12-5-2016

Práctica Larga

Sokoban con búsqueda heurística

Javier García Pérez - 100290698
José Manuel Fernández Ruiz - 100290892

MÁSTER EN INGENIERÍA INFORMÁTICA
CURSO 1
DISEÑO DE SISTEMAS INTELIGENTES
UC3M

Índice

Índice de Tablas

Índice de ilustraciones

1. Introducción

El siguiente documento recoge la memoria de la realización de la práctica larga de la asignatura Diseño de Sistemas Inteligentes del Máster en Ingeniería Informática de la Universidad Carlos III de Madrid. Esta práctica tratará sobre la implementación de un solver basado en búsqueda heurística para el juego del Sokoban.

Este documento contendrá un resumen sobre el trabajo realizado, su arquitectura, una descripción técnica del solver, los experimentos realizados junto a sus resultados, posibles mejoras y una guía de implantación de la aplicación para poder ser usada en Ubuntu o Windows.

Este trabajo ha sido realizado por el grupo compuesto por los alumnos José Manuel Fernández Ruiz y Javier García Pérez.

2. Sokoban

En este apartado contiene toda la información relacionada con el juego Sokoban, así como un resumen de la idea implementada que se explicará de una manera más técnica en los apartados siguientes. Con este apartado se quiere conseguir que el lector conozca la forma de jugar y los aspectos claves antes de profundizar en la implementación del solver que se ha realizado.

2.1. Historia y conceptos básicos

El Sokoban es un videojuego de rompecabezas inventado en Japón en 1981 que trata de empujar y colocar cajas en las posiciones objetivo de un almacén. Fue creado por Hiroyuki Imabayashi. Sokoban significa *'encargado de almacén'* en japonés y de ahí se puede obtener una ligera idea sobre el objetivo de este juego.[1]

El objetivo del juego es empujar las cajas hasta la posición destino reduciendo el número de pasos realizados para colocarlas y el tiempo invertido para su realización. Las restricciones básicas de este juego se muestran a continuación:

- El jugador puede moverse en horizontal o en vertical, pero nunca en diagonal.
- El jugador solo puede empujar las cajas, pero no puede tirar de las cajas.
- El jugador solo puede empujar una caja a la vez.
- Para poder empujar una caja la siguiente casilla debe de encontrarse vacía y no debe de ser una pared.

En su primera versión el Sokoban contaba con un total de 20 niveles, pero con el paso de los años y gracias a la popularidad que ha tenido el juego se han ido creando una gran cantidad de sets de nuevos niveles en los cuales se aumenta progresivamente la dificultad. De esta manera se ha conseguido mantener activos a los usuarios del Sokoban al proponerles nuevos desafíos con los niveles más difíciles.

La versión del Sokoban implementada para esta práctica es una aplicación que cuenta con un total de 150 niveles en los cuales se eleva progresivamente la dificultad del nivel aumentando el número de cajas que el usuario debe de colocar y la forma del escenario.

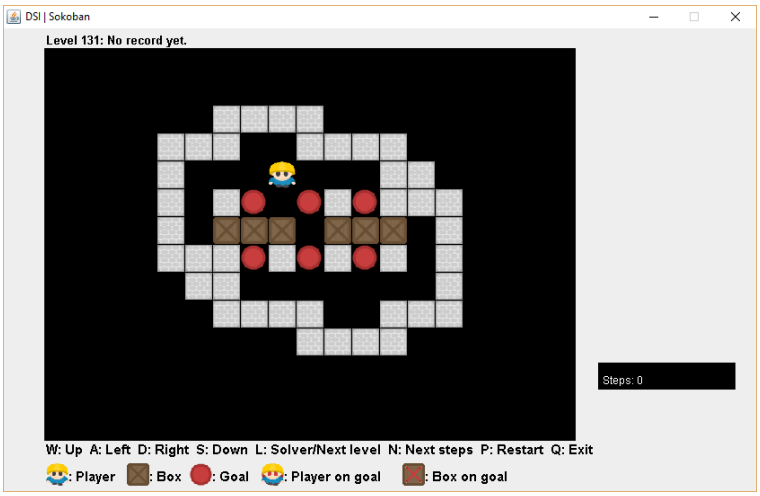


Ilustración 1 Interfaz gráfica

Por otro lado, la aplicación desarrollada será online y el progreso de cada jugador será guardado en nuestra base de datos. De esta manera se permite al usuario poder continuar con su nivel en cualquier momento y aumentar la competitividad del juego al poder batir el record de pasos establecidos para cada nivel por los diferentes usuarios. En cuanto a nosotros, poder disponer en la base de datos de las jugadas realizadas por los usuarios de forma detallada nos permite poder afinar la heurística y usar aprendizaje automático para hacer más potente el solver.

2.2. Resumen de la aplicación desarrollada

En este apartado se explica de forma general y a alto nivel el proyecto realizado. En el apartado *4. Solver* se encuentra de forma más detallada todos los aspectos principales de este trabajo.

La aplicación desarrollada del Sokoban se ha realizado en el lenguaje Java y la base de datos que da soporte a esta tiene como gestor MongoDB. La aplicación ha sido testeada y dado su visto bueno en cuanto a funcionamiento en los sistemas operativos Windows 10 y Ubuntu 14.04. En el apartado *ANEXO I: Guía de instalación* se puede encontrar un manual para su despliegue en estos sistemas operativos.

Como ya se ha mencionado anteriormente, contamos con un set de 150 niveles los cuales van en aumento en su complejidad conforme se va avanzado. En cada nivel se ofrece al usuario la posibilidad de resolver este manualmente o mediante un solver, este último permite resolver el nivel completamente o parcialmente (opción *next steps*).

El solver desarrollado utiliza búsqueda heurística para encontrar la solución usando una variante del algoritmo A* (AStar) con poda. Este solver recibirá el estado actual del jugador y mediante la expansión sucesiva de los nodos siguiendo una heurística y controlando una serie de restricciones dará con la solución, la cual buscará minimizar en el ratio coste/tiempo, este coste serán el número de pasos o movimientos realizados por el jugador para completar el nivel.

La función de evaluación que finalmente se ha decidido fijar está compuesta a su vez por tres funciones, las cuales se resumen a continuación:

- $i(x)$: Número de cajas colocadas en destino.
- $h(x)$: Coste aproximado de colocar las cajas.
- $g(x)$: Coste del recorrido hasta el nodo *x*.

En su conjunto trabajan de la siguiente manera: $f(x) = i(x) \mid g(x) + h(x)$. Por lo que la función de evaluación está definida o por el número de cajas colocadas correctamente (en un destino) o la suma de movimientos hasta la configuración *x* más el coste aproximado de llevar a los destinos las cajas restantes. En resumen, dentro de la cola abierta del algoritmo A* aparecerán primero los nodos con más cajas acomodadas correctamente y en caso de empate seguirán aquellos con menor costo acumulado de g(x) más el valor de la heurística h(x).[2]

Por último, todas las soluciones obtenidas por el solver, y los propios jugadores, en cada nivel son guardadas en la base de datos para su estudio. De esta manera también se evitan

demoras a la hora de volver a calcular la solución del nivel para otro usuario debido a que cuando un jugador solicite la resolución del nivel, la solución que se mueste puede ser la calculada por el solver o la realizada por otro jugador humano, siempre se mostrará la que menor número de movimientos utilice.

3. Arquitectura

La arquitectura que usa el sistema es modular. Sera una arquitectura que estará basada en los módulos representado en la imagen *Ilustración 2 Arquitectura modular* y que más adelante se explicarán.

El sistema tiene parte deliberativa porque conoce el objetivo para concluir un nivel y el estado actual del mundo. Además, se puede se considerar como reactiva en el caso del movimiento manual del usuario, debido a que el sistema obedece los movimientos del usuario sin tener un objetivo marcado al que dirigirse.

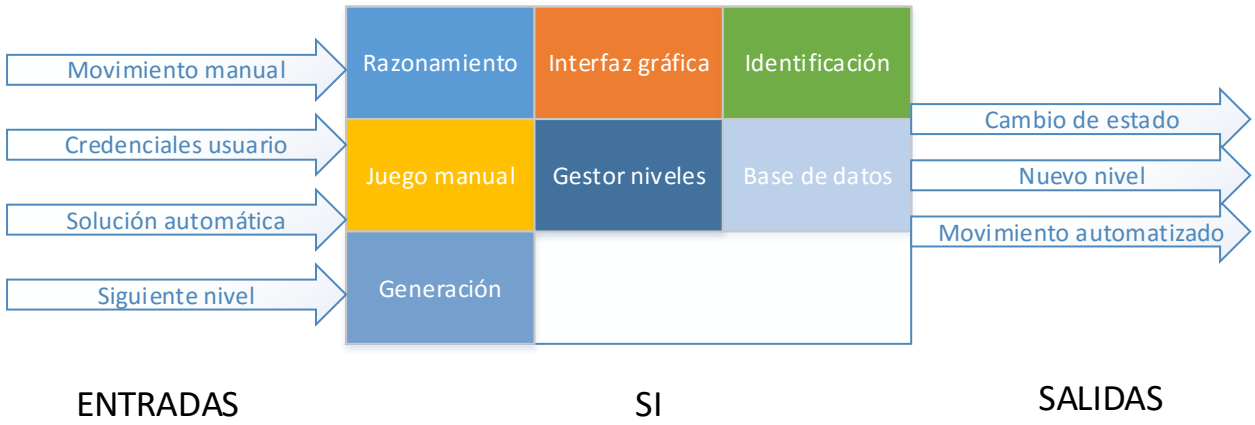


Ilustración 2 Arquitectura modular

La imagen anterior muestra las entradas y salidas del sistema, así como los módulos de los que se compone. En los siguientes apartados se explicarán la función de estos en el sistema. Aunque no se ha implementado se ha considerado como futura mejora añadir un módulo de aprendizaje automático que se apoye en toda la información recopilada de las jugadas realizadas por los jugadores y guardadas en la base de datos, para de esta manera perfeccionar la heurística usada en el algoritmo del solver o usar clasificadores para buscar la solución de un nivel.

3.1.1. Interfaz gráfica e identificación

Estos dos módulos se apoyan en entre sí. Ya que para realizar el proceso de identificación se proporcionará al usuario una interfaz gráfica para realizar esta tarea. Por otro lado, para interactuar con el juego el usuario también utilizará otra interfaz gráfica.

Se ha diseñado una interfaz gráfica sencilla y accesible en cuanto a uso. En primer lugar, el idioma elegido ha sido el inglés, de esta manera se hace más universal el uso de la aplicación al poder ser entendida por más usuarios.

Como primera interfaz, se ha realizado la de login, la cual únicamente pedirá el nombre y contraseña con el que el usuario accedió por última vez. En caso de ser su primera vez, el sistema registrará esos datos para futuros accesos. Por otro lado, en caso de ya existir un usuario con ese nombre o introducir mal la contraseña la interfaz mostrará un mensaje de error.

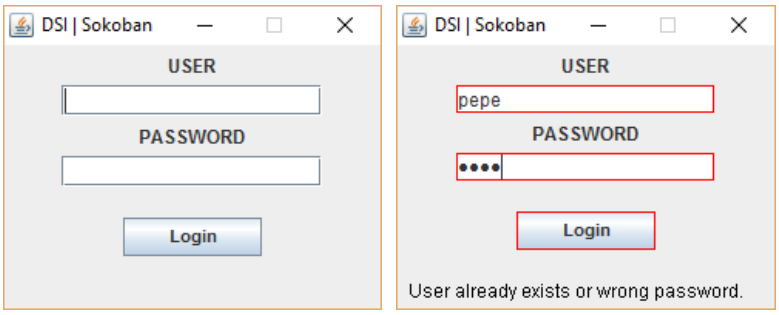


Ilustración 3 Interfaz gráfica de login

Tras realizar un login correcto el sistema mandará la información sobre el progreso actual del jugador en cuanto a niveles al módulo de gestión de niveles para que se encargue de cargar el que corresponde.

La segunda, y última interfaz realizada, ha sido la del propio juego. Como primera decisión se ha optado por interactuar con esta mediante teclado, en lugar de botones, de esta manera se ha conseguido dejar más espacio al escenario del nivel del Sokoban que se esté mostrando.

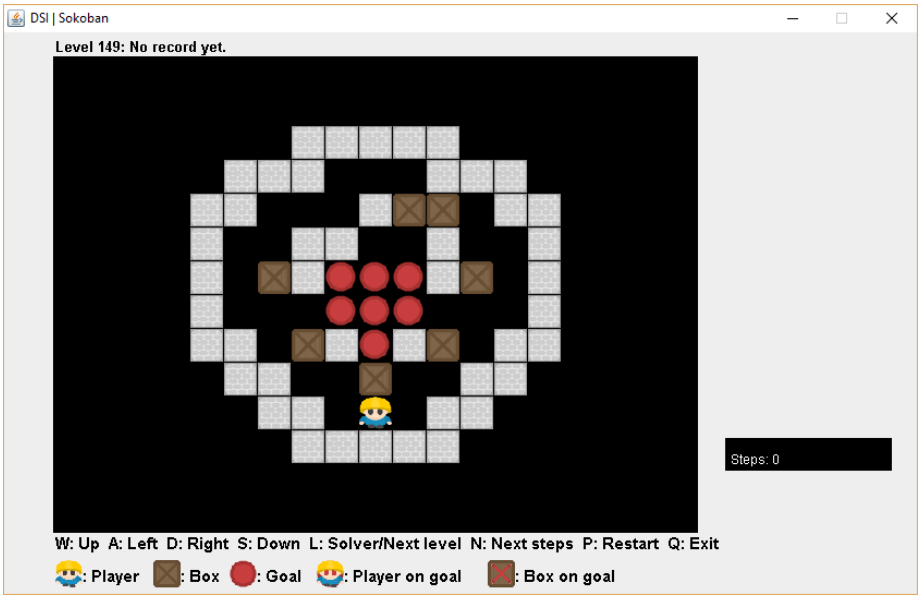


Ilustración 4 Interfaz gráfica juego

Como se puede apreciar en la imagen anterior, aparte del escenario del nivel, existen más elementos en ella. Estos elementos se explican a continuación:

- **Texto de nivel y record:** En la parte superior de la interfaz se mostrará un texto indicando el nivel en el que se encuentra el usuario seguido del record en pasos establecido para ese nivel y por quien fue logrado (usuario o solver). En caso de no existir record todavía se indicará este hecho.
- **Caja de pasos:** Situada en la parte inferior derecha de la interfaz muestra el contador de pasos realizados durante el transcurso de este nivel. Este contador se reseteará en caso de reiniciar el nivel o pasar al siguiente.

- **Instrucciones de juego:** Situadas en la parte inferior de la interfaz muestran las teclas de juego, así como el significado de cada uno de los elementos que aparecen en el nivel.

Para finalizar, mencionar que se ha bloqueado el redimensionamiento de ambas interfaces para no crear problemas de resolución y que así se descuadren sus elementos. Por otro lado, estas interfaces siempre aparecerán por defecto en el centro de la pantalla del usuario. Los gráficos con los que cuenta la interfaz de juego han sido extraídos de una página web donde se proporcionaban texturas para el juego Sokoban.[9]

3.1.2. Base de datos

El siguiente apartado describe la estructura de la base de datos y las diferentes comunicaciones y tareas que se realizan con ella durante la ejecución del Sokoban.

Las entradas de este módulo serán los nuevos niveles generados, las consultas de identificación de login, las peticiones para devolver la información de un nivel y las peticiones para actualizar la solución de un nivel o la información de un jugador.

Las salidas son todas las respuestas dadas a los módulos sobre consultas como por ejemplo la devolución de un nivel o la confirmación de que ya existe el nombre de usuario con el que alguien intenta acceder.

La base de datos está compuesta de dos colecciones, una para la información de los jugadores y otra para la información de los niveles. A continuación, se describe en detalle las características de cada una de ellas:

- **Jugadores:** Esta colección guarda toda la información relacionada con el usuario, así como su progreso en el juego. A continuación, se explican sus atributos.
    - **_id:** Nombre del usuario.
    - **Pass:** Contraseña del usuario.
    - **Progreso:** Lista con el avance del jugador en los niveles.
        - **Nivel:** ID del nivel.
        - **Jugada:** Compuesto por tecla, valor de la función de evaluación y mapa, guarda la información paso a paso de la jugada manual realizada por el usuario para completar el nivel. En caso de completar el nivel mediante el solver sólo se guardará el nivel completado en su progreso y no la jugada.

