si I, no contiene estados objetivo en algún punto la ruta óptima pasará por F. Sea p el estado en la ruta óptima que pertenece a F y q el estado interior en la ruta óptima justo antes de p. Después de actualizar,

   $h(i) = c(i, f) + h(f)$

   $\leq c(q, p) + h(p)$ (i, f) minimiza $c(v,w) + h(w)$

   $\leq k(i, p) + h(p)$ $k(i, p)$ incluye a $c(q, p)$

   $\leq k(i, p) + h^*(p)$ h es admisible

   $= h^*(i)$ porque es una ruta óptima

En ambos casos $h(i) \leq h^*(i)$, por tanto h se mantiene admisible. qed.

Con los mecanismos de selección y actualización de un espacio local de aprendizaje, los puntos débiles de la propagación basada en iteración de valores son resueltos parcial o totalmente. El punto 1 es totalmente resuelto cuando la heurística inicial es consistente y parcialmente resuelto cuando es admisible. El punto 2 es totalmente resuelto porque la estrategia de actualización considera cada estado en interiores sólo una vez. El punto 3 no es resuelto totalmente ya que el resultado final depende del orden de revisión de los sucesores y del valor de *k*. De todas maneras, se espera una menor dependencia al orden de los sucesores y mejor resultado para un mismo valor de *k*. Por ejemplo en la Figura 24 (i) con LRTA*(*k* = *3*), si el sucesor del oeste se revisa primero, el agente se mueve de *d* a *c*, y luego puede visitar *d* nuevamente. Con la propagación sobre un espacio local como se ve en la Figura 27, cualquiera que sea el orden de revisión de los sucesores, basta un *k* = 2, para que el agente se mueva de *d* a *c*, y luego a *b* sin visitar *d* nuevamente. Los resultados experimentales avalan esta suposición.

Es importante notar que la propagación sobre un espacio local de aprendizaje, realiza un aprendizaje más agresivo que la propagación basada en iteración de valores, porque actualiza un estado considerando los sucesores que están en la frontera del espacio local, en vez de considerar todos los sucesores. El valor heurístico de un estado, después de actualizar con los sucesores en la frontera, puede ser mayor que cuando se actualiza con todos los sucesores. Por ejemplo, la actualización del valor de *h*(*d*) en la Figura 24 (*i*) con LRTA*(*k*), se hace primero de 2 a 4 y luego de 4 a 6. En la Figura 27, *h*(*d*) cambia de 2 a 6 en una sola actualización.

LRTA*$_{LS}$(*k*) es un algoritmo basado en LRTA*, en el que se implementa la idea de propagación acotada sobre un espacio local de aprendizaje. El algoritmo se explica en la siguiente sección.

5.5 LRTA*$_{LS}(k)$

LRTA*$_{LS}(k)$ es un algoritmo similar a LRTA*(k). En vez de combinar la propagación con iteración de valores con LRTA*, combina la propagación acotada sobre un espacio local con LRTA*, de hecho LRTA* es un caso particular de LRTA*$_{LS}(k)$ con $k = 1$.

```
procedure LRTA-LS(k) (X, A, c, s0,G, k)
 1 for each x ∈ X do h(x) ← h0(x);
 2 repeat
 3   LRTA-LS-trial(X, A, c, s0, G, k);
 4 until h does not change;

procedure LRTA-LS-trial(X, A, c, s0, G, k)
 1 x ← s0;
 2 while x ∉ G do
 3   SelectUpdateLS(x, k);
 4   y ← argmin_{w∈Succ(x)} [h(w) + c(x,w)];(Break ties randomly)
 5   execute(a ∈ A such that a = (x, y));
 6   x ← y;

procedure SelectUpdateLS (x, k)
 1 Q ← ⟨x⟩; F ← ∅; I ← ∅; cont ← 0;
 2 while Q ≠ ∅ ∧ cont < k do
 3   v ← extract-first(Q);
 4   y ← argmin_{w∈Succ(v) ∧ w ∉ I} h(w) + c(v, w);
 5   if h(v) < h(y) + c(v, y) then
 6     I ←I ∪ {v};
 7     cont ← cont + 1;
 8     for each w ∈ Succ(v) do
 9       if w ∉ I ∧ w ∉ Q then Q ← add-last(Q,w);
10   else
11     if I ≠ ∅ then F ←F ∪ {v};
12 if Q ≠ ∅ then F ←F ∪ Q;
13 while I ≠ ∅ do
14   (i, f) ← argmin_{v∈I, w∈F, w∈Succ(v)} h(w) + c(v, w);
15   h(i) ← max[h(i), c(i, f) + h(f)];
16   I ←I - {i};
17   F ←F ∪ {i};
```

Figura 28. Algoritmo LRTA*$_{LS}(k)$.

Tal como LRTA* y LRTA*(k), LRTA*$_{LS}(k)$ guarda la información heurística entre ejecuciones. El algoritmo LRTA*$_{LS}(k)$ aparece en la Figura 28. Lo explicaremos teniendo en cuenta el algoritmo LRTA*. La principal diferencia

con LRTA* esta en el procedimiento LRTA-LS-Trial que llama a SelectionUpdateLS. Este procedimiento realiza la selección y actualización del espacio local alrededor del estado actual. El procedimiento SelectionUpdateLS primero selecciona el espacio local y luego lo actualiza. Para seleccionar el espacio local, se mantiene una cola Q de estados candidatos a ser incluidos en I o F. Q se inicializa con el estado actual y F e I con el conjunto vacío. Como máximo pueden entrar k estados en I. Esto es controlado por el contador *cont*. Se ejecuta el siguiente ciclo hasta que Q no contenga estados o *cont* sea igual a k. Se extrae el primer estado v de Q. Se calcula el estado $y \leftarrow \text{argmin}_{w \in Succ(v),\ w \notin I}\ h(w) + c(v, w)$. Si $h(v)$ satisface la condición de actualización, entonces v entra en I, se incrementa el contador y los sucesores de v que no perecen a I o Q se insertan al final de Q. En otro caso, v se inserta en F. Cuando se termina el ciclo, si Q aún contiene estados, estos se quitan de Q y se insertan en F. Una vez que el espacio local ha sido seleccionado, sus estados interiores son actualizados, ejecutando el siguiente ciclo hasta que I no contenga estados: Se selecciona el par $(i, f) = \text{argmin}_{v \in I,\ w \in F,\ w \in Succ(v)}\ h(w) + c(v, w)$, se actualiza $h(i)$, i se extrae de I y se inserta en F.

5.5.1 Prueba formal

LRTA*$_{LS}(k)$ es un algoritmo completo porque la heurística siempre se incrementa (Teorema 1 [Korf, 1990]). Para probar que converge a soluciones óptimas basta con probar que los valores heurísticos se mantienen admisibles después de la propagación acotada hacia un espacio local. Esto se garantiza con la **proposición 2** de la sección 5.4.

A partir de este resultado se garantiza la completitud y convergencia a rutas óptimas de LRTA*$_{LS}(k)$ en los mismos términos que en LRTA*. LRTA*$_{LS}(k)$ hereda las buenas propiedades de LRTA*.

Podemos comparar el funcionamiento de LRTA*(k) y LRTA*$_{LS}(k)$ observando los ejemplos de las Figuras 24 y 27, respectivamente, explicados en las secciones 5.1, 5.3 y 5.4. En el ejemplo de la Figura 24, (donde se aplica

LRTA*(k)), se necesita un k mayor que en la Figura 27 (donde se aplica LRTA*$_{LS}$(k)), para obtener la misma heurística. Para un mismo valor de k, LRTA*$_{LS}$(k) puede seleccionar el siguiente estado a visitar con una heurística más informada que LRTA*(k) y por tanto, obtener soluciones de mejor calidad. Los resultados experimentales avalan esta afirmación.

5.6 Resultados Experimentales

En esta sección se evalúa LRTA*$_{LS}$(k) para varios valores de k, en Cua-35, Lab-acic y Lab-cic. Se presentan resultados para: la primera ejecución, convergencia a soluciones óptimas y estabilidad del proceso de convergencia. Al igual que con los resultados presentados para LRTA*(k), se analiza la versión 1 y 2 en los tres escenarios de experimentación. Los resultados para los distintos k se presentan en términos de porcentaje respecto a los resultados obtenidos con LRTA* (LRTA*$_{LS}$(k = 1)). El tiempo total de búsqueda (T) esta el milisegundos, el coste (C) en cantidad de acciones, la memoria (M) en cantidad de estados que han aumentado su estimación heurística y el tiempo por paso, en milisegundos. Los resultados para convergencia incluyen el número de ejecuciones para converger a una solución óptima (E) y el tiempo total de convergencia una solución óptima en vez del tiempo total de búsqueda (T). La fila 100% de las tablas que se presentan en esta sección, contienen los resultados para LRTA*.

5.6.1 LRTA$_{LS}$(k) versus LRTA*(k)*

En esta sección presentamos los resultados experimentales de LRTA*$_{LS}$(k) y los compramos con LRTA*(k). Según veremos, LRTA* (k) consigue, con el mismo valor de k, menor número de actualizaciones, soluciones de mejor coste y tiempos de ejecución menores (y el peor caso sólo ligeramente mayores) que LRTA*(k). Todo ello nos permite afirmar que LRTA*$_{LS}$(k) implementa de forma más eficiente las ideas presentadas en el capítulo 4.

En la Figura 29 aparecen el número de actualizaciones realizadas por LRTA*(k) y LRTA*$_{LS}$(k) para diferentes valores de k en Cua-35 y Lab-acic en

la primera ejecución y convergencia con la versión 1 y 2. Podemos observar que LRTA*(k) hace más actualizaciones para todos los valores de k censados. Por razones de escala no se presenta en las gráficas el resultado para $k = \infty$ en la primera ejecución (excepto en Cua-35 con la versión 1). La diferencia en la cantidad de actualizaciones es resultado del punto débil número 2 de LRTA*(k). Un estado se puede actualizar varias veces en un episodio de planificación antes de obtener el valor final de la heurística, en cambio con LRTA*$_{LS}$(k) cada estado se actualiza sólo una vez por episodio de planificación. Como la diferencia en memoria utilizada por LRTA*$_{LS}$(k) y LRTA*(k) es pequeña (ver Tabla 1, 2, 4 y 5), concluimos que gran parte de la diferencia en actualizaciones entre los algoritmos es debida a que LRTA*(k) actualiza varias veces el mismo estado, mientras que LRTA*$_{LS}$(k) lo hace una vez. En Lab-acic la diferencia en la cantidad de actualizaciones entre LRTA*(k) y LRTA*$_{LS}$(k) es mayor que en Cua-35. Esto pasa porque la calidad de la heurística inicial en Lab-acic es mala y necesita más actualizaciones que Cua-35.

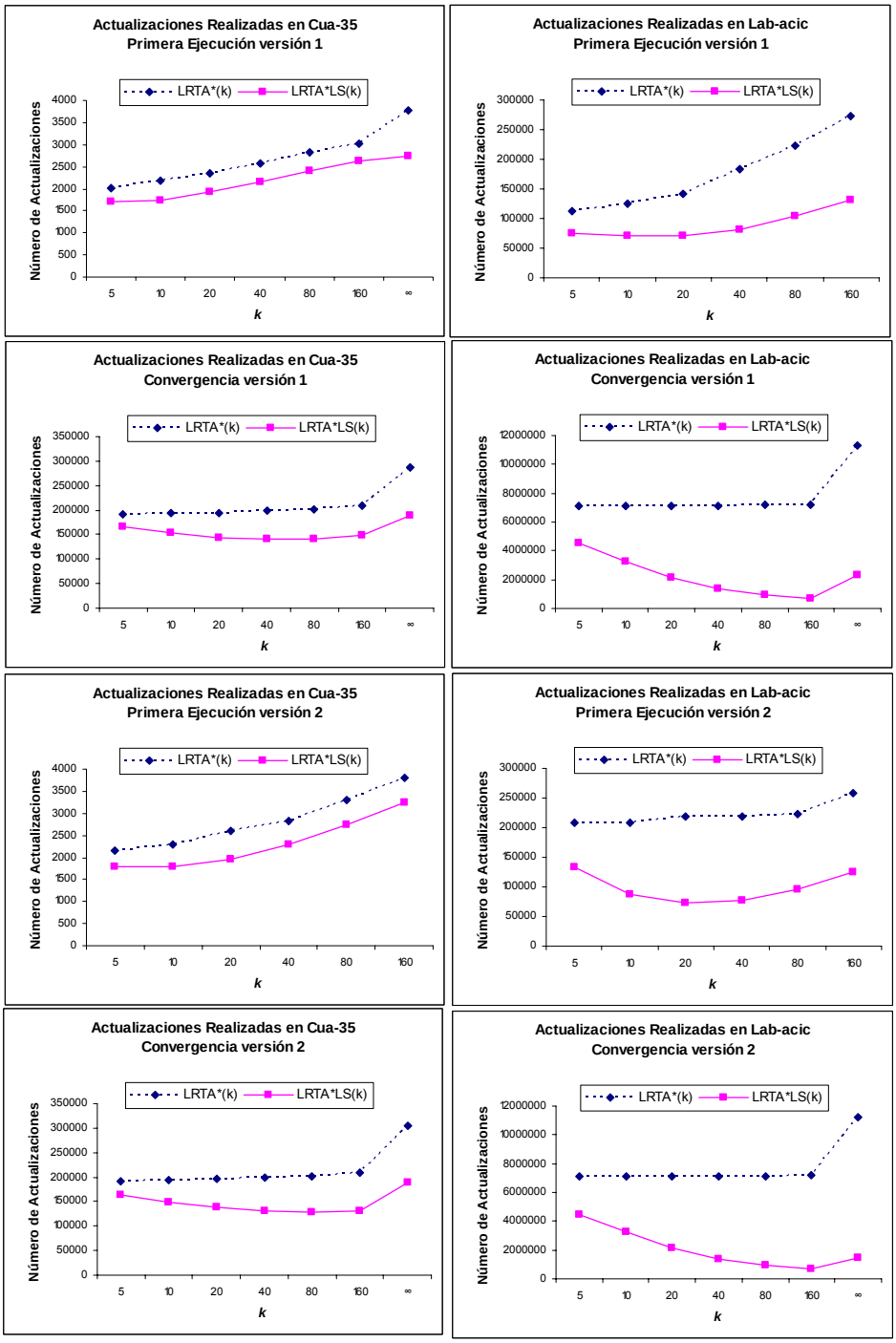


Figura 29. Resultados en Cua-35 y Lab-acic. Presentamos la cantidad de actualizaciones realizadas en la primera ejecución y convergencia con LRTA*(k) y LRTA*$_{LS}$(k) con la versión 1 y 2.

En la Figura 30 podemos ver las gráficas del coste de obtener una solución en Cua-35 y Lab-acic respecto a varios valores de k (eje horizontal), con la versión 1 y 2 de LRTA*(k) y LRTA*$_{LS}$(k) en la primera ejecución y

convergencia. El coste obtenido con LRTA*$_{LS}$(k) es menor que el obtenido con LRTA*(k) para todos los valores de k en las dos versiones (sólo con k = ∞ se obtiene una solución de la misma calidad). Esto nos indica que el mecanismo de propagación de LRTA*$_{LS}$(k) propuesto para mejorar los puntos débiles de LRTA*(k), efectúa un mejor uso de la k ya que se hacen menos actualizaciones y se mejora en coste para un valor de k fijo. Por ejemplo LRTA*$_{LS}$(k = 20) obtiene un coste similar a LRTA*(k = 40). En la primera ejecución la diferencia entre LRTA*(k) y LRTA*$_{LS}$(k) es más grande con la versión 2 en Cua-35 y con la versión 1 en Lab-acic. En el capitulo anterior vimos que una versión obtiene un coste menor que la otra en un escenario determinado. Con el nuevo mecanismo de propagación esta diferencia en coste se ve aumentada, porque se hace amplifica el efecto de la k y se mantiene las estrategias de actualización y movimiento.

El la Figura 31, podemos ver las gráficas del tiempo total de búsqueda en la primera ejecución y el tiempo en convergencia en Cua-35 y Lab-acic para varios valores de k, con la versión 1 y 2 de LRTA*(k) y LRTA*$_{LS}$(k). Podemos observar que LRTA*$_{LS}$(k) obtiene tiempo menor en la primera ejecución y convergencia que LRTA*(k) en Lab-acic en todos los valores de k, con ambas versiones. En Cua-35 el tiempo obtenido con LRTA*(k) es levemente menor en la primera ejecución y similar en convergencia que el obtenido con LRTA*$_{LS}$(k). La pequeña ventaja en tiempo de LRTA*(k) sobre LRTA*$_{LS}$(k) en Cua-35, se debe a que la mejora en coste y la disminución en el número de actualizaciones (el beneficio) no son tan sustanciales en este escenario. Además, el mecanismo de propagación de LRTA*$_{LS}$(k), es computacionalmente más costoso que el de LRTA*(k). Por tanto, la relación esfuerzo/beneficio en Cua-35 se ve reflejada en un leve aumento en el tiempo total de búsqueda de LRTA*(k) respecto a LRTA*$_{LS}$(k), en cambio en Lab-acic en una clara disminución. En convergencia la diferencia entre LRTA*(k) y LRTA*$_{LS}$(k) con la versión 1 y 2 son similares. En la primera ejecución hay diferencia. La diferencia es mayor con la versión 1 en la Lab-acic y con la

versión 2 en Cua-35. Este resultado es producto de las diferencias en coste que se aprecian en la Figura 30.

En la Figura 32 podemos apreciar el tiempo por paso en la primera ejecución y convergencia en Cua-35 y Lab-acic para a varios valores de *k* con la versión 1 y 2 de LRTA*(*k*) y LRTA*$_{LS}$(*k*). Para el mismo valor de *k* el tiempo por paso obtenido es similar en los dos escenarios con las dos versiones de los algoritmos en Lab-acic. En Cua-35 LRTA*(*k*) obtiene un tiempo levemente menor (excepto con k=∞ en convegencia en Cua-35). Las variaciones en el tiempo por paso se explican por las variaciones en el tiempo total o coste ya que es cociente de estos.

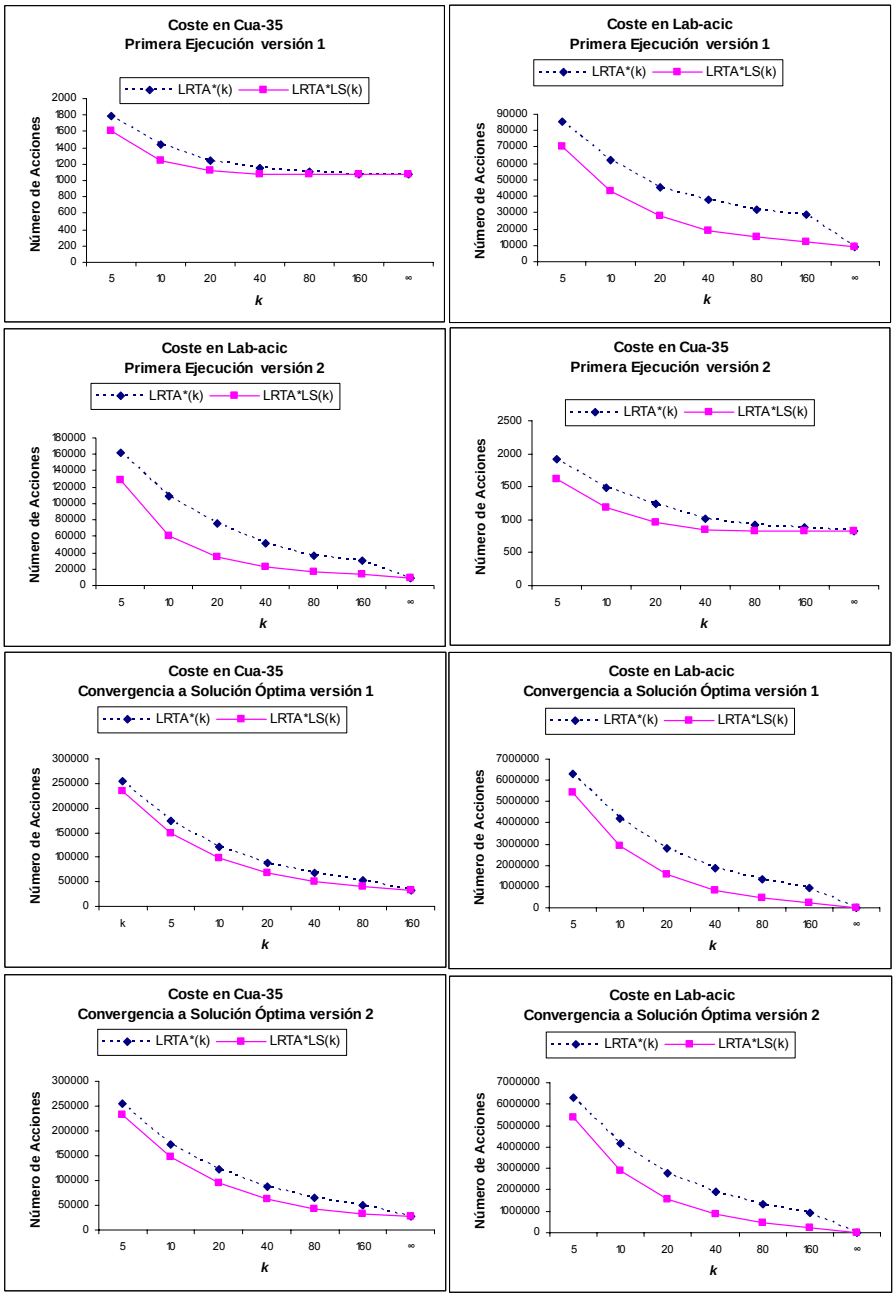


Figura 30. Resultados en Cua-35 y Lab-acic. Presentamos el coste de obtener una solución en la primera ejecución y convergencia con la versión 1 y 2 de LRTA*(*k*) y LRTA*$_{LS}$(*k*).

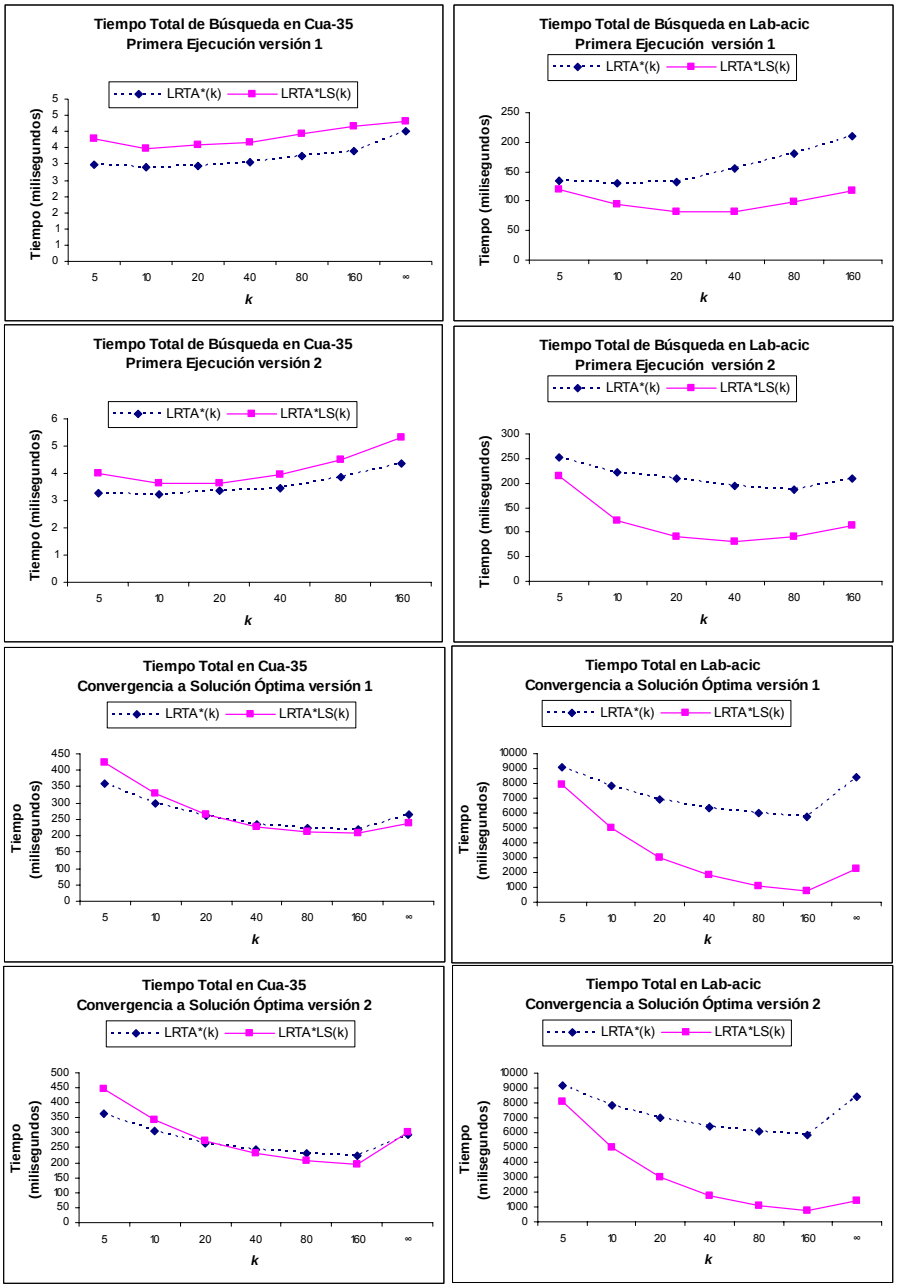


Figura 31. Resultados en Cua-35 y Lab-acic. Presentamos el tiempo total en la primera ejecución y convergencia con la versión 1 y 2 de LRTA*(*k*) y LRTA*$_{LS}$(*k*).

La mejora de LRTA*$_{LS}$(*k*) sobre LRTA*(*k*) es sustancial en Lab-acic. El Cua-35 la mejora es menos notoria. Esto se debe a la calidad de la heurística inicial y a la mejora en el mecanismo de actualización. LRTA*(*k*) itera para actualizar, cuando la heurística inicial es buena requiere pocas iteraciones por episodio de planificación. En este caso no hay mucha diferencia en la cantidad

de actualizaciones que se hacen con LRTA*$_{LS}$(k) (como en Cua-35 en la Figura 29). Por tanto, la diferencia en el desempeño de los algoritmos no es sustancial. En Lab-acic la heurística inicial es mala, por tanto la diferencia en el desempeño de los algoritmos es sustancial, ya que hay mucha diferencia en la cantidad de actualizaciones (ver Figura 29). Si se observan los resultados de las tablas en los capítulos 4 y 5, se puede ver que la mejora en rendimiento de LRTA*$_{LS}$(k) sobre LRTA*(k) en Lab-cic no es tan sustancial como en Lab-acic, pero mayor que la mejora en Cua-35, ya que la calidad de la heurística inicial en Lab-cic es mejor que en Lab-acic y peor que en Cua-35.

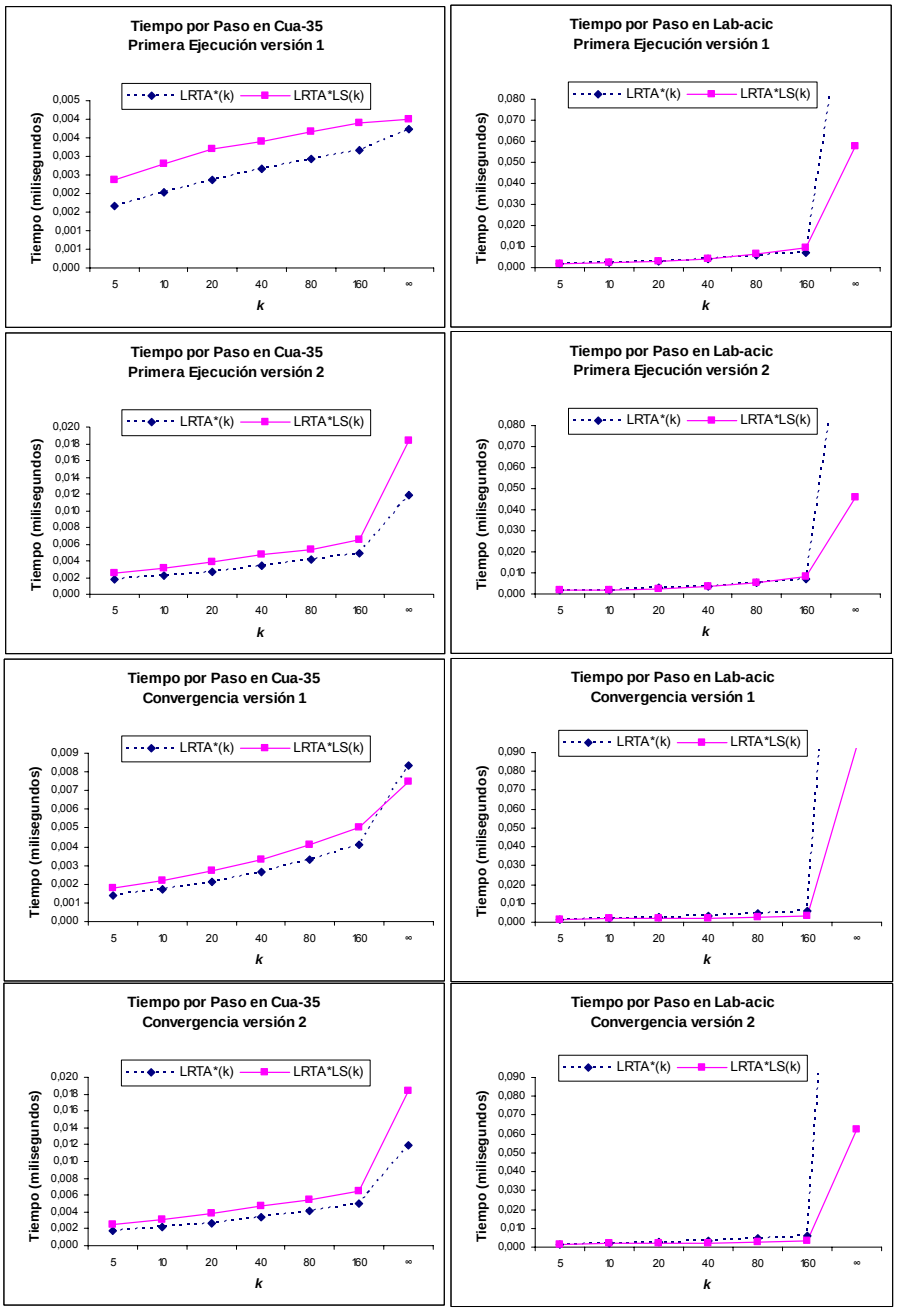


Figura 32. Resultados en Cua-35 y Lab-acic. Presentamos el tiempo por paso en la primera ejecución y convergencia con la versión 1 y 2 de LRTA*(k) y LRTA*$_{LS}$(k).

	Cua-35 versión 1				Cua-35 versión 2			
k	*T* %	*C* %	*M* %	*T/P* %	*T* %	*C* %	*M* %	*T/P* %
100%	5,3	4.124	499	0,0013	5,3	4.124	499	0,0013
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%
5	71%	39%	105%	183%	75%	39%	109%	191%
10	65%	30%	109%	216%	68%	28%	123%	237%
20	67%	27%	113%	246%	68%	23%	144%	294%
40	69%	26%	114%	263%	74%	20%	167%	362%
80	74%	26%	116%	282%	84%	20%	195%	417%
160	78%	26%	117%	300%	100%	20%	233%	497%
∞	80%	26%	116%	309%	285%	20%	660%	1422%
	Lab-acic versión 1				Lab-acic versión 2			
k	*T* %	*C* %	*M* %	*T/P* %	*T* %	*C* %	*M* %	*T/P* %
100%	425,5	391.940	5.370	0,0011	425,5	391.940	5370	0,0011
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%
5	28%	18%	94%	157%	50%	33%	108%	153%
10	22%	11%	101%	201%	29%	16%	116%	187%
20	19%	7%	105%	269%	21%	9%	122%	235%
40	19%	5%	104%	401%	19%	6%	126%	338%
80	23%	4%	104%	587%	22%	4%	128%	514%
160	28%	3,1%	105%	880%	26%	3,5%	128%	757%
∞	127%	2,4%	105%	5290%	99%	2,3%	135%	4226%
	Lab-cic versión 1				Lab-cic versión 2			
k	*T* %	*C* %	*M* %	*T/P* %	*T* %	*C* %	*M* %	*T/P* %
100%	133,7	117.995	2.795	0,0011	133,7	117.995	2.795	0,0011
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%
5	38%	22%	116%	172%	49%	30%	111%	161%
10	32%	14%	129%	230%	32%	16%	130%	209%
20	32%	10%	138%	315%	29%	10%	148%	278%
40	36%	8%	139%	445%	31%	8%	156%	397%
80	43%	7,1%	143%	611%	36%	6,4%	163%	564%
160	51%	6,3%	146%	813%	42%	5,7%	165%	742%
∞	91%	5,6%	145%	1620%	76%	5,1%	189%	1494%

Tabla 4. Resultados en la primera ejecución en Cua-35, Lab-acic y Lab-cic para la versión 1 y 2 de LRTA*$_{LS}$(*k*).

5.6.2 Primera Ejecución

La Tabla 4 contiene resultados para la primera ejecución. La relación versión 1/versión 2 es similar a la relación versión 1/versión 2 obtenida con LRTA*(*k*). A continuación analizamos cada una de las medidas de desempeño.

- Tiempo total de búsqueda: podemos observar que primero disminuye y luego aumenta a medida que *k* crece en las versiones 1 y 2. Además, es posible ver que en Cua-35, Lab-acic y Lab-cic una versión es mejor que la otra. La explicación de estos comportamientos es la misma que la dada en la sección 4.4.1. El esfuerzo lo podemos calcular multiplicando el número de estados extraídos[2] (*an*) de *Q* por el coste de procesar cada estado (primer while del procedimiento SelectionUpdateLS) más el número de actualizaciones realizadas (*ar*)

[2] Para continuar con la misma nomenclatura del capitulo 4, llamaremos actualizaciones no realizadas a las extracciones.

por el coste de actualizar cada estado (segundo while del procedimiento SelectionUpdateLS). En la Figura 33 podemos ver el esfuerzo con la versión 2 en Lab-cic es mayor que con la versión 1 con valores pequeños de k (5 y 10) porque la versión 2 invierte mucho esfuerzo en las actualizaciones no realizadas. Con valores más grandes de k el esfuerzo se nivela. En Cua-35 el esfuerzo es similar en ambas versiones, sólo con $k \geq 80$ la versión 2 invierte más en actualizaciones no realizadas y realizadas. Por tanto, el esfuerzo invertido explica el tiempo de búsqueda obtenido por cada versión en los dos escenarios.

- Coste: la calidad de la solución mejora monótonamente con k. A medida que k crece, el coste disminuye en ambas versiones (al menos no aumenta, ver Cua-35 con $k \geq 80$). Tal como en LRTA*(k), actualizar más (usar un mayor k) implica una mejora en la calidad de la solución. En Cua-35 la versión 2 obtiene un coste menor (excepto con $k = 5$ en el coste es el mismo). En Lab-acic se obtiene un coste menor con la versión 1. En Lab-cic se obtiene mejores resultados con la versión 1 para valores de k pequeños y con la versión 2 para valores de k grandes.

- Memoria: el consumo es mayor a medida que k aumenta porque se actualiza una mayor cantidad de estados (sólo Lab-acic con $k = 5$ y la versión 1 el consumo de memoria disminuye levemente). En los tres escenarios de prueba la versión 2 consume más memoria que la versión 1 (excepto con $k = 5$ en Lab-cic). Al igual que el LRTA*(k), esto se explica por el alcance de propagación. La versión 2 propaga sobre cualquier estado y la versión 1 sólo sobre estados visitados.

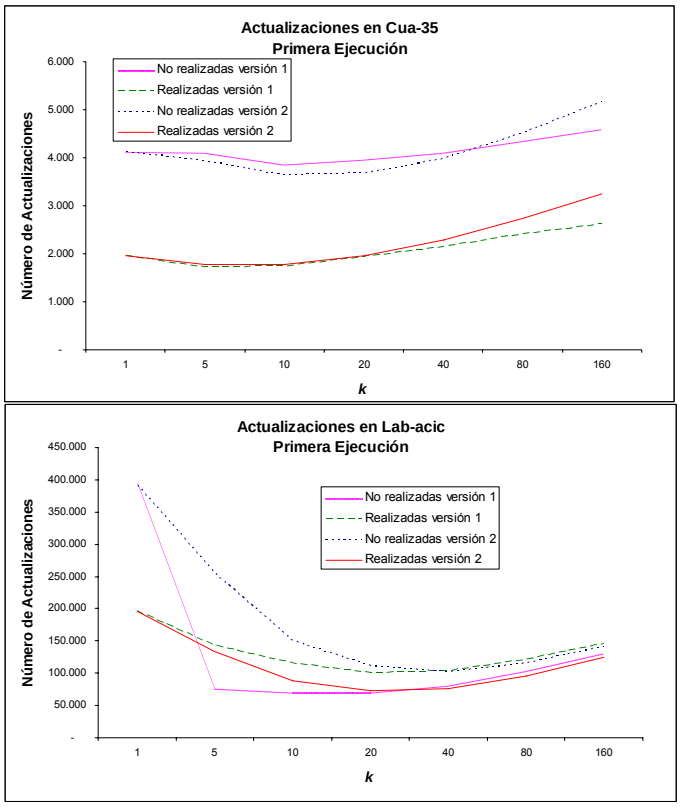


Figura 33. Resultados en Cua-35 y Lab-acic. Presentamos el numero de extracciones y actualizaciones realizadas en la primera ejecución con la versión 1 y 2 de LRTA*$_{LS}$(*k*).

- Tiempo por paso: podemos ver que aumenta a medida que *k* crece. Esto sucede porque se intentan hacer *k* actualizaciones en vez de una, en cada episodio de planificación. El tiempo por paso para un mismo *k*, es mayor en una de las dos versiones en cada escenario de prueba. Es mayor en la versión en que se hace un mayor esfuerzo por episodio de planificación, como en LRTA*(*k*). En la Figura 34 podemos ver que *an* y *ar* son mayores con la versión 1 en Lab-cic y con la versión 2 en Cua-35. Por tanto el esfuerzo por paso justifica el tiempo por paso obtenido con cada versión en Lab-cic y Cua-35.

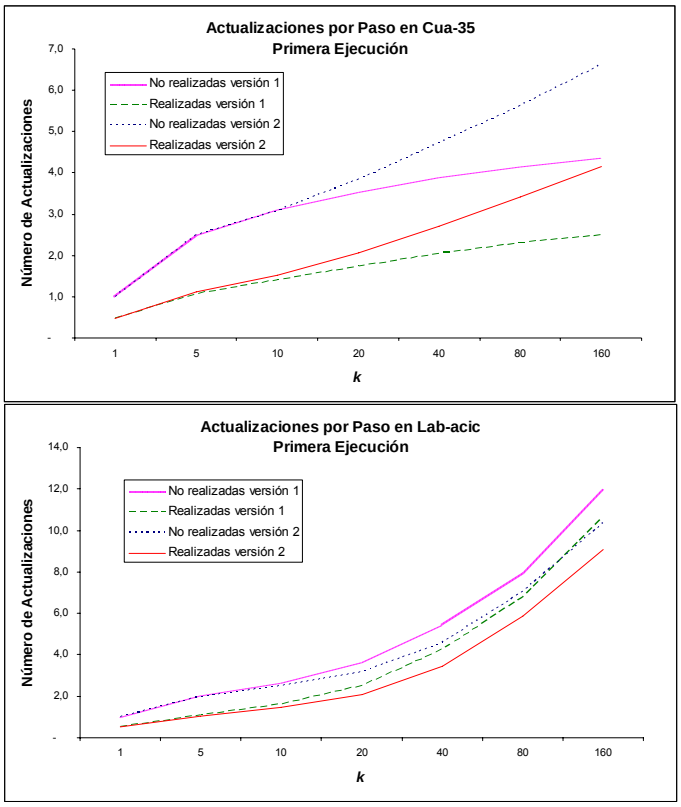


Figura 34. Resultados en Cua-35 y Lab-acic. Presentamos el numero de extracciones y actualizaciones realizadas por paso en la primera ejecución con la versión 1 y 2 de LRTA*$_{LS}(k)$.

Es importante notar que Cua-35 no conviene usar un $k > 80$, porque el coste no mejora y se obtienen peores resultados en tiempo total de búsqueda, tiempo por paso y memoria. Esto nos indica que el beneficio que se obtiene al aumentar k, esta limitado a cierta cantidad de actualizaciones. Usar un k superior a ese límite, es realizar trabajo que no mejora la calidad de la solución.

	Cua-35 versión 1					Cua-35 versión 2				
k	*T* %	*C* %	*E* %	*M* %	*T/P* %	*T* %	*C* %	*E* %	*M* %	*T/P* %
100%	687	577.072	656	12.962	0,0012	687	577.072	656	12.962	0,0012
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	62%	40%	56%	103%	152%	65%	40%	56%	103%	162%
10	48%	26%	37%	104%	184%	50%	25%	37%	109%	197%
20	39%	17%	24%	105%	226%	40%	16%	24%	116%	243%
40	33%	12%	16%	107%	280%	33%	11%	16%	122%	307%
80	31%	9%	11%	109%	347%	30%	8%	11%	128%	394%
160	30%	7,2%	7,6%	112%	420%	28%	5,7%	7,7%	136%	504%
∞	35%	5,5%	4,4%	115%	628%	44%	4,6%	4,8%	254%	955%
	Lab-acic versión 1					Lab-acic versión 2				
100%	17.483	16.614.587	1.054	9.835	0,0011	17.483	16.614.587	1.054	9.835	0,0011
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	46%	32%	42%	106%	140%	46%	32%	42%	105%	143%
10	28%	17%	24%	107%	163%	29%	17%	25%	111%	165%
20	17%	9%	14%	107%	180%	17%	9%	14%	115%	181%
40	10%	5%	8%	107%	203%	10%	5%	8%	118%	201%
80	6%	2,7%	4,2%	107%	238%	6%	2,7%	4,2%	119%	232%
160	4%	1,5%	2,4%	107%	303%	4%	1,4%	2,4%	120%	288%
∞	13%	0,1%	0,2%	107%	8892%	8%	0,1%	0,2%	123%	5941%
	Lab-cic versión 1					Lab-cic versión 2				
100%	5.633	5.084.859	536	10.328	0,0011	5.633	5.084.859	536	10.328	0,0011
1 (LRTA*)	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	49%	32%	46%	105%	151%	48%	32%	46%	106%	150%
10	31%	18%	28%	105%	177%	31%	18%	28%	112%	178%
20	19%	10%	16%	105%	200%	19%	10%	16%	115%	198%
40	13%	5%	9%	105%	236%	12%	5%	9%	117%	230%
80	9%	3,0%	6%	105%	300%	8%	3,0%	6%	118%	284%
160	7%	1,8%	3,5%	104%	416%	7%	1,7%	3,5%	119%	379%
∞	12%	0,4%	0,7%	102%	2656%	9%	0,4%	0,7%	120%	2108%

Tabla 5. Resultados en convergencia a soluciones óptimas en Cua-35, Lab-acic y Lab-cic para la versión 1 y 2 de LRTA*$_{LS}$(*k*).

5.6.3 Convergencia a soluciones óptimas

La Tabla 5 contiene resultados para convergencia a soluciones óptimas. La relación versión 1/versión 2, es similar a la relación versión 1/versión 2 obtenida con LRTA*(*k*). A continuación vemos las medidas de desempeño:

- Tiempo total de convergencia: en ambas versiones, el tiempo total de convergencia es similar. Podemos observar que este disminuye a medida que *k* crece en las dos versiones. Sólo con $k = \infty$ aumenta. Esto se debe a la relación esfuerzo/beneficio (como en LRTA*(*k*)).

- Coste: en ambas versiones el coste siempre disminuye a medida que *k* crece. Realizar una mayor cantidad de actualizaciones por episodio de planificación, tiene un efecto positivo en el proceso de convergencia. Los resultados que se obtienen en ambas versiones son similares, sólo en Cua-35 la versión 2 obtiene un coste algo menor para $k > 5$. A medida que se converge a la solución óptima, se va conociendo el

espacio de estados. Como en LRTA*(*k*) el efecto de la propagación en ambas versiones es similar en los 3 escenarios de prueba.

- Ejecuciones: el número de ejecuciones necesarias para converger a una solución óptima disminuye cuando *k* crece. La explicación de este efecto positivo de la propagación es la misma que la dada en la sección 4.4.2.

- Memoria: la memoria consumida aumenta cuando *k* crece. La versión 2 consume más memoria que la versión 1 (excepto con $k = 5$). La explicación de este comportamiento es la misma que en LRTA*(*k*).

- Tiempo por paso: aumenta a medida que *k* crece porque se hace una mayor cantidad de esfuerzo por episodio de planificación. En ambas versiones el tiempo promedio por paso a convergencia es similar en Lab-acic y Lab-cic (excepto con $k = \infty$). La explicación del comportamiento del tiempo paso se puede ver en la sección 5.6.2.

El rendimiento obtenido en convergencia, en términos de tiempo total de convergencia, coste, número de ejecuciones y tiempo por paso es similar en ambas versiones. Sólo en Cua-35 la versión 2 es un poco mejor en coste y un poco peor en tiempo por paso. Donde podemos ver una diferencia clara, es en la memoria consumida. La versión 2 consume más memoria que la versión 1 (excepto con $k = 5$).

5.6.4 Estabilidad del proceso de convergencia a soluciones óptimas

Los índices que miden la estabilidad del proceso de convergencia a soluciones óptimas se pueden ver en la Tabla 6. Menores valores en estos índices indican mejor calidad del proceso de convergencia. La tabla muestra los índices para la versión 1 (para la versión 2 el resultado es similar). Tal como en LRTA*(*k*), los índices mejoran monótonamente a medida que *k* crece, en los tres escenarios de prueba. Además, la mejora en los índices es mayor en el escenario con peor heurística inicial. Si se compara con los resultados de la

tabla 3 del capítulo 4, se puede observar que el valor de los índices que se obtienen con LRTA*$_{LS}$(*k*) son menores a los obtenidos con LRTA*(*k*), para todos los valores de *k* en los tres escenarios.

Cua-35					
k	IAE	ISE	ITAE	ITSE	SOD
5	1,1E+05	1,3E+08	3,1E+07	3,3E+10	6,5E+04
10	7,1E+04	7,0E+07	1,3E+07	1,2E+10	3,7E+04
20	4,8E+04	4,4E+07	5,2E+06	4,8E+09	2,4E+04
40	3,5E+04	3,5E+07	2,3E+06	2,3E+09	1,7E+04
80	2,9E+04	3,4E+07	1,2E+06	1,4E+09	1,4E+04
160	2,6E+04	3,8E+07	7,4E+05	1,1E+09	1,3E+04
Lab-acic					
k	IAE	ISE	ITAE	ITSE	SOD
5	4,1E+06	4,7E+11	4,8E+08	3,7E+13	2,5E+06
10	2,1E+06	1,6E+11	1,7E+08	8,8E+12	1,3E+06
20	1,2E+06	6,4E+10	5,6E+07	2,1E+12	7,0E+05
40	6,0E+05	2,6E+10	1,7E+07	5,0E+11	3,6E+05
80	3,2E+05	1,0E+10	5,2E+06	1,2E+11	1,8E+05
160	1,7E+05	4,2E+09	1,6E+06	3,0E+10	9,3E+04
Lab-cic					
k	IAE	ISE	ITAE	ITSE	SOD
5	1,4E+06	6,8E+10	1,0E+08	3,5E+12	8,1E+05
10	7,3E+05	2,5E+10	3,8E+07	8,9E+11	4,3E+05
20	4,0E+05	1,0E+10	1,3E+07	2,3E+11	2,3E+05
40	2,2E+05	4,5E+09	4,0E+06	6,0E+10	1,2E+05
80	1,2E+05	2,1E+09	1,3E+06	1,6E+10	6,3E+04
160	6,9E+04	9,9E+08	4,9E+05	5,1E+09	3,4E+04

Tabla 6. Índices de estabilidad del proceso de convergencia a soluciones óptimas en Cua-35, Lab-acic y Lab-cic para la versión 1 de LRTA*$_{LS}$(*k*).

5.6.5 Conclusiones

LRTA*$_{LS}$(*k*) mejora el coste obtenido con LRTA*(*k*) en todos los valores de *k* censados. LRTA*$_{LS}$(*k*) hace un mejor uso de la *k*. Por ejemplo, el coste que se obtiene con LRTA*$_{LS}$(*k* = 20) es similar al obtenido con LRTA*(*k*= 40). La mejora es sustancial en Lab-acic. El Cua-35 la mejora es menos notoria. El tiempo total y por paso en la primera ejecución y convergencia obtenido con LRTA*$_{LS}$(*k*) es similar o levemente peor al obtenido con LRTA*(*k*) en Cua-35 y mejor en Lab-acic en la mayoría de los valores de *k* censados. Observando estos resultados concluimos que LRTA*$_{LS}$(*k*) mejora el rendimiento obtenido con LRTA*(*k*) para un valor de *k*, porque el beneficio obtenido en coste es menor que el esfuerzo reflejado en tiempo.

Las dos versiones de LRTA*$_{LS}$(*k*) evaluadas muestran una mejora sustancial en rendimiento respecto a LRTA*, tanto en la primera ejecución, como en convergencia y estabilidad del proceso de convergencia. La mejora depende del parámetro *k*: a mayor *k* mejora la calidad de la solución (disminuye el coste obtenido) y en algunos casos aumenta el tiempo total de búsqueda o convergencia. Junto a la mejora en coste para un *k* determinado, se tiene un mayor tiempo de planificación por paso. Los resultados muestran que valores pequeños de *k* provocan importantes beneficios manteniendo un tiempo por paso razonable. La diferencia en rendimiento en las versiones de LRTA*$_{LS}$(*k*) se manifiesta principalmente en la primera ejecución, en convergencia el rendimiento es similar. En Cua-35 se obtienen mejores resultados con la versión 2, y en Lab-acic se obtiene mejores resultados con la versión 1. En Lab-cic con valores de *k* grandes se obtienen mejores resultados con la versión 2, y con valores pequeños de *k* con la versión 1. La relación versión 1/versión 2, es similar a la relación versión 1/versión 2 obtenida con LRTA*(*k*).

5.7 Resumen

En este capítulo se presentó una mejora al mecanismo de propagación acotada de valores heurísticos con iteración de valores, denominada propagación acotada sobre un espacio local. LRTA*(*k*) es un algoritmo que combina la propagación acotada con LRTA*. En el algoritmo LRTA*$_{LS}$(*k*), se implementa el método de propagación presentado. LRTA*$_{LS}$(*k*) se basa en LRTA*. Al comparar LRTA*$_{LS}$(*k*) con LRTA*(*k*), se puede apreciar que LRTA*$_{LS}$(*k*) mejora en el rendimiento obtenido con LRTA* (*k*) en la primera ejecución y en convergencia a soluciones óptimas en los escenarios de prueba usados. En los resultados experimentales se evalúan dos versiones de LRTA*$_{LS}$(*k*), presentadas en el capítulo 4. Los resultados muestran una mejora sustancial en el rendimiento respecto a LRTA* en la primera ejecución y en convergencia soluciones óptimas. La mejora depende del parámetro *k*, a mayor *k* mejores resultados con el coste de un mayor tiempo por episodio de planificación.

Capítulo seis

6 COMBINANDO ANTICIPACIÓN Y PROPAGACIÓN ACOTADA

El espacio local de búsqueda en los algoritmos LRTA*(k) y LRTA*$_{LS}$(k) esta constituido por el estado actual y sus sucesores inmediatos. En ambos algoritmos, la anticipación es de profundidad uno. Después de la anticipación, si el estado actual satisface la condición de actualización, se ejecuta la propagación acotada.

En varios algoritmos recientes de búsqueda heurística en tiempo real, la anticipación se controla por un parámetro que limita el tamaño del espacio local de búsqueda. En este capítulo exponemos los algoritmos existentes más relevantes y seleccionamos un mecanismo de anticipación para combinar con la propagación acotada. Además, se incluye una generalización del mecanismo de propagación acotada visto en el capítulo 5, que trabaja con el mecanismo de anticipación seleccionado. Después, discutimos sobre estrategias de movimiento del agente sobre el espacio local de búsqueda. Luego, presentamos una aproximación dinámica para el mecanismo de anticipación seleccionado, que además del parámetro considera la calidad de la heurística para limitar el tamaño del espacio local de búsqueda. Finalmente, introducimos el algoritmo LRTA*$_{LS}$(k, d), que combina anticipación y propagación acotada. En este algoritmo se implementan y evalúan experimentalmente las estrategias discutidas en las diferentes secciones y se compara su rendimiento con LRTA*(Koenig).

6.1 Mecanismos de anticipación

La anticipación en búsqueda heurística en tiempo real se puede implementar por diversos mecanismos, a continuación se describen los más relevantes.

6.1.1 Anticipación mediante búsqueda en anchura

Las dos estrategias de búsqueda heurística en tiempo real que se explican a continuación, realizan una anticipación implementada con una búsqueda en anchura a partir del estado actual. La profundidad de la anticipación esta limitada por un parámetro *d*. La diferencia entre ellas esta en el mecanismo de actualización de heurísticas.

- La estrategia Minimin [Korf, 1990]. En un estado *n* se usa la heurística $f(n) = g(n) + h(n)$, en donde $g(n)$ es la distancia desde el estado actual a *n*. En cada episodio de planificación, el agente realiza una búsqueda en anchura a profundidad *d*, siendo la raíz el estado actual. Si se expande el estado objetivo, el algoritmo detiene la anticipación. La estrategia Minimin propaga de las hojas a la raíz el mínimo valor de *f* encontrado en los sucesores de un nodo.

- El algoritmo Learning Real Time Search (LRTS) usa la estrategia Maxmin [Bulitko y Lee, 2006]. LRTS usa la heurística $f(n) = g(n) + h(n)$ al igual que Minimin. En cada episodio de planificación el agente realiza una búsqueda en anchura de profundidad *d*, siendo la raíz el estado actual. Calcula el valor mínimo de *f* a profundidad *t*, $\forall\ t \in [1, d]$. Si se expande el estado objetivo, el algoritmo detiene la anticipación a la profundidad en la que se encuentre. Sino, actualiza la raíz con el valor máximo entre los mínimos de cada profundidad (por esto se denomina Maxmin).

Es relevante mencionar que ambas estrategias mantienen la admisibilidad de la heurística. Esto es esencial para garantizar convergencia a soluciones óptimas.

*6.1.2 Anticipación mediante A**

En los trabajos recientes [Koenig, 2004] y [Koenig y Likhachev, 2006], se presentan los algoritmos LRTA*(Koenig) (una versión de LRTA*) y RTAA*

respectivamente. Ambos algoritmos usan una versión acotada de A* para realizar anticipación. A* se ejecuta desde el estado actual. La anticipación esta limitada por el número de estados que entran en la lista de CERRADOS. Pueden entrar hasta *d* (este parámetro no esta relacionado con el parámetro *d* que limita la profundidad en la sección 6.1.1) estados. La diferencia entre los algoritmos esta en el mecanismo de actualización de las heurísticas. En LRTA*(Koenig), se usa el algoritmo de caminos mínimos de Dijkstra, para actualizar la heurística de cada uno de los estados en CERRADOS. En RTAA* se actualiza la heurística de cada uno de los estados en CERRADOS con $g(b) + h(b)$ (b es el mejor estado en ABIERTOS después de insertar d estados en CERRADOS) menos el valor de g del estado en cerrados. La heurística que se obtiene después de actualizar con LRTA*(Koenig), es más informada que la que se obtiene con RTAA*, pero el mecanismo de actualización en RTAA* es más rápido y sencillo de implementar.

6.1.3 Selección del mecanismo de anticipación

Cuando se anticipa con A*, el tamaño del espacio local de búsqueda esta limitado por el número de estados que entran en CERRADOS. Esto permite un control fino sobre el tamaño del espacio local de búsqueda y por tanto, del tiempo de planificación dedicado a la anticipación. Cuando se anticipa con una búsqueda en anchura, el tamaño del espacio local de búsqueda esta acotado por la profundidad del árbol de búsqueda. El tamaño del espacio depende de la profundidad y del factor de ramificación. Debido a que no considera el factor de ramificación, este mecanismo no permite un control fino sobre el tamaño del espacio local. Por tanto, el mecanismo de anticipación que seleccionamos es A*, porque permite un control fino del tamaño del espacio local de búsqueda y es uno de los algoritmos más efectivos y usados, en sus diferentes versiones, en búsqueda de rutas [Ferguson et al. 2005].

6.2 Generalización de la propagación acotada

Si la heurística de un estado satisface la condición de actualización, entonces diremos que la heurística de ese estado es *inexacta*. La propagación acotada se inicia desde un estado con heurística inexacta. La detección de heurísticas inexactas esta relacionada con el tamaño del espacio local de búsqueda. Los algoritmos que se han presentado en los capítulos anteriores usan anticipación de profundidad uno, esto les permite detectar inexactitudes sólo en el estado actual. Por ello los mecanismos de propagación acotada vistos hasta ahora se inician a partir de un solo estado. En esta sección presentamos un mecanismo de propagación que trabaja con uno o varios estados con heurística inexacta. La motivación de este mecanismo es el aumento de la anticipación. Cuando se aumenta la anticipación se obtiene un espacio local de búsqueda de mayor tamaño, en el que es posible encontrar varios estados con heurística inexacta. A continuación se explica el proceso de detección de inexactitudes heurísticas y el nuevo mecanismo de propagación acotada.

6.2.1 Identificando heurísticas inexactas en la anticipación

La anticipación en LRTA*$_{LS}$(*k*) es de profundidad uno, por tanto el algoritmo sólo puede identificar una inexactitud heurística en el estado actual. Una vez identificada la inexactitud, LRTA*$_{LS}$(*k*) inicia la propagación acotada.

Podemos ver un ejemplo de LRTA*$_{LS}$(*k*) en la Figura 35. En (i), vemos una cuadrícula 4-conectada. Los números en las celdas son estimaciones heurísticas. El coste de las acciones es uno. El estado objetivo es *b2* y el estado actual es *a1*. Las celdas con recuadro remarcado conforman el espacio local de búsqueda (el estado actual *a1* y sus sucesores *a2* y *b1*). En (ii), vemos con fondo gris la celda que satisface la condición de actualización (el estado actual *a1*). Las celdas con fondo gris en (iii) conforman el espacio local de aprendizaje: el conjunto de estados interiores *I*:{*a1*, *b1*}, y en un gris suave el conjunto de estados en la frontera *F*:{*a2*, *b2*, *c1*}. En (iv) se puede ver la heurística de los estados en interiores, después de actualizar.

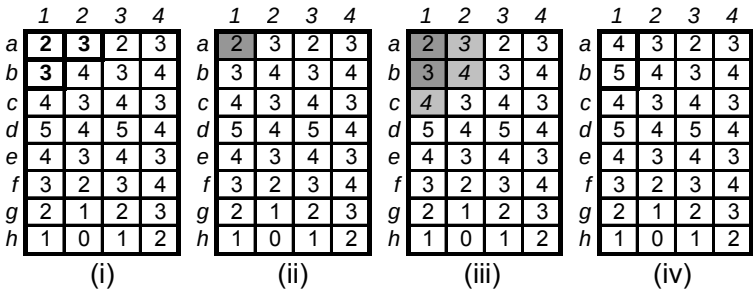


Figura 35. Ejemplo de propagación acotada con LRTA*$_{LS}(k)$

En el ejemplo de la Figura 35, después de identificar la inexactitud de la heurística del estado actual, se actualiza la heurística de 2 estados. Aunque se use un $k > 2$, por ejemplo $k = 9$, el número de actualizaciones será el mismo. En (i), podemos observar que existen otros estados con heurística inexacta (como *a3* y *c3*), que LRTA*$_{LS}(k)$ no detecta debido al alcance de su anticipación. Si se usara una anticipación de mayor profundidad, se podría identificar un mayor número de estados con heurística inexacta. Una vez identificados los estados, sería posible hacer un mayor número de actualizaciones.

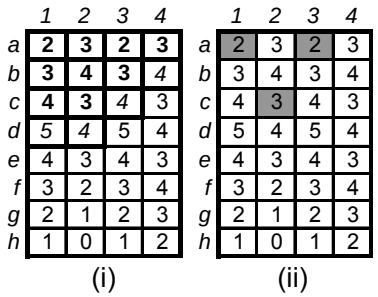


Figura 36. Ejemplo de identificación de estados con heurística inexacta cuando se anticipa con A*.

La identificación de los estados con heurística inexacta se hace durante la anticipación. Con A* se identifican entre los *d* estados que entran en CERRADOS y en el mejor de ABIERTOS (después de meter *d* estados en CERRADOS). La identificación se hace cuando se expande un estado que se inserta en CERRADOS y generando los sucesores del mejor estado en ABIERTOS. Por ejemplo, si usamos A* con |CERRADOS| = 9 en la Figura 35 (i), tenemos el espacio local de búsqueda que se aprecia en la Figura

36 (i). Las celdas con borde remarcado y números en negrita son los estados en CERRADOS y las celdas con borde remarcado y números en cursiva son los estados en ABIERTOS. La anticipación permite identificar una mayor cantidad de estados que satisfacen la condición de actualización: el estado actual *a1*, el estado *a3* y el estado *c2*, que pertenecen a CERRADOS como se ve en la Figura 36 (ii). El mejor estado de ABIERTOS no satisface la condición de actualización. Debido a que las inexactitudes se identifican mientras se genera el árbol de búsqueda, primero se identifica *a1*, luego *a3* y finalmente *c2*. A medida que se identifican los estados con heurística inexacta, se insertan en la lista *C*. Los estados de la lista *C* se ordenan por su distancia al estado actual. Los estados más cercanos al estado actual, se ubican al comienzo de la lista y los más lejanos al final. La razón de este orden es priorizar la actualización de la heurística del estado actual y los estados cercanos a este, porque según nuestra experiencia, influye positivamente en el rendimiento de la búsqueda.

Después de la identificación, es necesario seleccionar el espacio local de aprendizaje y actualizar. Los procedimientos de selección y actualización descritos en las secciones 5.1 y 5.2, están diseñados para trabajar a partir de un estado (el actual). A continuación se describe como se selecciona y actualiza un espacio local (*I*, *F*), a partir de varios estados con heurística inexacta.

6.2.2 Actualización de heurísticas

Los procedimientos de selección y actualización de un espacio local de aprendizaje a partir de varios estados, son una adaptación de los procedimientos descritos en las secciones 5.1 y 5.2. Primero describiremos el proceso de selección y luego el de actualización.

El espacio local de aprendizaje *I* se calcula usando la lista *C*. Debido a que la selección del espacio local se inicia desde estados que pueden estar alejados entre si, *I* puede ser no-conexo. El proceso es el siguiente:

4. Inicializar $I \leftarrow \varnothing$,
5. Extraer el estado x desde la primera posición de la lista C.
6. Inicializar $Q \leftarrow \{x\}$.
7. Realizar el siguiente ciclo hasta que Q este vacío o $|I|$ sea igual a k.
   7.1. Extraer el estado v desde la primera posición de Q.
   7.2. Si v es distinto a un estado objetivo, entonces:
      7.2.1. Obtener los sucesores de v que no están en I.
      7.2.2. Si $h(v)$ tiene oportunidad de cambiar (esto es, si $h(v) < \min_{w \in \{Succ(v) - I\}} h(w) + c(v, w)$), entonces, incluir v en I, e incluir los sucesores de v que no están en I en Q.
8. Si Q esta vacío y C es distinto de vacío y $|I|$ es menor que k, ir a 2.
9. El conjunto F es el que rodea completa e inmediatamente a I (I puede ser no-conexo).

La diferencia principal con el procedimiento descrito en la sección 5.1 es el uso de la lista C. En vez de inicializar Q con el estado actual se inicializa con el estado en la primera posición en C (línea 3). El ciclo en la línea 4 es igual al ciclo en la línea 2 del proceso en la sección 5.1. Una vez terminando el ciclo, si Q esta vacío y C contiene estados y el número de estados en interiores es menor que k, entonces se continua construyendo el espacio local de aprendizaje inicializando Q con el siguiente estado en C (línea 5).

Por ejemplo, en la Figura 36 (ii) C esta compuesto por {*a1*, *a3*, *c2*}. Si usamos un $k = 9$ y aplicamos el proceso de selección del espacio local de aprendizaje descrito, obtenemos lo que nos muestra la Figura 37.

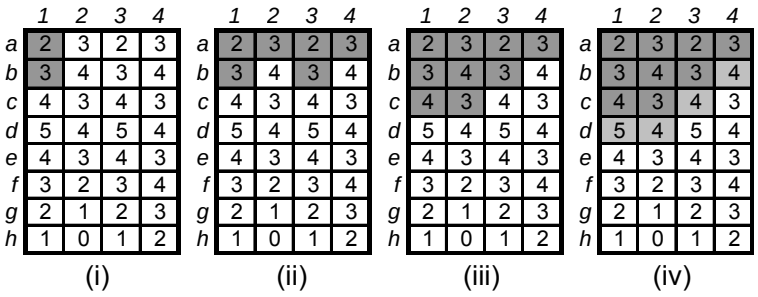


Figura 37. Ejemplo de selección del espacio local de aprendizaje a partir de varios estados con heurística inexacta.

En la Figura 37 (i) podemos ver los estados interiores (celdas en gris) que se seleccionan a partir del estado *a1*. En (ii) los estados seleccionados a partir de *a3* más los que ya estaban en interiores y en (iii) los estados seleccionados a partir de *c2* más los que ya estaban en interiores. En (iv), podemos observar el espacio local de aprendizaje completo: los estados en interiores *I*:{*a1*, *a2*, *a3*, *a4*, *b1*, *b2*, *b3*, *c1*, *c2*} (en gris) y los estados en la frontera *F*:{*b4*, *c3*, *d1*, *d2*} (en gris suave).

La actualización del espacio local de aprendizaje se hace con el mismo proceso descrito en la sección 5.4. El proceso en esta sección, requiere que el conjunto de estados frontera rodee completa e inmediatamente a los estados en interiores. Esto se cumple con el nuevo procedimiento, por tanto es correcto aplicar el proceso. Continuando con el ejemplo, en la Figura 38 podemos ver las heurísticas después de actualizar.

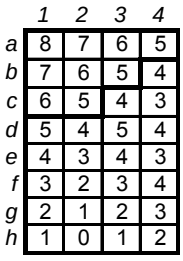


Figura 38. Ejemplo de actualización de la heurística con propagación acotada.

Al usar una mayor anticipación se pueden detectar varios estados con heurística inexacta dentro del espacio local de búsqueda. Esto permite realizar

un mayor número de actualizaciones. Después de cada episodio de planificación es posible obtener una heurística más informada, como se aprecia en el ejemplo descrito en esta sección. Algo importante es que al realizar una anticipación de mayor profundidad, se consume un mayor tiempo en computación. En la sección de resultados experimentales se discutirá sobre el beneficio y coste de combinar anticipación con el nuevo mecanismo de propagación acotada.

6.3 Estrategias de movimientos en el espacio local de búsqueda

Después de planificar, el agente se mueve desde el estado actual hacia un estado dentro del espacio local de búsqueda. Se distinguen dos estrategias de planificación de movimientos en búsqueda heurística en tiempo real [Hernández y Meseguer 2007b]: planificar un movimiento desde el estado actual hacia el mejor de los sucesores inmediatos, como se hace en LRTA*, LRTA*(k) y LRTA*$_{LS}$(k), y planificar varios movimientos desde el estado actual hacia el mejor estado en la frontera del espacio local de búsqueda, como se hace en LRTS [Bulitko y Lee, 2006], LRTA*(Koenig) [Koenig, 2004] y RTAA* [Koenig y Likhachev, 2006]. A continuación contrastamos las dos estrategias movimiento y presentamos una estrategia intermedia que combina ambas.

6.3.1 Un movimiento versus varios movimientos

Con la misma cantidad de anticipación se puede planificar uno o varios movimientos por episodio de planificación. Existe cierto debate sobre cual es la opción más adecuada. Típicamente planificar un movimiento produce trayectorias de mejor calidad (de menor coste). Sin embargo, el tiempo de CPU usado en la planificación de un movimiento generalmente es mayor que el usado por la otra aproximación, porque todo el esfuerzo dedicado a la anticipación produce un solo movimiento. Por esto, planificar varios movimientos es una opción atractiva que ha sido investigada desde diferentes enfoques [Koenig, 2004], [Bulitko & Lee, 2006].