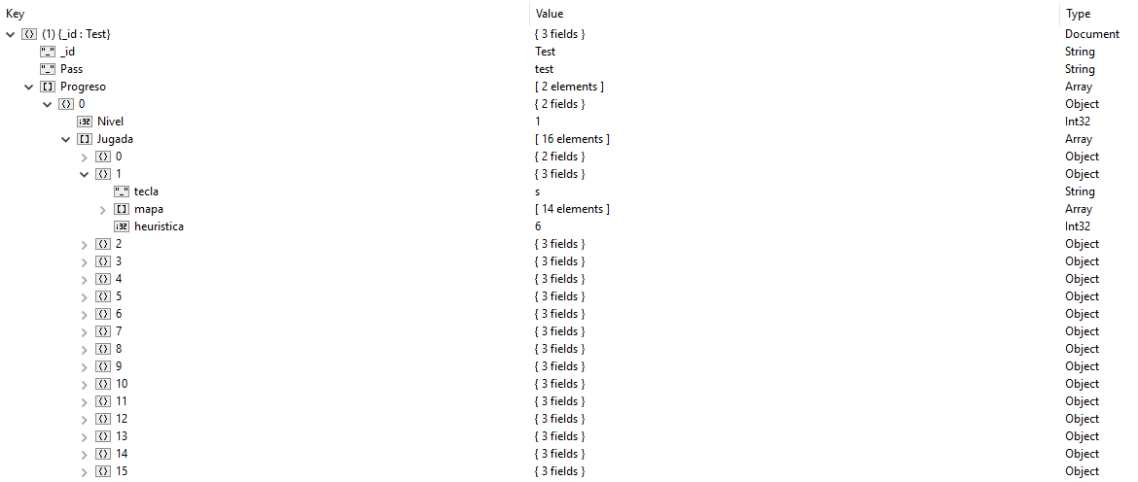


Ilustración 5 Ejemplo de documento en colección Jugadores.

- **Niveles:** Esta colección guarda toda la información relacionada con los niveles, así como las jugadas que se han usado para completarlos. A continuación, se explican sus atributos.
    - **_id:** Número de nivel.
    - **Mapa:** Escenario del nivel en su estado inicial.
    - **Jugada:** Compuesto por tecla, valor de la función de evaluación y mapa, guarda la información paso a paso de la jugada record realizada por el usuario o solver para completar el nivel.
    - **Jugador:** Nombre del jugador que ostenta el record en movimientos de completar el nivel.
    - **Time:** Tiempo de cómputo record para completar el nivel en caso de haber usado el solver.
    - **AStar/IDAStar:** Guarda los datos relacionados con el algoritmo que resolvió el nivel.
        - **Time:** Tiempo de cómputo para completar el nivel.
        - **Nodos:** Número total de nodos estudiados para completar el nivel.
        - **seq:** Compuesto por tecla, valor de la función de evaluación y mapa, guarda la información paso a paso de la jugada realizada por el solver para completar el nivel.

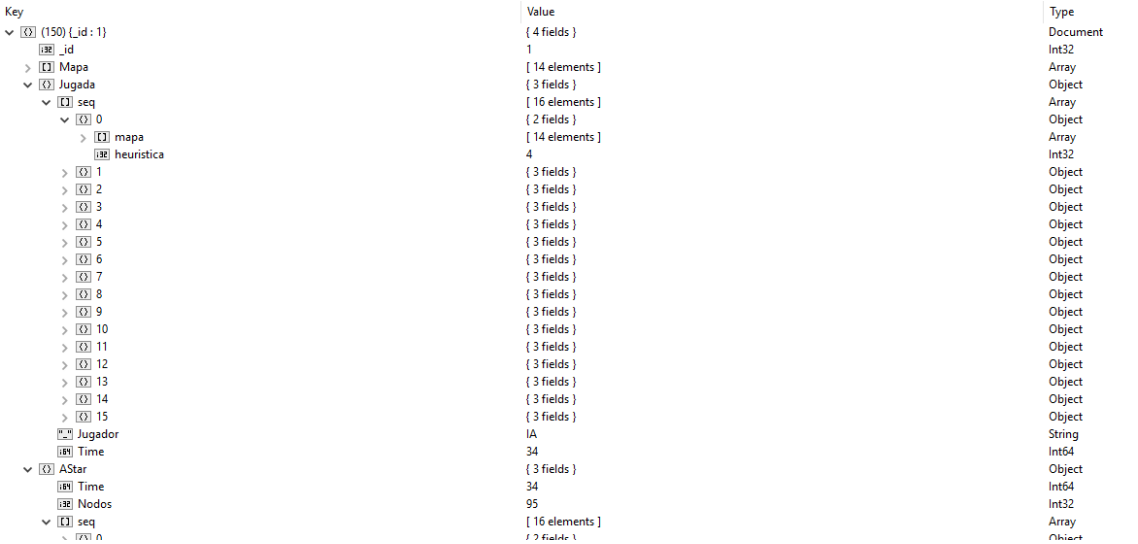


Ilustración 6 Ejemplo de documento en colección niveles

Una vez explicadas las colecciones, las tareas y comunicaciones que realiza el juego con la base de datos son las siguientes:

- **Generar niveles:** Si los niveles no están en la base de datos se crean e insertan estos en la colección *niveles* a partir del archivo *levels.txt*, el cual contiene los niveles en formato de caracteres.
- **Login:** Comprueba a la hora de acceder al juego si el usuario está registrado y si los datos introducidos son correctos. En caso de no estar registrado ese nombre lo crea en la colección *jugadores*. En caso de existir ya ese usuario y ser los datos incorrectos devuelve un mensaje de error que se muestra en la interfaz.
- **Cargar nivel:** Se encarga de cargar la información del nivel correspondiente desde la colección *niveles* a la interfaz gráfica (ajustando este basándose en su tamaño) para que el usuario pueda interactuar con él.
- **Reiniciar nivel:** En caso de que el usuario quiera reiniciar un nivel se buscará en la colección *niveles* el estado inicial de este para volverlo a mostrar en pantalla.
- **Resolver nivel:** En caso de que el usuario quiera resolver un nivel con el solver proporcionado se comprobará antes en la colección *niveles* si ya existe una solución para ese nivel usando el algoritmo actual.
- **Actualizar nivel:** Cuando el usuario o el solver completan un nivel se actualiza este siempre y cuando se haya batido el record de movimientos o se haya obtenido una solución con un algoritmo que no estaba guardada en la solución.
- **Actualizar usuario:** Cuando el usuario resuelve manualmente o con el solver un nivel se actualiza el progreso de niveles de este en la colección *jugadores* y se guarda la jugada realizada en caso de haber sido realizada manualmente.

3.1.3. Juego manual

Este módulo será el encargado de gestionar la iteración manual del usuario con el nivel, encargándose de realizar el cambio de estado en base al movimiento o petición solicitada por el usuario (solver, reiniciar nivel o siguiente nivel). Por otro lado, se encargará de ir

almacenando las teclas pulsadas por el usuario sobre la jugada realizada en el nivel para que en el caso de completarlo mandársela al gestor de niveles para que la guarde.

El módulo de juego manual está relacionado con el módulo de gestor de niveles que es el encargado de controlar el progreso del usuario y realizar las tareas necesarias una vez completado el nivel.

Las entradas de este módulo son las teclas pulsadas por el usuario para realizar los movimientos y peticiones del jugador.

La salida es la comunicación de estas teclas al módulo de gestor de niveles para que proceda a actualizar su estado.

3.1.4. Gestor de niveles

Es el modulo que realizará o tramitará (en caso de no ser de su competencia) todas las operaciones relacionadas con los niveles como carga, reinicio, siguiente nivel, activar el solver o actualizar el estado de un nivel tras un movimiento.

Este módulo tendrá como entradas la identificación de un usuario, la jugada manual de un usuario, la petición de solución automática, la petición de reinicio de nivel o el siguiente mapa de un nivel.

Las salidas del módulo son el cambio de estado de un nivel, un nuevo nivel para jugar o la información que envía al módulo de base de datos sobre jugadas que han completado el nivel.

3.1.5. Razonamiento

Este módulo será el encargado de aplicar la IA sobre un mapa del Sokoban para encontrar una solución.

Las técnicas que se aplicarán en el módulo de razonamiento para encontrar una solución a un mapa del Sokoban serán algoritmos de búsqueda heurística. Los detalles sobre el algoritmo elegido, así como la información relacionada con él, pueden consultarse en el apartado *4. Solver*.

El algoritmo elegido e implementado en el sistema ha sido probado con el Sokoban en un equipo con las siguientes características: procesador de 2.6 GHz con dos cores y 6 GB de memoria RAM. Obteniendo resultados satisfactorios para cada nivel del set de 150 elegido, marcándose como máximo las dos horas como tiempo tope de cálculo de solución para un nivel.

Este módulo tendrá relación con el módulo gestor de niveles del cual recibirá el mapa del nivel que debe encontrar la solución, posteriormente cuando finalice su ejecución debe devolver al módulo gestor la solución obtenida o un mensaje de que no fue encontrada.

Las entradas de este módulo serán los diferentes mapas de los cuales deben buscarse las soluciones.

Las salidas de este módulo son la secuencia de movimientos que debe realizar el jugador para completar el nivel o el mensaje de alerta para notificar que no se ha encontrado la solución.

3.1.6. Generación

Este módulo será el encargado de la generación de los 150 niveles de los que se compondrá el juego.

El módulo de generación tendrá relación con la base de datos, donde almacenará todos los mapas generados en base a al fichero *levels.txt* donde está la representación en caracteres de los niveles elegidos.

Las entradas de este módulo petición de generación de mapas y las salidas son los diferentes mapas que se generan y envían para almacenar en la base de datos.

Para garantizar que todos los niveles elegidos tienen solución se han escogido estos de diferentes sets de niveles que hay en Internet creados por amantes del Sokoban y los cuales ya han sido probados por ellos mismos. En el apartado *8. Referencias* puede consultarse el origen de los niveles elegidos.[3][4][5][6][7][8]

4. Solver

Este apartado recoge la descripción a un nivel más técnico del diseño del solver realizado para el juego Sokoban. En los siguientes apartados se explicará el problema, se justificará el algoritmo de búsqueda elegido, así como la función de evaluación establecida y las restricciones consideradas para el problema.

4.1. El problema

En este apartado se realizará la descripción del problema del Solver desde el punto de vista de la complejidad y la descripción del modelo para comprender el problema y poder implementar un algoritmo que lo resuelva.

4.1.1. Complejidad algorítmica

La implementación de un solver para el juego del Sokoban desde el punto de vista de la complejidad algorítmica es un problema de tipo NP-hard y EspacioP-completo, esto provoca que sea muy interesante para los investigadores de inteligencia artificial. La creación de un solver para el Sokoban es comparable a diseñar un robot que pueda moverse por un almacén y colocar las cajas deseadas. [2]

Sokoban es un problema muy complejo debido a que su factor de ramificación es muy elevado y la profundidad de su árbol de búsqueda puede alcanzar más de 1000 movimientos en los niveles más complejos que existen.[11]

Clasificar un problema como NP-hard implica decir que dicho problema pertenece al conjunto de los problemas que al menos son tan complejos como NP. Los problemas clasificados como NP son aquellos problemas que pueden ser resueltos en un tiempo polinómico por una máquina de Turing no determinista y de ahí surge el nombre "Nondeterministic Polynomial time" y que es abreviado como NP. [12][13]

El Sokoban es un ejemplo de problema del tipo EspacioP-Completo más conocido en su versión en inglés como *PSPACE-Complete*. Este tipo de problemas son los más complejos que se pueden encontrar en la clase de complejidad *PSPACE*. La definición de PSPACE-Completo está basada en la complejidad asintótica, es decir, el tiempo que tarda en resolverse un problema de tamaño *n* cuando *n* crece sin ningún límite. [14]

4.1.2. Descripción del modelo

La descripción del modelo estará formada por el espacio de estados, el estado inicial, el conjunto de estados meta, las acciones que permiten transformar un estado en otro con el coste asociado a dicha acción y por último la función de transición.

El espacio de estados *S* del problema del Sokoban será el conjunto de caracteres que representa el mapa de juego y la posición en la que se encuentran el jugador, las cajas, el destino de las cajas y las paredes que limitan el juego.

Los caracteres que pueden aparecer en la representación del mapa, los cuales se han elegido en base al formato estándar del Sokoban[10], son los siguientes:

- **Paredes:** Para la representación de las paredes que limitan el juego, ya sea en su condición de pared exterior o de pared interior, se utilizará el carácter *'#'*.
- **Jugador:** Para la representación de la posición del jugador se utilizará el carácter *'@'* cuando este se encuentre en una casilla vacía y en el carácter *'+'* cuando el jugador se encuentre sobre una casilla destino.
- **Cajas:** Para la representación de las cajas se utilizará el carácter *'$'* cuando las cajas se encuentren en una casilla vacía y el carácter *'*'* cuando la caja se encuentre colocada sobre una casilla destino.
- **Casillas vacías/destino:** Para la representación de las casillas vacías/destino que forman el mapa se utilizará el carácter espacio *' '* para representar que la casilla se encuentra vacía y el carácter *'.'* para representar que la casilla es un destino en el que se debe colocar una caja.

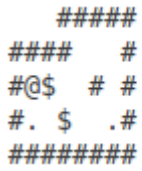


Ilustración 7: Ejemplo espacio de estados

El estado inicial en el cual comenzará el juego puede ser cualquier combinación de los caracteres antes mencionados que forman un mapa de juego con las condiciones de existir al menos una caja que no se encuentre colocada sobre una casilla destino, que existan los mismos destinos que cajas a colocar y que el jugador esté representado en el mapa. La imagen anterior, que representa un posible mapa, puede ser el estado inicial de un nivel del juego debido a que cumple las condiciones anteriores.

El estado meta es la combinación de los caracteres del mapa de forma que todas las cajas se encuentren colocadas sobre casillas destino, es decir, en el mapa no debe de aparecer ningún carácter '$' que hace referencia a cajas sin colocar, ni ningún carácter '.' que hace referencia a casillas destino vacías. Basándose en lo anterior, existe un conjunto de estados meta que son válidos, esto se debe a que la posición del jugador no es significativa en el estado meta y, por lo tanto, los estados meta están formados por las cajas colocadas y una posición cualquiera para el jugador. A continuación, se incluye un ejemplo de un estado meta para el nivel que ha sido mostrado en la imagen anterior:

Ilustración 8: Ejemplo estado meta

Las acciones que se pueden realizar durante la resolución del problema se listan a continuación:

- **Mover al jugador:** Si el jugador se encuentra colocado al lado de una casilla vacía o casilla destino puede moverse hacia ella dejando su casilla origen vacía. La posición de la casilla vacía debe encontrarse encima, debajo, a la izquierda o a la derecha del jugador, no siendo válidas las casillas que se encuentran en diagonal.
- **Empujar una caja:** Si el jugador se encuentra colocado al lado de una caja ya sea a encima, debajo, izquierda o derecha puede empujar la caja cuando al otro lado de la caja se encuentre una casilla vacía. En este caso la casilla que estaba ocupada por el jugador se quedará vacía, la casilla ocupada por la caja pasará a estar ocupada por el jugador y la casilla que se encontraba vacía pasará a estar ocupada por la caja.
- **Colocar una caja:** si el jugador se encuentra colocado al lado de una caja ya sea a encima, debajo, izquierda o derecha puede colocar la caja cuando al otro lado de la caja se encuentre una casilla destino. En este caso al igual que ocurría en el anterior la casilla ocupada por el jugador pasará a encontrarse vacía, la casilla ocupada por la caja pasará a estar ocupada por el jugador y la casilla destino estará ocupada por la caja y por lo tanto no se podrá colocar ninguna otra caja sobre ella.

El coste del problema se calcula basándose en el número de movimientos realizados para colocar todas las cajas en las casillas destino, por lo tanto, todas las acciones tienen coste 1, ya que se realiza un movimiento.

La función de transición utilizada es el movimiento del jugador para cambiar su posición en el mapa incluya el movimiento de una caja o no. El estado en el cual quedaría el mapa tras la realización de una acción ha sido explicado junto a la descripción de la acción.

Por último, la salida que generará el solver será la secuencia de teclas que el usuario debería pulsar para resolver el nivel de forma satisfactoria. Cada una de estas teclas se corresponden con un movimiento que realizará el jugador, pero al jugador se le mostrará en la interfaz gráfica la implementación de la secuencia para que pueda observar los movimientos que se realizan.

4.2. Algoritmo elegido

Dada la naturaleza del juego y su complejidad se ha optado por un algoritmo de búsqueda informada o heurística. Esto es debido a que gracias a la heurística que se proporcione, la cual se explicará en el apartado *4.3. La función de evaluación*, el algoritmo contará con un conocimiento extra del entorno que le ayudará a obtener una solución de una manera más rápida que al usar un algoritmo ciego o de fuerza bruta.

El algoritmo de búsqueda heurística elegido ha sido A* dado que es completo y óptimo, ya que siempre encontrará la solución (si existe) y siempre expandirá el nodo más prometedor (menor evaluación heurística). A este algoritmo se le ha aplicado poda y algunas variaciones del original. En cuanto a poda, esto quiere decir que habrá ciertos nodos que no se expandirán al no cumplir unas restricciones las cuales controlan que no se generen posiciones muertas. Estas restricciones pueden consultarse en el apartado *4.4. Las restricciones*. Por otro lado, las

variaciones mencionadas anteriormente son usar funciones de evaluación que no son la tradicional de $f(x) = g(x) + h(x)$ y aplicar un procedimiento diferente a la hora de comprobar si un nodo está en la lista de abiertos. Estos cambios se deben a que se quiere priorizar el tiempo de cálculo antes que la búsqueda de la solución óptima, por lo tanto, a la hora de generar un nuevo nodo se comprobará si este ya está en la lista de abiertos como de cerrados, y en caso de estar en alguna no se volverá a estudiar, aunque esto signifique que se haya podido llegar a ese nodo con un coste menor que el ya estudiado. Para simplificar, a la hora de mencionar este algoritmo en la memoria se seguirá mencionando como A*, pero se tendrá que tener en cuenta los cambios mencionados en este párrafo.

Por otro lado, se han realizado pruebas con otros algoritmos de búsqueda informada como el IDA*, cuyos resultados, junto al del A*, podrán verse en el apartado *5. Experimentos realizados*. Con el uso de IDA* se ha buscado un ahorro de memoria en cuanto a nodos estudiados.

La implementación del algoritmo A* para este trabajo, con sus variaciones, se ha realizado de la siguiente forma en Java:

- Se han implementado dos listas:
  - Una cola de prioridades para los nodos por estudiar donde el orden de posicionamiento será el valor devuelto por la función de evaluación de cada nodo.
  - Una lista para almacenar todos los nodos ya estudiados o repetidos (mismo estado de mapa, aunque diferente valor de evaluación) y así evitar volverlos a estudiar en un futuro y no entrar en bucles.
- El algoritmo recibirá como primer nodo el estado actual del jugador y expandirá este nodo con todos los hijos posibles que se puedan generar basado en los movimientos y las restricciones cumplidas. Estos hijos que cumplan con lo anterior serán añadidos a la cola de prioridades siempre que su estado del mapa no haya sido estudiado ya.
- Mientras la cola de prioridades tenga nodos, o el nodo que actualmente se esté estudiando no sea meta, el algoritmo expandirá el nodo que se esté estudiando siempre y cuando no haya sido estudiado previamente, guardando en orden todos sus hijos que cumplan las restricciones en la cola de prioridades.
- Si el nodo que actualmente se está estudiando es meta el algoritmo retornará una secuencia de caracteres con los movimientos a realizar desde la posición desde la que el jugador invocó al solver hasta la meta, además se devolverán el total de nodos estudiados y el total de tiempo invertido en la solución para futuros estudios.
- Si la cola de prioridades se queda vacía significará que el nivel no tiene solución y se retornará este hecho.
- El sistema actualizará en la base de datos el documento del nivel que se ha resuelto añadiéndole el tiempo total invertido, la cantidad de nodos estudiados y un documento embebido con los movimientos a realizar para dar la solución, donde cada entrada tendrá la tecla del movimiento y el valor de la función de evaluación y una representación del escenario para ese estado. Todos estos datos

serán guardados para perfeccionar la heurística o para encarar el problema una vez se tengan suficientes datos usando clasificadores.

A continuación, se adjunta en pseudocódigo el algoritmo implementado para el solver del Sokoban:

```
FUNCTION A* (Nodo padre)
    abiertos.push(padre);
    WHILE abiertos NO VACIO
        Actual = abiertos.pop();
        IF Actual = HAS_GANADO
            RETURN SOLUCIÓN;
        ELSE
            cerrados.add(Actual);
            hijos = getHijos(Actual);
            WHILE hijos NO VACIO
                IF hijos[i] NO ESTA EN abiertos AND cerrados
                    abiertos.push(hijos[i]);
                END IF
            END WHILE
        END IF
    END WHILE
    RETURN NULL;
END FUNCTION
```

```
FUCTION getHijos (Nodo padre)
    i = 0;
    WHILE i MENOR MÁXIMO_MOVIMIENTOS
        mapa = padre.getMapa();
        IF realizarMovimiento(mapa, i) VERDADERO
        posicion = mapa.cajaMovida(padre.getMapa());
            IF posicion NOT NULL
                IF comprobarRestricciones(posicion,mapa) VERDADERO
                    hijos.add(mapa.getNodo());
                END IF
            ELSE
                hijos.add(mapa.getNodo());
            END IF
        END IF
        i++;
    END WHILE
    RETURN hijos;
END FUNCTION
```

Ilustración 9 Pseudocódigo A y expansión*

4.3. La función de evaluación

A la hora de buscar la función de evaluación a usar para la búsqueda informada se han considerado varias candidatas, teniendo como meta elegir la función de evaluación que menos ratio nºpasos/tiempo utilice para completar el global del set de 150 niveles.

En primer lugar, se ha optado por probar una función de evaluación con una heurística admisible:

- $h(x)$: Coste aproximado de colocar las cajas.
- $g(x)$: Coste del recorrido hasta el nodo *x*.

Donde $f(x) = g(x) + h(x)$. Con esta función de evaluación garantizamos la solución óptima, sin importar las modificaciones hechas en el A*, ya que para este dominio cualquier nodo repetido que llegue siempre tendrá una evaluación mayor que el estudiado previamente. Por el contrario, el tiempo de cómputo de los niveles es demasiado elevado ya que el solver pierde el tiempo al priorizar nodos que distan bastante de tener solución próxima, pero se estudiaban al tener como resultado una evaluación baja para ese nodo.

Por otro lado, se ha decidido también probar una función de evaluación con una heurística voraz, $f(x) = h(x)$, donde *h(x)* es el número de cajas colocadas en destinos. Esta heurística es de doble filo, ya que puede crear algunos casos donde se den nodos prioritarios debido a que se tienen cajas colocadas pero la situación actual del escenario no permite colocar las restantes sin tener que mover previamente estas, por lo tanto, primero se tendrán que estudiar todos los nodos y sus diferentes expansiones basándose en el estado actual con las

cajas anteriores en su destino antes de plantearse moverlas. Por el contrario, muchas veces al colocar la primera caja se puede descubrir ya una ruta a seguir para colocar las restantes sin que el colocar una suponga bloquear este camino, esto puede darse en escenarios donde se presenten todos los destinos en la misma zona.

A continuación, se exponen dos ejemplos para comprender lo explicado anteriormente sobre la heurística voraz, donde en un caso esta heurística no es la adecuada y en otro donde funciona perfectamente.

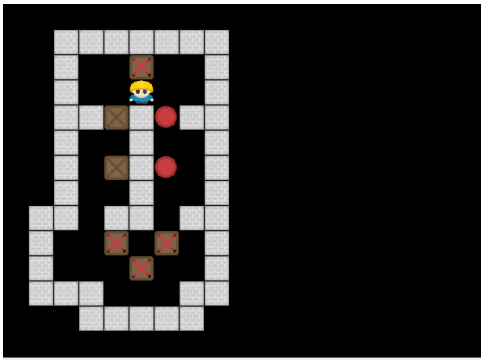


Ilustración 10 Nivel donde heurística voraz falla

En la imagen anterior podemos apreciar como el nivel tienen las posiciones destino dispersas por todo el escenario, y algunas de ellas con cajas ya colocadas. Si aplicáramos la heurística voraz para resolverlo primero movería todas las cajas que no están en destino a lo largo de todas las posiciones posibles del escenario, pero no se podrían llevar a su destino debido a que es necesario mover las colocadas previamente. Hasta que el solver decidiera mover una caja en destino pasaría bastante tiempo y se habrían estudiado muchos nodos inútiles ya que no proporcionaban solución para ese estado.

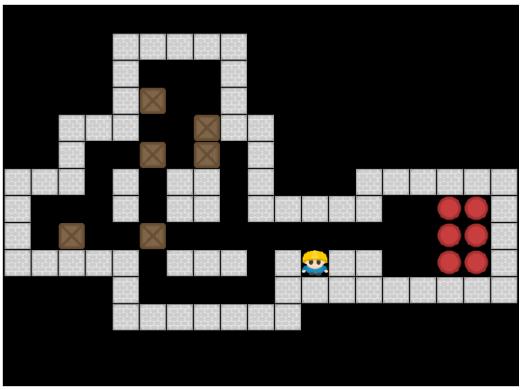


Ilustración 11 Nivel donde heurística voraz funciona

Por el contrario, en esta imagen podemos ver todas las posiciones destino en una misma zona y, además, se puede apreciar que al colocar una caja no se bloqueará ningún camino para poder llevar las restantes, por lo tanto, en este caso si funcionaría correctamente la heurística voraz.

Para tratar de evitar los problemas que puedan surgir con la heurística voraz vista anteriormente se ha considerado otra función de evaluación que deja de priorizar por cajas colocadas y prioriza por distancia Manhattan, la cual es una heurística más informada que la

anterior. Con esta explicación parece que esta función es igual que la función de evaluación admisible explicada en párrafos anteriores, pero se le ha realizado un cambio. Ahora el coste acumulado (g(x)), sólo se tiene en cuenta en caso de empate de h(x). Se ha decido probar esta forma debido a que se detectaba en los resultados probados con la función admisible que añadir g(x) siempre al valor de evaluación del nodo no implicaba que guiará bien en la búsqueda debido a las modificaciones realizadas en el A*. Por lo tanto, la nueva función de evaluación está compuesta por $f(x) = h(x) \mid g(x)$. Donde:

- $h(x)$: Coste aproximado de colocar las cajas.
- $g(x)$: Coste del recorrido hasta el nodo *x*.

En este caso se priorizará primero los nodos cuyo coste aproximado de colocar las cajas restantes (h(x)) sea menor, sin tener en cuenta las cajas ya colocadas. En caso de empate se tendrá en cuenta el coste acumulado (g(x)) para deshacer la igualdad y establecer el orden.

Finalmente, tras estudiar las tres funciones de evaluación descritas anteriormente se ha decidido elegir una que es combinación de ellas y la cual ya se introdujo en el apartado *2.2. Resumen de la aplicación desarrollada*. Esta función de evaluación está formada por las siguientes funciones:

- $i(x)$: Número de cajas colocadas en destino.
- $h(x)$: Coste aproximado de colocar las cajas.
- $g(x)$: Coste del recorrido hasta el nodo *x*.

Donde $f(x) = i(x) \mid g(x) + h(x)$. Con esta función de evaluación se pretende conseguir una solución buena en cuanto a pasos (heurística admisible h(x)) en un tiempo rápido (heurística voraz i(x)). Por lo tanto, la función de evaluación queda definida o por el número de cajas colocadas correctamente (en un destino) o la suma de movimientos hasta la configuración *x* más el coste aproximado de llevar a los destinos las cajas restantes. En resumen, dentro de la cola abierta del algoritmo A* aparecerán primero los nodos con más cajas acomodadas correctamente y en caso de empate seguirán aquellos con menor costo acumulado de g(x) y h(x).[2]

A continuación, se pasan a detallar cada una de las partes que componen nuestra función de evaluación elegida para usarla con el algoritmo A*:

- $g(x)$: Es el coste acumulado hasta el nodo actual, es decir, la suma total de movimientos hasta llegar a esa posición. En cada transición este coste será incrementado en uno, debido a que las reglas del Sokoban solo permiten desplazarse de una en una casilla vertical u horizontalmente.
- $h(x)$: Se ha definido como el coste aproximado de llevar todas las cajas restantes a sus destinos. Para calcular este coste de una manera realista se usará la distancia Manhattan, ya que es la que mejor se adapta a nuestras restricciones de sólo mover vertical u horizontalmente. Además, la distancia Manhattan garantiza que nuestra heurística sea admisible.
- $i(x)$: Simplemente es el cálculo de las cajas que actualmente ya se encuentran en una posición destino. Para su obtención se recorrerá el escenario en busca de destinos ocupados y se guardará este valor.

Para finalizar este apartado matizar que se ha probado el set de 150 niveles propuesto con cada una de las funciones de evaluación mencionadas en este apartado, los resultados obtenidos en estas pruebas, y los cuales justifican nuestra elección, pueden ser consultados en el apartado *5. Experimentos realizados*.

4.4. Las restricciones

En este apartado se explicarán las decisiones tomadas en cuanto a restricciones para conseguir que el solver sea lo más eficiente posible a la hora de encontrar la solución para un nivel. De esta manera se podarán los nodos que generen posiciones muertas dentro del algoritmo A*.

Las restricciones consideradas sólo podrán comprobarse una vez expandido el nodo, ya que el movimiento se podrá realizar, pero el resultado de este dejará un escenario cuya situación hará que el nivel sea imposible de resolver. En caso de que un nodo no las cumpla se descartará y no se añadirá a la cola de prioridades. De esta manera, al descartar los nodos que originan posiciones muertas se consigue un ahorro considerable en cómputo al no seguir expandiéndolos y estudiándolos en el solver.

Este tipo de restricciones puede dividirse en cinco grupos: esquinas, bloques, paredes limitadas, camino bloqueante y bloques especiales. Estas restricciones sólo se comprobarán en el solver en caso de que al expandir un nodo se haya movido una caja, por lo que se estudiará si el movimiento de esa caja ha creado alguna posición muerta al no cumplir una o varias de estas restricciones. En los siguientes párrafos se pasan a detallar los cinco grupos de restricciones consideradas.

Las esquinas son posiciones del escenario donde nunca se debería llevar una caja, a no ser que allí haya un destino, porque generaría una posición muerta. La detención de esta restricción es muy simple, para la caja desplazada sólo hay que ver si tiene en lados consecutivos (sur-oeste, sur-este, norte-oeste, este-norte) paredes.

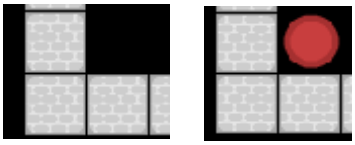


Ilustración 12 Restricción de esquinas

Los bloques son composiciones cuadradas de cajas, muros y casillas vacías que hacen imposible desplazar las cajas de las que están compuestos. Para esta práctica se ha realizado la comprobación de bloques 2x2 y 3x3, ya que bloques superiores serían una composición de estos dos últimos.

En primer lugar, los bloques 2x2, son sencillos de detectar, ya que sólo hay que comprobar para la caja movida si en sus alrededores hay un total de casillas cuya suma de muros y cajas (incluyéndose a sí misma) da un valor de cuatro. Lo que implicaría un bloque sólido imposible de desplazar.

Por otro lado, en los bloques 3x3 hay más variantes, ya que puede darse el caso que incluso con casillas vacías en sus límites (no es un bloque sólido) se dé una posición muerta. En nuestro caso consideramos bloque 3x3 que provoca posición muerta las siguientes variantes: bloque 3x3 sólido (está compuesto en sí mismo por un bloque de 2x2), bloque 3x3 con centro vacío, bloque 3x3 con centro vacío y una esquina vacía y bloque 3x3 con centro vacío y esquinas opuestas vacías. Para detectarlos el método es similar al de los bloques 2x2, se estudiará los alrededores de la caja movida y se comprobará el número de cajas y muros que hay, así como sus posiciones respecto a la de la caja movida.

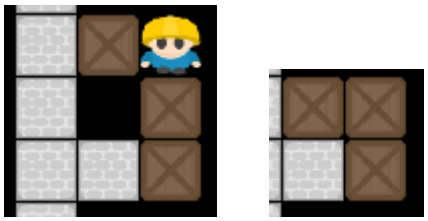


Ilustración 13 Restricción de bloques

Las paredes limitadas son posiciones del escenario donde al llevar una caja restringes para siempre el movimiento de esta a derecha e izquierda o arriba y abajo. Estas situaciones se evitarán siempre y cuando en esa zona de paredes no haya una casilla destino vacía. Para detectar cuando una zona es una pared limitada se comprobará al mover una caja si esta está pegada a una pared, si es así, se recorrerá tanto a derecha como a izquierda o arriba y abajo (dependiendo de la posición de la pared) hasta dar con unas casillas vacías a derecha e izquierda o arriba y abajo que permitan sacar la caja de esa zona, si no es este el caso, se recorrerá hasta dar con una esquina, esto indicará que esa zona es una pared limitada y se descartará ese nodo si no existe ninguna posición destino libre en esa área.