Planificar un movimiento por episodio de planificación es una estrategia conservadora. El agente puede calcular la mejor trayectoria (varios movimientos) hacia un estado en frontera del espacio local de búsqueda, pero desde una perspectiva global no esta seguro si esta trayectoria efectivamente lo acerca al estado objetivo, o lo lleva por una ruta errónea hacia un obstáculo (la ruta errónea se conoce después de algunos movimientos, a medida que el entorno es descubierto). En esta situación puede ser adecuada una estrategia de mínimo compromiso y planificar una sola acción: moverse desde el estado actual hacia el mejor de los sucesores inmediatos.

Planificar varios movimientos por episodio de planificación es una estrategia arriesgada. Dado que el espacio local de búsqueda es pequeño respecto a todo el espacio de estados, no está claro si la mejor trayectoria dentro del espacio local también es buena a nivel global. Seguir la mejor trayectoria en el espacio local es arriesgado porque (i) puede no ser buena a nivel global, y (ii) si finalmente es errónea, debido a que incluye varios movimientos, se requerirá cierto esfuerzo en volver atrás. Por otro lado, si la trayectoria es buena, realizar varios movimientos por episodio de planificación es mejor que realizar un movimiento, porque el agente se aproxima más al estado objetivo.

6.3.2 Movimientos y calidad de la heurística: estrategia dinámica

Una estrategia intermedia para planificar movimientos es la que presentamos en [Hernández y Meseguer 2007b]. La estrategia toma en cuenta la calidad de la heurística en los estados del espacio local de búsqueda. Si el agente descubre cualquier indicio que revele que la calidad de la heurística en el espacio local de búsqueda no es perfecta (es decir, que algún estado tiene heurística inexacta), entonces sólo planifica un movimiento. Sino, planifica varios movimientos. La estrategia es la siguiente (la anticipación se hace con A*). El agente planifica un movimiento si: (i) encuentra algún estado con heurística inexacta en CERRADOS o (ii) el mejor estado en ABIERTOS tiene heurística inexacta (después de completar el espacio local de búsqueda, es decir los *d* estados en CERRADOS). De lo contrario, el agente planifica

varios movimientos. Si el mejor estado en ABIERTOS tiene heurística inexacta, se inserta en *C* para considerarlo en la propagación. Estrategia permite identificar una heurística inexacta en el mejor estado de la frontera, lo que evita que el agente se mueva hacia un mínimo local.

En la sección de resultados experimentales, se hace una evaluación de las estrategias de movimiento.

6.4 Aproximación dinámica de la anticipación

El mecanismo que presentamos en [Hernández y Meseguer 2007b] toma en cuenta la calidad de la heurística de los estados mientras se realiza la anticipación, de manera similar a la estrategia de movimientos dinámica. Si se descubre cualquier indicio que revele que la calidad de la heurística en el espacio local de búsqueda no es perfecta (es decir, que algún estado satisface la condición de actualización) mientras se realiza la anticipación, entonces no confiamos en la heurística como guía de la búsqueda y se detiene la anticipación. De lo contrario, la anticipación continúa hasta que llegue a la cota establecida por el parámetro. En este esquema, el espacio local de búsqueda en cada episodio de planificación no tiene un tamaño fijo. El tamaño del espacio local de búsqueda depende de la calidad de la heurística.

La aproximación dinámica usa A* como mecanismo de anticipación. Cada vez que se generan los sucesores de un estado durante la anticipación, se verifica si tiene una heurística inexacta. La anticipación se detiene cuando hay *d* estados en CERRADOS o se expande un estado con heurística inexacta. Cuando se detecta un estado con heurística inexacta, se inserta en *C* para considerarlo en la propagación acotada. En la sección de resultados experimentales, se evalúa el mecanismo de anticipación con A* presentado en la sección 6.1 y la aproximación dinámica.

6.5 LRTA*$_{LS}$(*k*, *d*)

LRTA*$_{LS}$(*k*, *d*) es un algoritmo basado en LRTA*, que combina anticipación con propagación acotada. El parámetro *k* se usa para limitar la propagación y

el parámetro *d* para limitar la anticipación. La propagación acotada se hace con el procedimiento expuesto en la sección 6.2. La anticipación tiene dos versiones. Primero, la aproximación estática: se anticipa con A* hasta meter *d* estados en CERRADOS. Segundo, la aproximación dinámica: se anticipa con A* hasta expandir un estado con heurística inexacta o hasta meter *d* estados en CERRADOS. En el algoritmo se pueden implementar las distintas estrategias de movimiento presentadas. Primero presentamos LRTA*$_{LS}$(*k*, *d*) con las estrategias de varios movimientos, un movimiento y movimiento dinámico, anticipando con la aproximación estática. Después presentamos el algoritmo con las tres estrategias de movimiento anticipando con la aproximación dinámica.

6.5.1 Aproximación estática de la anticipación

La versión del algoritmo LRTA*$_{LS}$(*k*, *d*) que anticipa con la aproximación estática y planifica varios movimientos aparece en la Figura 40. La planificación de varios movimientos no siempre es al mejor de los estados en la frontera como en los algoritmos mencionados en la sección 6.3. En la Figura 39 podemos ver dos elipses que representan el espacio local de búsqueda y el espacio local de aprendizaje, respectivamente. El espacio local de búsqueda se generó desde el estado actual y el de aprendizaje desde un estado con heurística inexacta, como se puede apreciar en la figura. En este caso puede suceder que el mejor estado de ABIERTOS este en *I* y haya cambiado su estimación heurística. Si cambió, ya no es muy conveniente usarlo para el cálculo de varios movimientos. Definimos ABIERTOS2 como (ABIERTOS – *I*) unido a (*F*-(ABIERTOS + CERRADOS)). En LRTA*$_{LS}$(*k*, *d*) el cálculo de varios movimientos se hace hacia el mejor estado de ABIERTOS2 que no tenga heurística inexacta. Si todos los estados en ABIERTOS2 tienen heurística inexacta, se calcula un movimiento hacia el mejor estado de los sucesores inmediatos del estado actual.

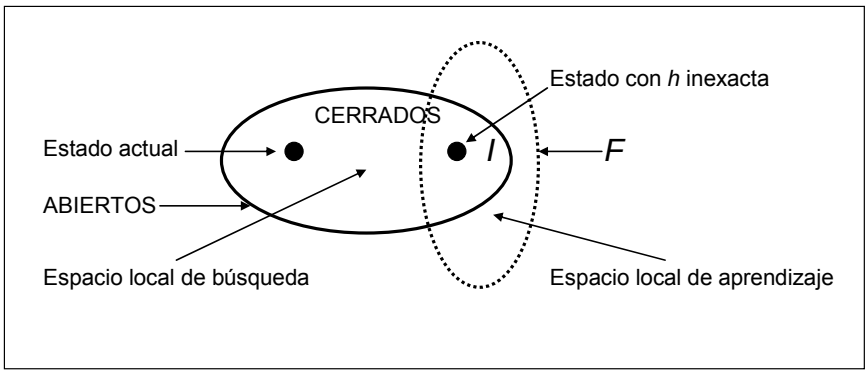


Figura 39. Ejemplo usado para explicar el cálculo de varios movimientos con LRTA*$_{LS}$(*k*, *d*).

El procedimiento LRTA-LS(k,d) inicializa la heurística de cada estado en el espacio de estados (línea 1) y ejecuta el procedimiento LRTA-LS-Trial, hasta que se produzca la convergencia (cuando *h* no cambia) (líneas 2, 3 y 4). El procedimiento LRTA-LS-Trial inicializa el estado actual *x* con el estado inicial s_0 (línea 1). Después se realiza el siguiente ciclo hasta que se encuentra el estado objetivo. Primero, se ejecuta A* desde el estado actual. Se anticipa con A* hasta que el número de estados en CERRADOS es *d*, o hasta que se encuentra un estado objetivo. A* retorna ABIERTOS, CERRADOS y *C* la secuencia de estados con heurística inexacta, ordenados por su cercanía al estado actual. El estado más cercano al estado actual esta primero en *C* y el estado más lejano esta al final de *C* (línea 3). Es importante recordar que cada vez que A* genera los sucesores de un estado, se verifica si este tiene heurística inexacta. Después se llama a la función SelectionLS y al procedimiento UpdateLS, siempre y cuando *C* contenga estados (línea 4). La función SelectionLS selecciona el espacio local a partir de los estados en *C* y retorna (*I*, *F*) (línea 5). El procedimiento UpdateLS realiza la actualización del espacio local de aprendizaje (línea 6). El procedimiento PATH retorna una secuencia de estados *path* que comienza con el estado actual *x* y termina con el mejor estado en ABIERTOS2 que no tiene heurística inexacta o el mejor estado entre los sucesores inmediatos o un estado objetivo (línea 7). Una vez realizada la actualización de heurísticas, se ejecutan los movimientos

a través de la secuencia de estados en *path* (líneas 8-12) considerando la suposición de espacio libre.

La función SelectionLS selecciona el espacio local. Para seleccionar el espacio local, se mantiene una cola Q de estados candidatos a ser incluidos en I o F. Además, se usa la secuencia de estados C. Q, F e I se inicializan con vacío (línea 1). Como máximo pueden entrar k estados en I. Esto es controlado por el contador *cont* que se inicializa con cero (línea 1). Se ejecuta el siguiente ciclo hasta que C no contenga estados (línea 2). Se extrae p el primer estado en C (línea 3). Si p no es nulo y p no pertenece a I y Q es vacío y *cont* es menor que k, entonces se inserta p en Q (línea 4). Esta línea es importante porque controla la inserción de un estado de C en Q. Debido a que la función SelectionLS se llama cuando C tiene por lo menos un elemento, cuando se extrae el primer estado en C todas las condiciones son verdaderas y p se inserta en Q. Después se ejecuta el siguiente ciclo hasta que Q no contenga estados o *cont* sea igual a k (línea 5). Se extrae el primer estado v de Q (línea 6). Se calcula el estado $y \leftarrow \text{argmin}_{w \in Succ(v),\, w \notin I}\, h(w) + c(v, w)$ (línea 7). Si $h(v)$ satisface la condición de actualización (línea 8), entonces v entra en I (línea 9), se incrementa el contador (línea 10) y los sucesores de v que no pertenecen a I o Q se insertan al final de Q (líneas 11-12). En otro caso v se inserta en F (línea 13). Cuando termina este ciclo (líneas 5-13), si aún hay espacio en interiores (*cont* < k) y C contiene elementos el otro ciclo (línea 2) continua iterando. Cuando termina el ciclo (línea 2), si Q aún contiene estados estos se extrae de Q y se insertan en F (línea 14).

El procedimiento UpdateLS actualiza el espacio local de aprendizaje. Una vez que el espacio local ha sido seleccionado, sus estados interiores son actualizados, ejecutando el siguiente ciclo hasta que I no contenga estados: Se selecciona el par $(i, f) = \text{argmin}_{v \in I,\, w \in F,\, w \in Succ(v)}\, h(w) + c(v, w)$, se actualiza $h(i)$, i se extrae de I y se inserta en F (línea 1-5).

El algoritmo LRTA*$_{LS}$(*k*, *d*) es completo porque la heurística siempre se incrementa (Teorema 1 [Korf, 1990]). Para probar que converge a soluciones óptimas basta con probar que los valores heurísticos se mantienen admisibles después de la propagación acotada sobre un espacio local de búsqueda. Esto se garantiza con la **proposición 2** de la sección 5.4, porque el mecanismo de actualización es el mismo de LRTA*$_{LS}$(*k*).

A partir de este resultado se garantiza la completitud y convergencia a rutas óptimas de LRTA*$_{LS}$(*k*, *d*) en los mismos términos que en LRTA*. LRTA*$_{LS}$(*k*, *d*) hereda las buenas propiedades de LRTA*.

```
procedure LRTA-LS(k,d) (X, A, c, s0, G, k, d)
1  for each x ∈ X do h(x) ← h0(x);
2  repeat
3     LRTA-LS-trial(X, A, c, s0, G, k, d);
4  until h does not change;

procedure LRTA-LS-trial(X, A, c, s0, G, k, d)
1  x ← s0;
2  while x ∉ G do
3     (ABIERTOS, CERRADOS, C) ← A*(x, d, G);
4     if C ≠ ∅ then
5        (I,F) ← SelectLS(k, C);
6        UpdateLS(I,F);
7     path ← PATH(ABIERTOS, CERRADOS, F, I);
8     x ← extract-first (path);
9     while path ≠ ∅ do
10       y ← extract-first (path);
11       execute(a ∈ A such that a = (x, y));
12       x ← y;

function SelectLS (k, C): pair of sets;
1  Q ← ∅; F ← ∅; I ← ∅; cont ← 0;
2  while C ≠ ∅ do
3     p ← extract-first(C);
4     if p ∉ I ∧ Q = ∅ ∧ cont < k then Q ← {p};
5     while Q ≠ ∅ ∧ cont < k do
6        v ← extract-first(Q);
7        y ← argmin_{w∈Succ(v) ∧ w ∉ I} h(w) + c(v, w);
8        if h(v) < h(y) + c(v, y) then
9           I ←I ∪ {v};
10          cont ← cont + 1;
11          for each w ∈ Succ(v) do
12             if w ∉ I ∧ w ∉ Q then Q ← add-last(Q,w);
13       else F ←F ∪ {v};
14 if Q ≠ ∅ then F ←F ∪ Q;
15 return (I, F);


procedure UpdateLS (I, F)
1  while I ≠ ∅ do
2     (i, j) ← argmin_{v∈I, w∈F, w∈Succ(v)} h(w) + c(v, w);
3     h(i) ← max[h(i), c(i, j) + h(j)];
4     I ←I - {i};
5     F ←F ∪ {i};
```

Figura 40. Versión del algoritmo LRTA*$_{LS}(k, d)$ que anticipa con la aproximación estática y planifica varios movimientos.

```
procedure LRTA-LS-trial(X, A, c, s0, G, k, d)
1  x ← s0;
2  while x ∉ G do
3     (path, C) ← A*(x, d, G);
4     if C ≠ ∅ then
5        (I,F) ← SelectLS(k, C);
6        UpdateLS(I,F);
7        y ← argmin_{w∈Succ(x)} [h(w) + c(x,w)];
8        execute(a ∈ A such that a = (x, y));
9        x ← y;
10    else
11       x ← extract-first (path);
12       while path ≠ ∅ do
13          y ← extract-first (path);
14          execute(a ∈ A such that a = (x, y));
15          x ← y;
```

Figura 41. Versión del algoritmo LRTA*$_{LS}(k, d)$ que anticipa con la aproximación estática y planifica movimientos con la estrategia de movimiento dinámico. Modificación al algoritmo de la Figura 40.

Para implementar la versión que planifica un movimiento es necesario realizar algunos cambios menores a LRTA*$_{LS}(k, d)$ de la Figura 40. Los cambios se deben hacer en el procedimiento LRTA-LS-Trial. Los cambios son: Primero, no retornar ABIERTOS y CERRADOS cuando se ejecuta A*. Segundo, remplazar las líneas 7 a 12 que ejecutan varios movimientos, por el cálculo de un movimiento como se hace en LRTA*$_{LS}(k)$.

La estrategia de movimiento dinámico (uno o varios movimientos), planifica el movimiento dependiendo de la calidad de la heurística. Si *C* contiene algún estado, entonces se actualiza la heurística y se ejecuta un movimiento, de lo contrario se ejecutan varios movimientos. Para implementar esta versión, hay que realizar cambios en el procedimiento LRTA-LS-Trial. Los cambios los podemos ver el la Figura 41. A* retorna una secuencia de estados *path* que comienza con el estado actual *x* y termina con el mejor estado en ABIERTOS o un estado objetivo, además tiene que verificar si el mejor estado en ABIERTOS tiene heurística inexacta, e incluirlo en *C* si esto sucede (línea 3). Si se encuentra algún estado con heurística inexacta (línea 4) en CERRADOS

o en el mejor de la frontera (ABIERTOS), entonces se ejecuta la propagación acotada, se planifica un movimiento y se ejecuta (líneas 5-9). De lo contrario se ejecutan varios movimientos (líneas 11-15).

6.5.2 Aproximación dinámica de la anticipación

En la aproximación dinámica, la anticipación con A* se debe detener cuando se meten *d* estados en CERRADOS o se encuentra un estado con heurística inexacta mientras se expanden los *d* estados que entran en CERRADOS. Si se encuentra un estado con heurística inexacta, este es insertado en *C* para ser considerado en la propagación acotada.

La versión de varios movimientos es similar a la versión de la Figura 40. La única diferencia está en el procedimiento LRTA-LS-Trial. A* debe parar cuando se expanda un estado con heurística inexacta, sino continua hasta completar *d* estados en CERRADOS (línea 4). *C* puede tener como máximo un estado. Desde este estado se ejecuta la propagación acotada (líneas 5-6). Luego ejecuta varios movimientos (líneas 7-12). La versión de un movimiento, en lugar de planificar varios movimientos, siempre planifica y ejecuta un movimiento (al mejor de los estados sucesores inmediatos).

La versión de movimiento dinámico es similar a la versión de la Figura 41. Tal como en las otras versiones, A* debe parar cuando se expanda un estado con heurística inexacta (línea 3). *C* puede tener como máximo un elemento: un estado con heurística inexacta encontrado mientras se están expandiendo los *d* estados de CERRADOS o el mejor estado en la frontera. Si se encuentra un estado con heurística inexacta se ejecuta la propagación acotada desde ese estado y se planifica y ejecuta un movimiento (líneas 5-9). De lo contrario ejecutan varios movimientos (líneas 11-15).

En la sección de resultados experimentales, se evalúan las 6 versiones de $LRTA^*_{LS}(k, d)$ en dos escenarios de prueba.

6.6 Resultados Experimentales

En este sección comparamos LRTA*$_{LS}(k)$ y LRTA*$_{LS}(k, d)$ (con aproximación estática de la anticipación y estrategia de un movimiento), para observar el beneficio y coste de usar una anticipación mayor y propagar con el nuevo método. Segundo, presentamos resultados para las 6 versiones de LRTA*$_{LS}(k, d)$:

1. La versión con aproximación estática de la anticipación y estrategia de varios movimientos (est-var).
2. La versión con aproximación estática de la anticipación y estrategia de un movimiento (est-un).
3. La versión con aproximación estática de la anticipación y estrategia de movimiento dinámico (est-din).
4. La versión con aproximación dinámica de la anticipación y estrategia de varios movimientos (din-var).
5. La versión con aproximación dinámica de la anticipación y estrategia de un movimiento (din-un).
6. La versión con aproximación dinámica de la anticipación y estrategia de movimiento dinámico (din-din).

Veremos cual es la más conveniente, a partir de los resultados obtenidos en los escenarios de prueba. Finalmente, comparamos LRTA*$_{LS}(k, d)$ (din-din) con LRTA*(Koenig).

6.6.1 LRTA$_{LS}(k)$ v/s LRTA*$_{LS}(k, d)$*

Evaluamos LRTA*$_{LS}(k, d)$ (est-un) para distintos valores de k y d. Cuando el parámetro $d = 1$, LRTA*$_{LS}(k, d)$ no tiene el mismo comportamiento que LRTA*$_{LS}(k)$[3]. LRTA*$_{LS}(k, d = 1)$ verifica si el estado actual y el mejor de la

[3] Se considera la versión 2 de LRTA*$_{LS}(k)$ ya que LRTA*$_{LS}(k, d)$ propaga sobre cualquier estado.

frontera (en este caso el mejor vecino inmediato) tienen heurística inexacta, en cambio LRTA*$_{LS}(k)$ sólo verifica la heurística del estado actual. El rendimiento de LRTA*$_{LS}(k, d = 1)$ y LRTA*$_{LS}(k)$ es similar (en coste, tiempo y actualizaciones), por tanto en esta sección en la que comparamos LRTA*$_{LS}(k)$ con LRTA*$_{LS}(k, d)$ consideramos que LRTA*$_{LS}(k, d = 1)$ equivale a LRTA*$_{LS}(k)$. La comparación se realizará sobre en Cua-35 y Lab-acic en la primera ejecución y en convergencia a soluciones óptimas. Los valores de k censados son 10, 40 y 160, un valor pequeño, mediano y grande respectivamente. Los valores de d censados son 1, 2, 4, 8, 16 y 32. Los resultados para los distintos k y d, se presentan en términos de actualizaciones (cantidad de actualizaciones), tiempo total de búsqueda (en milisegundos) y coste (en cantidad de acciones) y tiempo por paso (en milisegundos).

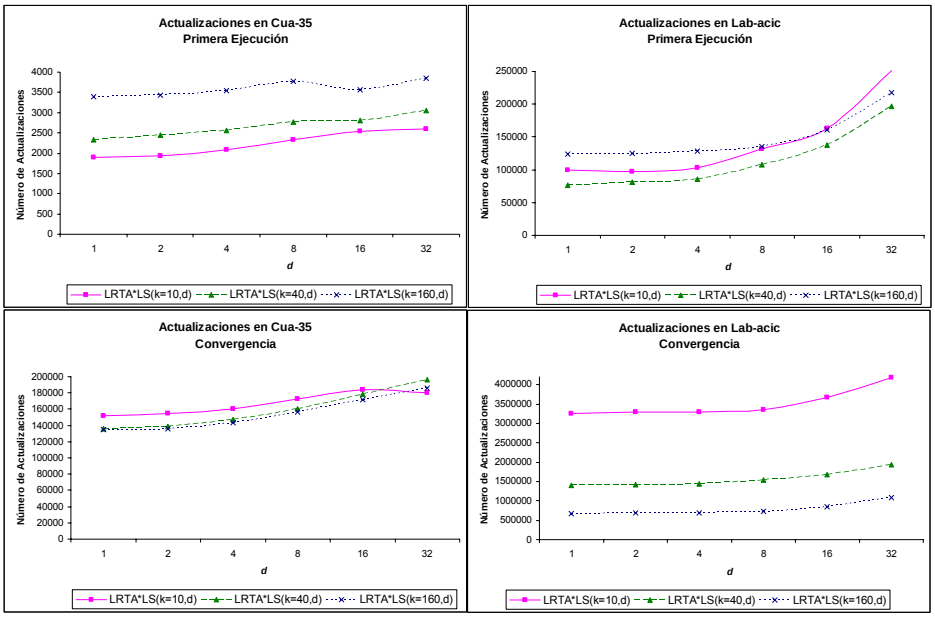


Figura 42. Resultados experimentales en Cua-35 y Lab-acic. Presentamos el número de actualizaciones que hace LRTA*$_{LS}(k,d)$ (est-un), para distintos valores de k y d en la primera ejecución y en convergencia.

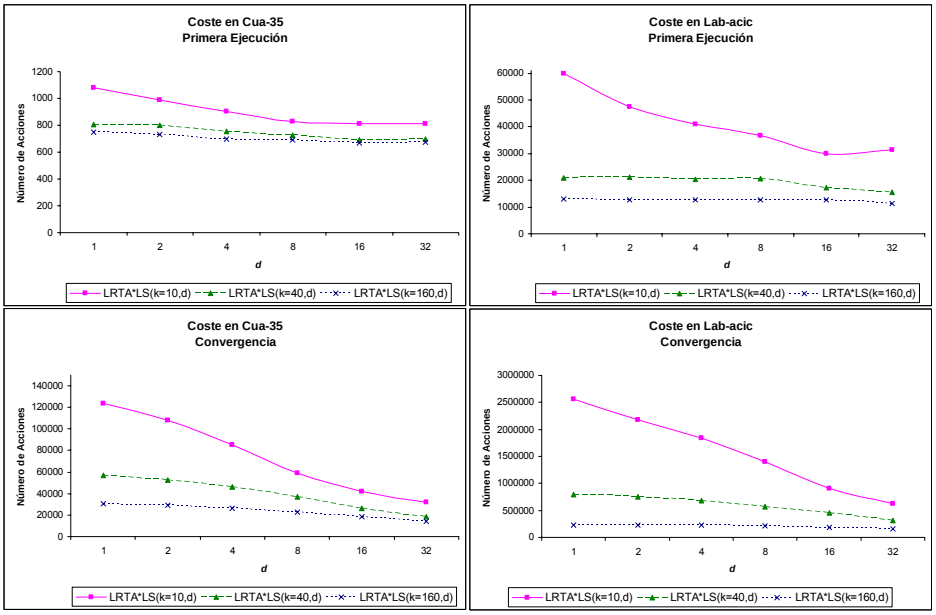


Figura 43. Resultados experimentales en Cua-35 y Lab-acic. Presentamos el coste obtenido con LRTA*$_{LS}$(k,d) (est-un), para distintos valores de k y d en la primera ejecución y en convergencia.

Uno de los efectos que se espera al aumentar la anticipación y aplicar el mecanismo de propagación sobre varios estados, es que al aumentar d para un k fijo, aumente el número de actualizaciones. En la Figura 42, se puede apreciar este efecto en Cua-35 y Lab-acic en la primera ejecución y en convergencia. En la primera ejecución, el aumento de actualizaciones es más notorio en Lab-acic. En Cua-35 el aumento es leve. En convergencia, el aumento en el número de actualizaciones es mucho más tenue, sobre todo en Lab-acic (para k grande la curva es casi plana). En general, no se observa un aumento considerable en el número de actualizaciones al aumentar la anticipación. Sólo en la primera ejecución en Lab-acic el aumento es considerable. Esto pasa porque la heurística inicial en Lab-acic es mala y la anticipación permite identificar varios estados con heurística inexacta dentro del espacio local de búsqueda. A mayor d se identifica una mayor cantidad de inexactitudes y se hace una mayor cantidad de actualizaciones.

Respecto al coste, hay un patrón común en los escenarios de prueba tanto para la primera ejecución como en convergencia que se puede observar en la Figura 43: Primero, el coste disminuye a medida que d aumenta. Segundo, a

mayor k la disminución es más leve. Por ejemplo, en convergencia en Lab-acic, para k=10 el coste disminuye 4.6 veces, desde 2.897.963 (LRTA*$_{LS}(k)$, d=1) a 634.929 (d=32). Para k=160 el coste disminuye 1.6 veces, desde 240.046 (LRTA*$_{LS}(k)$, d=1) a 150.716 (d=32). Podemos observar que con un valor de k grande (k =160), la influencia de la anticipación es pequeña.

Respecto al tiempo, se puede apreciar en las gráficas de la Figura 44 que el tiempo aumenta a medida que d aumenta. El esfuerzo (mayor cantidad de actualizaciones y mayor anticipación) para obtener un coste menor al aumentar d, se refleja en el aumento del tiempo total de búsqueda. El aumento en el tiempo total es menor cuando k es grande, sobre todo en convergencia.

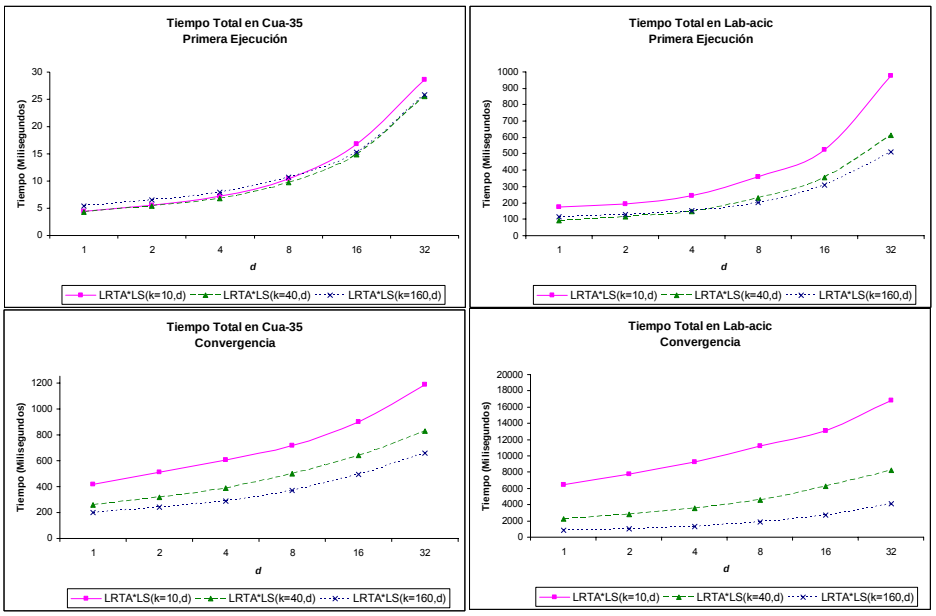


Figura 44. Resultados experimentales en Cua-35 y Lab-acic. Presentamos el tiempo total de búsqueda obtenido con LRTA*$_{LS}(k,d)$ (est-un), para distintos valores de k y d en la primera ejecución y en convergencia.

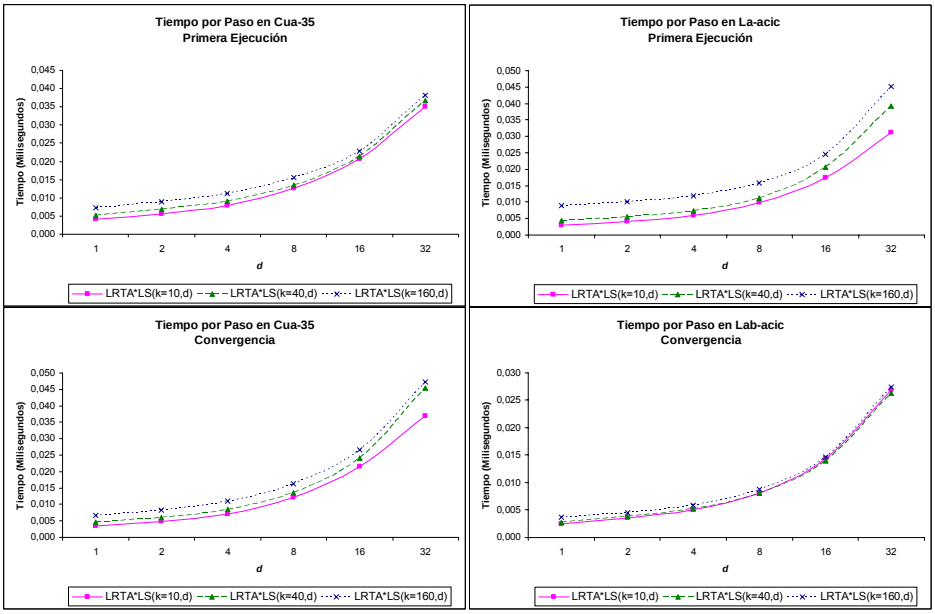


Figura 45. Resultados experimentales en Cua-35 y Lab-acic. Presentamos el tiempo por paso obtenido con LRTA*$_{LS}(k,d)$ (est-un), para distintos valores de k y d en la primera ejecución y en convergencia.

Para un k fijo el tiempo por paso aumenta cuando d crece, como se observa en la Figura 45. Para un d fijo el tiempo por paso aumenta levemente cuando k crece, excepto en convergencia en Lab-acic. En convergencia en Lab-acic las curvas son muy similares. Podemos observar que a menor k, menor aumento en el tiempo por paso cuando d crece (excepto en convergencia en Lab-acic con d =16 y 32).

Observando los resultados podemos concluir que para obtener una solución de buena calidad, conviene usar tiempo de la planificación en propagar con un k grande, en lugar de usar una mayor anticipación. Por ejemplo, en convergencia en Lab-acic con k = 160 y d = 1 se obtiene un coste de 240.046 con un tiempo total de búsqueda de 728 ms y un tiempo por paso de 0.0030 ms. Con k = 40 y d = 32 se obtiene un coste de 322.369 con un tiempo total de 8.499 ms y un tiempo por paso de 0,0264 ms.

6.6.2 Comparación entre las distintas versiones de LRTA$_{LS}(k,d)$*

La comparación se realizará sobre en Cua-35 y Lab-acic en la primera ejecución y en convergencia a soluciones óptimas en términos de coste

(número de acciones) y tiempo total de búsqueda. El objetivo de comparar las versiones es seleccionar la mejor en términos de coste y tiempo. Los valores de *k* censados son 10, 40 y 160. Los valores de *d* censados son 2, 4, 8, 16 y 32.

Antes de comparar las versiones, queremos destacar lo siguiente. Primero, el rendimiento de todas las versiones es similar con *d* pequeño tanto en tiempo como en coste en todos los valores de *k* en los dos escenarios. Las diferencias se manifiestan cuando *d* crece. Segundo, cuando *d* crece, en general se produce una mejora en el coste en distintas las versiones, pero en algunos casos el coste empeora. Este comportamiento ha sido estudiado en [Bulitko y Luvstrek, 2006] y se denomina patología de la anticipación en búsqueda de rutas en tiempo real. Respecto al tiempo total, podemos observar que siempre aumenta cuando *d* crece y dependiendo de la versión el aumento es más o menos pronunciado. Tercero, cuando *k* crece en primera ejecución y en convergencia el coste disminuye en los dos escenarios para todos los valores de *d*. Cuando *k* crece el tiempo total en la primera ejecución mantiene su valor o se incrementa o disminuye levemente. En convergencia el tiempo total disminuye cuando *k* crece en los dos escenarios de prueba en la mayoría de los casos.

En la Tabla 7 podemos apreciar los resultados para Cua-35. Respecto al coste y el tiempo total, se repite el mismo patrón con las versiones con anticipación dinámica y estática. En la mayoría de los casos la versión que realiza un movimiento obtiene el mejor coste, pero con un tiempo total mayor, y la versión que realiza varios movimientos obtiene el menor tiempo total, pero con el peor coste. Esto sucede en la primera ejecución y convergencia. La versión con movimiento dinámico es un caso intermedio. Esta versión obtiene una solución con un coste bueno, pero en general un poco peor que la versión de un movimiento y mejor que la versión de varios movimientos. El tiempo que obtiene la versión de movimiento dinámico no es tan bueno como el que obtiene la versión de varios movimientos, pero mejor que el de la

versión de un movimiento. Por tanto las versiones est-din y did-din son las mejores en la relación coste/tiempo en primera ejecución y convergencia.

En la Tabla 8 podemos ver los resultados para Lab-acic. En este escenario se presenta el mismo patrón que en Cua-35. Por tanto las versiones est-din y did-din son las mejores en la relación coste/tiempo en primera ejecución y convergencia.