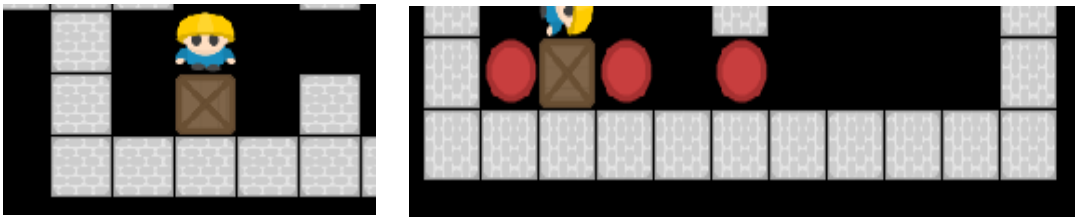


Ilustración 14 Restricción de paredes limitadas

Los caminos bloqueantes son situaciones del escenario donde una caja se encuentra con paredes a ambos lados, tanto a arriba y abajo o derecha e izquierda. Parecida a la restricción de paredes limitadas, pero en este caso, aunque haya una salida, habrá que comprobar si hay algún elemento por el camino que haga imposible sacar la caja (como puede ser otra caja) o si se da este caso, si el jugador se encuentra en el medio de estos dos elementos tendría la oportunidad de deshacer esta posición muerta. El método para detectar estas situaciones es similar al de paredes limitadas, pero ahora se tendrá en cuenta la posición del jugador y la de los elementos que puedan surgir por el camino.

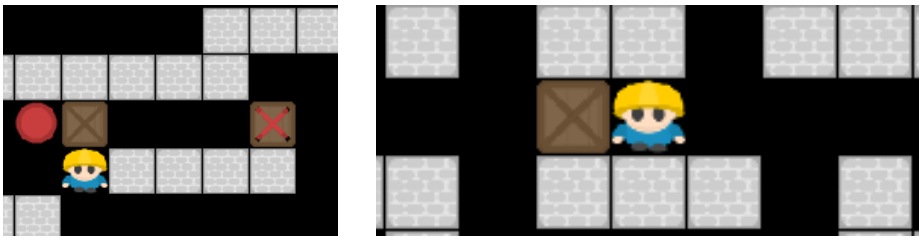


Ilustración 15 Restricción de camino bloqueante

Por último, tenemos los bloques especiales, estos son estados del escenario que dada su naturaleza no pueden agruparse dentro de bloques 2x2 o 3x3, pero provocan una posición muerta. Para detectarlos se ha tenido que meter la comprobación exacta del bloque tras mover una caja en el escenario. Por motivos de rendimiento no se han considerado todos estos tipos de bloques y únicamente se comprueban los que pueden ser más comunes a la hora de completar un nivel.

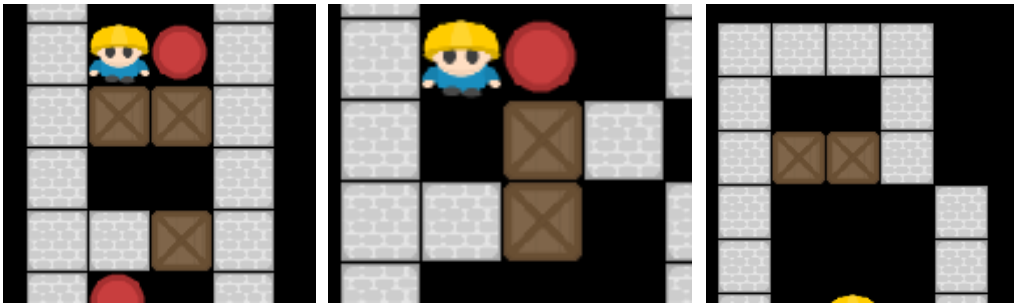


Ilustración 16 Restricción de bloques especiales

Para finalizar, como ya se ha mencionado anteriormente, aclarar que las restricciones sólo se comprobarán en el caso de que al realizar un movimiento se haya desplazado una caja, entonces sólo para esa caja se comprará si cumple con las restricciones explicadas anteriormente, y en caso de no cumplirlas no se generará ese nuevo nodo y se descartará.

5. Experimentos realizados

En este apartado se explicarán los experimentos que han sido realizados para probar el Solver antes explicado, mostrando los resultados que se han obtenido de los experimentos para poder realizar un análisis de dichos resultados.

Para la realización de los experimentos se han utilizado los dos algoritmos mencionados anteriormente, el A*, con sus variaciones, y el IDA*. Dentro del A*, el cual ha sido el elegido finalmente, se ha experimentado con diferentes funciones de evaluación para conocer cuál de ellas muestra unos mejores resultados en base a nuestro objetivo de reducir el ratio nºpasos/tiempo. Estas funciones de evaluación pueden ser consultadas en el apartado *4.3. La función de evaluación*.

En primer lugar, se realizarán los experimentos para conocer cuál de las funciones de evaluación es mejor para el algoritmo A*. Esta comparación se realizará en base a los parámetros:

- **Número de pasos**: Se tendrá en cuenta el número de pasos que el solver ha necesitado para completar el nivel. Este parámetro es similar al número de movimientos que realizaría el usuario para completar un nivel, y por lo tanto, el objetivo es minimizar este número de pasos.
- **Nodos**: Se tendrá en cuenta el número de nodos que el algoritmo ha necesitado expandir para conseguir encontrar una solución con cada función de evaluación. El número de nodos afecta a la cantidad de memoria que se necesita para completar el nivel, y por lo tanto, el objetivo es minimizar el número de nodos para minimizar el consumo de memoria.
- **Tiempo de búsqueda**: Se tendrá en cuenta el tiempo que el algoritmo ha necesitado con cada una de las funciones de evaluación para encontrar una solución al nivel. De esta manera se busca minimizar el tiempo de ejecución necesario para encontrar una solución.

La comparación de los algoritmos se realizará únicamente en base al tiempo invertido y nodos estudiados para encontrar una solución debido a que se conoce que el algoritmo IDA* tiene un menor consumo de memoria que el algoritmo A*. Por otro lado, para el IDA* se ha usado como función de evaluación la función admisible descrita en apartados anteriores, por lo que la comparación con el A* se realizará usando esa misma función de evaluación.

Una vez conocidos los algoritmos y las funciones de evaluación que se utilizaran en los experimentos es necesario incluir que dichos experimentos serán realizados con una suite de problemas del Sokoban que está formada por un total de 150 niveles. Para comprobar la escalabilidad de los algoritmos y las funciones de evaluación se utilizarán niveles de diferente dificultad de manera que los primeros niveles a los que se enfrentarán estarán formados por un total de 2 cajas y escenarios pequeños, mientras que los niveles de dificultad más elevada contendrán un total de 7 cajas y escenarios más grandes.

Los niveles utilizados se pueden encontrar en el archivo "*levels.txt*" que se encuentra en la siguiente ruta */Sokoban/niveles/levels.txt* del proyecto entregado.

Una vez explicada la forma en la que se realizarán los experimentos se explica la forma en la que se incluyen los resultados obtenidos para permitir una correcta visualización y una rápida comparación.

Después de la ejecución del Solver utilizando cada algoritmo y función de evaluación se realiza una separación de los niveles en base al número de cajas que contiene cada nivel:

- **2 cajas**: Los niveles que tienen dos cajas son los contenidos entre el nivel 1 y el 35 ambos incluidos.
- **3 cajas**: Los niveles que tienen tres cajas son los contenidos entre el nivel 36 y el 69 ambos incluidos.
- **4 cajas**: Los niveles que tienen cuatro cajas son los contenidos entre el nivel 70 y el 101 ambos incluidos.
- **5 cajas**: Los niveles que tienen cinco cajas son los contenidos entre el nivel 102 y el 124 ambos incluidos.
- **6 cajas**: Los niveles que tienen seis cajas son los contenidos entre el nivel 125 y el 143 ambos incluidos.
- **7 cajas**: Los niveles que tienen siete cajas son los contenidos entre el nivel 144 y el 150 ambos incluidos.

Para mejorar la visualización de los resultados y de la comparación se muestran gráficas que contienen los parámetros especificados anteriormente en base al número de cajas que contiene cada uno de los niveles. Una vez se hayan mostrado los resultados desglosados por el nivel al que corresponden se realizará una comparación a nivel global y por categoría de los resultados obtenidos por cada algoritmo y función de evaluación en los apartados *5.4. Conclusiones funciones de evaluación y 5.7. Conclusiones de algoritmos*.

5.1. Tiempo de ejecución por funciones de evaluación

En este punto se mostrarán los resultados de los tiempos de ejecución que se han obtenido para cada uno de los niveles utilizando las diferentes funciones de evaluación para el algoritmo A*.

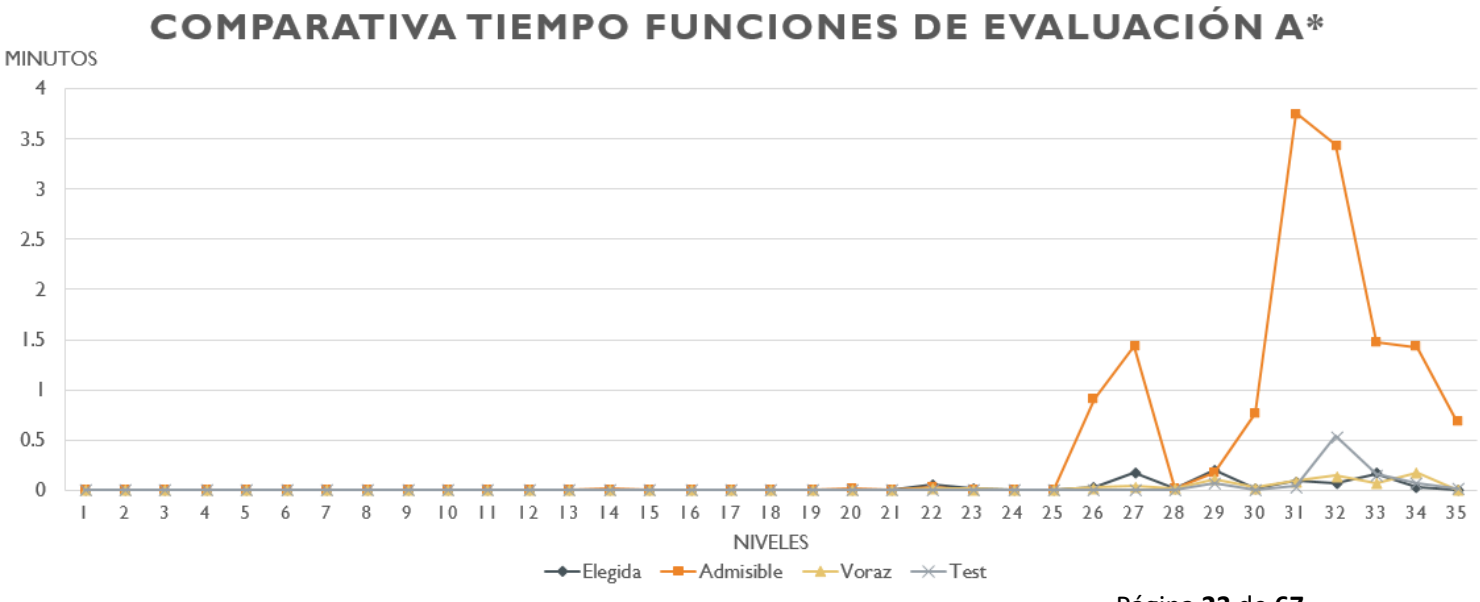


Ilustración 17: Tiempo de ejecución niveles de 2 cajas

La gráfica anterior refleja el tiempo invertido por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de dos cajas. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

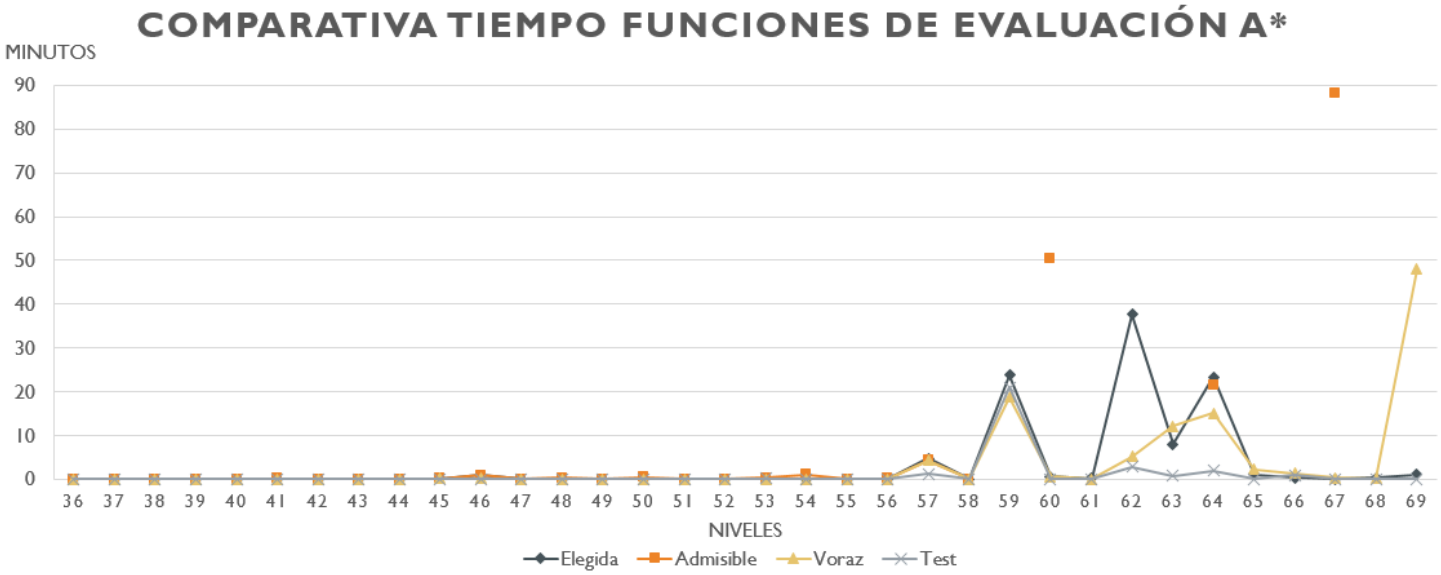


Ilustración 18: Tiempo de ejecución niveles de 3 cajas

La gráfica anterior refleja el tiempo invertido por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de tres cajas. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

Se puede observar en la gráfica anterior como para alguna función de evaluación en algunos niveles no aparece reflejado el tiempo de ejecución, esto se debe a que se ha fijado un tiempo máximo de ejecución de dos horas para cada nivel y si en ese tiempo no se encontraba una solución se saltaba al siguiente nivel. De esta manera se pretende obtener un buen rendimiento en el algoritmo y función de evaluación elegidas. Este mismo comportamiento se verá reflejado en las gráficas siguientes de mayor número de cajas por nivel y complejidad de escenario.