Los resultados confirman lo esperado, cuando la anticipación es estática o dinámica, la estrategia de movimiento conservadora (un movimiento) en la mayoría de los casos realiza menos acciones a costa de un mayor tiempo de computación (sobre todo para los valores más grandes de *d*). En cambio la estrategia arriesgada (varios movimientos) obtiene un tiempo menor, pero con mayor coste. La estrategia de movimiento dinámico es un caso intermedio que parece ser la mejor opción en la relación coste/tiempo.

Realizamos una comparación más exhaustiva entre las versiones est-din y did-din para determinar cual es la más conveniente. En la Figura 46 podemos apreciar el coste y el tiempo total en Cua-35 en primera ejecución y convergencia para distintos valores de *k* y *d*. El coste para es similar en ambas versiones para todos los *k*, sólo con $d > 8$ est-din presenta una ventaja relativamente apreciable. La versión din-din obtiene un tiempo total menor que est-din, esta ventaja es apreciable con $d > 8$ cuando est-din obtiene un mejor coste. En la Figura 47 podemos apreciar el coste y el tiempo total en Lab-acic en primera ejecución y convergencia para distintos valores de *k* y *d*. Podemos apreciar que el efecto de la anticipación en el coste cuando *k* es 40 o 160 es despreciable en la primera ejecución (sólo con d=32 se observa una mejora). No hay un ganador claro respecto al coste, ambas versiones obtienen soluciones de una calidad similar. Donde si hay un ganador claro es el el tiempo total en primera ejecución y convergencia, din-din obtiene mejor tiempo que est-din para la mayoría de los valores de *k* y *d*. En la Figura 48, podemos ver el tiempo promedio por episodio de planificación. Podemos

observar que din-din consume menos tiempo que est-din por episodio de planificación en todos los valores *d* censados para un *k* fijo en primera ejecución y convergencia en los dos escenarios de prueba.

Si miramos los primeros valores de *d* (que son los que importan, porque el tiempo total aumenta con *d* en mayor proporción que la disminución del coste, Figuras 43 y 44), se observa que est-din es siempre mejor (por poco) que din-din en coste, y din-din es siempre mejor (por poco) en tiempo. Entonces elegimos din-din como la mejor opción (aunque est-din sigue siendo una buena alternativa). En estas gráficas podemos observar nuevamente que realizar mucha propagación y poca anticipación es más conveniente que realizar mucha anticipación y poca propagación en términos de coste y tiempo en primera ejecución y convergencia.

	Coste Primera Ejecución Cua-35																	
	k=10						*k*=40						*k*=160					
	est-uno	est-var	est-din	din-uno	din-var	din-din	est-uno	est-var	est-din	din-uno	din-var	din-din	est-uno	est-var	est-din	din-uno	din-var	din-din
LRTA*$_{LS}$(*k*,*d=2*)	988	1.034	1.026	993	1.055	1.024	800	819	813	808	816	799	730	748	752	733	752	740
LRTA*$_{LS}$(*k*,*d=4*)	904	994	934	923	1.006	964	755	778	775	776	812	785	699	734	707	710	733	727
LRTA*$_{LS}$(*k*,*d=8*)	831	946	847	888	985	898	727	762	730	740	782	746	689	710	692	692	711	696
LRTA*$_{LS}$(*k*,*d=16*)	813	998	805	864	983	885	691	728	699	728	791	738	669	687	678	690	708	689
LRTA*$_{LS}$(*k*,*d=32*)	813	1.034	814	866	989	877	695	728	687	724	782	728	677	676	676	683	710	680
	Coste Convergencia Cua-35																	
LRTA*$_{LS}$(*k*,*d=2*)	107.774	148.621	114.382	108.367	131.773	115.651	52.781	64.373	54.718	52.930	58.219	55.237	29.106	33.651	29.843	29.249	30.792	30.069
LRTA*$_{LS}$(*k*,*d=4*)	85.098	132.140	98.214	87.386	123.266	101.433	46.001	60.169	50.153	46.448	56.179	51.314	26.265	31.766	27.864	26.519	29.701	28.435
LRTA*$_{LS}$(*k*,*d=8*)	58.846	119.426	74.432	63.075	113.246	80.163	37.138	54.510	43.005	38.324	53.104	44.783	22.831	28.771	24.775	23.044	28.251	25.538
LRTA*$_{LS}$(*k*,*d=16*)	41.705	132.012	52.704	47.750	104.068	59.774	26.442	47.777	32.734	28.673	49.101	35.904	18.607	25.265	20.920	19.252	26.412	22.002
LRTA*$_{LS}$(*k*,*d=32*)	32.232	166.649	38.395	39.212	99.336	45.889	18.253	45.362	22.343	21.733	45.591	26.544	13.937	21.554	16.042	15.175	24.453	17.753
	Tiempo de Búsqueda Primera Ejecución Cua-35																	
	k=10						*k*=40						*k*=160					
	est-uno	est-var	est-din	din-uno	din-var	din-din	est-uno	est-var	est-din	din-uno	din-var	din-din	est-uno	est-var	est-din	din-uno	din-var	din-din
LRTA*$_{LS}$(*k*,*d=2*)	5,5	4,4	4,7	6,0	4,3	4,3	5,4	4,6	4,6	6,1	4,4	4,3	6,5	5,8	5,9	7,3	5,7	5,5
LRTA*$_{LS}$(*k*,*d=4*)	7,2	4,7	5,3	7,3	4,3	4,7	6,8	4,6	5,1	7,5	4,7	4,7	7,9	6,0	6,1	8,6	5,8	5,8
LRTA*$_{LS}$(*k*,*d=8*)	10,4	5,2	7,2	9,7	4,8	5,6	9,7	5,3	6,3	9,7	5,0	5,4	10,6	6,5	7,3	10,9	6,1	6,4
LRTA*$_{LS}$(*k*,*d=16*)	16,8	6,6	10,9	13,5	5,7	7,3	14,8	6,5	8,9	13,9	6,2	7,2	15,3	7,8	9,5	15,3	7,2	7,9
LRTA*$_{LS}$(*k*,*d=32*)	28,5	9,3	17,9	21,2	7,6	9,6	25,5	9,3	13,8	22,0	8,0	9,5	25,9	10,4	14,0	23,0	9,1	10,3
	Tiempo de Búsqueda Convergencia Cua-35																	
LRTA*$_{LS}$(*k*,*d=2*)	513,1	417,1	398,0	586,0	410,7	384,9	316,4	254,2	252,5	360,9	252,0	244,7	240,7	204,1	204,9	273,2	202,0	198,1
LRTA*$_{LS}$(*k*,*d=4*)	604,2	397,8	405,8	652,3	388,0	380,4	389,2	248,6	260,9	427,5	248,2	249,6	288,8	200,3	211,4	317,8	200,8	203,3
LRTA*$_{LS}$(*k*,*d=8*)	716,7	409,6	469,0	696,3	379,1	392,7	499,8	259,2	299,7	524,9	253,3	275,6	371,4	210,0	236,5	394,8	209,5	224,2
LRTA*$_{LS}$(*k*,*d=16*)	895,0	474,3	621,3	764,8	377,6	429,9	638,5	286,9	380,1	618,3	264,2	322,2	494,4	238,5	290,0	508,5	227,5	266,5
LRTA*$_{LS}$(*k*,*d=32*)	1.187,6	628,7	884,3	895,4	382,5	474,6	828,7	339,7	535,3	751,5	279,8	388,7	659,9	291,4	383,2	641,7	254,1	330,8

Tabla 7. Resultados en Cua-35 Presentamos el coste y tiempo total en primera ejecución y convergencia de las distintas versiones de LRTA*$_{LS}$(k,d).

	Coste Primera Ejecución Lab-acic																	
	k=10						*k=40*						*k=160*					
	est-uno	est-var	est-din	din-uno	din-var	din-din	est-uno	est-var	est-din	din-uno	din-var	din-din	est-uno	est-var	est-din	din-uno	din-var	din-din
LRTA*$_{LS}$(*k*,*d=2*)	47.519	49.240	47.195	48.761	51.006	51.085	21.122	19.806	21.132	21.202	20.213	21.155	12.792	12.797	12.880	12.802	13.022	13.188
LRTA*$_{LS}$(*k*,*d=4*)	40.812	48.303	44.274	39.976	46.300	45.390	20.318	21.306	21.479	20.339	20.796	21.496	12.643	13.349	13.033	12.922	12.672	12.712
LRTA*$_{LS}$(*k*,*d=8*)	36.807	49.094	41.050	35.900	44.241	39.665	20.533	21.829	21.157	19.406	20.599	21.481	12.757	13.551	12.911	13.181	13.328	13.526
LRTA*$_{LS}$(*k*,*d=16*)	29.890	65.837	32.538	28.434	41.869	31.294	17.181	21.419	22.440	15.434	19.360	20.782	12.601	13.664	13.556	12.212	13.531	12.970
LRTA*$_{LS}$(*k*,*d=32*)	31.347	104.690	32.302	22.498	43.099	24.594	15.567	22.681	18.503	13.155	18.711	16.155	11.299	13.464	12.322	11.023	13.400	12.291
	Coste Convergencia Lab-acic																	
LRTA*$_{LS}$(*k*,*d=2*)	2.178.548	2.414.819	2.348.924	2.177.916	2.421.189	2.357.899	749.701	772.175	768.321	749.442	774.007	768.337	228.961	230.546	231.130	228.293	231.336	231.206
LRTA*$_{LS}$(*k*,*d=4*)	1.838.192	2.184.339	2.031.629	1.845.275	2.180.984	2.043.727	684.104	744.673	730.165	684.117	743.168	732.114	221.826	228.834	226.700	221.884	227.930	226.757
LRTA*$_{LS}$(*k*,*d=8*)	1.396.037	2.087.627	1.696.385	1.410.139	1.952.763	1.721.259	568.843	700.813	658.707	569.327	694.738	663.369	208.948	224.034	218.278	209.427	222.245	219.379
LRTA*$_{LS}$(*k*,*d=16*)	904.621	2.991.035	1.225.184	916.335	1.825.321	1.229.664	450.243	637.295	536.644	455.089	622.145	544.419	186.322	216.495	204.527	187.201	213.882	206.082
LRTA*$_{LS}$(*k*,*d=32*)	627.151	5.303.609	830.714	628.682	1.755.500	853.709	313.249	652.528	412.284	317.808	562.307	419.728	149.210	205.843	178.152	151.663	203.652	183.038
	Tiempo de Búsqueda Primera Ejecución Lab-acic																	
	k=10						*k=40*						*k=160*					
	est-uno	est-var	est-din	din-uno	din-var	din-din	est-uno	est-var	est-din	din-uno	din-var	din-din	est-uno	est-var	est-din	din-uno	din-var	din-din
LRTA*$_{LS}$(*k*,*d=2*)	195,1	146,4	142,2	231,3	146,6	146,6	116,8	87,3	92,6	134,6	87,5	88,7	129,3	111,9	116,1	149,6	115,8	113,7
LRTA*$_{LS}$(*k*,*d=4*)	242,8	159,5	164,2	252,9	143,5	152,3	148,3	95,5	102,4	162,4	90,7	95,3	151,8	117,1	124,6	171,2	116,9	115,7
LRTA*$_{LS}$(*k*,*d=8*)	360,8	187,4	228,6	326,9	153,8	169,7	231,9	115,2	133,1	224,0	99,3	114,9	200,6	131,7	142,6	215,5	128,3	132,1
LRTA*$_{LS}$(*k*,*d=16*)	524,0	222,4	350,5	385,9	167,8	203,3	353,9	153,8	231,0	284,9	113,0	160,5	308,7	168,1	202,1	293,1	145,7	161,7
LRTA*$_{LS}$(*k*,*d=32*)	976,8	320,7	761,5	422,5	193,9	265,6	611,9	213,6	399,5	387,3	129,3	206,8	511,7	241,7	324,6	425,9	168,8	227,2
	Tiempo de Búsqueda Convergencia Lab-acic																	
LRTA*$_{LS}$(*k*,*d=2*)	7.727,3	5.816,4	5.709,9	9.112,5	5.717,9	5.557,1	2.831,9	1.885,7	1.957,5	3.350,0	1.898,9	1.909,6	1.038,3	726,6	775,9	1.240,9	756,6	758,8
LRTA*$_{LS}$(*k*,*d=4*)	9.239,9	5.494,8	5.587,0	10.206,5	5.236,4	5.239,1	3.523,5	1.768,4	1.878,2	4.005,2	1.755,2	1.820,2	1.315,4	699,6	758,0	1.505,4	720,5	735,6
LRTA*$_{LS}$(*k*,*d=8*)	11.220,2	5.717,7	6.130,0	11.282,0	4.999,2	5.070,4	4.619,3	1.740,3	1.912,0	4.978,1	1.690,6	1.798,7	1.822,9	703,5	782,5	1.988,5	713,2	748,7
LRTA*$_{LS}$(*k*,*d=16*)	13.069,6	7.015,4	7.957,2	11.472,0	4.994,7	5.130,2	6.279,2	1.791,2	2.126,5	6.374,7	1.685,2	1.862,5	2.740,0	758,2	901,2	2.832,4	732,3	820,8
LRTA*$_{LS}$(*k*,*d=32*)	16.823,9	10.590,6	12.472,9	11.520,6	5.077,5	5.979,1	8.220,4	2.005,6	2.741,5	7.594,7	1.686,3	1.984,1	4.079,2	892,3	1.217,6	3.995,6	769,8	976,1

Tabla 8. Resultados en Lab-acic Presentamos el coste y tiempo total en primera ejecución y convergencia de las distintas versiones de LRTA*$_{LS}$(k,d).

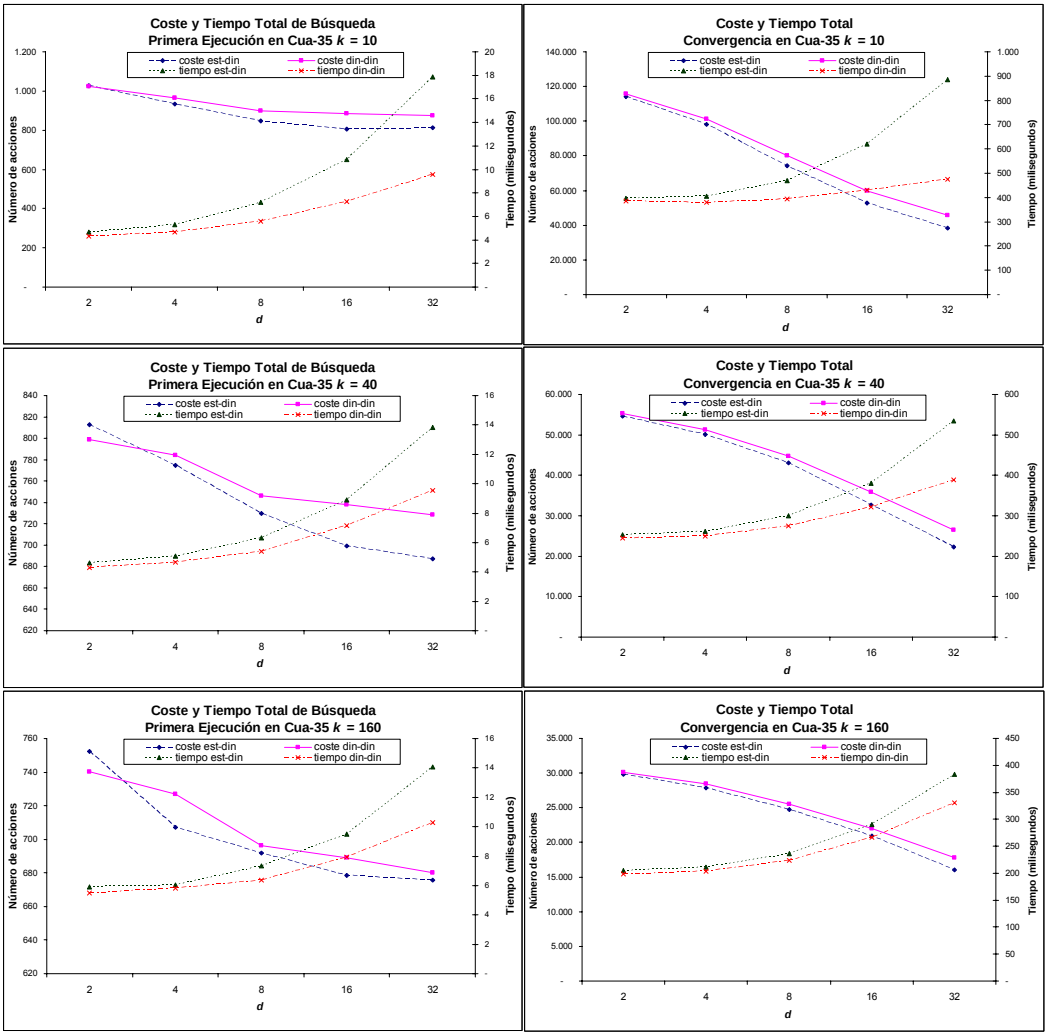


Figura 46. Resultados experimentales en Cua-35. Presentamos el coste y el tiempo total obtenido con LRTA*$_{LS}$(k,d) versión est-din y din-din, para distintos valores de k y d en la primera ejecución y en convergencia.

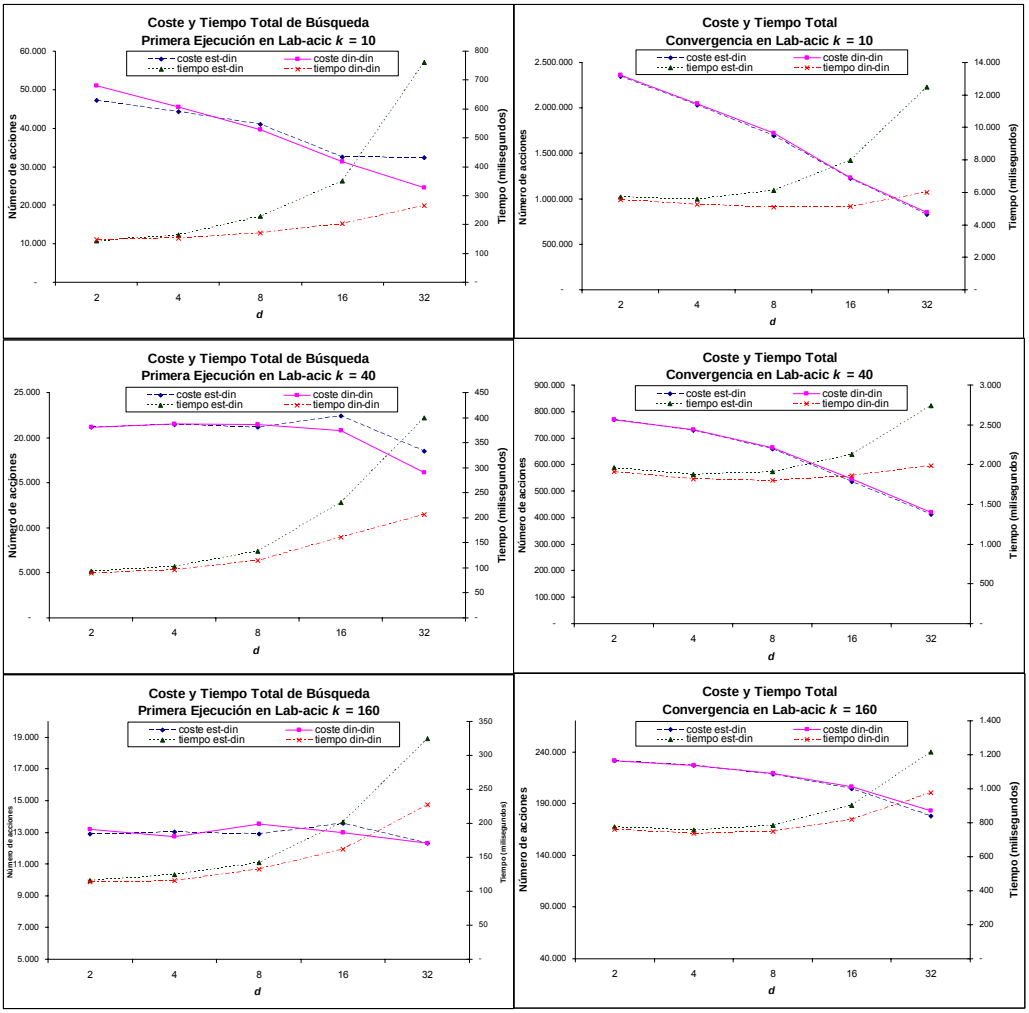


Figura 47. Resultados experimentales en Lab-acic. Presentamos el coste y el tiempo total obtenido con LRTA*$_{LS}(k,d)$ versión est-din y din-din, para distintos valores de k y d en la primera ejecución y en convergencia.

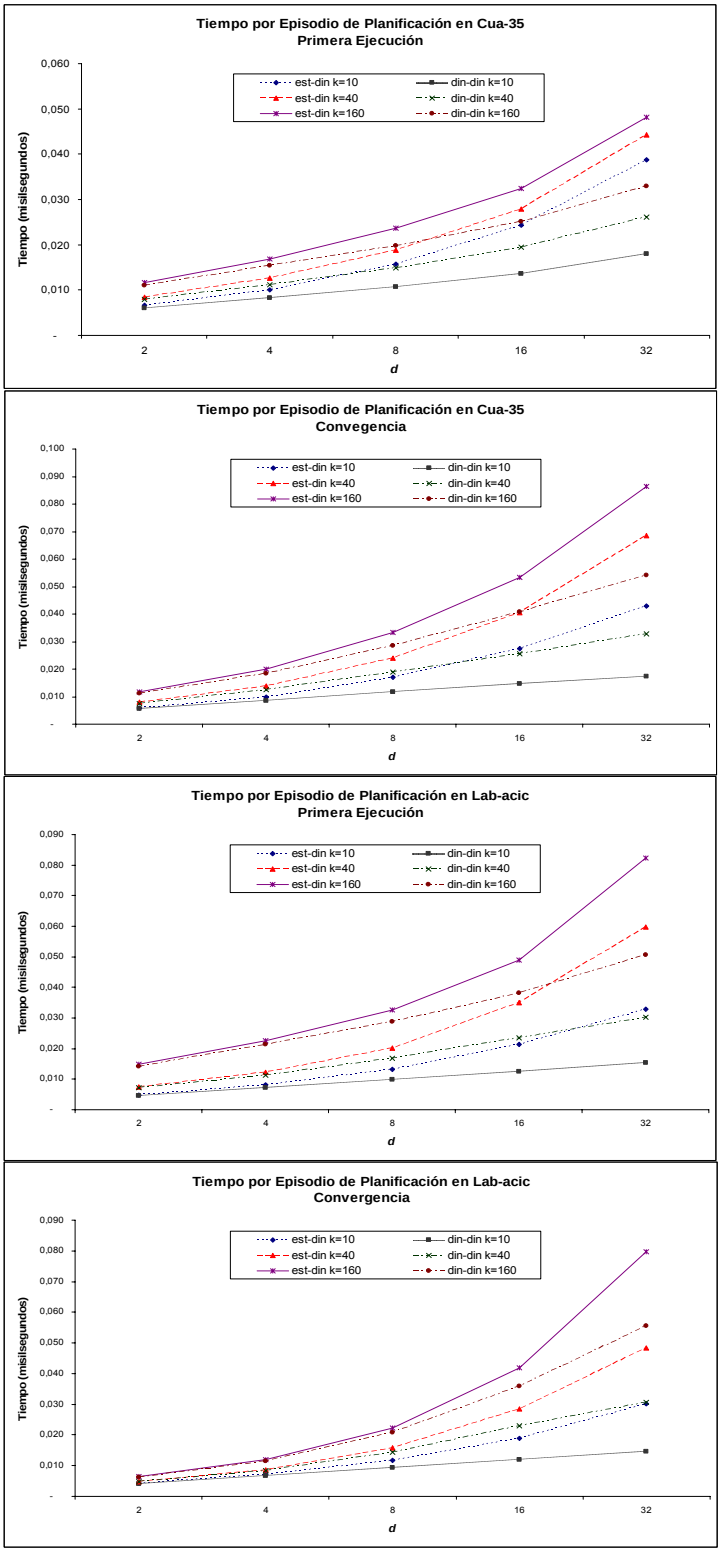


Figura 48. Resultados experimentales en Cua-35 y Lab-acic. Presentamos tiempo total promedio por episdio de planificación obtenido con LRTA*$_{LS}$(k,d) versión est-din y din-din, para distintos valores de k y d en la primera ejecución y en convergencia.

6.6.3 Comparando con LRTA(Koenig).*

El tamaño del espacio local de búsqueda y el espacio local de aprendizaje en LRTA*(Koenig) están determinados por el parámetro *d*. El algoritmo anticipa hasta completar *d* estados en CERRADOS, luego actualiza la heurística de estos estados como se explica en la sección 6.1 y se ejecutan varios movimientos hacia el mejor estado de ABIERTOS. Hacemos la comparación con la versión de LRTA*$_{LS}$(*k*, *d*) que usa la aproximación dinámica en la anticipación y la estrategia de movimiento dinámico. Los valores de *k* censados son 10, 40 y 160, los valores de *d* son 2, 4, 8, 16, 32, 64 y 128. Presentamos el coste, el tiempo total de búsqueda (convergencia) y el tiempo por episodio de planificación en Cua-35 y Lab-acic para la primera ejecución y convergencia a soluciones óptimas.

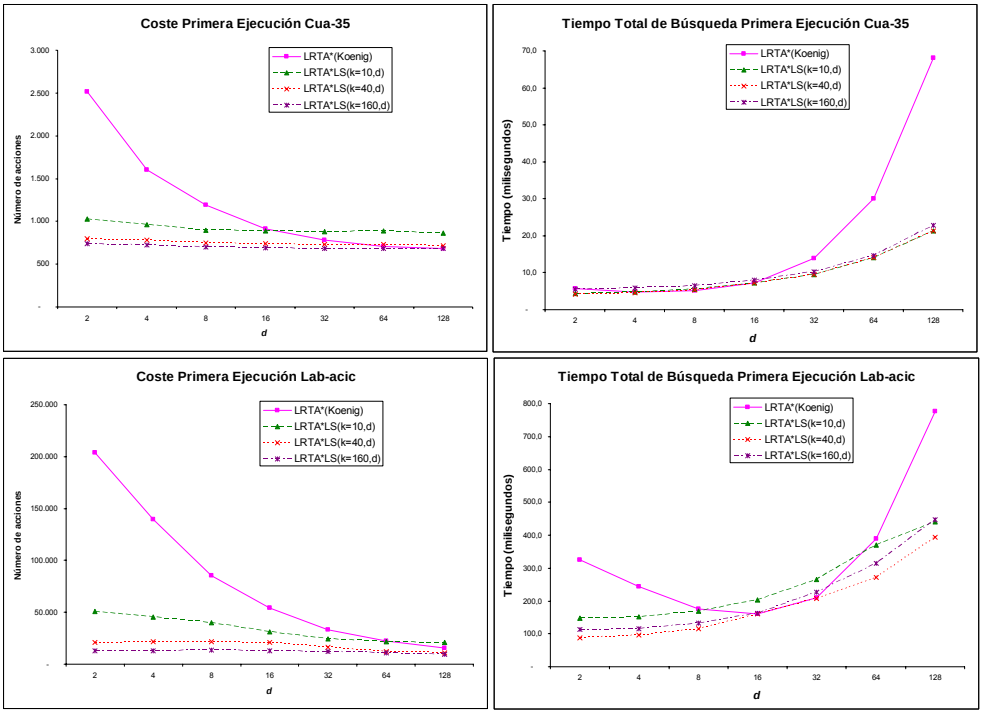


Figura 49. Resultados experimentales en Cua-35 y Lab-acic con LRTA*(Koenig) y LRTA*$_{LS}$(*k*,*d*) (din-din) en la primera ejecución, para distintos valores de *k* y *d*.

En la Figura 49 podemos apreciar los resultados en la primera ejecución. A continuación comentamos las medidas de desempeño.

- Coste: LRTA*(Koenig) disminuye su coste a medida que *d* crece. Con valores pequeños e intermedios de *d*, LRTA*$_{LS}$(*k*, *d*) obtiene un coste menor. Con valor grandes de *d*, el coste obtenido con LRTA*(Koenig) es similar al de LRTA*$_{LS}$(*k*, *d*). En LRTA*$_{LS}$(*k*, *d*) a mayor *k* menor coste. Podemos apreciar que con el valor intermedio (40) y grande (160) de *k*, el efecto de la anticipación es prácticamente despreciable.

- Tiempo total de búsqueda: LRTA*(Koenig) obtiene un tiempo similar al obtenido con LRTA*$_{LS}$(*k*, *d*) con valores pequeños e intermedios de *d*. Con valores grandes, el tiempo obtenido con LRTA*(Koenig) empeora considerablemente. El tiempo obtenido con LRTA*$_{LS}$(*k*, *d*) para los diferentes *k*, es similar (sobre todo en Cua-35) y aumenta de manera suave a medida que *d* crece. Podemos ver que con *k* = 40, se obtiene el mejor tiempo.

Es importante destacar que en primera ejecución, cuando LRTA*(Koenig) tiene un tiempo similar a LRTA*$_{LS}$(*k*, *d*) ofrece soluciones mucho peores. Por otro lado, cuando ofrece soluciones de calidad similar a LRTA*$_{LS}$(*k*, *d*), requiere tiempos de ejecución mucho mayores. Esto nos indica que LRTA*$_{LS}$(*k*, *d*) con anticipación dinámica y movimiento dinámico es una buena aproximación que mejora a LRTA*(Koenig).

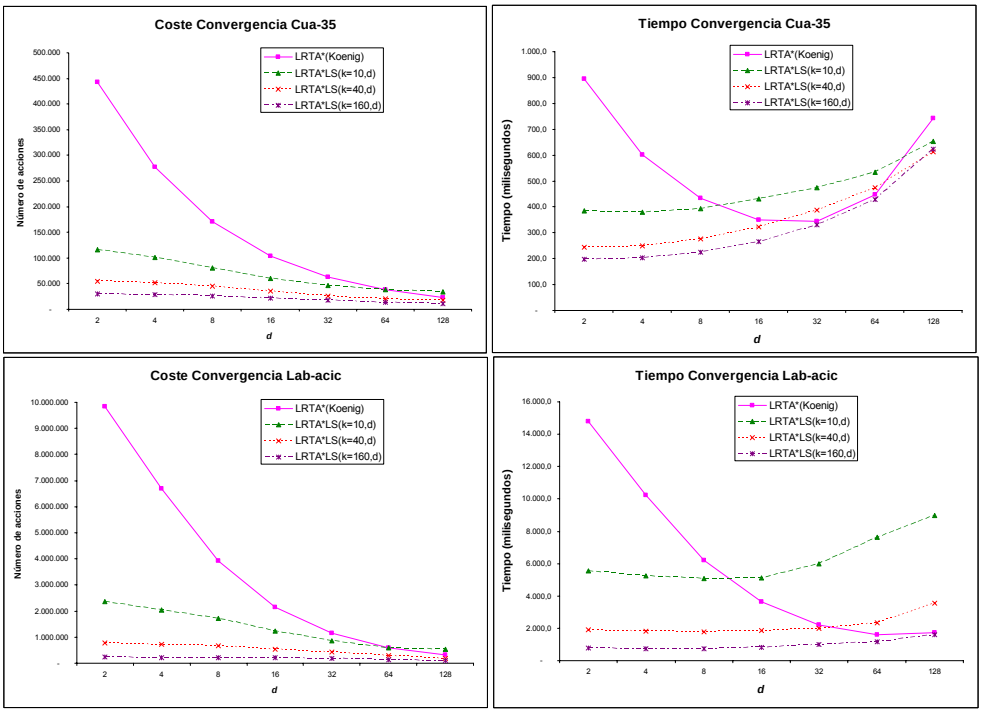


Figura 50. Resultados experimentales en Cua-35 y Lab-acic con LRTA*(Koenig) y LRTA*$_{LS}$(k,d) (din-din) en convergencia, para distintos valores de k y d.

En la Figura 50 podemos apreciar los resultados para la convergencia. A continuación comentamos las medidas de desempeño.

- Coste: LRTA*(Koenig) disminuye su coste a medida que d crece. En todos los valores d LRTA*$_{LS}$(k=40,160, d) obtiene un coste menor. LRTA*(Koenig) sólo obtiene un coste menor que LRTA*$_{LS}$(k=10, d) con d grande ($d \geq 64$). En LRTA*$_{LS}$(k, d) a mayor k menor coste. Podemos apreciar que con el valor intermedio (40) y grande (160) de k, el efecto de la anticipación es prácticamente despreciable.

- Tiempo total de convergencia: en LRTA*(Koenig) el tiempo primero disminuye y luego aumenta desde un valor de d (d=32 en Cua-35 y d=64 en Lab-acic). Con d pequeño LRTA*(Koenig) obtiene un tiempo mayor que al obtenido con LRTA*$_{LS}$(k, d). A partir de cierto valor de d, LRTA*(Koenig) mejora a LRTA*$_{LS}$(k=10,40, d) (excepto con d = 128 en Cua-35). El tiempo obtenido con LRTA*$_{LS}$(k, d) para

los diferentes *k*, disminuye cuando *k* crece. LRTA*$_{LS}$(*k*=160, *d*) siempre obtiene el menor tiempo total de convergencia.

Estos resultados nos muestran que LRTA*(Koenig) es competitivo en coste con LRTA*$_{LS}$(*k*, *d*) sólo cuando se hace suficiente anticipación. En convergencia LRTA*(Koenig) con *d* grandes mejora en tiempo a LRTA*$_{LS}$(*k*, *d*) con k=10 y 40, pero siempre es peor con *k* =160. Claramente propagar mucho y anticipar poco (*k*=160 y *d*=2) con LRTA*$_{LS}$(*k*, *d*) es mejor que anticipar mucho con LRTA*(Koenig) en la relación coste/tiempo.

6.6.4 Conclusiones

Aumentar la anticipación por episodio de planificación en LRTA*$_{LS}$(*k*, *d*) de un movimiento, mejora la calidad de la solución de LRTA*$_{LS}$(*k*). El esfuerzo invertido en anticipar para obtener un menor coste, se refleja en un aumento importante del tiempo total de búsqueda. El aumento en la anticipación para un *k* fijo, siempre aumenta el tiempo total de búsqueda. En cambio, el aumento de propagación para un *d* fijo, disminuye o aumenta levemente el tiempo de búsqueda. Los resultados muestran que es más conveniente (para obtener soluciones de buena calidad en un tiempo moderado), usar una anticipación baja con un valor de *k* mediano o grande, en vez de invertir esfuerzos en una mayor cantidad de anticipación.