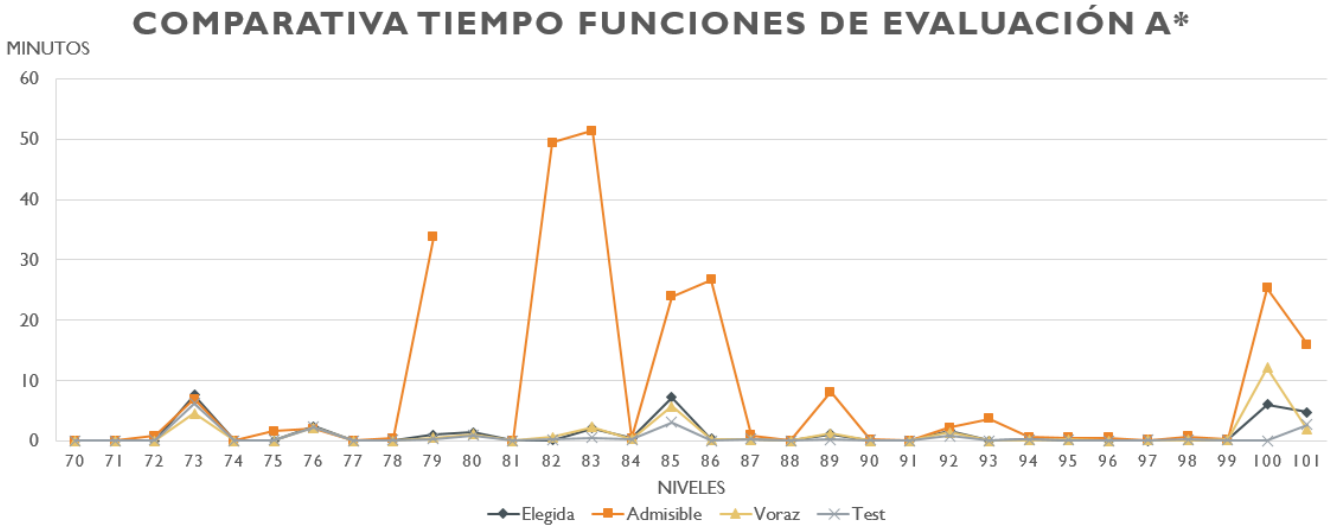


Ilustración 19: Tiempo de ejecución niveles de 4 cajas

La gráfica anterior refleja el tiempo invertido por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de cuatro cajas. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

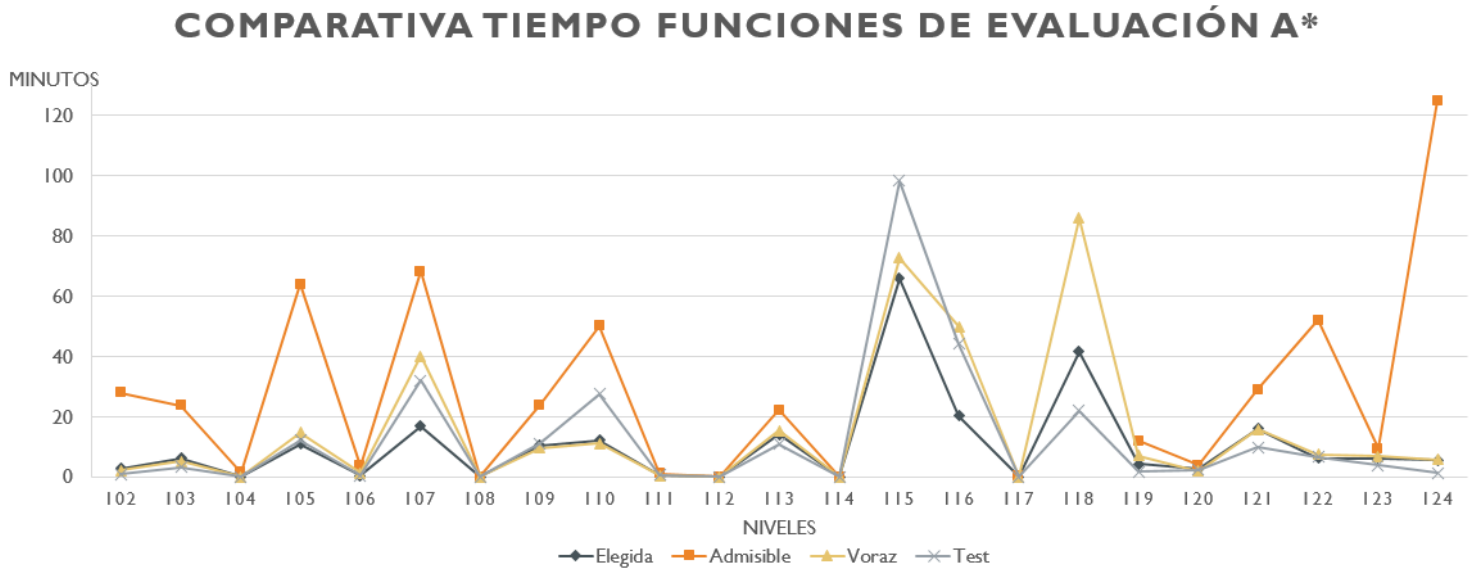


Ilustración 20: Tiempo de ejecución niveles de 5 cajas

La gráfica anterior refleja el tiempo invertido por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de cinco cajas. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

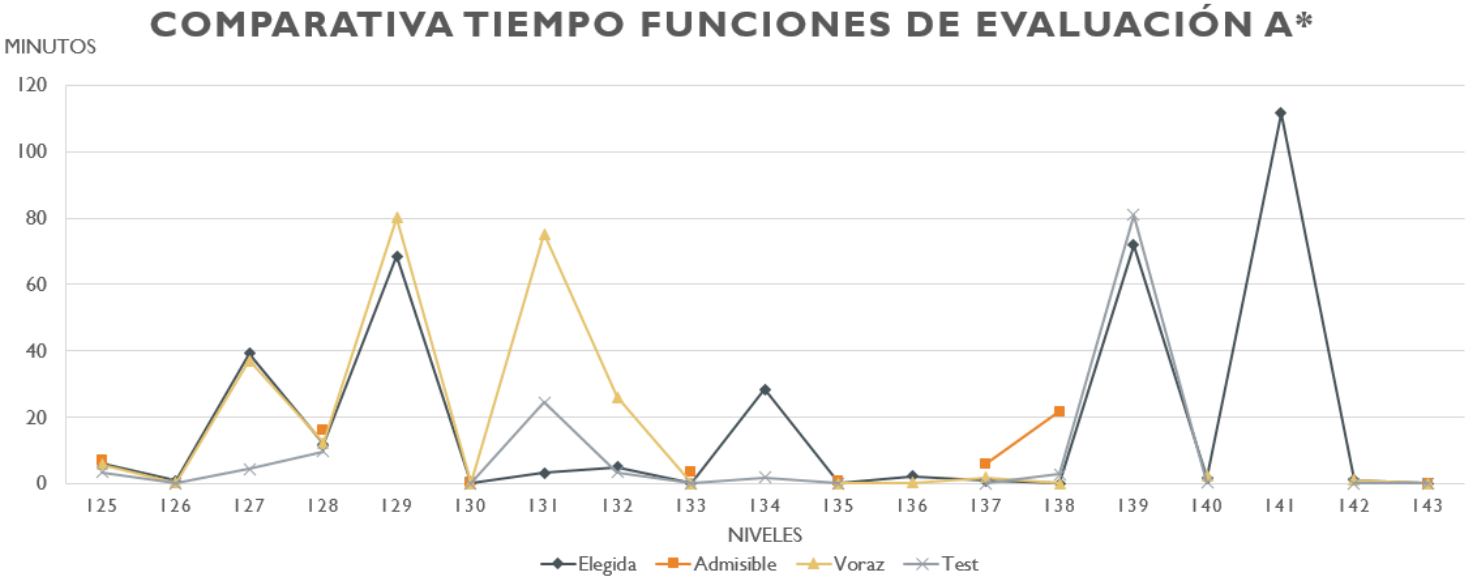


Ilustración 21: Tiempo de ejecución niveles de 6 cajas

La gráfica anterior refleja el tiempo invertido por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de seis cajas. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

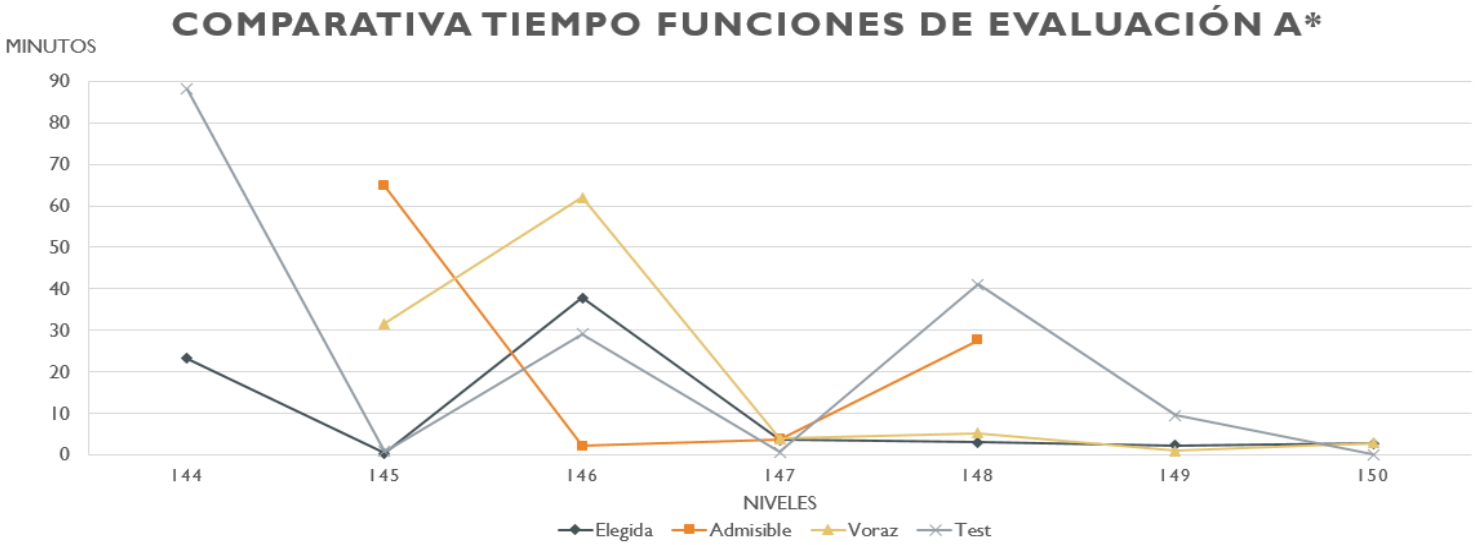


Ilustración 22: Tiempo de ejecución niveles de 7 cajas

La gráfica anterior refleja el tiempo invertido por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de siete cajas. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

Tras la realización de un análisis de los resultados mostrados en las gráficas anteriores y que se pueden observar en forma de tabla en el *ANEXO II: Tablas de datos*, se puede observar como la única función de evaluación que completa todos los niveles en un tiempo inferior a dos horas por nivel es la función de evaluación nombrada como "Elegida". De forma complementaria se puede observar como la función de evaluación "Elegida" es la que aporta un menor tiempo de ejecución de forma general, aunque algunas como la voraz y test aportan mejores resultados en para este valor en niveles de dos a cuatro cajas.

En el apartado *5.4. Conclusiones funciones de evaluación* se puede observar la comparativa general de las diferentes funciones de evaluación incluyendo el tiempo total invertido por cada una de las funciones para completar los 150 niveles y por categoría de número de cajas.

5.2. Nodos por función de evaluación

En este punto se mostrarán los resultados de los nodos que se han expandido para cada uno de los niveles utilizando las diferentes funciones de evaluación para el algoritmo A*.

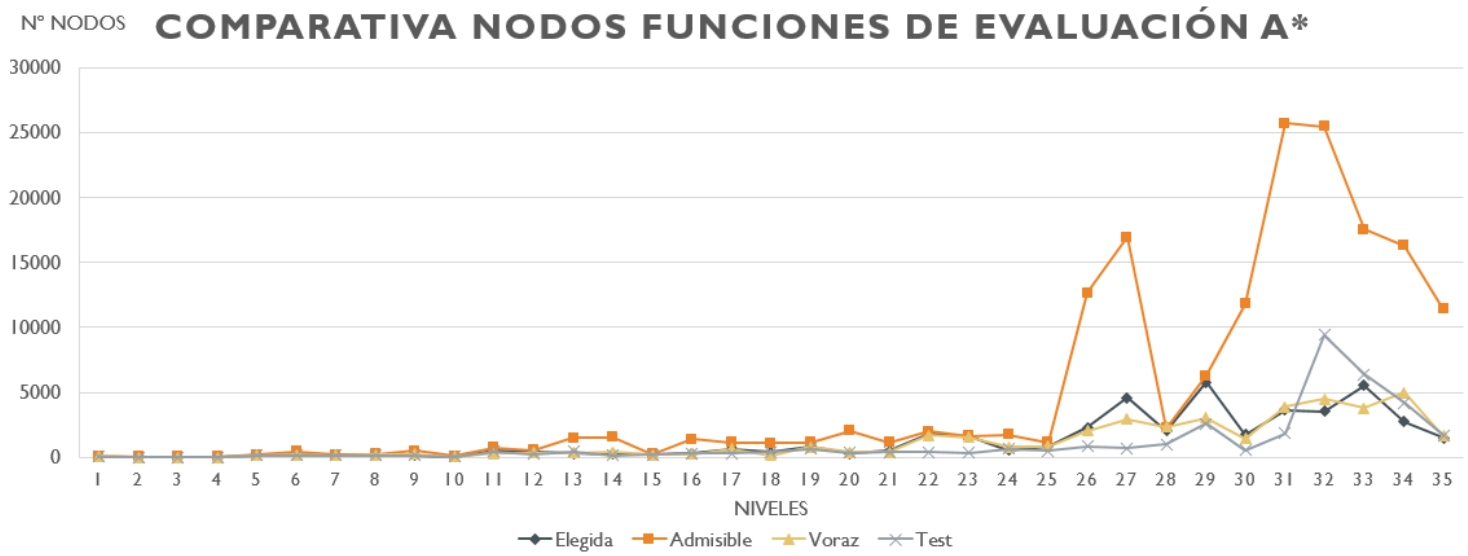


Ilustración 23: Nº nodos niveles 2 cajas

La gráfica anterior refleja el número de nodos expandidos por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de dos cajas.

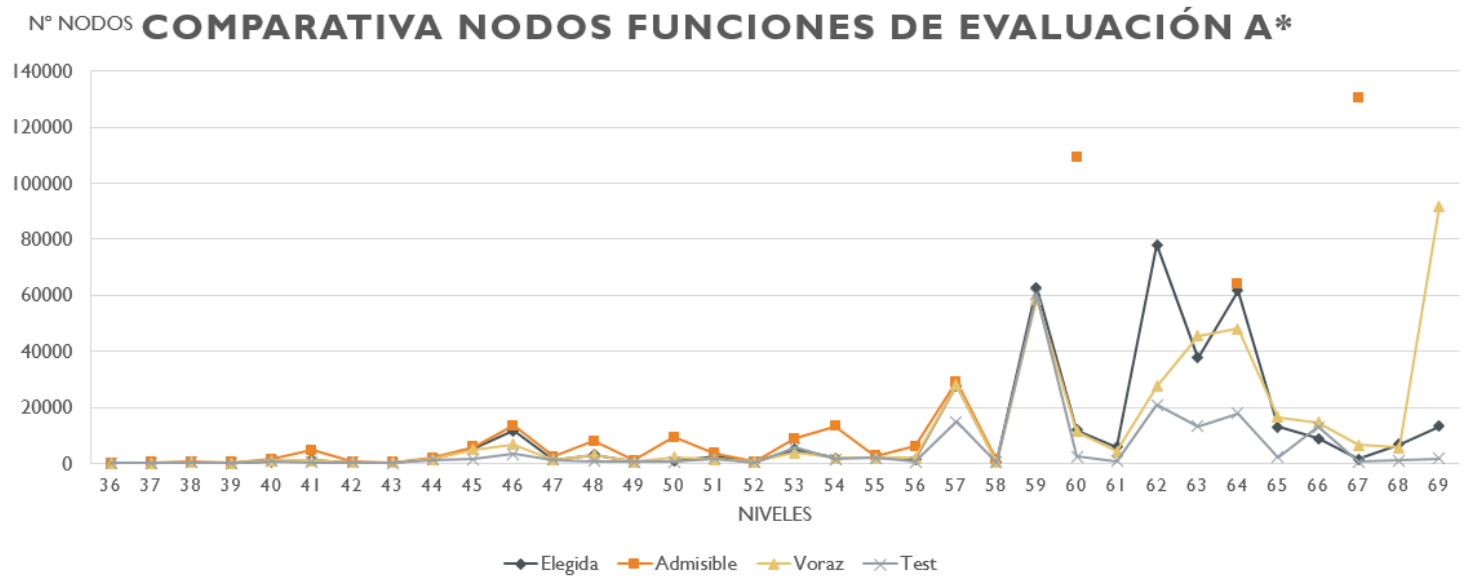


Ilustración 24: Nº nodos niveles 3 cajas

La gráfica anterior refleja el número de nodos expandidos por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de tres cajas.

Se puede observar en la gráfica anterior que para algunos niveles no aparece ningún número de nodos expandidos. Esto es debido a que como se ha comentado anteriormente se ha puesto un límite máximo de tiempo de dos horas de ejecución, y todos aquellos niveles que no han obtenido una solución en esas dos horas de tiempo no tendrán un numero de nodos expandidos. Por lo tanto, aquellos niveles que no tienen un numero de nodos es debido a que no tienen una solución para esa función de evaluación en un tiempo inferior a dos horas.

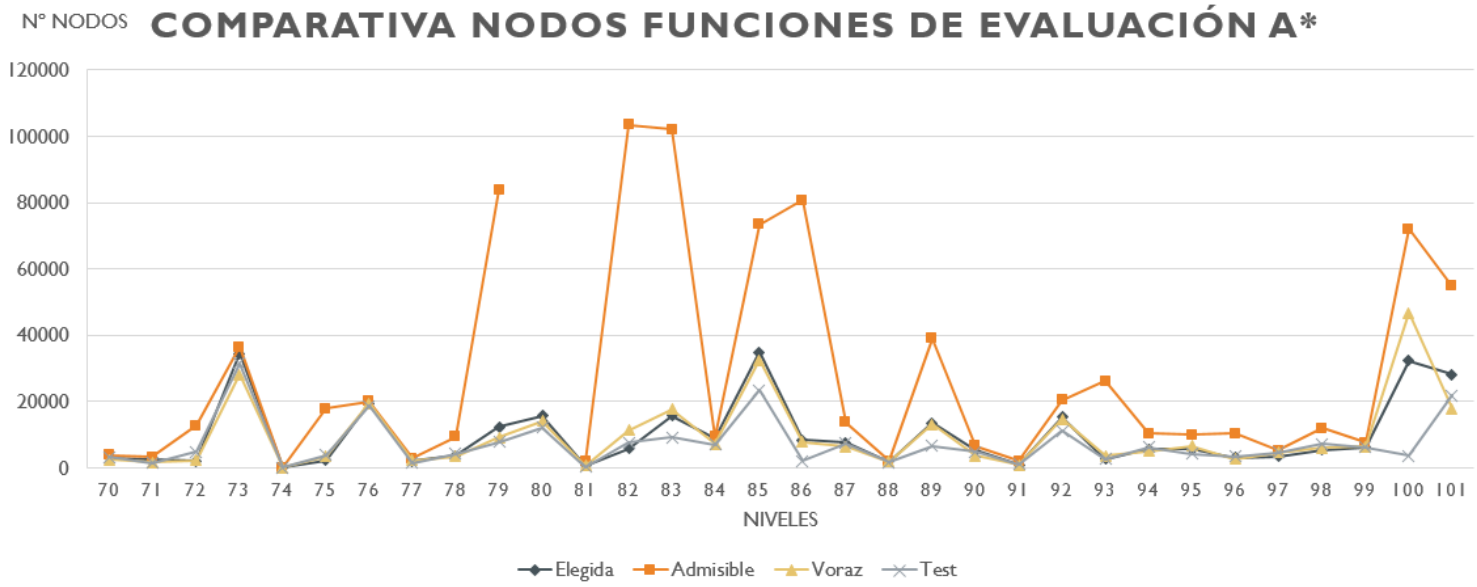


Ilustración 25: Nº nodos niveles 4 cajas

La gráfica anterior refleja el número de nodos expandidos por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de cuatro cajas.

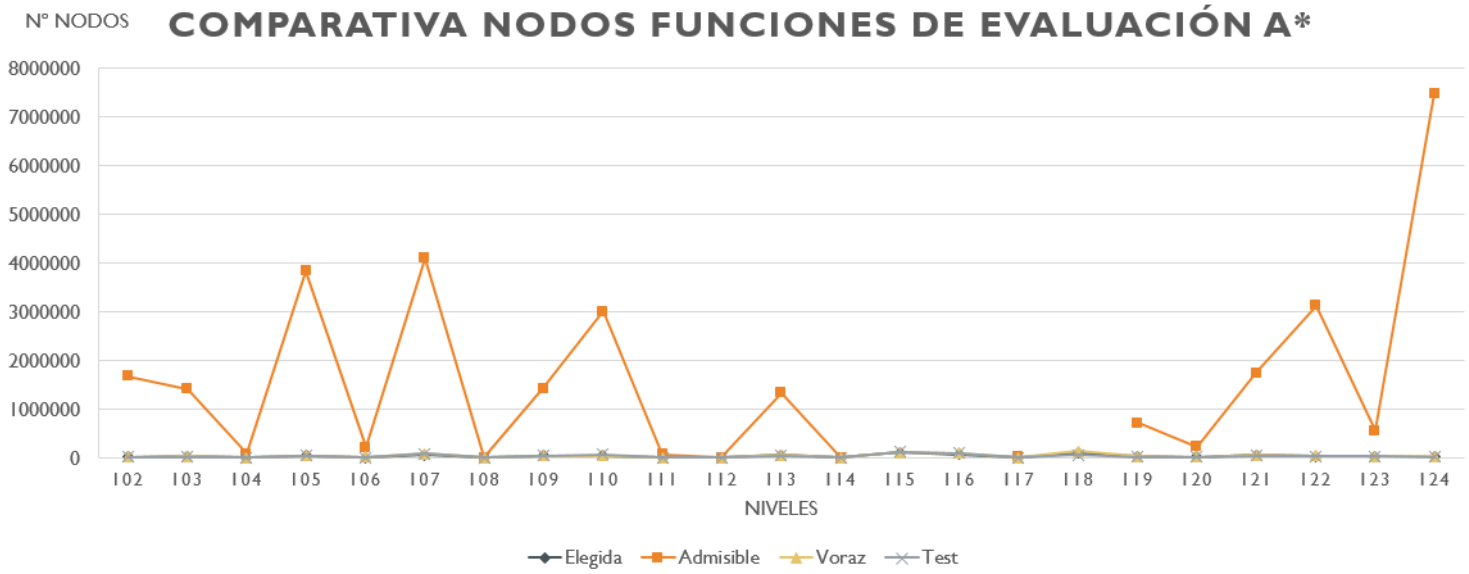


Ilustración 26: Nº nodos niveles 5 cajas

La gráfica anterior refleja el número de nodos expandidos por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de cinco cajas.

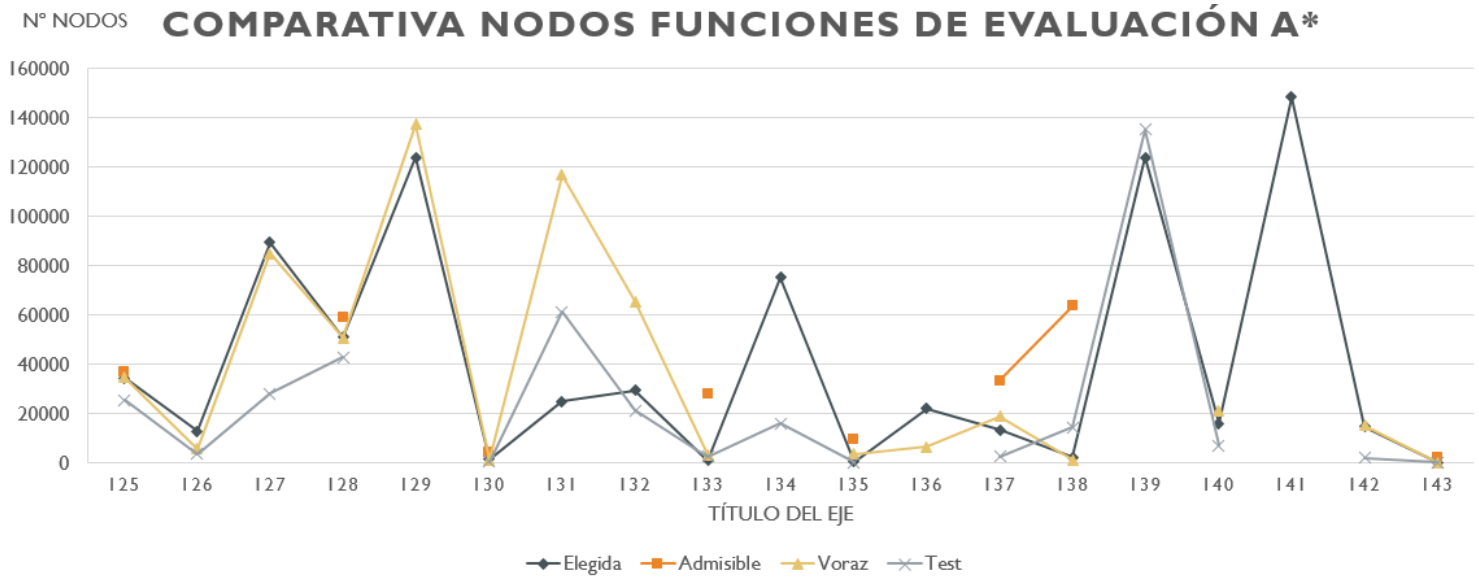


Ilustración 27: Nº nodos niveles 6 cajas

La gráfica anterior refleja el número de nodos expandidos por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de seis cajas.

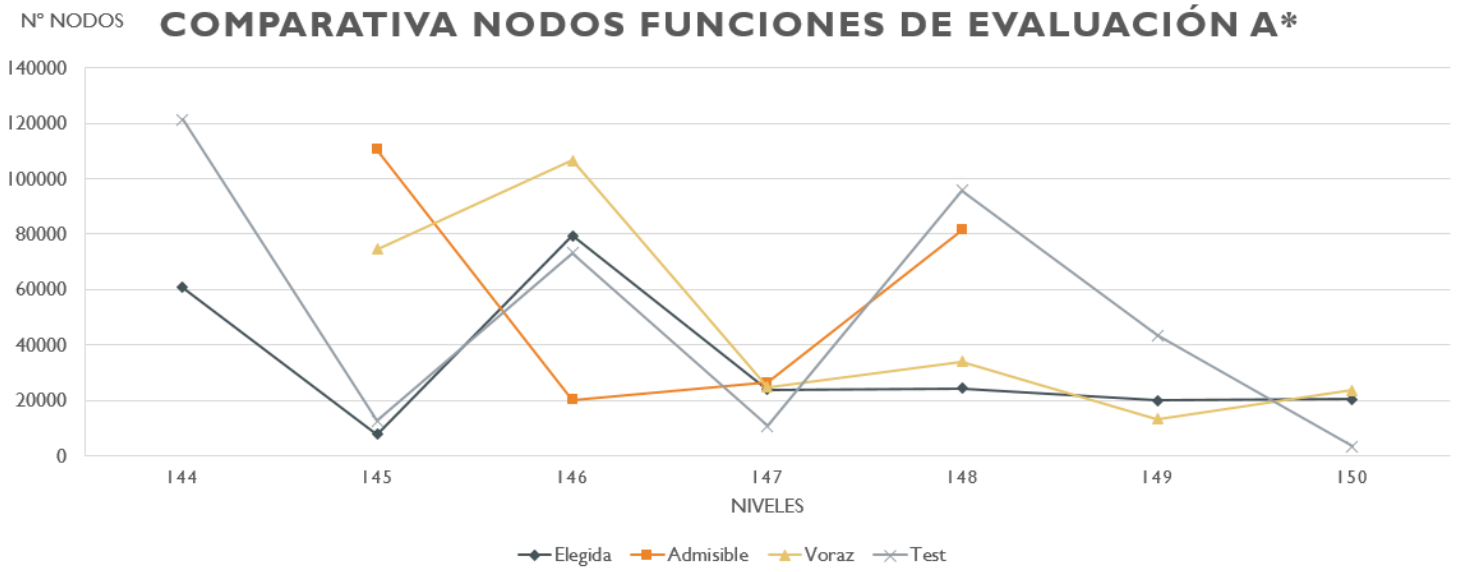


Ilustración 28: Nº nodos niveles 7 cajas

La gráfica anterior refleja el número de nodos expandidos por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de siete cajas.

Tras la realización de un análisis de los resultados mostrados en las gráficas anteriores y que se pueden observar en forma de tabla en el *ANEXO II: Tablas de datos*, se puede observar que las funciones de evaluación llamadas “Voraz” y “Test” tienen de forma general un número de nodos expandidos algo menor que la función de evaluación elegida, pero tienen la ventaja que solo reflejan los nodos utilizados en los niveles que han encontrado solución, por lo tanto, en base al tiempo que han utilizado en los niveles que no han encontrado solución dentro de

las dos horas se puede obtener la conclusión que cuando estas funciones de evaluación completaran los niveles, sin importar el tiempo, su número de nodos sería superior.

5.3. Pasos por función de evaluación

En este punto se mostrarán los resultados del número de pasos que es necesario realizara para completar cada uno de los niveles utilizando las diferentes funciones de evaluación para el algoritmo A*.

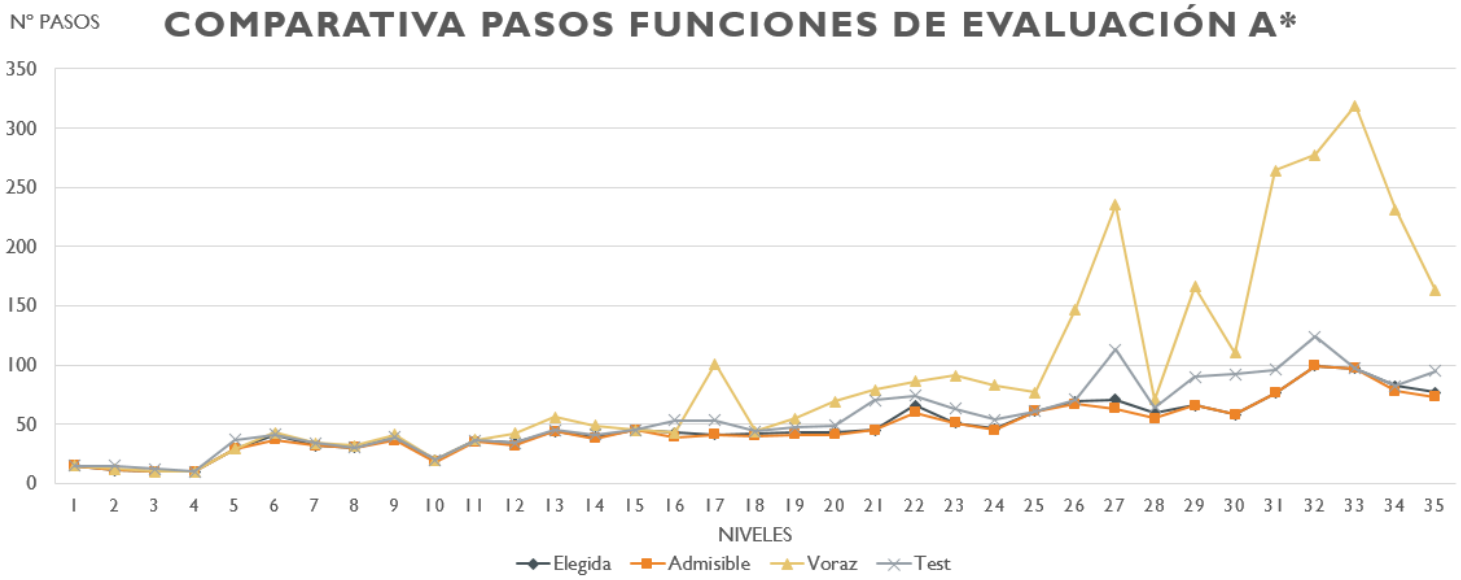


Ilustración 29: Pasos niveles 2 cajas

La gráfica anterior refleja el número de pasos utilizados por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de dos cajas.

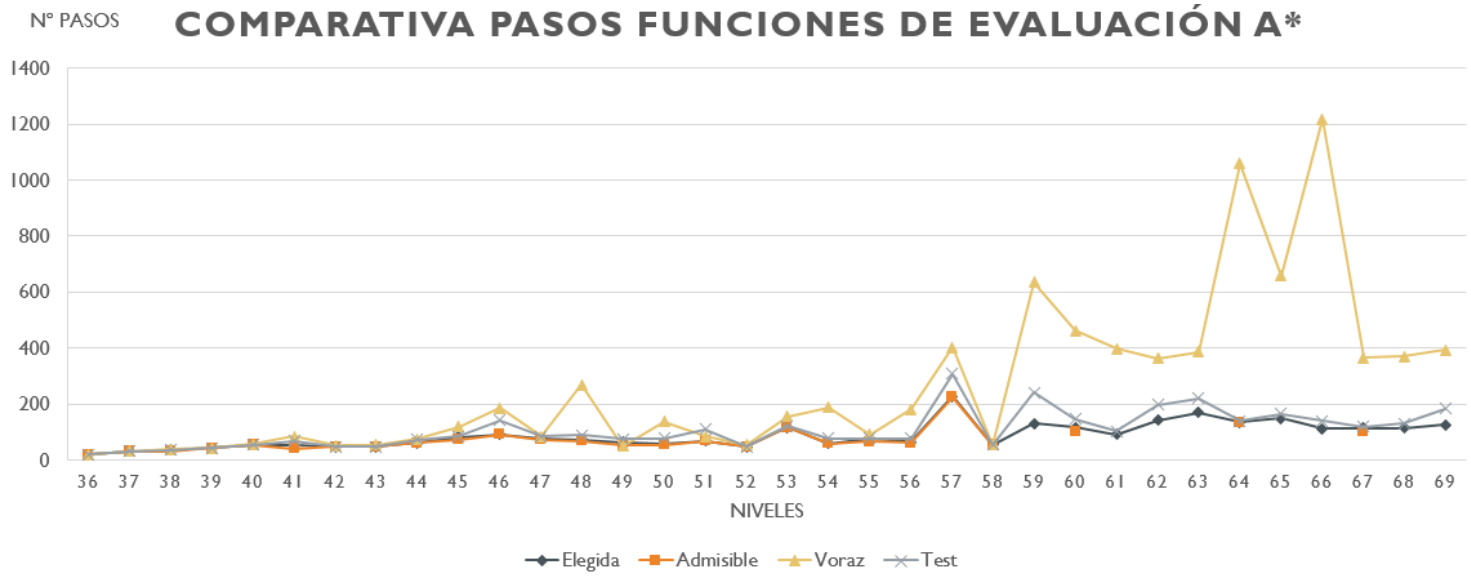


Ilustración 30: Pasos niveles 3 cajas

La gráfica anterior refleja el número de pasos utilizados por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de tres cajas.