En la mayoría de los casos evaluados, las versiones de LRTA*$_{LS}$(*k*, *d*) que realizan un movimiento obtienen el mejor resultado en coste y las que realizan varios movimientos el mejor resultado en tiempo total. La versión con aproximación dinámica de la anticipación y estrategia de movimiento dinámico de LRTA*$_{LS}$(*k*, *d*) es la mejor en la relación Coste/Tiempo. En esta versión el espacio local de búsqueda puede contener hasta *d* estados y el espacio local de aprendizaje hasta *k*. El tamaño de ambos espacios locales y el número de movimientos que realiza el agente dependen de la calidad de la heurística.

La versión de LRTA*$_{LS}(k, d)$ con aproximación dinámica en la anticipación y la estrategia de movimiento dinámico, obtiene un mejor desempeño que LRTA*(Koenig) en términos coste/tiempo. Este resultado se obtiene cuando LRTA*$_{LS}(k, d)$ invierte un mayor esfuerzo en propagar en lugar de anticipar.

6.7 Resumen

En este capítulo vimos los métodos de anticipación más relevantes en búsqueda heurística en tiempo real. Entre estos seleccionamos A* para hacer una combinación con la propagación acotada. Con la anticipación es posible identificar varios estados con heurística inexacta en el espacio local de búsqueda, debido a esto generalizamos el mecanismo de propagación acotada para que se pueda iniciar desde más de un estado, lo que permite hacer una mayor cantidad actualizaciones por episodio de planificación. Luego presentamos tres estrategias de movimiento en el espacio de búsqueda y un nuevo método de anticipación, basado en A*, que toma en cuenta la calidad de la heurística para hacer una selección dinámica del tamaño del espacio de búsqueda. Los mecanismos de anticipación y estrategias de movimiento se implementan en distintas versiones del algoritmo LRTA*$_{LS}(k, d)$.

En la sección de resultados experimentales, analizamos el beneficio/coste de aumentar anticipación, evaluamos las distintas versiones de LRTA*$_{LS}(k, d)$ y comparamos nuestra aproximación con el algoritmo LRTA*(Koenig). La versión con aproximación dinámica de la anticipación y estrategia de movimiento dinámico de LRTA*$_{LS}(k, d)$ obtiene el mejor resultado en la relación coste/tiempo. En esta versión del algoritmo el tamaño del espacio local de búsqueda, el tamaño del espacio local de aprendizaje y el número de movimientos que realiza el agente no son fijos, sino que dependen de la calidad de la heurística.

Los resultados experimentales, muestran que en cada episodio de planificación, es preferible invertir tiempo en propagar en vez de anticipar. El aumento de la anticipación (para un k fijo), generalmente disminuye el coste

de obtener una solución, pero siempre aumenta (en algunos casos considerablemente) el tiempo total de búsqueda. El aumento de la propagación (para un *d* fijo), disminuye el coste de obtener una solución, y disminuye o aumenta levemente el tiempo total de búsqueda.

Capítulo siete

7 MEJORANDO HLRTA*

Aplicaremos los métodos de propagación desarrollados en los capítulos anteriores a HLRTA*. Esta combinación se justifica porque los beneficios obtenidos en LRTA* al incluir propagación, también se pueden obtener en HLRTA*, tanto en primera ejecución como en convergencia. El contenido del capítulo es el siguiente: primero, describiremos el comportamiento de HLRTA* (qué lo diferencia de LRTA*), luego veremos los algoritmos HLRTA*(k), HLRTA*$_{LS}$(k) y HLRTA*$_{LS}$(k, d), una combinación entre HLRTA* y propagación acotada con iteración de valores, propagación sobre un espacio local de reparación y anticipación con propagación acotada. Después se presentaran resultados experimentales en distintos escenarios de prueba. Finalmente presentamos el algoritmo RTA*$_{LS}$(k) una combinación entre RTA* y propagación acotada sobre un espacio local y resultados experimentales para este algoritmo. Terminamos el capítulo con las conclusiones.

7.1 Comportamiento de HLRTA*

Como se mencionó en el estado del arte, HLRTA* es un algoritmo que usa dos estimaciones heurísticas y almacena la dirección del movimiento del agente en cada estado. Cuando el agente se mueve desde un estado y hacia x, se guarda $d(y) = x$, la dirección del movimiento desde el estado y. Cada vez que HLRTA* actualiza las heurísticas del estado actual x, guarda el primer ($h_1(x)$) y segundo ($h_2(x)$) mínimo de $c(x, y) + H(y)$, $y \in succ(x)$, en donde $H(y) = h_2(y)$ si $d(y)=x$, o $H(y) = h_1(y)$ en otro caso. Estas características, incorporan al algoritmo dos particularidades vinculadas a su desempeño:

1. El mantenimiento de información sobre la mejor ruta alternativa en el camino recorrido. Cuando el agente obtiene la heurística de un estado vecino, cuya dirección de movimiento es el estado actual (el vecino es parte del camino recorrido), observa la heurística h_2. Lo que en realidad

observa el agente, es la estimación del coste de la mejor ruta alternativa desde el estado vecino. En esta situación, dependiendo del valor de la estimación heurística de los otros vecinos, el agente puede considerar más conveniente volver por el camino recorrido que seguir adelante.

2. La actualización con información de la mejor ruta alternativa. Cuando el agente decide regresar por el camino recorrido, actualiza la heurística h_1 del estado actual usando la heurística h_2 del estado vecino. Debido a que $h_2 \geq h_1$, se realiza una actualización más agresiva que la de LRTA*, manteniendo la admisibilidad de la heurística [Thorpe, 1994].

Por ejemplo, en la parte superior de la Figura 51, se puede ver una cuadrícula 4-conectada. El estado inicial es la celda *a1*. El estado objetivo es *c4*. El coste de cualquier movimiento es uno. A continuación, veremos como un agente encuentra una ruta entre el estado inicial y el objetivo, usando HLRTA*. En la Figura 51 (A), podemos ver al agente (punto negro) en el estado inicial. El rango de visibilidad del agente es uno. Los números en las celdas son las estimaciones heurísticas h_1/h_2. Supondremos que los empates se resuelven en orden sur-oeste-norte-este. En (A), el agente actualiza las heurísticas h_1/h_2 de *a1*, selecciona el mejor sucesor para moverse y guarda la dirección del movimiento. El resultado de la actualización, una flecha que indica la dirección del movimiento y la nueva posición del agente se pueden ver en (B). En (B), el agente hace lo mismo. En este estado tiene 3 opciones de movimiento. Desde *a2*, puede ir a *a3* o *b2* o regresar a *a1*. La opción de regresar es descartada, porque la mejor ruta alternativa desde *a1* tiene valor heurístico infinito. Se selecciona *a3* como próximo estado (por definición de empates). El agente continúa actualizando heurísticas y realizando movimientos, hasta que en (D) encuentra un camino sin salida. En (D), la mejor opción es regresar desde *c2* por el camino recorrido. La heurística de mejor ruta alternativa desde *b2* es 5. Por tanto, se actualiza h_1/h_2 de *c2* con $6/\infty$ y luego el agente se mueve a *b2*. En esta situación, LRTA* actualizaría $h_1(c2)$ con 4 en vez de 6. En (E) se puede ver el resultado de las

actualizaciones y la nueva posición del agente. En el estado *b2*, el agente tiene dos opciones de movimiento, desde *b2*, puede ir a *a2* o regresar a *c2*. La opción de regresar es descartada, porque la mejor ruta alternativa desde *c2* tiene valor heurístico infinito. Se selecciona *a2* como próximo estado. En esta situación, LRTA* podría moverse a *a2* o *c2* con la misma probabilidad (si los empates se resuelven de manera aleatoria), ya que ambos estados tendrían un heurística de valor 4. Después, el agente continúa actualizando heurísticas y realizando movimientos, hasta que encuentra la solución. Es fácil ver que en una segunda ejecución, el agente recorrerá una ruta óptima hacia el objetivo.

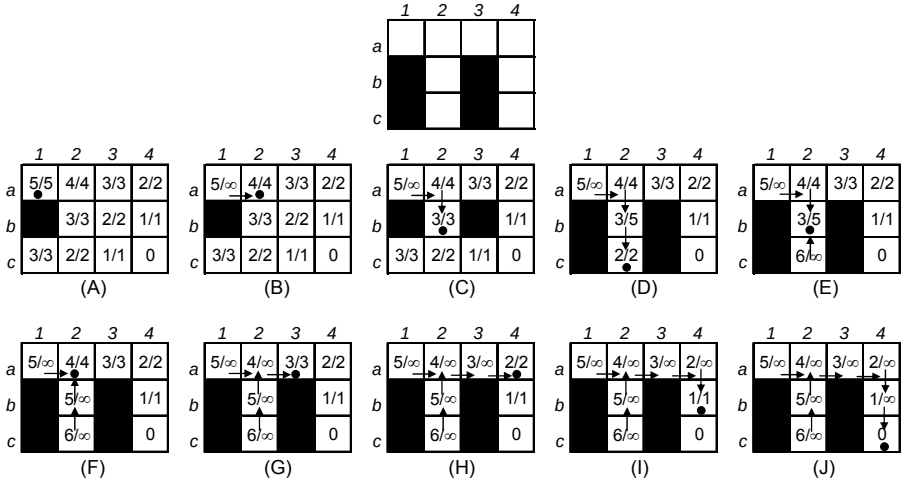


Figura 51. Ejemplo del comportamiento de HLRTA*.

En el ejemplo se puede apreciar el efecto positivo de usar dos heurísticas y la dirección del movimiento en HLRTA*.

Otro punto a destacar es que HLRTA* actualiza la heurística h_1 del estado actual x, usando h_2 de un sucesor z sólo si $d(z) = x$ y $c(x, z) + h_2(z)$ es la mejor opción entre todos los sucesores. En [Thorpe, 1994] se demuestra que esta actualización mantiene la admisibilidad y como resultado asegura la convergencia a rutas óptimas. Es fácil ver que, para asegurar la admisibilidad de h_1, es necesario que $d(z) = x$. Lo podemos ver en el ejemplo de la Figura 52. En (i) vemos la situación inicial. Los estados son los puntos de color negro, todas las acciones valen 1, los valores iniciales de h_1/h_2 son los

números que se aprecian junto a cada estado (h_1 es admisible). En (ii) podemos apreciar la trayectoria que ha seguido el agente, indicada por las flechas, y los valores heurísticos después de actualizar h_1/h_2, suponiendo que $H(z)$ es $h_2(z)$ si z ha sido visitado(sin exigir que $d(z) = x$). Después del cuarto movimiento, el agente actualiza la heurística del estado actual h_1 con 5, lo que sobrestima la distancia real al objetivo, haciendo h_1 no admisible. Este valor es el resultado de actualizar h_1 del estado actual usando h_2 de los sucesores sin considerar la dirección del movimiento. En este estado el agente ve un 4 en el sucesor que acaba de visitar y un 4 en el otro.

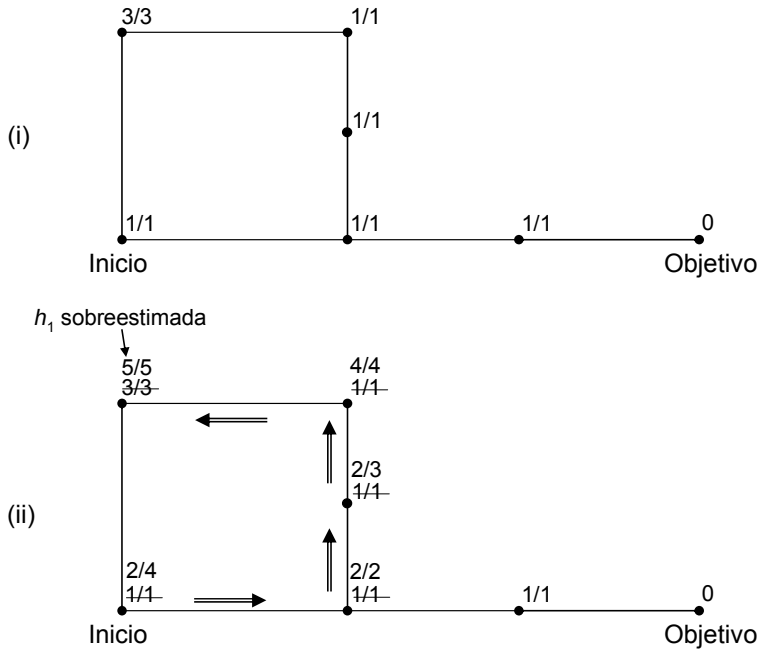


Figura 52. Ejemplo de sobreestimación de la heurística por HLRTA* al dejar de usar la dirección del movimiento del agente.

7.2 HLRTA*(*k*)

HLRTA*(*k*) [Hernandez y Meseguer, 2005c], es un algoritmo que combina HLRTA* y propagación acotada. La propagación acotada en HLRTA*(*k*) sólo se realiza sobre h_1. La propagación sobre h_2, puede provocar que h_1 pierda la admisibilidad. Por ejemplo, propaguemos el cambio en las dos heurísticas. Supongamos que después de actualizar las heurísticas del estado *b2* de 3/5 a

5/∞ en la Figura 51 (E) se propaga el cambio al estado *a2*. Las heurísticas de *a2* cambian de 4/4 a 4/6, si propaga este cambio a *b2*, las heurísticas de *b2* cambian de 5/∞ a 7/∞. Con este último cambio, $h_1(b2)$ ya no es admisible.

El algoritmo HLRTA*(*k*) aparece en la Figura 53. La principal diferencia con HLRTA* (sección 3.3.3) esta en el procedimiento HLRTA-TRIAL: HLRTA-LookaheadUpdate1 es remplazado por HLRTA-LookaheadUpdateK (línea 4) y se inserta HLRTA-LookaheadUpdate2min (línea 5). HLRTA-LookaheadUpdateK realiza la propagación acotada. La actualización del segundo (h_2) y primer (h_1) mínimo se hace con el procedimiento HLRTA-LookaheadUpdate2min y la función HLRTA-LookaheadUpdate1min, respectivamente. La heurística h_1 se actualiza en la propagación acotada y después de propagar se actualiza h_2 (antes de cada movimiento). En la función HLRTA-LookaheadUpdate1min se asignan los soportes. Esta retorna verdadero cuando h_1 cambia y falso si no cambia.

En HLRTA-LookaheadUpdateK, Q es la cola de propagación con capacidad para k elementos, esta se inicializa con el estado actual x (línea 1). En seguida se ejecuta el ciclo en donde se hace la propagación. El segundo mínimo no se incluye en la propagación, porque puede afectar la admisibilidad de h_1. El ciclo es el siguiente. Mientras Q no este vacío, se extrae el primer elemento v de Q y se actualiza $h_1(v)$ (con HLRTA-LookaheadUpdate1min) (líneas 3-4). Si esta actualización causa algún cambio en $h_1(v)$, este cambio es propagado a un sucesor w siempre y cuando cumpla las siguientes condiciones: v es soporte de w ($v = supp(w)$), hay espacio en Q ($cont > 1$), y el número de soportes de w es 1 ($|SUPP(w)|=1$). Si el sucesor cumple estas condiciones entonces se agrega a Q. En este punto, si v es soporte de w, se quita del conjunto de soportes de w, porque $h(v)$ cambió y por tanto no cumple la condición de soporte para w (líneas 5-9).

```
procedure HLRTA(k) (X, A, c, s0, G, k)
1  for each x ∈ X do h1(x) ← h0(x); h2(x) ← h0(x); SUPP(x) ← ∅;
2  repeat
3     HLRTAK-trial(X, A, c, s0, G, k);
4  until h1 does not change;

procedure HLRTAK-trial(X, A, c, s0, G, k)
1  x ← s0;
2  while x ∉ G do
3     d(x) ← null;
4     HLRTA-LookaheadUpdateK(x, k);
5     HLRTA-LookaheadUpdate2min(x);
6     y ← argmin_{w∈Succ(x)} [c(x,w) + H(w,x)]; (Break ties randomly)
7     execute(a ∈ A such that a = (x, y));
8     d(x) ← y;   x ← y;

procedure HLRTA-LookaheadUpdateK(x, k, path)
1  Q ← ⟨x⟩; cont ← k – 1;
2  while Q ≠ ∅ do
3     v ← extract-first(Q);
4     if HLRTA-LookaheadUpdate1min(v) then
5        for each w ∈ Succ(v) do
6           if v ∈ SUPP(w) then
7              if cont > 0 ∧ |SUPP(w)| = 1 then
8                 Q ← add-last(Q,w); cont ← cont – 1;
9           SUPP(w) ← SUPP(w) – {v};

procedure HLRTA-LookaheadUpdate2min(x)
1  z ← arg2ndmin_{v∈Succ(x)}[c(x,v) + H(v,x)];
2  if h2(x) < c(x, z) + H(z,x) then h2(x) ← c(x, z) + H(z,x);

function HLRTA-LookaheadUpdate1min(x): boolean;
1  y ← argmin_{v∈Succ(x)}[c(x,v) + H(v,x)];
2  for each v ∈ Succ(x) do
3     if c(x,w) + H(w,x) = c(x,y) + H(y,x) then
4        SUPP(x) ← SUPP(x) ∪ {w};
5  if h1(x) < c(x, y) + H(y,x) then h1(x) ← c(x, y) + H(y,x); return true; else return false;

function H(v, from): real;
1  if d(v) = from then return max{h1(v),h2(v)}; else return h1(v);
```

Figura 53. Algoritmo HLRTA*(*k*).

No es difícil ver que HLRTA*(*k*) es un algoritmo completo. El Teorema 2 de [Thorpe, 1994] sobre la completitud de HLRTA* continúa siendo valido, debido a que los valores heurísticos que se guardan, siempre aumentan a medida que se recorre el espacio de estados. Para demostrar la convergencia hacia rutas óptimas, se requiere que h_1 mantenga la admisibilidad después de la

propagación acotada. Lo cual se prueba con el siguiente lema, basado en el Teorema 3 en [Thorpe 1994].

Lema 1 (de [Edelkamp and Eckele, 1997], modificado). Sea $x \in X - G$ y h_1, h_2 y H como en HLRTA*. Si $h_1(x) \leftarrow \max[h_1(x), \min_{v \in Sucs(x)}(c(x, v) + H(v))]$ entonces $h_1(x) \leq h^*(x)$.

Dem. Si $h_1(x) \geq \min_{v \in Sucs(x)}(c(x, v) + H(v))$ no hay nada que probar. Si no, existe una ruta óptima desde x al estado objetivo que pasa por sucesor w. Entonces $h^*(x) = c(x, w) + h^*(w) \geq c(x, w) + H(w)$. Para ver esto, recuerde que $H(w)$ puede ser $h_1(w)$ (en este caso $H(w)$ es obviamente admisible) o $h_2(w)$. En general, $h_2(w)$ no es admisible, excepto cuando la ruta entre x y w nos dice que la última vez que w fue visitado, el algoritmo seleccionó x como siguiente estado. En este caso particular $h_2(w)$ es admisible (Teorema 3 [Thorpe 1994]). En este caso cuando $d(w) = x$, $H(w)$ puede tomar el valor de $h_2(w)$. De esta manera $H(w)$ es admisible a través de la ruta de x a w, por lo que podemos escribir $h^*(x) = c(x, w) + h^*(w) \geq c(x, w) + H(w)$. En particular, esto es cierto para el mínimo $c(x, v) + H(v)$ entre los sucesores de x, esto es, $h^*(x) \geq \min_{v \in Sucs(x)}(c(x, v) + H(v))$. Así $h_1(x) \leq h^*(x)$. q.e.d.

Para que el documento sea auto-contenido, se incluye el teorema que garantiza que h_1 permanece admisible tras la actualización de HLRTA*. Este teorema esta sacado de la tesis de master de Thorpe.

Teorema 3 [Thorpe 1994]. Con arcos de coste positivo y una heurística inicial admisible, la heurística $h_1(n)$ almacenada por HLRTA* nunca sobreestima el coste de alcanzar un estado objetivo desde el estado n, y cada $h_2(n)$ (segundo mejor valor) almacenado por HLRTA* nunca sobreestima el coste de alcanzar un estado objetivo desde n por cualquier ruta que se inicia en n, excluyendo la ruta que contiene el arco desde n al sucesor $d(n)$ (la dirección del movimiento).

Dem. Inicialmente, tenemos una estimación heurística admisible $h_0(n)$ y ningún valor almacenado en $h_1(n)$, $h_2(n)$ y $d(n)$ (la dirección del movimiento). Usaremos esto como base para demostrar por inducción en el número de movimientos.

La hipótesis de inducción es la siguiente: si los valores de $h_1(n)$ y $h_2(n)$ tienen la propiedad de no sobreestimar, antes del movimiento número x de HLRTA*, entonces los valores de $h_1(n)$ y $h_2(n)$ y $h_0(n)$ mantendrán la propiedad de no sobreestimar después de movimiento número x.

Primero se prueba que antes del movimiento número x de HLRTA*, $f(n) = c(z, n) + H(n)$ no sobreestima el coste de llegar al objetivo desde el estado actual z, en una ruta que comienza con el arco desde z a n y no incluye el arco que regresa de n a z.

Lema 3.1 [Thorpe 1994]. Se asume que el coste de los arcos es positivo y que la heurística inicial es admisible. También se asume que $h_1(n)$ es siempre admisible en cualquier ruta desde n y que $h_2(n)$ es admisible por cualquier ruta que excluye el arco desde n a $d(n)$. Dado estos supuestos, $f(n)$ no sobreestima el coste de llegar al objetivo desde el estado actual z, en una ruta que comienza con el arco desde z a n y no incluye el arco que regresa de n a z.

Hay tres maneras de evaluar $f(n)$ en HLRTA*.

1. Si el sucesor n nunca a sido visitado por HLRTA*, $f(n) = c(z, n) + h_0(n)$. La heurística h_0 es admisible y $c(z, n)$ es el coste del arco entre z y n. Por tanto, si n nunca a sido visitado por HLRTA*, $f(n)$ es admisible desde el estado actual z, en una ruta que comienza con el arco entre z y n.

2. Si la dirección de movimiento desde n no es el estado actual $d(n) \neq z$, $f(n) = c(z, n) + h_1(n)$. Por hipótesis de inducción $h_1(n)$ es admisible, por tanto si la dirección de movimiento desde n ($d(n)$) no es el estado

actual z, $f(n)$ es admisible desde el estado actual z, en una ruta que comienza con el arco entre z y n.

3. Si la dirección de movimiento $d(n)$ es el estado actual z, $f(n) = c(z, n) + h_2(n)$. Por hipótesis de inducción $h_2(n)$ no sobreestima el coste de alcanzar la meta desde n por cualquier ruta desde n que excluya el arco desde n a $d(n)$. El coste del arco entre z y n es $c(z, n)$. En este caso $d(n)$ es el estado actual z. Por tanto, si $d(n)$ de n es el estado actual z, $f(n)$ es admisible por cualquier ruta desde el estado actual z que excluya el arco desde n hacia z.

Hemos demostrado que $f(n)$ es admisible antes del movimiento número x, veamos ahora los valores de $h_1(n)$ y $h_2(n)$ después del movimiento número x.

HLRTA* selecciona el sucesor j con menor valor de f entre los sucesores de z, y guarda $f(j)$ como el nuevo $h_1(z)$ antes de realizar el movimiento. La única manera en que $f(j)$ se base en $h_2(j)$ del sucesor j es cuando los valores de $f(n)$ de todos los sucesores de z, son mayor o igual a $f(j)$, y $d(j)=z$. En este caso, $f(j)$ basado en $h_2(j)$ del sucesor seleccionado es admisible desde z, porque $f(j)$ es admisible en cualquier ruta que no contiene el arco entre j y z, y j es el estado con mínimo f entre los sucesores de z.

Debido a que el mínimo $f(n)$ para el estado actual es admisible, la actualización mantiene la admisibilidad de $h_1(z)$.

HLRTA* guarda el sucesor j con mínimo f como la dirección de movimiento $d(z)$ desde el estado actual y selecciona un sucesor k con el mínimo valor de f entre los sucesores restantes de z para guardar $h_2(z)$ con $f(k)$. La única manera en que $f(k)$ pueda estar basado en $h_2(k)$ es cuando el $f(n)$ de todos los sucesores de z, excluyendo j, son mayor o igual a $f(k)$. En este caso, $f(k)$ basado en $h_2(k)$ es admisible desde z excluyendo cualquier ruta que contenga el arco entre z y j. Debido a que $f(k)$ debe ser admisible para el estado actual excluyendo cualquier ruta que contenga el arco entre z y j, la actualización de

$h_2(z)$ es admisible para todos los sucesores excluyendo cualquier ruta que contenga el arco entre z y j. Notar que si j es el único sucesor de z, el valor infinito almacenado en $h_2(z)$ también es admisible debido a que no existe una ruta que no contenga el arco desde z a j.

Por inducción en el número de movimientos realizados por HLRTA* (siempre a través de ejecuciones consecutivas), cada $h_1(n)$ guardado por HLRTA* es admisible desde n, y cada $h_2(n)$ guardado por HLRTA* es admisible desde n por cualquier ruta que excluya el arco entre n y $d(n)$, si la heurística inicial es admisible. q.e.d.

Con estos resultados, la prueba de convergencia a rutas óptimas de HLRTA* (Teorema 5 en [Thorpe, 1994]) es también valida para HLRTA*(k), ya que se mantiene la admisibilidad de los valores heurísticos almacenados en h_1. Como consecuencia, HLRTA*(k) es un algoritmo correcto y completo que converge a rutas óptimas si se ejecuta repetidamente sobre la misma instancia de un problema.

7.3 HLRTA*$_{LS}(k)$

HLRTA*$_{LS}(k)$ [Hernandez y Meseguer, 2007c], es un algoritmo que combina HLRTA* y propagación acotada sobre un espacio local. Esta combinación se justifica porque la propagación en HLRTA*$_{LS}(k)$ es una mejora de la propagación en HLRTA*(k).

La selección del espacio local se hace usando la heurística H de HLRTA*. Esta heurística puede tomar el valor h_1 o h_2, dependiendo de la dirección del movimiento. El proceso de actualización se hace sobre h_1, con los valores de H en la frontera del espacio local. Sabemos que los valores en H son admisibles. Si la heurística inicial es admisible, entonces el proceso de actualización mantiene la admisibilidad [Hernández y Meseguer 2007a].

El algoritmo HLRTA*$_{LS}(k)$ aparece en la Figura 54. La principal diferencia con HLRTA* esta en el procedimiento HLRTA-LS-TRIAL. En el ciclo que itera hasta que el estado actual x sea un estado objetivo (línea 2), se llama a la función Changes? (línea 3). La función retorna verdadero cuando x tiene heurística inexacta o de lo contrario retorna falso. Si retorna verdadero, se selecciona el espacio local de reparación (I,F) con la función SelectLS (línea 4) y se actualiza con el procedimiento UpdateLS (línea 5). Después se actualiza el segundo mínimo del estado actual (línea 6) y se planifica y ejecuta un movimiento (líneas 7-8).

No es difícil ver que HLRTA*$_{LS}(k)$ hereda las propiedades de HLRTA*(k): el algoritmo es correcto y completo. Además, debido a que se mantiene la admisibilidad de la heurística, se asegura la convergencia a soluciones óptimas.

```
procedure HLRTA-LS (k) (X, A, c, s0, G, k)
1  for each x ∈ X do h1(x) ← h0(x); h2(x) ← h0(x); d(x) ← null;
2  repeat
3     HLRTA-LS-trial(X, A, c, s0, G, k);
4  until h1 does not change;

procedure HLRTA-LS-trial(X, A, c, s0, G, k)
1  x ← s0;
2  while x ∉ G do
3     if Changes?(x) then
4        (I, F) ← SelectLS(x, k);
5        UpdateLS(I, F);
6     HLRTA-LookaheadUpdate2min(x);
7     y ← argmin_{w∈Succ(x)} [c(x,w) + H(w,x)];
8     execute(a ∈ A such that a = (x, y));
9     d(x) ← y;   x ← y;

function SelectLS(x, k): pair of sets;
1  Q ← ⟨x⟩; F ←∅; I ←∅; cont ← 0;
2  while Q ≠ ∅ ∧ cont <k do
3     v ← extract-first(Q);
4     y ← argmin_{w∈Succ(v)} [c(v, w) + H(w, v)];
5     if h1(v) < c(v, y) + H(y, v) then
6        I ← I ∪ {v}; cont ← cont + 1;
7        for each w ∈ Succ(v) do
8           if w ∉ I ∧ w ∉ Q then Q ← add-last(Q,w);
9     else F ←F ∪ {v};
10 if Q ≠ ∅ then F ←F ∪ Q;
11 return (I, F);

procedure UpdateLS(I, F)
1  while I ≠ ∅ do
2     (i, f) ← argmin_{v∈I, w∈F, w∈Succ(v)} c(v, w) + H(w,v);
3     h1(i) ← max[h1(i), c(i, f) + H(f, i)];
4     I ←I - {i};
5     F ←F ∪ {i};

procedure HLRTA-LookaheadUpdate2min(x)
1  z ← arg2ndmin_{v∈Succ(x)}[c(x,v) + H(v,x)];
2  if h2(x) < c(x, z) + H(z,x) then h2(x) ← c(x, z) + H(z,x);

function Changes?(x): boolean;
1  y ← argmin_{w∈Succ(x)} [c(v,w) + H(w,v)];
2  if h1(x) < c(x, y) + H(y,x) then return true; else return false;

function H(v, from): real;
1  if d(v) = from then return max{h1(v),h2(v)}; else return h1(v);
```

Figura 54. Algoritmo HLRTA*$_{LS}(k)$.

7.4 HLRTA*$_{LS}(k,d)$

HLRTA*$_{LS}(k,d)$ [Hernandez y Meseguer, 2007b], es un algoritmo que implementa en HLRTA*, anticipación con propagación acotada sobre un espacio local. Este algoritmo fue pensado inicialmente para calcular una cantidad de movimientos fija (lo que se usaba como límite para la cantidad de anticipación). En el capítulo 6 analizamos distintas formas de hacer anticipación y ejecutar movimientos. Como resultado del análisis elegimos el mecanismo de anticipación dinámica y la estrategia de movimiento dinámico (la versión din-din) la más conveniente. Debido a esto, en esta sección desarrollaremos la versión din-din del algoritmo HLRTA*$_{LS}(k,d)$.

Se presenta el algoritmo HLRTA*$_{LS}(k, d)$ en la Figura 55. Los procedimientos y funciones que no aparecen, se pueden ver en la Figura 54. Las diferencias principales con la versión din-din de LRTA*$_{LS}(k, d)$, son el uso de las dos heurísticas y el almacenaje de la dirección del movimiento.

Como la heurística siempre aumenta, se garantiza que el algoritmo es completo. Si la heurística inicial es admisible, la actualización del espacio local mantiene la admisibilidad, por tanto la convergencia a rutas óptimas esta garantizada. Es fácil ver que HLRTA*$_{LS}(k, d)$ hereda las buenas propiedades de HLRTA*$_{LS}(k)$ y HLRTA*.

```
procedure HLRTA-LS(k, d) (X, A, c, s0, G, k, d)
1  for each x ∈ X do h1(x) ← h0(x); h2(x) ← h0(x); d(x) ← null;
2  repeat
3     HLRTA-LS-trial(X, A, c, s0, G, k, d);
4  until h1 does not change;

procedure HLRTA-LS-trial(X, A, c, s0, G, k, d)
1  x ← s0;
2  while x ∉ G do
3     (path, C) ← A*(x, d, G);
4     if C ≠ ∅ then
5        (I,F) ← SelectLS(k, C);
6        UpdateLS(I,F);
7        HLRTA-LookaheadUpdate2min(x);
8        y ← argmin_{w∈Succ(x)} [c(x,w) + H(w,x)];
9        execute(a ∈ A such that a = (x, y));
10       d(x) ← y;   x ← y;
11    else
12       x ← extract-first (path);
13       while path ≠ ∅ do
14          HLRTA-LookaheadUpdate2min(x);
15          y ← extract-first (path);
16          execute(a ∈ A such that a = (x, y));
17          d(x) ← y; x ← y;


function SelectLS (k, C): pair of sets;
16Q ← ∅; F ← ∅; I ← ∅; cont ← 0;
17while C ≠ ∅ do
18   p ← extract-first(C);
19   if p ∉ I ∧ Q = ∅ ∧ cont < k then Q ← {p};
20   while Q ≠ ∅ ∧ cont < k do
21      v ← extract-first(Q);
22      y ← argmin_{w∈Succ(v) ∧ w ∉ I} H(w, v) + c(v, w);
23      if h1(v) < c(v, y) + H(y, v) then
24         I ←I ∪ {v};
25         cont ← cont + 1;
26      for each w ∈ Succ(v) do
27         if w ∉ I ∧ w ∉ Q then Q ← add-last(Q,w);
28      else F ←F ∪ {v};
29 if Q ≠ ∅ then F ←F ∪ Q;
30 return (I, F);
```

Figura 55. Algoritmo HLRTA*$_{LS}$(*k*, *d*) versión din-din.

7.5 Resultados Experimentales

En esta sección, presentamos resultados para HLRTA*(*k*), HLRTA*$_{LS}$(*k*) y HLRTA*$_{LS}$(*k*, *d*) versión din-din (en adelante HLRTA*$_{LS}$(*k*, *d*)). Primero comparamos los resultados de HLRTA*(*k*) con los de HLRTA*$_{LS}$(*k*), luego comparamos HLRTA*$_{LS}$(*k*) con HLRTA*$_{LS}$(*k*, *d*) y finalmente vemos el

rendimiento de HLRTA*$_{LS}$(k, d) v/s LRTA*$_{LS}$(k, d). En HLRTA*$_{LS}$(k, d) y LRTA*$_{LS}$(k, d) usamos la versión di-din.

7.5.1 Comparando HLRTA(k) con HLRTA*$_{LS}$(k)*

La propagación sobre un espacio local (capítulo 5) es una mejora a la iteración de valores (capítulo 4). En esta sección queremos confirmar esta apreciación sobre HLRTA*. Para ello compararemos coste y tiempo total obtenido con las versión 1 y 2 de HLRTA*(k) y HLRTA*$_{LS}$(k) en primera ejecución y convergencia en Cua-35 y Lab-acic.

En la Figura 56 podemos observar los resultados en la primera ejecución. En Cua-35, HLRTA*$_{LS}$(k) mejora en coste a HLRTA*(k) en todos los valores de k con la versión 1 y 2. Respecto al tiempo total de búsqueda, HLRTA*(k) obtiene un tiempo menor que HLRTA*$_{LS}$(k) con la versión 1, esto se explica porque la diferencia en coste en esta versión es pequeña y el mecanismo de propagación de HLRTA*$_{LS}$(k) es más computacionalamente más costoso que el de HLRTA*(k). Con la versión 2, HLRTA*$_{LS}$(k) obtiene un mejor tiempo que HLRTA*(k) (excepto con k=5). Las gráficas nos permiten apreciar que HLRTA*(k) y HLRTA*$_{LS}$(k) obtienen soluciones de mejor calidad con la versión 2 para k grandes. Con k pequeño las soluciones son similares en ambas versiones. El lab-acic sucede algo peculiar relacionado con la estructura de los laberintos acíclicos y el comportamiento de HLRTA* con la distancia manhattan como heurística inicial. Una de las mejores estrategias conocidas para "salir" de un laberinto es avanzar marcando el camino que se recorre para evitar pasar nuevamente por este. Sólo se puede avanzar por el camino recorrido cuando se entra a un pasillo sin salida para volver atrás. HLRTA* imita esta estrategia en laberintos acíclicos. Es fácil ver que un laberinto acíclico con la distancia manhattan como heurística inicial, cuando el agente llega al estado x desde el estado y, el $H(y)$ será mayor que el H de todos los sucesores de x que no se han visitado, por tanto el agente seleccionará uno de los estados no visitados como próximo estado a visitar (recuerde que el coste de las acciones es 1). Además, cuando el agente detecta un pasillo sin salida

vuelve atrás actualizando h_2 con infinito. Esto garantiza que el agente no vuelva a entrar en este porque verá *H* igual a infinito en el estado de la entrada del pasillo. Ahora si en HLRTA* incluimos propagación, se seguirá imitando la estrategia para salir de laberintos siempre y cuando la propagación se realice sobre estados visitados. En los resultados para Lab-acic podemos apreciar esto, vemos que el coste con la versión 1 de HLRTA*(*k*) y HLRTA*$_{LS}$(*k*) es el mismo para todos los valores de *k*. Con la versión 2 el coste empeora en ambos algoritmos y no mejora cuando *k* crece (empeora). El coste empeora con la versión 2, porque en algunos casos el agente puede volver por el camino recorrido. Esto sucede porque en un estado en concreto, el *H* de los sucesores no visitados puede ser mayor que el *H* de los sucesores visitados producto de la propagación. Respecto al tiempo total de búsqueda, crece a medida que *k* aumenta y es similar en los dos algoritmos en los dos versiones, apreciando una leve ventaja en HLRTA*(*k*).

En la Figura 57 podemos observar los resultados en convergencia. Podemos apreciar que HLRTA*$_{LS}$(*k*) obtiene un coste y tiempo total menor para todos los *k* censados en los dos escenarios con las dos versiones (excepto con *k* pequeño en Cua-35 versión 1) y que las diferencias entre los resultados obtenidos con los algoritmos son mayores en la versión 2.

Estos resultados nos permiten concluir que el afecto de la propagación sobre HLRTA* es similar al efecto sobre LRTA*. Podemos observar que el rendimiento de HLRTA*$_{LS}$(*k*) (con propagación acotada sobre un espacio local) mejora el de HLRTA*(*k*) (con propagación con iteración de valores), como sucede con LRTA*$_{LS}$(*k*) respecto a LRTA*$_{LS}$(*k*). Con respecto al tiempo por paso, este aumenta como podemos ver en la Figura 58 (tiempo por paso en Cua-35 en la primera ejecución con la versión 1). Un caso particular es el comportamiento de los algoritmos en laberintos. HLRTA*(*k*) y HLRTA*$_{LS}$(*k*) heredan de HLRTA* el excelente rendimiento en laberintos acíclicos en la primera ejecución.

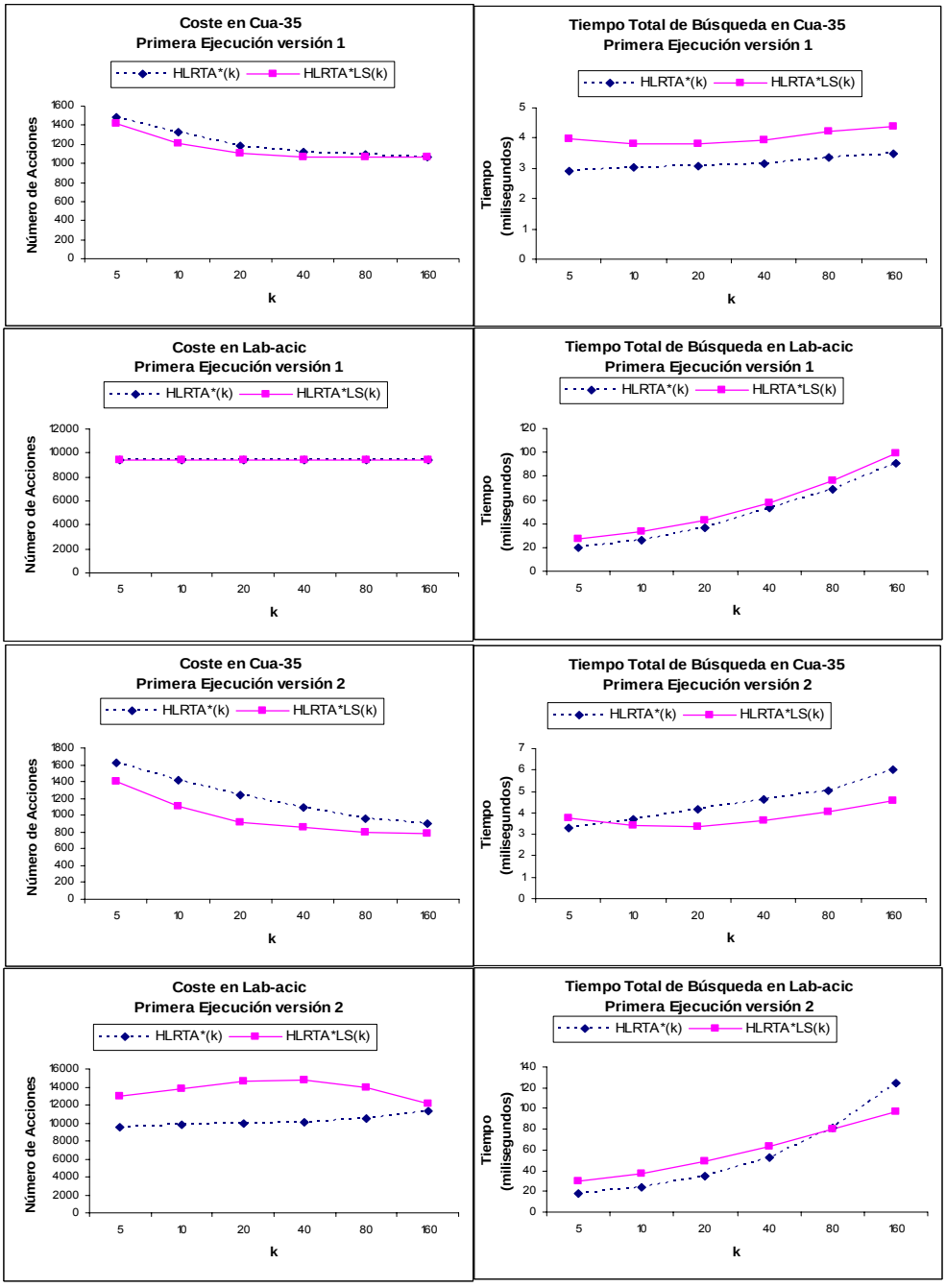


Figura 56. Resultados experimentales en Cua-35 y Lab-acic. Presentamos tiempo total y coste con la versión 1 y 2 de HLRTA*(*k*) y HLRTA*$_{LS}$(*k*) para distintos valores de *k* en la primera ejecución.

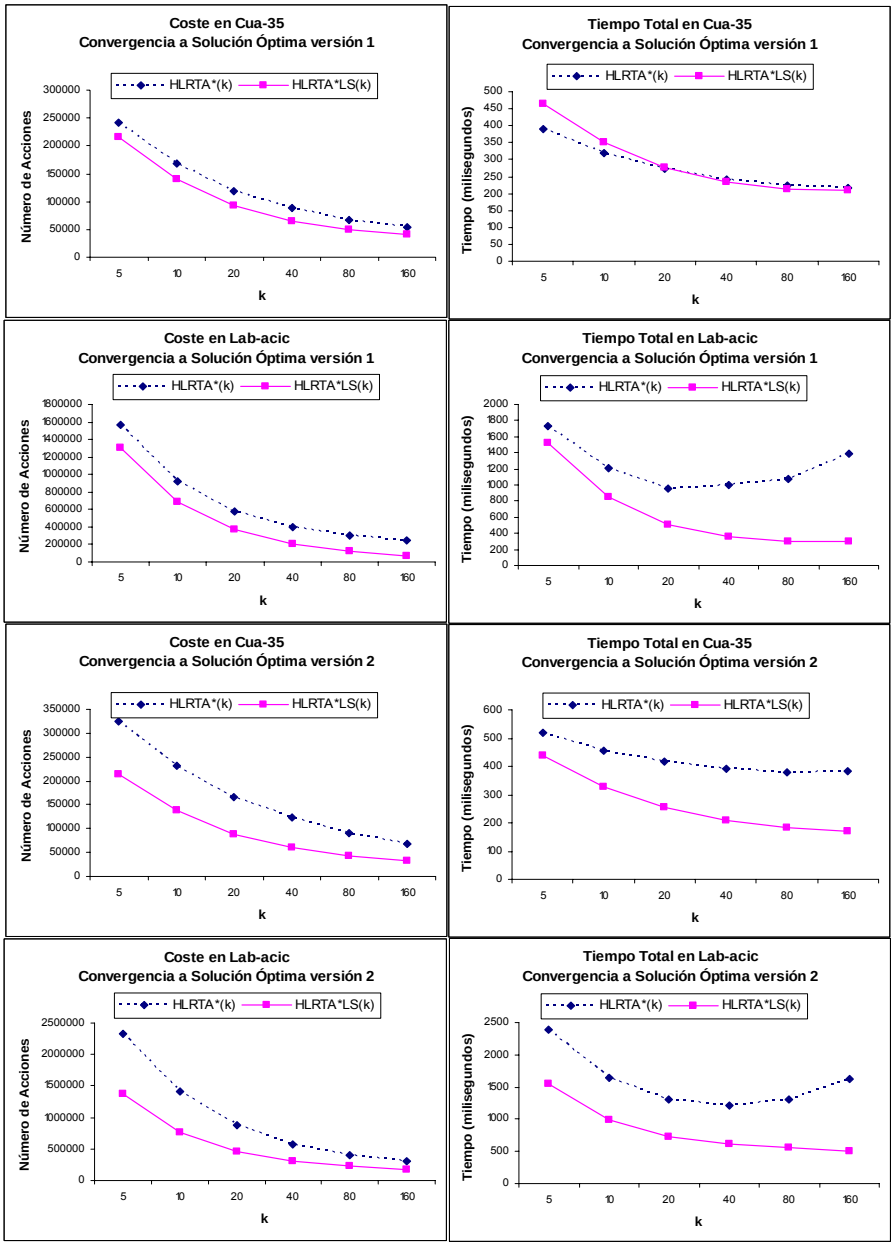


Figura 57. Resultados experimentales en Cua-35 y Lab-acic. Presentamos tiempo total y coste con la versión 1 y 2 de HLRTA*(k) y HLRTA*$_{LS}$(k) para distintos valores de k en convergencia.

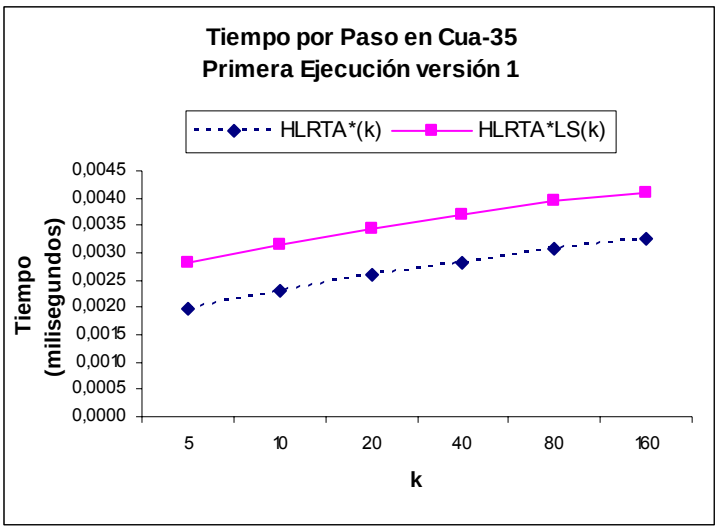


Figura 58. Resultados experimentales en Cua-35. Presentamos el tiempo por paso con la versión 1 HLRTA*(*k*) y HLRTA*$_{LS}$(*k*) para distintos valores de *k* en primera ejecución.

7.5.2 Comparando HLRTA$^*_{LS}$*(k) con HLRTA*$^*_{LS}$*(k, d)*

La posibilidad de calcular varios movimientos por episodio de planificación, combinada con anticipación ha supuesto una mejora en coste aunque no en tiempo en LRTA*. Para confirmar este efecto sobre este algoritmo, comparamos HLRTA*$_{LS}$(*k*) con HLRTA*$_{LS}$(*k*, *d*). Para realizar la comparación consideramos HLRTA*$_{LS}$(*k*, *d*=1) como HLRTA*$_{LS}$(*k*). Los valores de *k* censados son 10, 40, y 160, los valores de *d* son 1, 2, 4, 8, 16 y 32. Veremos el coste de obtener una solución y tiempo total en primera ejecución y convergencia. Usaremos Cua-35 como escenario de pruebas, consideramos que es suficiente para ilustrar las diferencias en el rendimiento de cada algoritmo.

Los resultados aparecen en la Figura 59. Podemos apreciar que el coste disminuye a medida que *k* crece en primera ejecución y convergencia. También podemos ver que el tiempo total aumenta cuando que *k* crece en primera ejecución y convergencia. Esto nos indica que HLRTA*$_{LS}$(*k*, *d*) obtiene soluciones de mejor calidad que HLRTA*$_{LS}$(*k*) para un mismo valor de *k* cuando *d* crece, pero con un tiempo total mayor. En la Figura 60 podemos ver que el tiempo por paso también aumenta cuando *k* crece. Estos

resultados nos indican que la relación entre HLRTA*$_{LS}(k)$ y HLRTA*$_{LS}(k, d)$, es similar a la de LRTA*$_{LS}(k)$ con LRTA*$_{LS}(k, d)$.

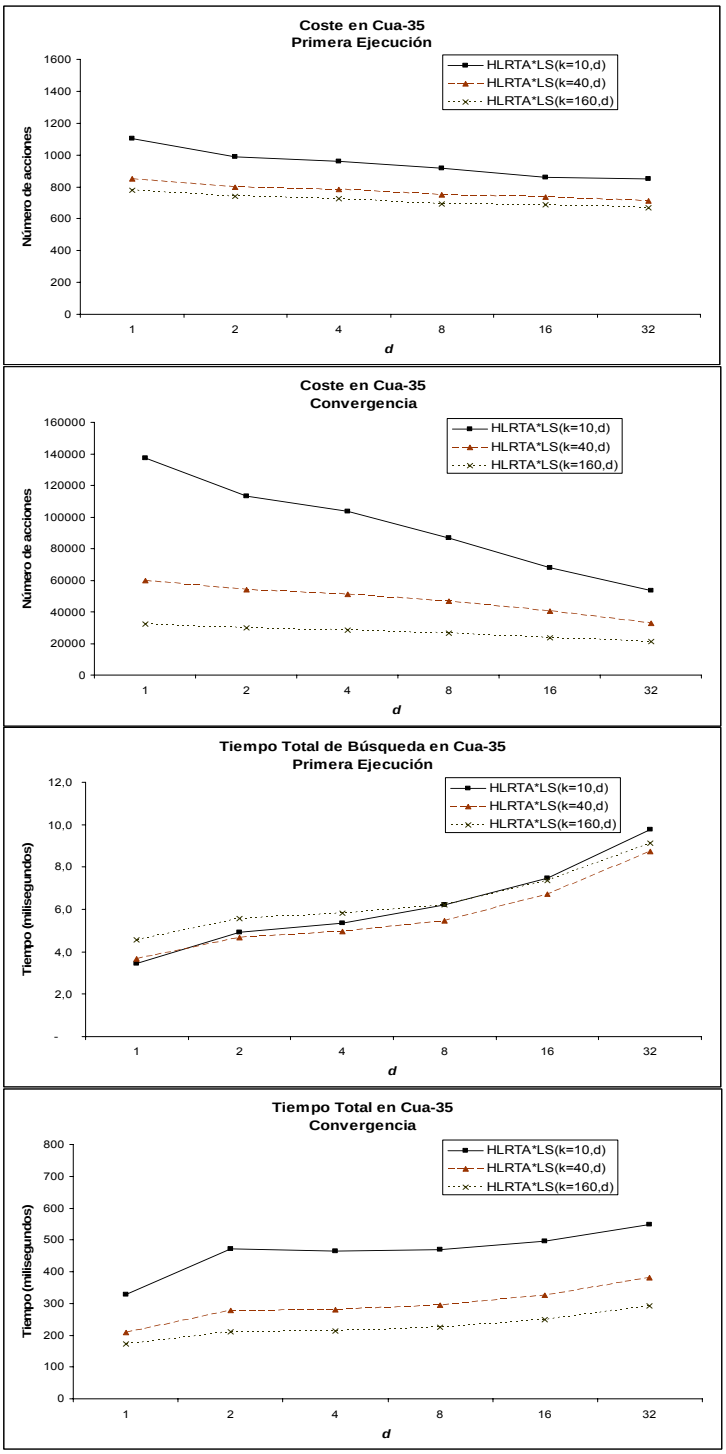


Figura 59. Resultados en Cua-35. Presentamos el coste y el tiempo total en primera ejecución y convergencia con HLRTA*$_{LS}(k,d)$.

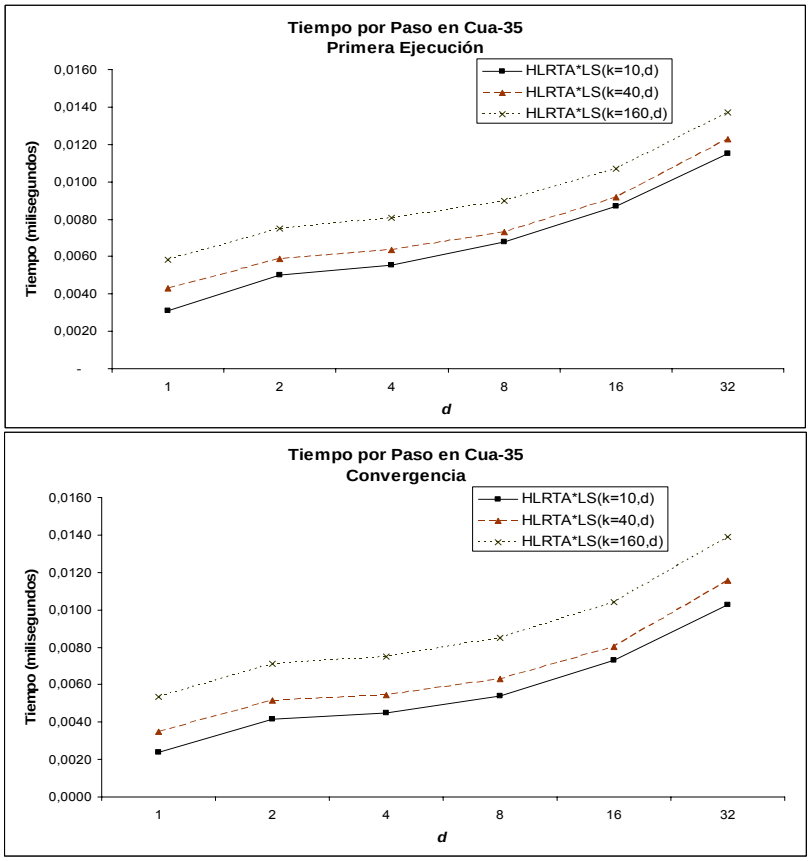


Figura 60. Resultados en Cua-35. Presentamos el tiempo por paso en primera ejecución y convergencia con HLRTA*$_{LS}$(k,d).

7.5.3 Comparando HLRTA$_{LS}$(k, d) con LRTA*$_{LS}$(k, d)*

HLRTA* es un algoritmo más informado que LRTA*, y por tanto sería esperable observar un mejor rendimiento. Comparamos HLRTA*$_{LS}$(k,d) con LRTA*$_{LS}$(k,d) en Cua-35 y Lab-acic. Los valores de k censados son 10, 40, y 160 los valores de d son 2, 4, 8, 16 y 32.

En la Figura 61, podemos ver los resultados para el coste en primera ejecución y convergencia. En primera ejecución el coste obtenido con HLRTA*$_{LS}$(k,d) es similar al de LRTA*$_{LS}$(k,d) en Cua-35. La curvas resultantes son muy parecidas, sólo con k = 10 HLRTA*$_{LS}$(k,d) es levemente mejor que LRTA*$_{LS}$(k,d) (excepto con d = 8) y con d = 32 en los otros valores de k. En Lab-acic HLRTA*$_{LS}$(k,d) es mejor que LRTA*$_{LS}$(k,d) en todos los valores de k

y *d*. En convergencia sucede algo similar en Cua-35 no hay mucha diferencia enre los algoritmos, en cambio en Lab-acic HLRTA*$_{LS}$(*k*,*d*) es mejor que LRTA*$_{LS}$(*k*,*d*) en todos los valores de *k* y *d* censados.

En la Figura 62 podemos ver el tiempo total en primera ejecución y convergencia. En Cua-35 no hay mucha diferencia en el tiempo obtenido en primera ejecución y convergencia. Sólo con *k* = 10 LRTA*$_{LS}$(*k*,*d*) es un poco mejor que HLRTA*$_{LS}$(*k*,*d*). El Lab-acic HLRTA*$_{LS}$(*k*,*d*) obtiene un tiempo menor que LRTA*$_{LS}$(*k*,*d*) en todos los valores de *k* y *d* en primera ejecución y convergencia.

En conclusión, HLRTA*$_{LS}$(*k*,*d*) es siempre mejor que LRTA*$_{LS}$(*k*,*d*) en Lab-acic en primera ejecución y convergencia. En Cua-35 el rendimiento es similar. Esto se debe a la estructura del problema. HLRTA*$_{LS}$(*k*,*d*) hereda el buen desempeño de HLRTA* en laberintos.

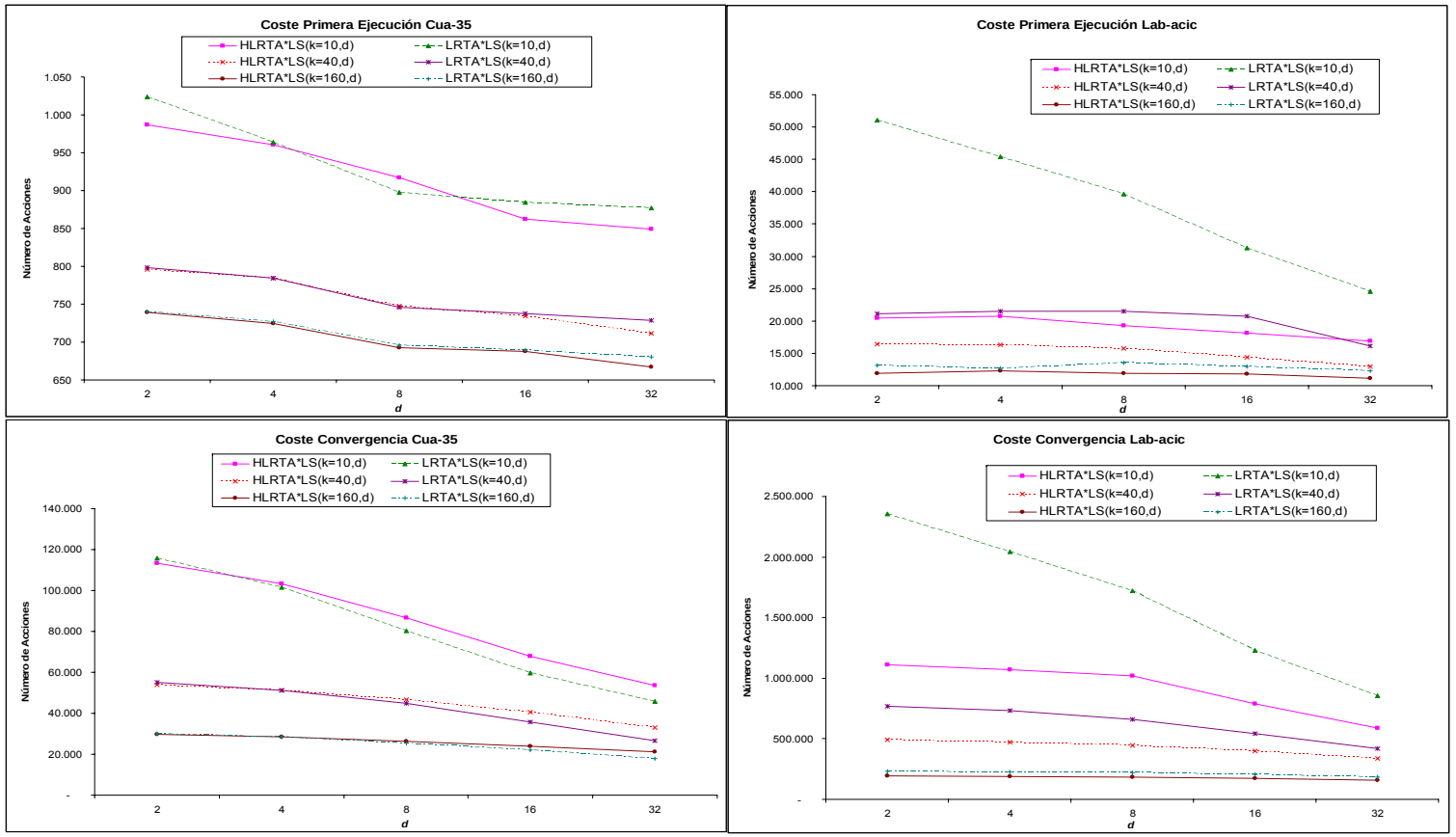


Figura 61. Resultados en Cua-35 y Lab-acic. Presentamos el coste en primera ejecución y convergencia con HLRTA*$_{LS}(k,d)$ y LRTA*$_{LS}(k,d)$.

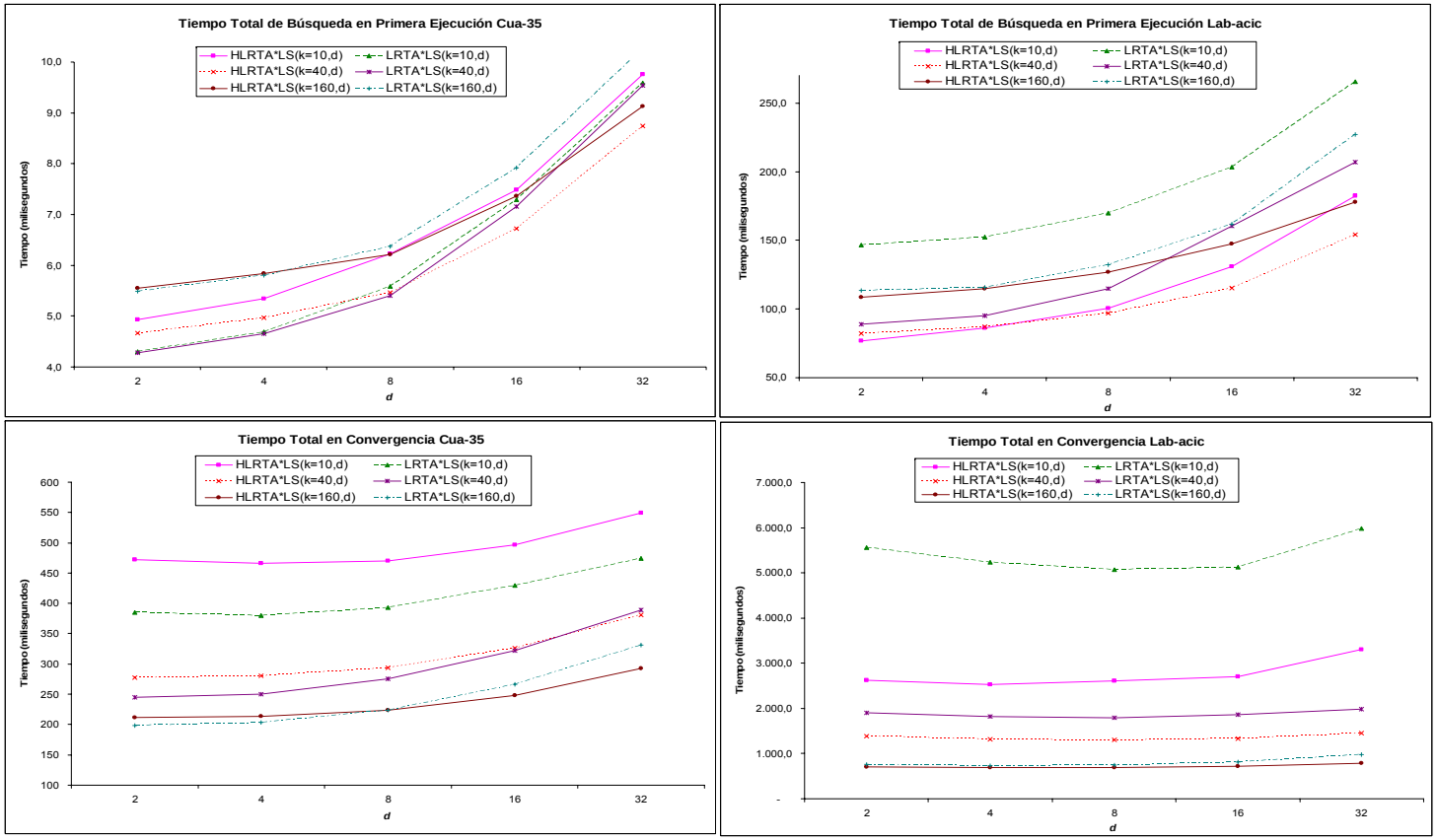


Figura 62. Resultados en Cua-35 y Lab-acic. Presentamos el tiempo total en primera ejecución y convergencia con HLRTA*$_{LS}(k,d)$ y LRTA*$_{LS}(k,d)$.

7.5.4 Conclusiones

El efecto de la propagación en HLRTA*(*k*) y HLRTA*$_{LS}$(*k*) es similar al efecto en LRTA*(*k*) y LRTA*$_{LS}$(*k*) en primera ejecución. En la mayoría de los casos HLRTA*$_{LS}$(*k*) mejora el rendimiento de HLRTA*(*k*) debido a al mecanismo de propagación que utiliza cada uno. HLRTA*$_{LS}$(*k*, *d*) obtiene soluciones de mejor calidad que HLRTA*$_{LS}$(*k*), pero con un mayor tiempo paso. El algoritmo HLRTA*$_{LS}$(*k*,*d*) sólo mejora a LRTA*$_{LS}$(*k*,*d*) en Lab-acic, en Cua-35 tiene un desempeño similar, por tanto en problemas en donde la heurística inicial es buena, sería conveniente usar LRTA*$_{LS}$(*k*,*d*) porque es un algoritmo más sencillo de implementar y utiliza menos información por estado. En laberintos es donde se aprecia un mejor rendimiento de los algoritmos presentados, debido a la estructura del problema y al comportamiento que heredan de HLRTA*.

7.6 RTA*$_{LS}$(*k*)

El algoritmo RTA* (descrito en la sección 3.3.1) exhibe un buen desempeño en la primera ejecución [Furcy y Koenig 2001] [Shimbo y Ishida, 2003] respecto a LRTA* y HLRTA*. Como la propagación mejora el desempeño de LRTA* y HLRTA* en primera ejecución, es de esperar que suceda lo mismo al aplicarla sobre RTA* y que mantenga su ventaja sobre los otros algoritmos. En esta sección presentamos RTA*$_{LS}$(*k*), un algoritmo que combina RTA* con propagación acotada sobre un espacio local. Como se ha visto en capítulos anteriores, este mecanismo de propagación trabaja con heurísticas admisibles. Debido a que RTA* puede sobreestimar la heurística, proponemos realizar la combinación de la siguiente manera: usamos dos heurísticas h_1 y h_2. En un comienzo se inicializan con h_0, una heurística inicial admisible. Supongamos que x es el estado actual. En cada episodio de planificación, primero se ejecuta la propagación acotada desde x sobre h_1 (como LRTA*$_{LS}$(*k*) en la sección 5.5), luego el agente actualiza $h_2(x)$ con el segundo mínimo de $c(x, y) + \max(h_1(y), h_2(y))$, $y \in \text{succ}(x)$. Esta actualización de $h_2(x)$ es más agresiva que la de RTA*, ya que se puede usar $h_1(y)$ en lugar de

$h_2(y)$ cuando $h_1(y) > h_2(y)$ por causa de la propagación. Después el agente se mueve al estado z con mínimo valor $c(x, z) + \max(h_1(y), h_2(z))$, $z \in \text{succ}(x)$. Note que las únicas diferencias con RTA* son: propagar sobre h_1 en cada episodio de planificación y usar el máximo entre $h_1(y)$ y $h_2(y)$ en lugar de $h_2(y)$, cuando se actualiza $h_2(x)$ y se ejecuta el movimiento.

```
procedure RTA-LS(X, A, c, s0, G, k)
 7  for each x ∈ X do h1(x) ← h0(x); h2(x) ← h0(x);
 8  x ← s0;
 9  while x ∉ G do
10     SelectUpdateLS(x, k);
11     h2(x) ← max[h2(x), secondmin_{n∈Succ(x)} [max[h1(n), h2(n)] + c(x,n)]];
12     y ← argmin_{n∈Succ(x)} [max[h1(n), h2(n)] + c(x,n)];
13     execute(a ∈ A such that a = (x, y));
14     x ← y;
```

Figura 63. Algoritmo RTA*$_{LS}(k)$.