Se puede observar en la gráfica anterior que para algunos niveles no aparece ningún número de pasos. Esto es debido a que como se ha comentado anteriormente se ha puesto un límite máximo de tiempo de dos horas de ejecución, y todos aquellos niveles que no han obtenido una solución en esas dos horas de tiempo no tendrán un numero de pasos. Por lo tanto, aquellos niveles que no tienen un numero de pasos es debido a que no tienen una solución para esa función de evaluación en un tiempo inferior a dos horas.

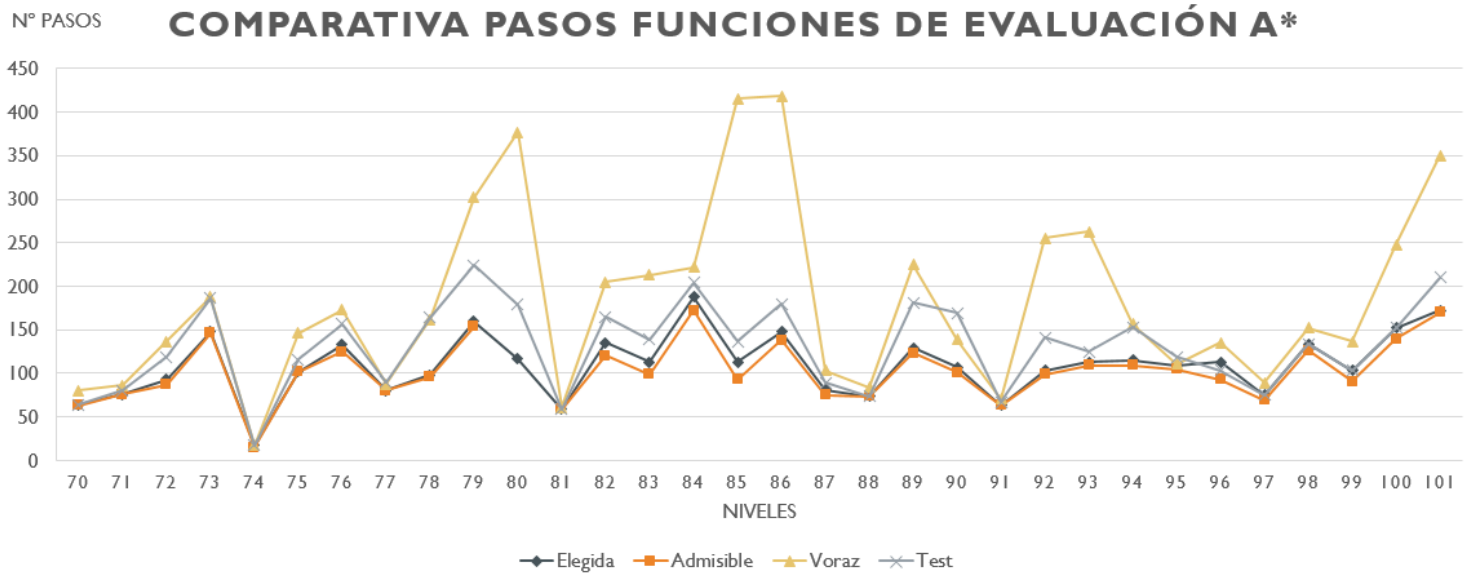


Ilustración 31: Pasos niveles 4 cajas

La gráfica anterior refleja el número de pasos utilizados por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de cuatro cajas.

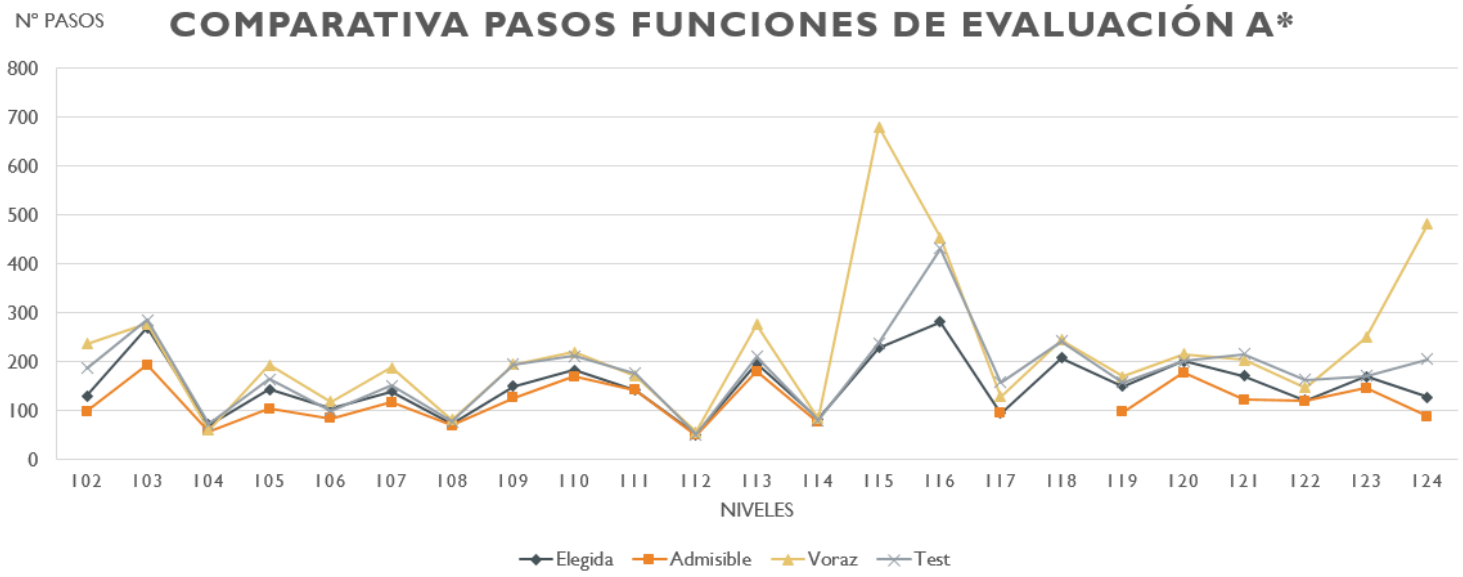


Ilustración 32: Pasos niveles 5 cajas

La gráfica anterior refleja el número de pasos utilizados por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de cinco cajas.

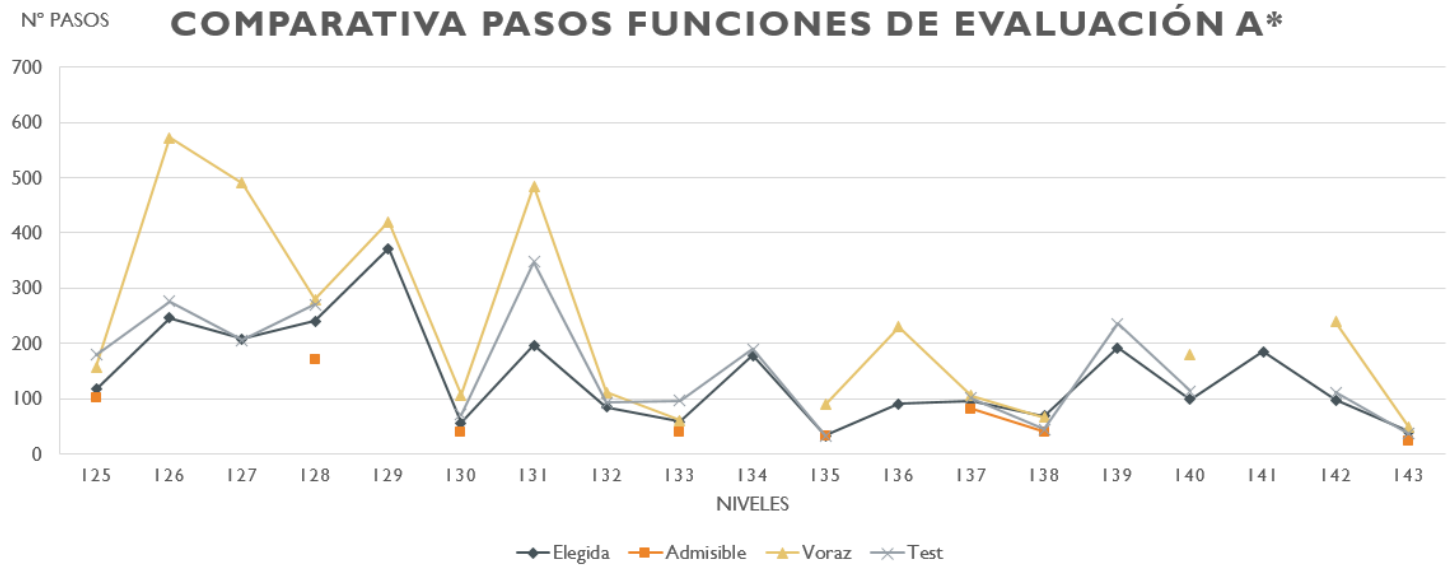


Ilustración 33: Pasos niveles 6 cajas

La gráfica anterior refleja el número de pasos utilizados por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de seis cajas.

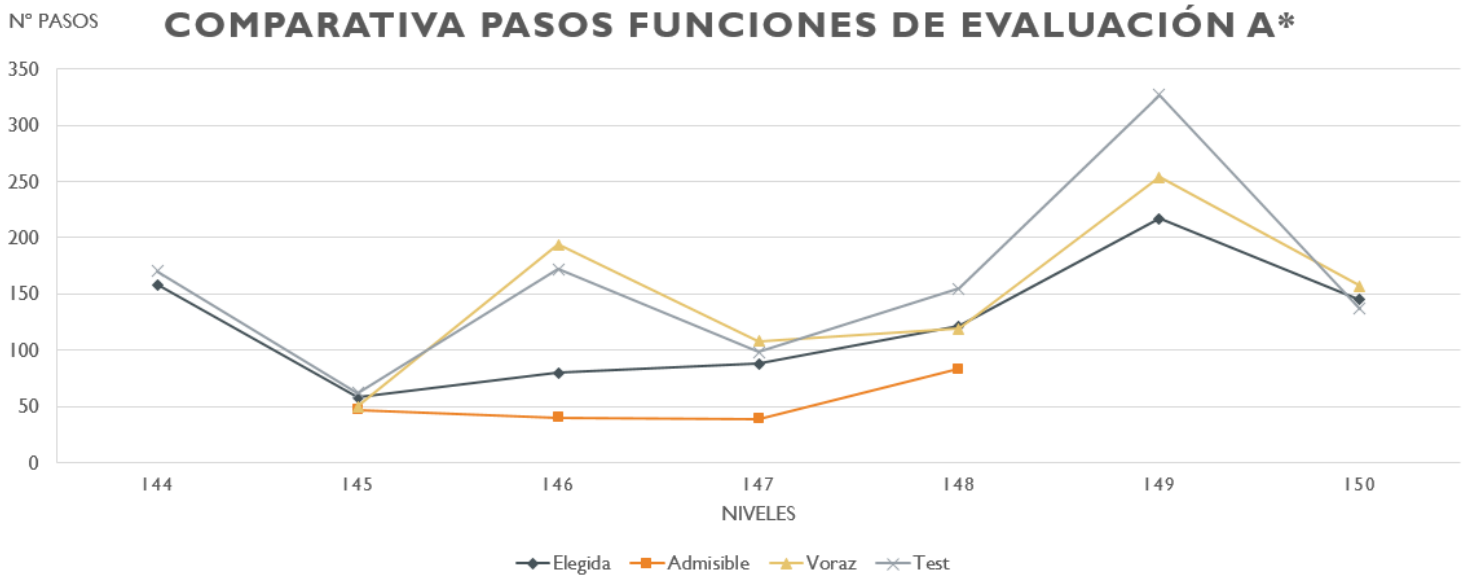


Ilustración 34: Pasos niveles 7 cajas

La gráfica anterior refleja el número de pasos utilizados por el algoritmo A* con las diferentes funciones de evaluación para resolver los niveles de siete cajas.

Tras la realización de un análisis de los resultados mostrados en las gráficas anteriores y que se pueden observar en forma de tabla en el *ANEXO II: Tablas de datos*, se puede observar que la función de evaluación "Admisible" tiene un número de pasos menor que la función de evaluación elegida para aquellos niveles que ha encontrado solución en un tiempo inferior a las dos horas. Por lo tanto, se puede deducir de las gráficas anteriores que todos los niveles para los cuales la función de evaluación "Admisible" ha encontrado solución es igual o mejor a la encontrada por el resto de funciones de evaluación.

5.4. Conclusiones funciones de evaluación

Después de realizar un análisis de los resultados obtenidos para las diferentes funciones de evaluación de manera desglosada por niveles en este punto se incluye la evaluación global y por categorías de las diferentes funciones de evaluación. De esta manera se pretende justificar la elección de la función de evaluación usada finalmente en el Sokoban.

Para evaluar cuál es la mejor función de evaluación, en primer lugar, se compara el número de niveles completados por cada una de las funciones:

Ilustración 35: Niveles resueltos por función

Se puede observar en la gráfica anterior como la única función de evaluación que ha conseguido completar la suite de 150 problemas en un tiempo inferior a dos horas por cada nivel ha sido la función de evaluación “Elegida”, mientras que la función “Test” y “Voraz” se han quedado en 147 y 146 niveles respectivamente. Finalmente, la “Admisible” ha quedado muy por detrás.

Después de conocer el número de niveles resueltos por cada función es necesario conocer el tiempo total que han invertido para resolverlos.

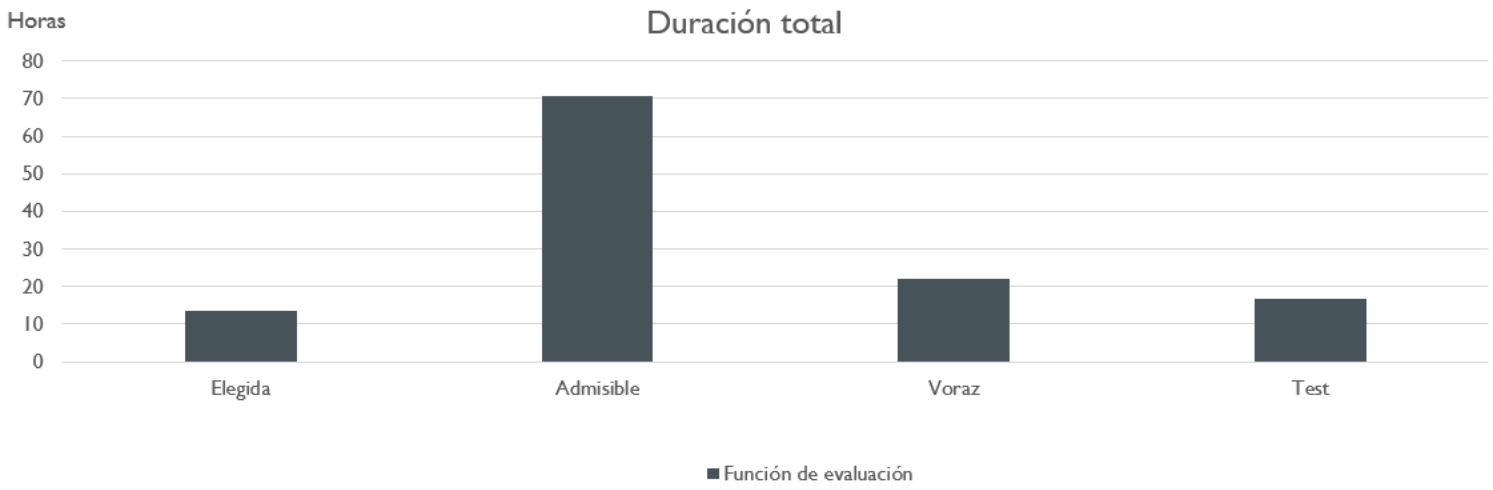


Ilustración 36: Duración total por función

La gráfica anterior refleja el tiempo que ha estado en ejecución el solver para intentar encontrar la solución de todos los niveles con las diferentes funciones de evaluación, realizando el cambio de nivel a las dos horas de ejecución en caso de no haber encontrado la solución en ese tiempo.