En la Figura 63 podemos apreciar el algoritmo RTA*$_{LS}(k)$. Al comienzo se inicializan las heurísticas (línea 1). Luego se inicializa x con el estado inicial y se realiza el siguiente ciclo hasta que encontrar el estado objetivo (líneas 2-3): primero se ejecuta la propagación acotada desde el estado actual con el mismo procedimiento de LRTA*$_{LS}(k)$ en la sección 5.5 sobre h_1 (línea 4). Después, se actualiza $h_2(x)$ (línea 5) y se selecciona y el siguiente estado a visitar (línea 6). Finalmente se ejecuta la acción y se cambia de estado (líneas 7-8), y el ciclo continúa iterando.

Debido a que la heurística siempre aumenta, es fácil ver que en un espacio de estados finito con costes de arcos positivo, valores heurísticos positivos y en donde se puede alcanzar el estado objetivo desde cualquier estado, RTA*$_{LS}(k)$ es correcto y completo (Teorema 1 y 2 en [Korf, 1990]).

7.6.1 Resultados experimentales

En esta sección, comparamos los resultados obtenidos con versión 1 y 2 de RTA*$_{LS}(k)$ usando como referencia a HLRTA*$_{LS}(k)$ en Cua-35 y Lab-cic. Presentamos resultados de coste y tiempo total de búsqueda en la primera

ejecución con diferentes valores de *k*. Los valores de *k* censados son 1, 5, 10, 20, 40, 80 y 160.

En la Figura 64 podemos apreciar los resultados en Cua-35. En ambas versiones el coste de RTA*$_{LS}(k)$ es menor que el de HLRTA*$_{LS}(k)$ sólo con $k = 1$. Con $1 < k < 80$ HLRTA*$_{LS}(k)$ tiene menor coste, con *k* grande el coste es similar. Con *k* pequeño RTA*$_{LS}(k)$ tiene menor tiempo, con otros valores de *k* el tiempo es similar. La ventaja de RTA* (RTA*$_{LS}(k=1)$) sobre HLRTA* (HLRTA*$_{LS}(k=1)$) en la primera ejecución, no se mantiene con $k > 1$ en Cua35. La propagación no tiene un efecto tan positivo sobre RTA* como lo tiene sobre HLRTA*. Sólo con *k* grande el efecto de la propagación en ambos algoritmos es similar.

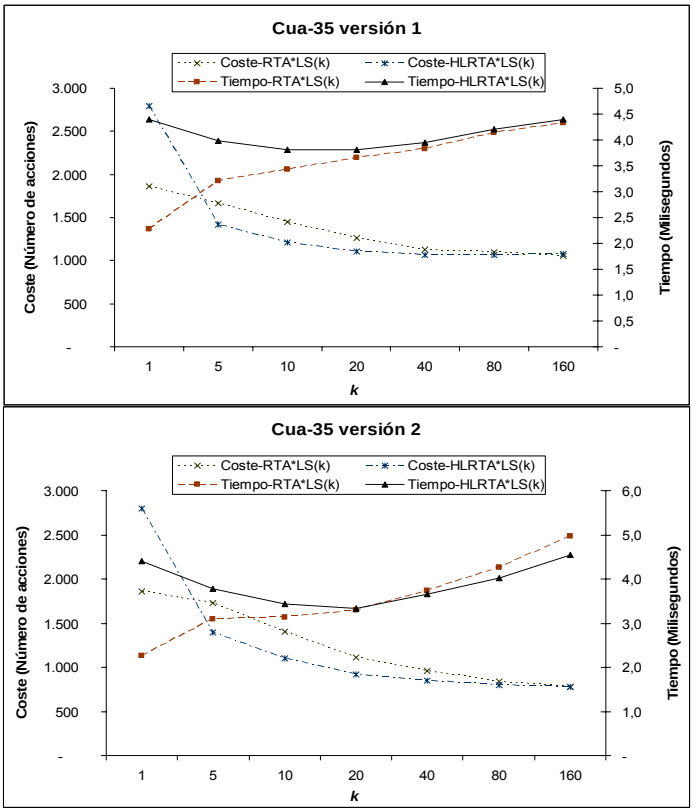


Figura 64. Resultados experimentales en Cua-35. Presentamos el coste y el tiempo total de búsqueda con la versión 1 y 2 de RTA*$_{LS}(k)$ y HLRTA*$_{LS}(k)$, para distintos valores de *k* en primera ejecución.

En la Figura 65 vemos los resultados en Lab-cic. Con k = 20, RTA*$_{LS}(k)$ y HLRTA*$_{LS}(k)$ tienen un desempeño similar (en coste y tiempo) en ambas versiones. Con k < 20, RTA*$_{LS}(k)$ tiene un desempeño cada vez mejor que HLRTA*$_{LS}(k)$ cuando k disminuye. Con k > 20, HLRTA*$_{LS}(k)$ tiene un desempeño un poco mejor que HLRTA*$_{LS}(k)$ cuando k crece. En este escenario de pruebas, el efecto de la propagación tampoco es muy positivo porque el coste obtenido por RTA*$_{LS}(k$=1) mejora poco cuando k crece y el tiempo total de búsqueda aumenta considerablemente en las dos versiones.

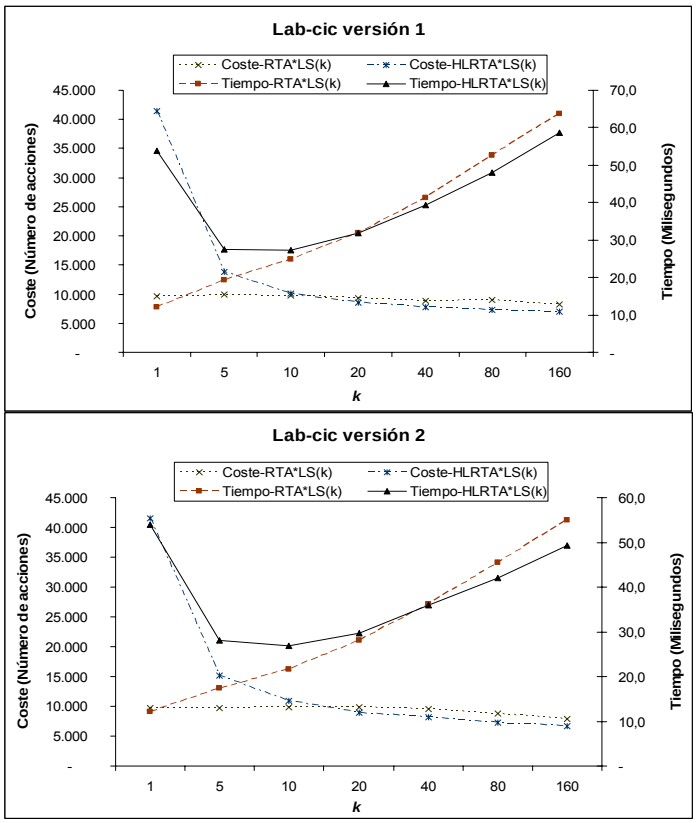


Figura 65. Resultados experimentales en Lab-cic. Presentamos el coste y el tiempo total de búsqueda con la versión 1 y 2 de RTA*$_{LS}(k)$ y HLRTA*$_{LS}(k)$, para distintos valores de k en primera ejecución.

En conclusión, los resultados experimentales nos muestran que el efecto de la propagación sobre RTA* no es tan positivo como sobre LRTA* y HLRTA*. Podemos apreciar que RTA* tiene un buen coste y tiempo el laberintos, y que al combinar con propagación el coste mejora poco y el tiempo empeora. En

Cua-35 RTA* también tiene un buen rendimiento. Al combinar con propagación, RTA*$_{LS}(k)$ tiene un peor desempeño que HLRTA*$_{LS}(k)$.

7.7 Resumen

En este capítulo describimos el comportamiento del algoritmo HLRTA*, combinado con los métodos de propagación presentados en los capítulos anteriores. Esto ha generado tres algoritmos: HLRTA*(k), HLRTA*$_{LS}(k)$ y HLRTA*$_{LS}(k,d)$ versión din-din.

En la sección de resultados experimentales se puede apreciar que el efecto de la propagación sobre HLRTA* genera unos beneficios similares a los obtenidos sobre LRTA*. También se puede ver que HLRTA*$_{LS}(k)$ mejora HLRTA*(k) en la mayoría de los casos y que HLRTA*$_{LS}(k, d)$ obtiene soluciones de mejor calidad que HLRTA*$_{LS}(k)$, pero con un mayor tiempo paso. El rendimiento de HLRTA*$_{LS}(k,d)$ es mejor que el rendimiento de LRTA*$_{LS}(k,d)$ en Lab-acic y similar en Cua-35. Los algoritmos presentados en este capítulo tienen muy buen desempeño en laberintos porque heredan el comportamiento de HLRTA*.

En los resultados experimentales del algoritmos RTA*$_{LS}(k)$, se puede ver que la propagación acotada combinada con RTA* (a diferencia de LRTA* y HLRTA*), no ofrece una mejora significativa sobre el rendimiento.

Capítulo ocho

8 APLICACIÓN

En este capítulo evaluamos los algoritmos presentados en este documento en mapas de juegos de estrategia en tiempo real para ordenador. Los mapas utilizados fueron extraídos desde los juegos comerciales Baldur's Gate [BioWare Corp., 1998] y WarCraft III [Blizzard Entertainment., 2002]. Primero describimos el dominio de la aplicación y los mapas que se utilizan y después realizamos un análisis experimental en donde comparamos el resultado de nuestros métodos con los obtenidos por los algoritmos más relevantes en búsqueda heurística en tiempo real.

8.1 Búsqueda de rutas para un agente en mapas de juegos

En juegos para ordenador en tiempo real los personajes necesitan buscar una ruta desde el estado inicial al objetivo en terreno que inicialmente desconocen [Koenig 2004] [Bulitko y Lee 2006]. Los personajes pueden censar cierta porción del entorno alrededor de su posición actual y recordarlo para un uso posterior. Una práctica común en juegos es modelar el terreno en una cuadrícula bidimensional constituida de celdas que tienen un estatus libre o bloqueado [Bjornsson et. al 2003]. En la cuadrícula, el agente debe ir desde una coordenada (x, y) a un estado objetivo (x_{obj}, y_{obj}). Como el agente desconoce a-priori cuales celdas están bloqueadas mas allá de su rango de visibilidad, asume que las celdas que no ha censado están libres (suposición de espacio libre). El agente puede realizar 8 movimientos desde una celda (cuadrícula 8-conectada). Se puede mover hacia cualquier estado en los cuatro puntos cardinales o hacia cualquiera en diagonal En coste de ir de una celda a otra tiene coste 1. La heurística inicial de una celda (x, y) respecto al estado objetivo (x_{obj}, y_{obj}) es $h(x, y) = \max(\text{abs}(x_{obj} - x), \text{abs}(y_{obj} - y))$ [Koenig y Likhachev, 2005]. En rango de visibilidad que usaremos en nuestros experimentos es 1, es decir el agente sólo puede censar sus sucesores inmediatos. A continuación describimos los mapas utilizados.

8.1.1 Mapas de Baldur's Gate

Los mapas de Baldur's Gate utilizados se pueden ver en la Figura 66. Los mapas 1, 2, 3, 4 y 5 tienen 2.765, 7.637, 13.765, 14.098 y 16.142 estados libres, respectivamente. El agente sólo puede recorrer el terreno de color blanco, el color negro corresponde a obstáculos. Generamos 2.000 ejemplares para cada mapa de manera aleatoria, asegurando que exista una ruta entre el estado inicial y objetivo. Los resultados que se presentan son promedios sobre 10.000 ejemplares.

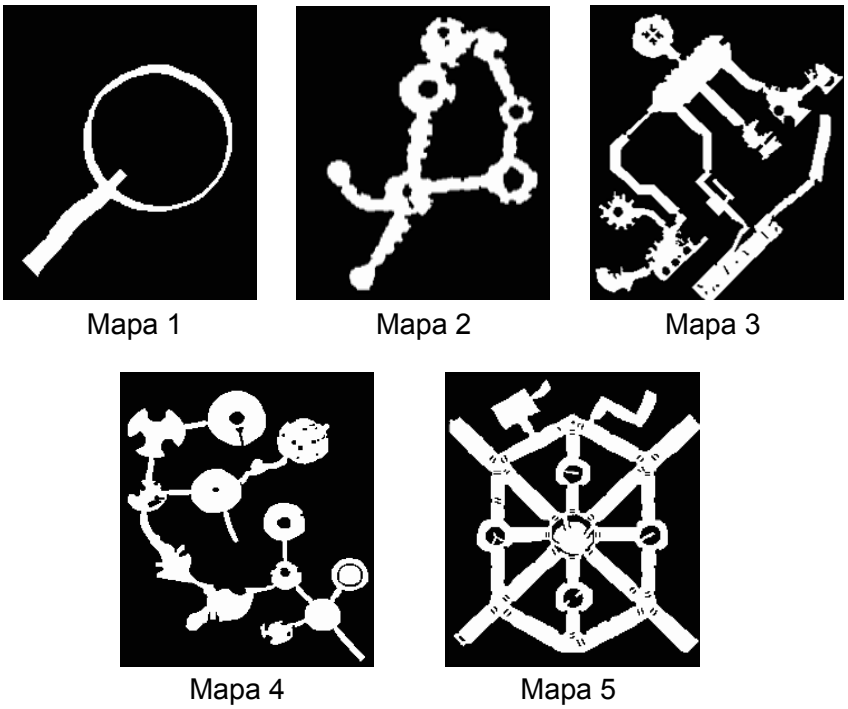
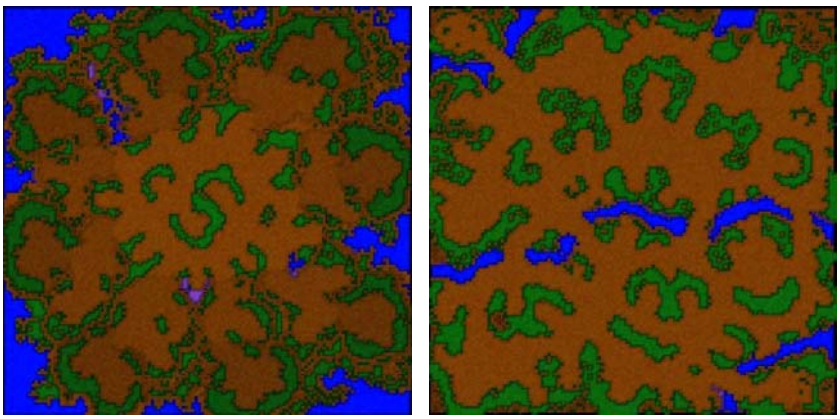


Figura 66. Mapas de Baldur's Gate utilizados en el análisis experimental.

8.1.2 Mapas de WarCraft III

Los mapas de WarCraft III utilizados se pueden ver en la Figura 67. Los mapas 1, 2 y 3 tienen 10.242, 10.691 y 19.253 estados libres, respectivamente. El terreno en los mapas contiene lagos o lagunas, bosque, pantanos y suelo llano. El agente sólo puede recorrer el terreno llano (de color café). Generamos 2.000 ejemplares para cada mapa de manera aleatoria, asegurando que exista una ruta entre el estado inicial y objetivo. Los resultados que se presentan son promedios sobre 6.000 ejemplares.



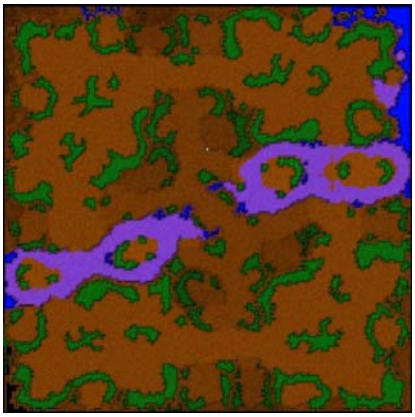


Figura 67. Mapas de WarCraft III utilizados en el análisis experimental.

8.2 Resultados experimentales

En esta sección evaluamos el desempeño de los distintos algoritmos presentados en esta tesis en mapas de juegos. En la aplicación sólo son relevantes los resultados en la primera ejecución [Bulitko y Lee, 2006]. De todas maneras mostramos algunos resultados en convergencia. En la primera ejecución se evalúa los algoritmos que se mencionan a continuación.

- Las versiones din-din de LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$. Se utilizan estas versiones porque son las que han obtenido las soluciones de mejor calidad en los escenarios de pruebas usados en los capítulos anteriores.

- Los algoritmos LRTA*(Koenig), RTAA* y LRTS(γ=1, T = ∞). Se compara nuestra aproximación con estos algoritmos porque son los más relevantes en el estado del arte.

En convergencia comparamos LRTA* $_{LS}$(k, d) con LRTA*(Koenig). Descartamos RTAA* y LRTS(γ=1, T = ∞) porque su mecanismo de actualización es peor que el de LRTA*(Koenig). La idea es comparar una de nuestras mejores aproximaciones con una de las mejores aproximaciones del estado del arte.

8.2.1 Resultados en la primera ejecución

Una medida de rendimiento importante en esta aplicación es el tiempo que tarda el agente antes de realizar el primer movimiento en la primera ejecución [Bulitko y Lee, 2006]. Una forma de medir el retardo en el primer movimiento, que sirve para comparar los algoritmos, es contar el número de estados expandidos en el primer episodio de planificación. Los algoritmos expanden estados en la anticipación y en la actualización de las heurísticas (en la propagación). Notemos que RTAA* no realiza expansiones cuando actualiza la heurística de los estados en CERRADOS. Por simplicidad en la presentación de los resultados, no presentaremos el esfuerzo que hace RTAA* en actualizar en las gráficas.

Las medidas de desempeño que usaremos son: el coste (en cantidad de acciones), el tiempo total de búsqueda (en milisegundos), la memoria (en cantidad de estados que han aumentado su estimación heurística), el tiempo por episodio de planificación (en milisegundos) y el número de estados expandidos antes del primer movimiento. Los valores de k censados son 10, 40 y 160, los valores de d son 2, 4, 8, 16, 32, 64 y 128. Comentamos cada una de las medidas de desempeño. En las gráficas que mostramos a continuación se excluye LRTS(γ=1, T = ∞). Por razones de escala los resultados de este algoritmo se presentan aparte.

- Coste: el la Figura 68 podemos observar los resultados para el coste en los dos tipos de mapa. El resultado es similar para ambos tipos. Podemos apreciar que LRTA*(Koenig) y RTAA* necesitan una gran cantidad de anticipación para lograr un coste similar al de LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$. El efecto de la anticipación en LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$ es bajo (las curvas son casi rectas). Con d pequeños LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$ obtienen muy buenos resultados que poco mejoran cuando d crece. HLRTA*$_{LS}(k, d)$ tiene un rendimiento levemente mejor que LRTA*$_{LS}(k, d)$ con $k = 10$, con los otros valores de k, los resultados son muy similares.

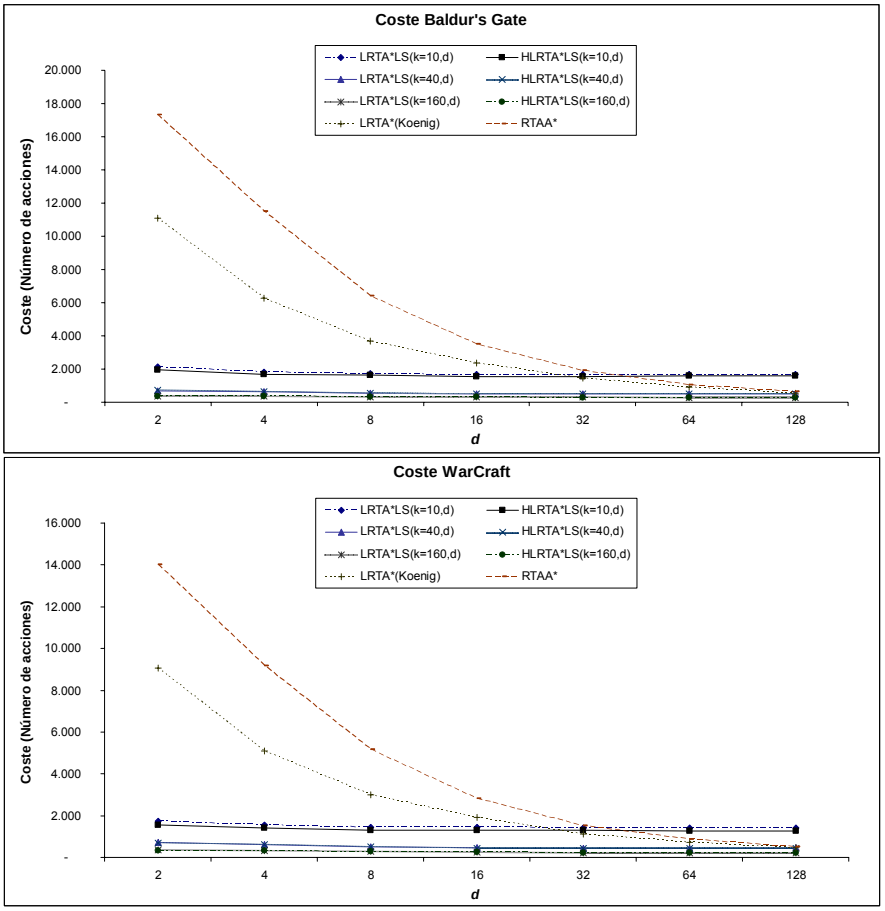


Figura 68. Resultados experimentales en mapas de Bladur's Gate y WarCraft III. Presentamos el coste obtenido con las versiones din-din de LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$, LRTA*(Koenig) y RTAA* para distintos valores de d y k en primera ejecución.

- Tiempo total de búsqueda: el la Figura 69 podemos observar los resultados en los dos tipos de mapa. El resultado es similar para ambos tipos. Podemos apreciar que el tiempo total de búsqueda primero disminuye y luego sube con LRTA*(Koenig) y RTAA* y que en LRTA*$_{LS}$(*k*, *d*) y HLRTA*$_{LS}$(*k*, *d*) el tiempo siempre aumenta. Para valores pequeños de *d*, el tiempo de RTAA* y LRTA*(Koenig) es peor que el de LRTA*$_{LS}$(*k*, *d*) y HLRTA*$_{LS}$(*k*=40,160 , *d*). LRTA*$_{LS}$(*k*=40,160, *d*) siempre obtiene menor tiempo que LRTA*(Koenig) en todos los valores de *d*. RTAA* es el que obtiene mejor resultado con valores de *d* intermedios y grandes. El tiempo que obtiene HLRTA*$_{LS}$(*k*, *d*) en mayor que el que obtiene LRTA*$_{LS}$(*k*, *d*) en todos los valores de *k* y *d*. Esto sucede porque la gestión de las heurísticas de HLRTA*$_{LS}$(*k*, *d*) necesita una mayor cantidad de operaciones que LRTA*$_{LS}$(*k*, *d*). Es importante destacar que el mejor tiempo de búsqueda los obtiene LRTA*$_{LS}$(*k*=160, *d*) con valores de *d* pequeños.

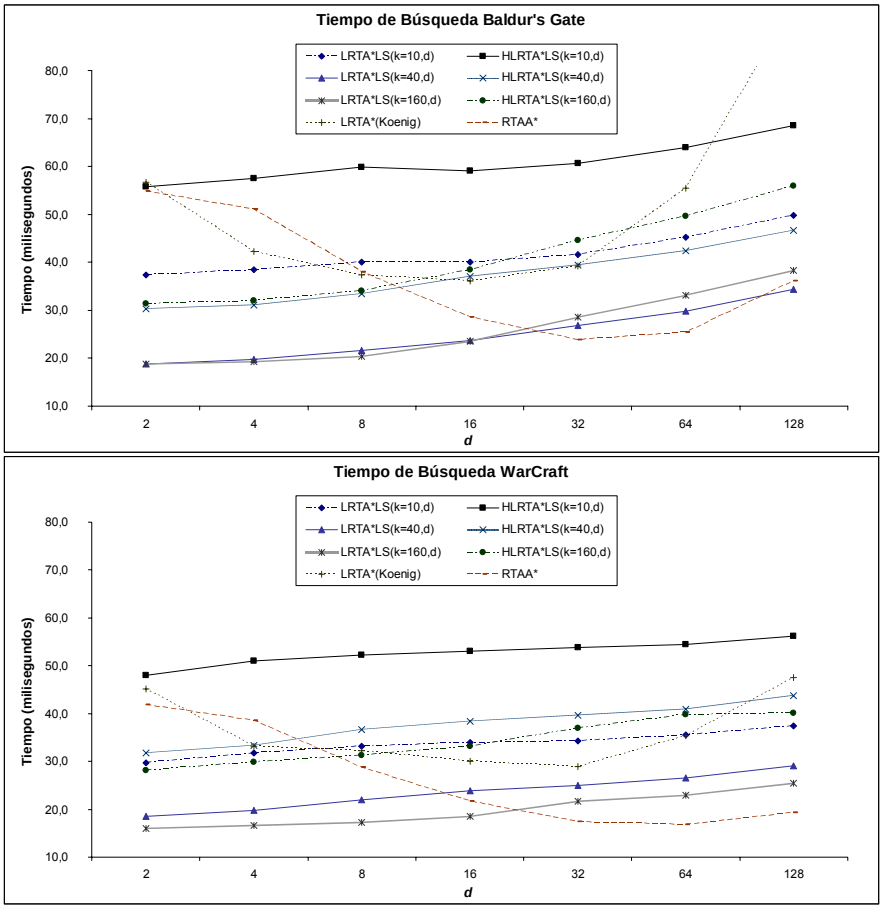


Figura 69. Resultados experimentales en mapas de Bladur's Gate y WarCraft III. Presentamos el tiempo total de búsqueda obtenido con las versiones din-din de LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$, LRTA*(Koenig) y RTAA* para distintos valores de d y k en primera ejecución.

- Tiempo por episodio de planificación: el la Figura 70 vemos los resultados en los dos tipos de mapas. El tiempo por episodio de planificación siempre aumenta cuando d crece en todos los algoritmos. Con d pequeño LRTA*(Koenig) y RTAA* obtiene el menor tiempo, a partir de d = 32 LRTA*$_{LS}(k=10, d)$ obtiene el tiempo menor y el de LRTA*(Koenig) y RTAA* aumentan drásticamente (sobre todo LRTA*(Koenig)). LRTA*$_{LS}(k, d)$ obtiene un tiempo por episodio de planificación menor que el de HLRTA*$_{LS}(k, d)$ por las razones que se explicaron en el punto anterior. Un punto a destacar es que el valor de esta medida de desempeño en LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$ también depende del

valor de *k*. Con un k más pequeño, *k* = 5 por ejemplo, se obtendrían tiempos menores.

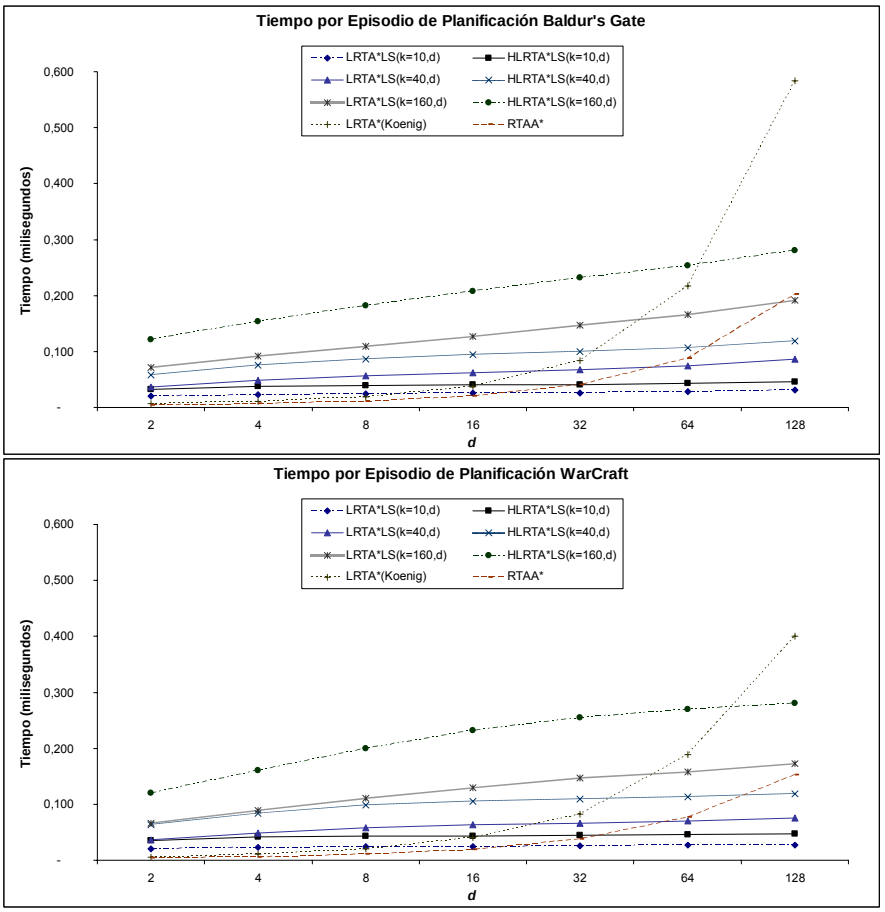


Figura 70. Resultados experimentales en mapas de Bladur's Gate y WarCraft III. Presentamos el tiempo por episodio de planificación obtenido con las versiones din-din de LRTA*$_{LS}$(*k*, *d*) y HLRTA*$_{LS}$(*k*, *d*), LRTA*(Koenig) y RTAA* para distintos valores de *d* y *k* en primera ejecución.

- Número de estados expandidos antes del primer movimiento: el la Figura 71 vemos los resultados en los dos tipos de mapas. El número de expandidos siempre aumenta cuando *d* crece en todos los algoritmos. Con *d* pequeño se obtienen los mejores resultados. En las gráficas se observa que LRTA*(Koenig) es el que obtiene los peores resultados a medida que *d* crece. Los otros algoritmos obtienen resultados similares. Recordemos que no se esta considerando el esfuerzo en actualizar en RTAA*, en cambio en LRTA*$_{LS}$(*k*, *d*) y HLRTA*$_{LS}$(*k*, *d*) si se considera. Esto nos permite concluir que

LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$ son los algoritmos que realizan menos esfuerzo antes del primer movimiento.

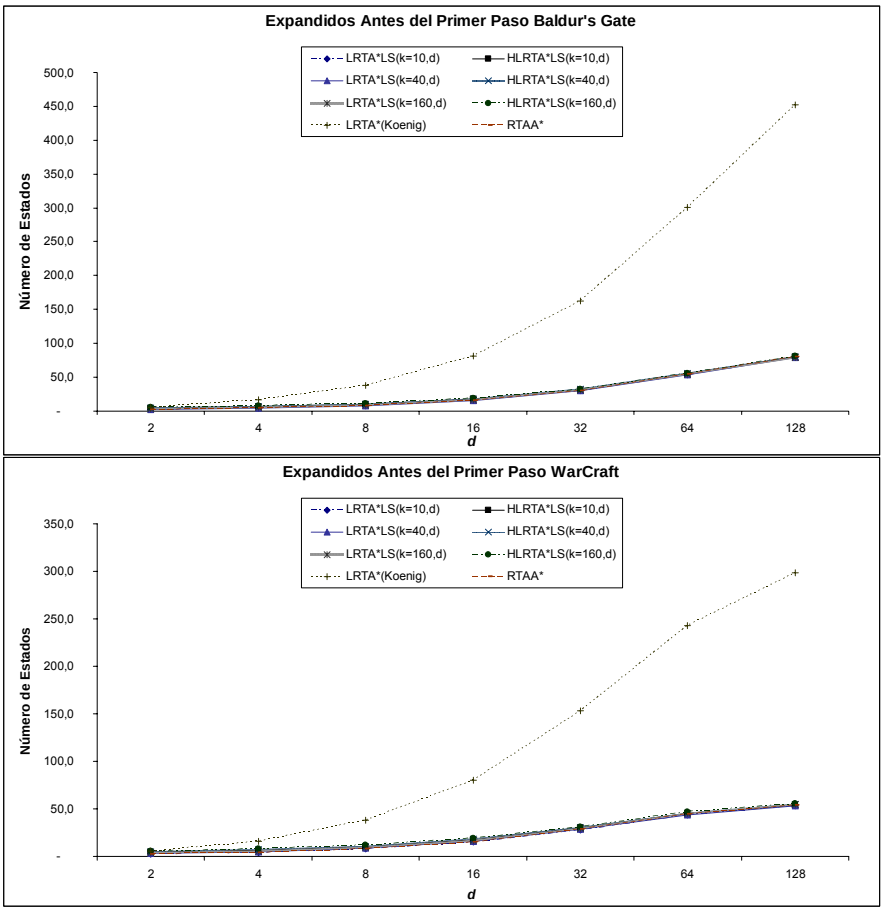


Figura 71. Resultados experimentales en mapas de Bladur's Gate y WarCraft III. Presentamos el número de estados expandidos antes del primer movimiento del agente con las versiones din-din de LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$, LRTA*(Koenig) y RTAA* para distintos valores de d y k en primera ejecución.

- Memoria: el la Figura 72 vemos los resultados en los dos tipos de mapas. Los algoritmos que usan menos memoria son LRTA*(Koenig) y RTAA* (excepto en Baldur's Gate con d = 64 y 128). Los otros algoritmos usan una cantidad mayor de memoria y es similar para un valor de k fijo. A mayor k se usa una mayor cantidad de memoria. Este resultado se explica porque los algoritmos LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$ propagan sobre cualquier estado, en

cambio la actualización de heurísticas en LRTA*(Koenig) y RTAA* se concentra en los estados en CERRADOS.

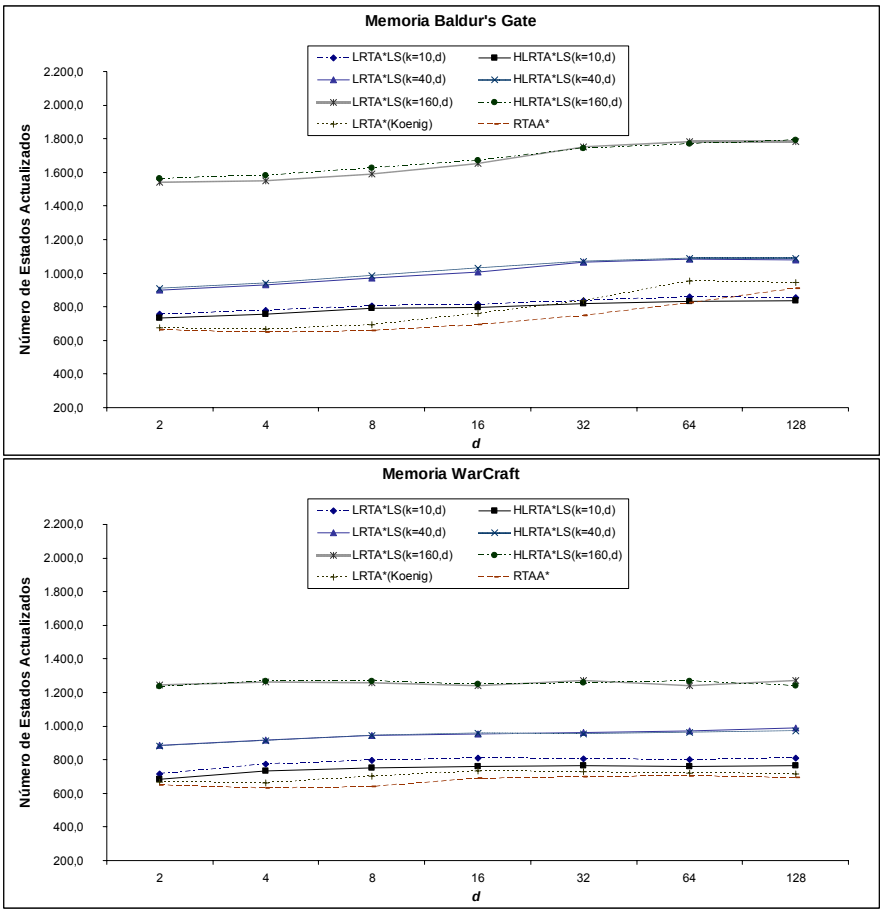


Figura 72. Resultados experimentales en mapas de Bladur's Gate y WarCraft III. Presentamos la memoria consumida por las versiones din-din de LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$, LRTA*(Koenig) y RTAA* para distintos valores de d y k en primera ejecución.

Baldur's Gate					
Anticipación	Coste	Tiempo Total	Tiemp/Plani.	Memoria	Expandidos
3	19.974,3	100,4	0,015	649,8	23,4
6	19.400,6	178,1	0,054	603,6	117,8
9	18.393,4	234,5	0,110	560,2	281,6
12	17.303,3	276,8	0,179	517,5	512,4

War Craft					
Anticipación	Coste	Tiempo Total	Tiemp/Plani.	Memoria	Expandidos
3	15.738,5	76,9	0,015	609,4	22,7
6	15.092,7	135,1	0,053	529,0	114,0
9	14.246,4	179,0	0,109	470,7	269,0
12	13.742,0	220,4	0,182	437,2	479,4

Tabla 9. Resultados experimentales en mapas de Bladur's Gate y WarCraft III. Presentamos el coste, el tiempo total, el tiempo por episodio de planificación, la memoria consumida y los estado expandidos por LRTS(γ=1, $T = \infty$) para distintas profundidades de anticipación en primera ejecución.

En la Tabla 9 están los resultados para LRTS(γ=1, $T = \infty$). Utilizamos como profundidad del árbol de búsqueda (anticipación) 3, 6, 9 y 12. Usamos estos valores porque el tamaño del espacio local de búsqueda que producen es similar a los producidos con los valores de *d* por A*. Podemos apreciar que todas las medidas de desempeño son peores que la de los otros algoritmos excepto la memoria. De hecho es el algoritmo que consume la menor cantidad de memoria. Esto se debe a que actualiza la heurística de un estado por episodio de planificación, lo que implica una mal desempeño en las otras medidas de rendimiento.

Entre nuestras aproximaciones vemos que LRTA*$_{LS}(k, d)$ tiene un coste similar al de HLRTA*$_{LS}(k, d)$, pero HLRTA*$_{LS}(k, d)$ tiene un tiempo total y por episodio de planificación mayor. Por tanto elegimos LRTA*$_{LS}(k, d)$ como nuestro mejor algoritmos en los mapas de juegos. Los resultados nos muestran que cuando LRTA*(Koenig) y RTAA* tienen un tiempo (total y por episodio de planificación) similar o un poco mejor que a LRTA*$_{LS}(k, d)$ ofrece soluciones mucho peores. Por otro lado, cuando ofrece soluciones de calidad similar a LRTA*$_{LS}(k, d)$, LRTA*(Koenig) requiere tiempos de ejecución mucho mayores y RTAA* tiempos similares o un poco mayores. LRTA*$_{LS}(k, d)$ con *d* pequeño obtiene los mejores resultados en tiempo total (y una cantidad de tiempo por episodio de planificación moderada) y soluciones de mejor calidad que LRTA*(Koenig), RTAA* y LRTS. Estos nos indica claramente que es más conveniente utilizar recursos en propagar que en anticipar. Respecto al tiempo de retraso del primer movimiento medido con el número de estados expandidos, LRTA*$_{LS}(k, d)$ mejora a LRTA*(Koenig), RTAA* y LRTS. LRTA*$_{LS}(k, d)$ gasta mayor cantidad de memoria que sus competidores. Este gasto se explica por el mecanismo de propagación. Podemos decir que la ventaja de LRTA*$_{LS}(k, d)$ sobre los otros algoritmos en la mayoría de las medidas de desempeño, implica un mayor gasto de memoria.

8.2.2 Resultados en convergencia

En la Figura 73 podemos ver el resultado para el coste y tiempo total de convergencia. LRTA*$_{LS}$(*k*=40,160, *d*) obtiene menor coste y tiempo en todos los valores de *d* censados en los dos tipos de mapas. Hay una situación similar a la primera ejecución, cuando LRTA*(Koenig) obtiene costes parecidos a LRTA*$_{LS}$(*k*, *d*) su tiempo total comienza a crecer. LRTA*$_{LS}$(*k*, *d*) consigue soluciones de buena calidad en menos tiempo que LRTA*(Koenig) con *d* pequeño.

La Figura 74 muestras resultados para el tiempo por episodio de planificación y el número de ejecuciones. LRTA*(Koenig) tiene un tiempo menor que LRTA*$_{LS}$(*k*, *d*) con *d* pequeño, que aumenta drásticamente a partir de $d = 16$. Cuando el coste de LRTA*(Koenig) tiende a acercarse a los valores de LRTA*$_{LS}$(*k*, *d*) su tiempo de planificación por paso crece considerablemente. El comportamiento del número de ejecuciones es similar al de coste. En LRTA*$_{LS}$(*k*, *d*) la anticipación tiene poca influencia: a medida que *d* crece el número de ejecuciones disminuye levemente. En LRTA*(Koenig) disminuye notoriamente desde un número de ejecuciones alto (con $d = 2$) a valores similares a los de LRTA*$_{LS}$(*k*, *d*) (con $d = 128$).

En la Figura 75 podemos ver la memoria usada en convergencia. LRTA*(Koenig) consume menos memoria con $d \leq 16$. Para valores de *d* mayores LRTA*$_{LS}$(*k*=10, *d*) es el que menos consume. Podemos apreciar que los buenos resultados obtenidos con LRTA*$_{LS}$(*k*=40,160, *d*) en coste, tiempo y número de ejecuciones implican un mayor consumo de memoria.

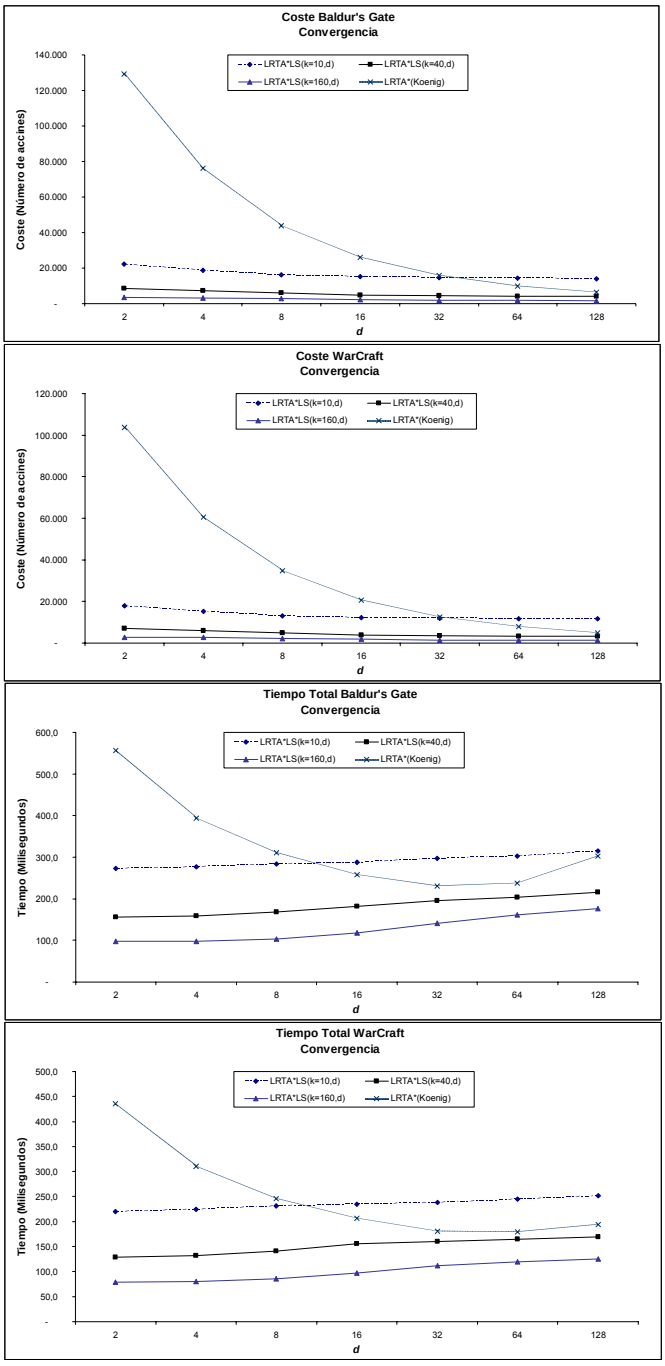


Figura 73. Resultados experimentales en mapas de Bladur's Gate y WarCraft III. Presentamos el coste y el tiempo total gastado por las versiones din-din de LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$, LRTA*(Koenig) para distintos valores de d y k en convergencia.

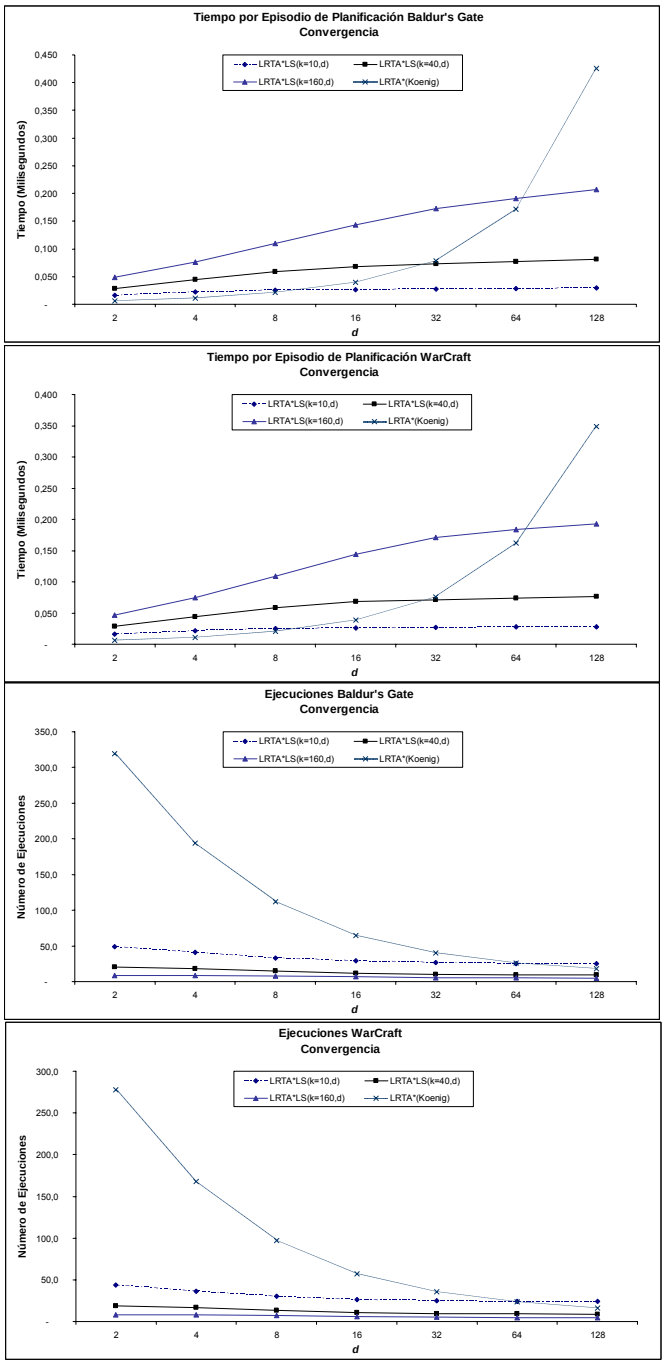


Figura 74. Resultados experimentales en mapas de Bladur's Gate y WarCraft III. Presentamos el tiempo por episodio de planificación y el número de ejecuciones de las versiones din-din de LRTA*$_{LS}(k, d)$ y HLRTA*$_{LS}(k, d)$, LRTA*(Koenig) para distintos valores de d y k en convergencia.

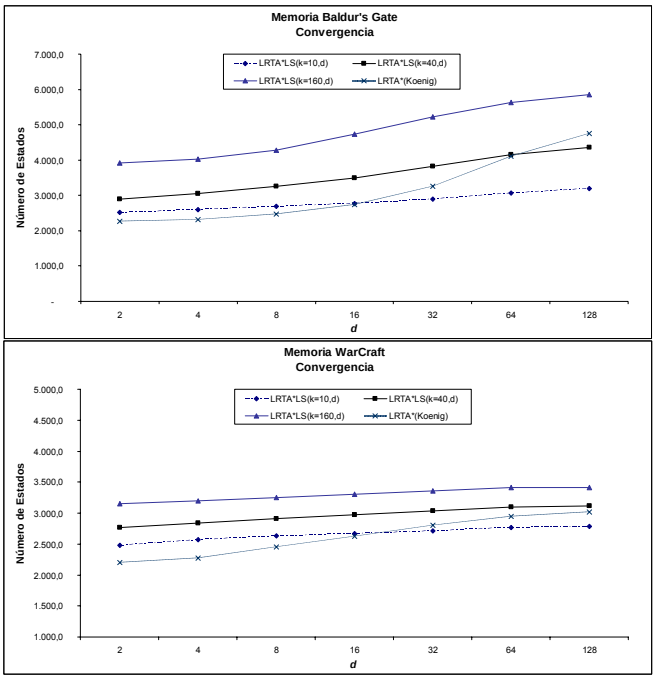


Figura 75. Resultados experimentales en mapas de Bladur's Gate y WarCraft III. Presentamos la memoria gastada con las versiones din-din de LRTA*$_{LS}$(*k*, *d*) y HLRTA*$_{LS}$(*k*, *d*), LRTA*(Koenig) para distintos valores de *d* y *k* en convergencia.

8.2.3 Conclusión

Nuestras aproximaciones consiguen las soluciones de mejor calidad en los mapas de Baldur's Gate y WarCraft III. En primera ejecución, la anticipación afecta poco el desempeño de LRTA*$_{LS}$(*k*, *d*) y HLRTA*$_{LS}$(*k*, *d*), en cambio tiene un efecto considerable en LRTA*(Koenig) y RTAA*. LRTA*(Koenig) y RTAA* necesitan realizar gran cantidad de anticipación para obtener soluciones similares a las de LRTA*$_{LS}$(*k*, *d*) y HLRTA*$_{LS}$(*k*, *d*), que les significa un aumento en el número de estados expandidos antes del primer movimiento y en el tiempo de búsqueda y planificación por paso. LRTS obtiene el peor rendimiento entre los algoritmos evaluados, excepto en el consumo de memoria.

En convergencia se repiten los resultados: la anticipación afecta poco a LRTA*$_{LS}$(*k*, *d*) y mucho a LRTA*(Koenig). Para obtener soluciones de calidad similar a LRTA*$_{LS}$(*k*, *d*), LRTA*(Koenig) necesita realizar mucha anticipación

que le significa un mayor tiempo de convergencia y tiempo por episodio de planificación.

Nuestras aproximaciones exhiben un mayor consumo de memoria. Esto se debe al mecanismo de propagación. La propagación se realiza sobre cualquier estado, en cambio las otras aproximaciones sólo actualizan los estados en CERRADOS o el estado actual.

El parámetro fundamental en LRTA*$_{LS}$(*k*, *d*) y HLRTA*$_{LS}$(*k*, *d*) es la *k*. A mayor *k*, mantenido un *d* pequeño, mejor calidad de la solución con un tiempo por episodio planificación moderado. El aumento en la anticipación mejora levemente los resultados, pero empeora el tiempo por episodio de planificación de manera más importante.

8.3 Resumen

En este capítulo describimos la búsqueda de rutas en mapas de juegos en tiempo real para ordenador. Luego evaluamos experimentalmente dos de nuestros mejores algoritmos en mapas de Baldur's Gate y WarCraft III y los compramos con las mejores aproximaciones desarrolladas en la comunidad de búsqueda en heurística tiempo real.

Los resultados experimentales muestran que nuestras aproximaciones mejoran a las existentes en los mapas de juegos. Además, podemos apreciar que la anticipación no es un parámetro fundamental en nuestros algoritmos (a diferencia de los otros algoritmos, en donde si es un parámetro fundamental para obtener soluciones de buena calidad) para obtener buenas soluciones. En nuestras aproximaciones es más importante utilizar recursos en propagar que en anticipar. A mayor cantidad de propagación, manteniendo una anticipación baja, obtenemos soluciones de mejor calidad en términos de coste y tiempo.

Capítulo nueve

9 CONCLUSIONES Y TRABAJO FUTURO

La búsqueda heurística en tiempo real permite resolver problemas no abordables por mecanismos tradicionales de búsqueda heurística. A largo de este trabajo hemos combinado la propagación acotada de cambios heurísticos con varios algoritmos de búsqueda heurística en tiempo real para resolver problemas de búsqueda de rutas para un agente en entornos inicialmente desconocidos. Los algoritmos propuestos fueron evaluados considerando principalmente dos medidas de desempeño: calidad de la solución y tiempo de búsqueda.

9.1 Conclusiones

De nuestro trabajo podemos extraer las siguientes conclusiones:

- La propagación acotada mejora claramente el desempeño de los algoritmos de búsqueda heurística en tiempo real. Esta mejora la hemos observado al aplicar propagación acotada a LRTA*, HLRTA* y FALCONS [Hernández y Meseguer, 2005d]. Los beneficios de la propagación dependen del parámetro k. Vimos que a mayor k se obtiene una mejora considerable en la calidad de la solución, con un aumento razonable en el tiempo por episodio de planificación. Sólo con $k = \infty$ (propagación no acotada) el tiempo por episodio de planificación aumenta de manera importante en algunos casos.

- Al combinar anticipación con propagación acotada es mejor utilizar más recursos en propagar que en anticipar. En los resultados experimentales observamos que con valores medianos y grandes de k, al aumentar la anticipación se produce una mejora poco relevante en la calidad de la solución y un aumento considerable en el tiempo de búsqueda y el tiempo por episodio de planificación. En concreto, con un k grande y poca anticipación se consiguen soluciones de buena calidad y cantidades bajas

de tiempo. Cuando se aumenta la anticipación, se consiguen soluciones de calidad similar en tiempos mucho mayores.

- Cuando permitimos realizar varios movimientos por episodio de planificación, podemos concluir:

  - Si estamos interesados en obtener el menor coste, la estrategia adecuada es hacer siempre un movimiento.

  - Si estamos interesados en minimizar el tiempo a la solución, la estrategia a utilizar es la de varios movimientos.

  - La aproximación dinámica ofrece una solución de buena calidad, con un tiempo razonable.

- Los mecanismos de propagación y anticipación aplicados sobre LRTA*, tienen un efecto similar sobre HLRTA*. Debido a la estructura del problema, HLRTA* y los algoritmos basados en este, tienen un alto desempeño en laberintos.

- Los algoritmos presentados en la tesis obtuvieron excelentes resultados aplicados a problemas reales. Nuestros resultados mejoran a las principales aproximaciones del estado del arte en búsqueda de rutas para juegos en tiempo real de ordenador. Utilizamos mapas extraídos de Baldur's Gate y WarCraft III. La ventaja se puede apreciar en ambos tipos de mapas. En nuestros métodos la anticipación no es un parámetro fundamental para obtener buenas soluciones. El patrón que observamos en los escenarios de prueba se repite en los mapas de juegos para ordenador: en los métodos es más importante utilizar recursos en propagar que en anticipar. A mayor cantidad de propagación, manteniendo una anticipación baja, obtenemos soluciones de mejor calidad en términos de coste y tiempo.

9.2 Trabajo futuro

Consideramos como trabajo fututo los siguientes puntos:

- Implementar la propagación acotada a otros métodos y entornos en búsqueda heurística en tiempo real.

  - Búsqueda en tiempo real para múltiples agentes que buscan un objetivo. Este tipo de búsqueda se describió en la sección 3.3.5. Vemos como trabajo desarrollar métodos de búsqueda para múltiples agentes que usan propagación acotada.

  - Los algorimos ε-search y δ-search [Shimbo e Ishida, 2003]. Combinar la propagación acotada con los algoritmos ε-search, un algoritmo permite encontrar soluciones subóptimas o ε-óptimas rápidamente, y δ-search, un algoritmo que balancea la relación entre exploración y explotación del espacio de búsqueda que mejora la estabilidad del proceso de convergencia a una solución óptima.

  - Entornos en que la heurística disminuye debido a que un estado pasa del "status" bloqueado a libre. En los métodos que se presentaron la heurística siempre crece. Como trabajo futuro vemos el desarrollo de métodos más generales de búsqueda heurística en tiempo real, en donde la heurística puede crecer o disminuir.

  - El problema de búsqueda de rutas cooperativo [Silver 2005]. En este problema existen múltiples objetivos que múltiples agentes deben buscar de forma cooperativa. El agente a_i debe buscar el objetivo o_i sin entorpecer la búsqueda que realizan los demás agentes, lo más rápido posible. Vemos como trabajo futuro aplicar la propagación acotada en nuevas técnicas que permitan resolver este problema.

  - Problemas clásicos de búsqueda con espacios de estados de gran tamaño, por ejemplo n-puzzle con n > 15. En [Yokoo y Kitamura,

1996] se resuelven ejemplares de 35-puzzle y 48-puzzle con métodos de búsqueda en tiempo real para múltiples agentes. Un trabajo futuro es verificar si la propagación acotada combinada con esta aproximación mejora la calidad de las soluciones.

- Aplicar la propagación acotada en otras técnicas de búsqueda heurística y resolución de problemas.

  - Búsqueda incremental [Koenig et al. 2004]. La búsqueda incremental reutiliza información de búsquedas previas para solucionar una serie de problemas similares potencialmente más rápido. Algunos de estos métodos aprenden mejores valores heurísticos mientras solucionan un problema [Koenig y Likhachev, 2005]. Debido a que la propagación acotada es un mecanismo de aprendizaje de heurísticas, vemos como trabajo futuro desarrollar mecanismos de búsqueda incremental basados en propagación acotada.

  - Programación dinámica en tiempo real y aprendizaje por refuerzo. Estas dos áreas son cercanas a la búsqueda heurística en tiempo real [Barto et al. 1995]. Un trabajo futuro es verificar si los beneficios obtenidos con nuestras contribuciones en búsqueda en tiempo real se pueden obtener en estas áreas. Un trabajo reciente en esta dirección ha sido desarrollado por [Liang et al. 2007].

- Aplicaciones.

  - Una aplicación futura es la implementación de nuestros algoritmos en robots físicos. En [Koenig, 2001b] se aplica un algoritmo en robots. Es interesante realizar este trabajo porque a la fecha no se ha realizado una investigación en profundidad sobre la búsqueda de rutas para robots físicos con algoritmos de búsqueda heurística en tiempo real. Estos métodos pueden ser competitivos con los algoritmos de

búsqueda heurística incremental, los más utilizados recientemente en planificación de rutas para robots autónomos [Ferguson et al. 2005].

REFERENCIAS

[Barto et al. 1995] A. Barto, S. Bradtke y S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*. 72:81-138. 1995.

[BioWare Corp., 1998] Baldur's Gate. *Interplay*. http://www.bioware.com/games/baldur_gate

[Blizzard Entertainment, 2002] Warcraft 3: Reign of chaos. http://www.blizzard.com/war3

[Bonnet y Geffner, 2006] B. Bonet y H. Geffner. Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and its application to MDPs. *In Proceedings of the 16th International Conference on Automated Planning and Scheduling* (ICAPS). 2006.

[Bulitko, 2004] V. Bulitko, Learning for adaptive real-time search. *The Computing Research Rep.* (CoRR): cs.DC/0407017, 2004.

[Bulitko y Lee, 2005] V. Bulitko y G. Lee. Learning in Real Time Search: A Unifying Framework. *Journal of Artificial Intelligence Research*. 25:119 - 157. 2006.

[Bulitko y Luvstrek, 2006] V. Bulitko y M. Lookahead Pathology in Real-Time Path-Finding. *National Conference on Artificial Intelligence* (AAAI), Member Abstracts and Posters. Boston, Massachusetts. July 19, 2006.

[Cakir y Polat, 2002] A.Cakir y F.Polat. Coordination of Intelligent Agents in Real-Time Search. *Expert Systems*, Blackwell Pub. Vol. 19, No.2, 80-87. 2002.

[Edelkamp and Eckerle, 1997] S. Edelkamp and J. Eckerle, New strategies in learning real time heuristic search. *In Proceedings of AAAI Workshop on On-Line Search*. 30–35. 1997.

[Ferguson et al. 2005]. D. Ferguson, M. Likhachev y A. Stentz, "A Guide to Heuristic-based Path Planning". *In Proceedings of ICAPS Workshop on Planning under Uncertainty for Autonomous Systems*. 2005.

[Furcy y Koenig, 2000] D. Furcy y S. Koenig. Speeding up the convergence of real-time search. *In Proceedings of AAAI*. 2000.

[Furcy y Koenig, 2001] D. Furcy y S. Koenig, Combining two fast-learning real-time search algorithms yields even faster learning. *In Proceedings of the 6th European Conference on Planning*. 2001.

[Goldenber et al. 2003] M. Goldenberg, A. Kovarksy, X. Wu, and J. Schaeffer, Multiple agents moving target search. *In Proceedings of the 18th International Joint Conference on Artificial Intelligence* (IJCAI). 2003.

[Hart, Nilsson y Raphel, 1968] P. Hart, N. Nilsson, and B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100--107, 1968.

[Hernández y Meseguer, 2005a] Carlos Hernández y Pedro Meseguer. LRTA*(k). *In Proceedings of the 19th International Joint Conference on Artificial Intelligence* (IJCAI). 2005.

[Hernández y Meseguer, 2005b] Carlos Hernández y Pedro Meseguer. Improving convergence of LRTA*(k). *In Proceedings of Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains IJCAI*. 2005.

[Hernández y Meseguer, 2005c] Carlos Hernández y Pedro Meseguer. Propagating Updates in Real-Time Search: HLRTA*(k). *In Proceedings of the 11th Conference of the Spanish Association for Artificial Intelligence, CAEPIA. Lecture notes in computer science*. 2005.

[Hernández y Meseguer, 2005d] Carlos Hernández y Pedro Meseguer. Propagating Updates in Real-Time Search: FALCONS*(k). *In Proceedings of the 25th International Conference of the Chilean Computer Science Society IEEE CS*. 2005.

[Hernández y Meseguer, 2007a] Carlos Hernández y Pedro Meseguer. Improving LRTA*(k). *In Proceedings of the 20th International Joint Conference on Artificial Intelligence* (IJCAI). 2007.

[Hernández y Meseguer, 2007b] Carlos Hernández y Pedro Meseguer. *Improving Real-Time Heuristic Search on Initially Unknown Maps. In Proceedings of Workshop on Planning in Games ICAPS*. 2007.

[Hernández y Meseguer, 2007b] Improving HLRTA*(*k*). *In Proceedings of the 12th Conference of the Spanish Association for Artificial Intelligence* (CAEPIA). *Lecture notes in computer science*. 2007

[Howlett, 2002] Jason Howlett. Path Planning for Sensing Multiple Targets from an Aircraft. *Master's thesis*. Mechanical Engineering, Brigham Young University, USA. 2002.

[Ishida, 1993] Toru Ishida. Moving Target Search with Intelligence. *National Conference on Artificial Intelligence* (AAAI), pp. 525-532, 1992.

[Ishida y Korf, 1991] Toru Ishida y Richard Korf. Moving Target Search. *International Joint Conference on Artificial Intelligence* (IJCAI). pp. 204-210, 1991.

[Ishida y Korf, 1995] T. Ishida y R. Korf. Moving target search: A real-time search for changing goals. *IEEE PAMI*. 17(6):609-619, 1995.

[Kitamura et al. 1996] Y. Kitamura, K. Teranishi, y S. Tatsumi. Organizational Strategies for Multiagent Real-time Search. *Proceedings International Conference on Multi-Agent Systems* (ICMAS). 150-156, 1996.

[Knight, 1993] K. Knight, Are many reactive agents better than a few deliberative ones?. *In Proceedings of the 13th International Joint Conference on Artificial Intelligence* (IJCAI). 1993.

[Koenig, 2001a] S. Koenig. Agent-Centered Search. *Artificial Intelligence Magazine*. 22, (4), 109-131. 2001.

[Koenig, 2001b] S. Koenig. Minimax Real-Time Heuristic Search. *Artificial Intelligence Journal,* 129, (1-2), 165-197, 2001.

[Koenig y Likhachev, 2002] S. Koenig y M. Likhachev. D*lite. *In Proceedings of the National Conference on Artificial Intelligence* (AAAI). 476–483. 2002.

[Koenig et al. 2003] S. Koenig, C. Tovey y Y. Smirnov. Performance bounds for planning in unknown terrain. *Artificial Intelligence*. 147, 253-279, 2003.

[Koenig et al. 2004] S. Koenig, M. Likhachev, Y. Liu y D. Furcy. Incremental heuristic search in AI. *Artificial Intelligence Magazine*. Vol. 25, 99 – 112. 2004.

[Koenig, 2004] S. Koenig. A comparison of fast search methods for real-time situated agents. *In Proceedings of the* 3rd *International Conference on Autonomous Agents and Multiagent Systems* (AAMAS), 2004.

[Koenig y Likhachev, 2005] Adaptive A*. *In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems* (AAMAS). 1311-1312. 2005.

[Koenig y Likhachev, 2006] S. Koenig y M. Likhachev. Real-time adaptive A*. *In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems* (AAMAS). 281–288. 2006.

[Koenig et al. 2007] S. Koenig, M. Likhachev y X. Sun. Speeding up Moving-Target Search. *In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems* (AAMAS), 2007.

[Korf, 1985] R.E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*. 27, 97 - 109. 1985.

[Korf, 1990] Richard E. Korf. Real-Time Heuristic Search. *Artificial Intelligence*. 42(2-3): 189-211. 1990

[Korf, 1999] Richard E. Korf, Artificial Intelligence Search Algorithms. *Algorithms and Theory of Computation Handbook CRC Press*. 1999.

[Liang et al. 2007] T. Liang, J. Sun y M. Yin. Improving the Convergence of RTDP. *Proceeding of International Conference on Fuzzy Systems and Knowledge Discovery*. 613-617. 2007.

[Nilsson, 1971] N. J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

[Pearl, 1984] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.

[Pemberton y Korf, 1992] J. Pemberton y R. Korf. Making locally optimal decisions on graphs with cycles. *Technical Report 920004*. Computer Science Dep. UCLA, 1992.

[Ramachandra et al. 2003] K. Ramachandra, S. Braynov y J. Llinas. Multi-agent moving target search in a hazy environment. *Proceedings of International Conference on Integration of Knowledge Intensive Multi-Agent Systems*. 275- 278, 2003.

[Rayner et al. 2007] C. Rayner, K. Davison, V. Bulitko, K Anderson y J. Lu. Real-Time Heuristic Search with a Priority Queue. *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI). pp. 2372 - 2377. 2007.

[Russell y Norvig, 2003] S. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*. Second Edition. Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey. 2003.

[Shang et al. 2003] Y. Shang, M. Fromherz, Y. Zhang, y L. Crawford, Constraint-based routing for ad-hoc networks. *IEEE International Conference on Information Technology: Research and Education* (ITRE). 2003.

[Shimbo e Ishida, 2003] M. Shimbo y T. Ishida. Controlling the learning process of real-time heuristic search. *Artificial Intelligence*, 146(1):1–41, 2003.

[Silver, 2005]. D. Silver. Cooperative Pathfinding. *In Artificial Intelligence and Interactive Digital Entertainment* (AIIDE). 2005.

[Stentz, 1995] A. Stentz. The focussed D* algorithm for real-time replanning. *In Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI). 1652–1659. 1995.

[Thorpe, 1994] P. E. Thorpe. A hybrid learning real-time search algorithm. *Master's thesis*. Computer Science Dep., UCLA, 1994.

[Yokoo y Kitamura, 1996] M. Yokoo y Y. Kitamura. Multiagent real-time A* with selection: Introducing competition in cooperative search. *Proceedings of the First International Conference on Multi--Agent Systems* (ICMAS). 409-416, 1996.

[Zhang and Fromherz, 2004] Ying Zhang y Markus P.J. Fromherz. Search-based Adaptive Routing Strategies for Sensor Networks. *In AAAI Workshop on Sensor Networks*. July 2004.