Se puede observar que la función de evaluación con un menor tiempo de evaluación es la función "Elegida" cuyo tiempo total es de 13 horas seguida de la función "Test" con un tiempo de 16 horas.

Los datos de las gráficas que se muestran a continuación para cada función de evaluación hacen referencia únicamente a los niveles para los cuales dicha función ha encontrado una solución, debido a que el número de nodos expandidos y el número de pasos de la solución son calculados una vez el nivel ha sido completado y se ha considerado no estimarse en caso de superarse las dos horas de cálculo de solución.

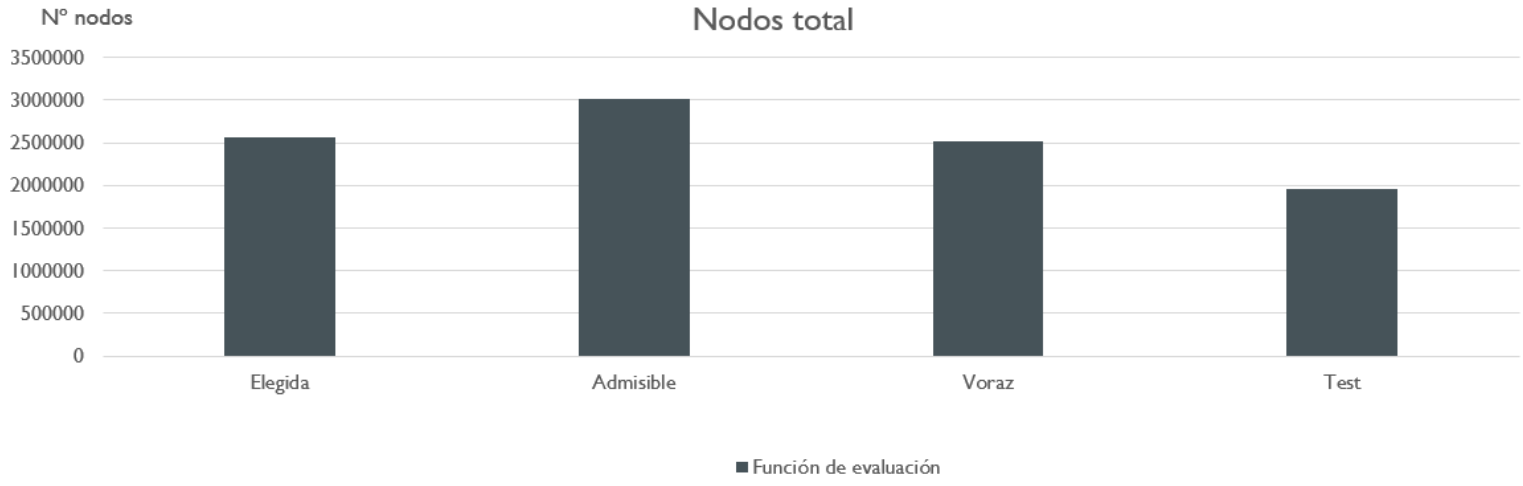


Ilustración 37: Nodos totales por función

En la gráfica anterior se muestran el número total de nodos que ha expandido cada una de las funciones en los niveles que ha completado. Se puede observar como las funciones "Voraz"

y “Test” tienen un número de nodos inferior a la función “Elegida” pero habiendo resuelto 4 y 3 niveles menos respectivamente.

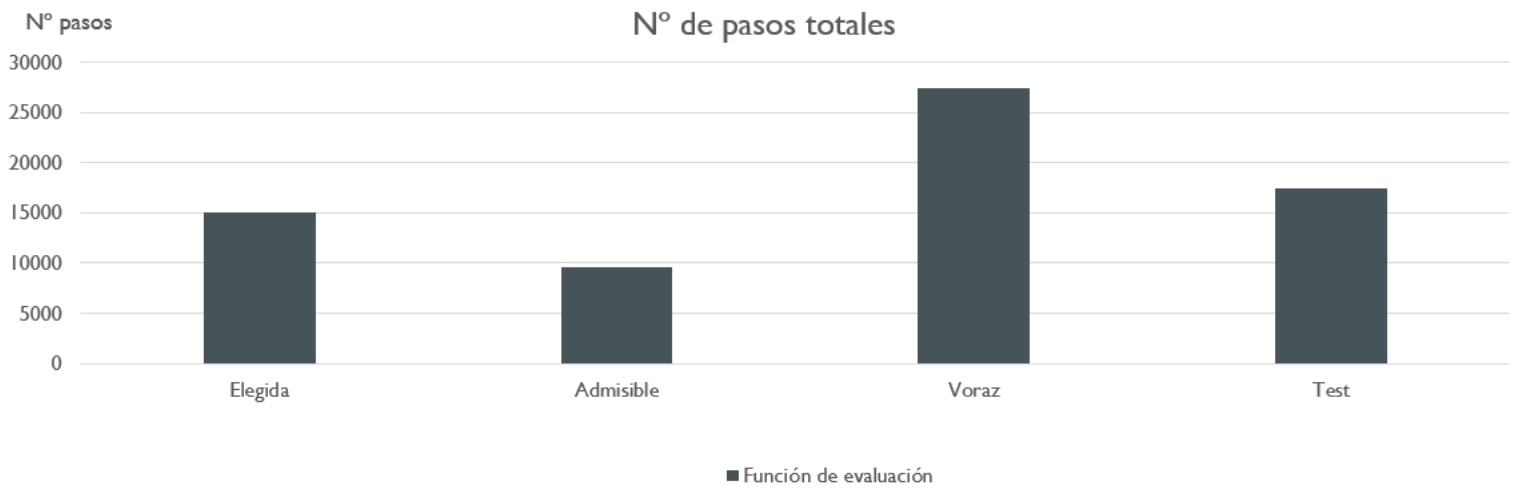


Ilustración 38: Pasos totales por función

En la gráfica anterior se muestra el número de pasos totales utilizados para completar los 150 niveles de la suite probada. Se puede observar como a pesar de disponer de un menor número de niveles resueltos, la función “Admisible” es la que obtiene un numero de pasos más bajo. Se puede deducir que esta función será la que encuentre una solución óptima en pasos en comparación a sus competidoras para los diferentes niveles estudiados, mientras que el resto de funciones devuelven una solución que puede ser igual o menos buena como ocurre en el caso de la función “Elegida” que devuelve una solución igual a la “Admisible” en algunas ocasiones, mientras que la “Voraz” devuelve una solución en un numero de pasos mucho mayor, ya que prioriza por tiempo.

Una vez se conocen los datos de manera total a continuación se muestran los datos diferenciando por la categoría del número de cajas a la cual pertenecen.

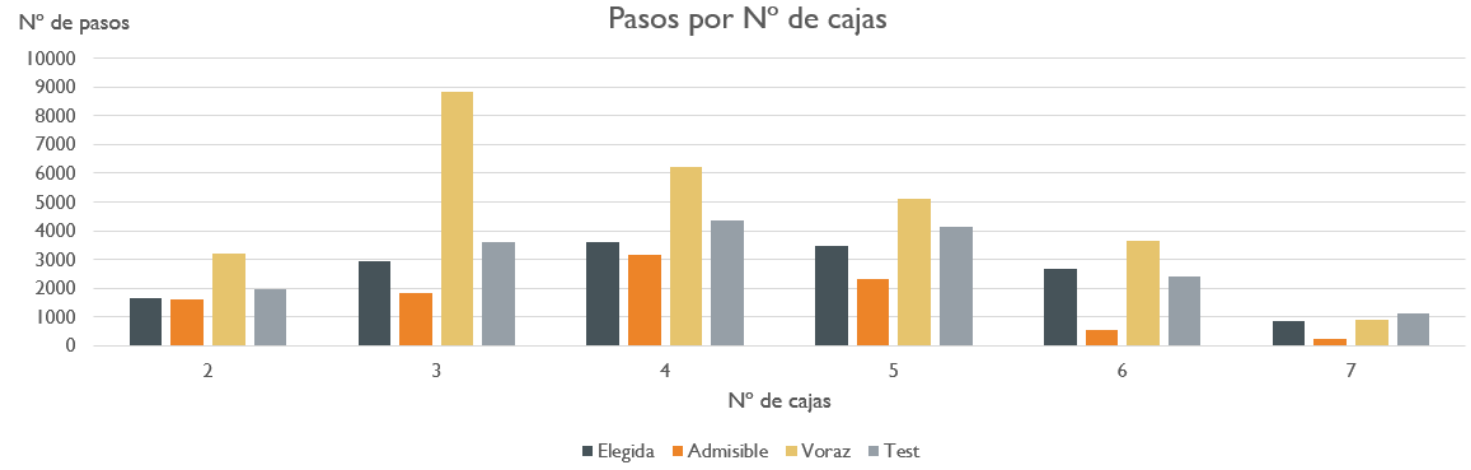


Ilustración 39: Pasos por categoría

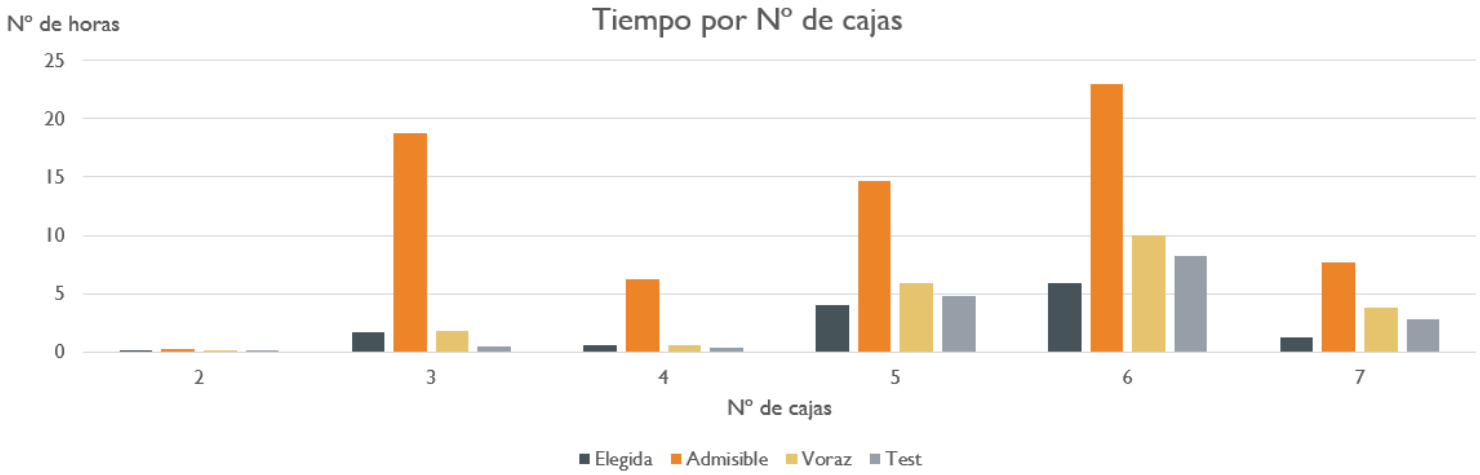


Ilustración 40: Tiempo por categoría

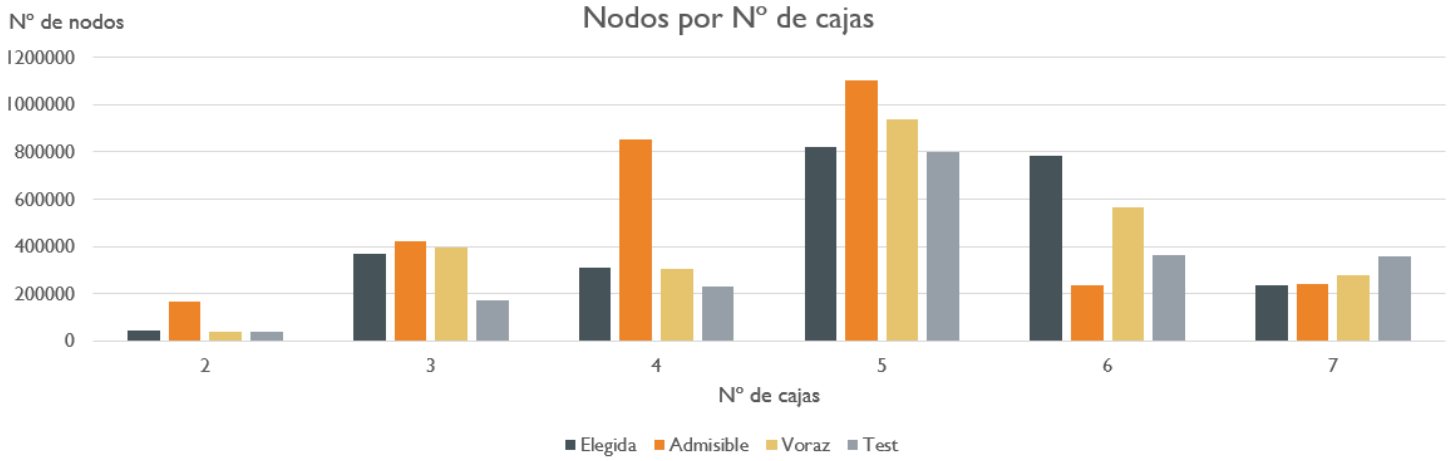


Ilustración 41: Nodos por categoría

Observando las gráficas que muestran del número de nodos por niveles se puede ver que la función "Voraz" superará de forma inmediata el número de nodos de la función "Elegida" en cuanto encuentre solución a los cuatro niveles que le faltan, mientras que observando los niveles que le faltan por resolver a la función "Test" su número de nodos una vez completados estos niveles será muy semejante al número de nodos de la función "Elegida".

Una vez que se han mostrado todos los datos de cada parámetro para permitir el estudio y la comparativa entre las diferentes formas a continuación se incluye una tabla en la que se podrán analizar los ratios nº pasos/tiempo y nodos/tiempo para conocer el rendimiento que se ha obtenido de cada una de las funciones de evaluación, para su cálculo se ha empleado el tiempo en horas:

Funcion de evaluación	Nº pasos/tiempo	Nodos/tiempo
Elegida	1128,78	190176,59
Admisible	136,78	42782,18
Voraz	1257,43	113899,78

Test	1055,537	117872,65

Tabla 1: Ratios por función de evaluación

Después de estudiar los valores de todos los parámetros que permiten comparar las diferentes funciones de evaluación que se han utilizado para el algoritmo A* se puede concluir que la mejor de ellas en términos generales es la función de evaluación “Elegida”.

Esta elección de la función de evaluación “Elegida” se debe a que es la única que consigue completar la suite de niveles en un tiempo inferior a dos horas por nivel, siendo el tiempo de ejecución un parámetro clave en la elección de la función, decisión que es reafirmada debido a que si se observan las gráficas del resto de parámetros la función “Elegida” ofrece el segundo menor número de pasos y esta igualada en el número de nodos expandidos.

Si en nuestra elección primara el número de pasos la mejor de las funciones de evaluación sería la función “Admisible” que se ha demostrado que encuentra siempre la solución óptima a pesar de que necesita una cantidad de tiempo muy superior.

Para concluir con la elección de la función concluir que la función de evaluación “Elegida” ofrece la mejor relación nº de pasos/tiempo y nodos expandidos/tiempo a excepción de con la función “Voraz” debido a que esta no ha completado todos sus niveles y esto altera un poco el valor. A pesar de eso se ha observado que la función “Voraz” encuentra soluciones que tienen un mayor número de pasos que las soluciones encontradas por la función “Elegida” y esto hace que se descarte.

5.5. Tiempo de ejecución por algoritmo

En este punto se mostrarán los resultados de los tiempos de ejecución que se han obtenido para cada uno de los niveles utilizando el algoritmo A y el algoritmo IDA* * con la función de evaluación “Admisible”.

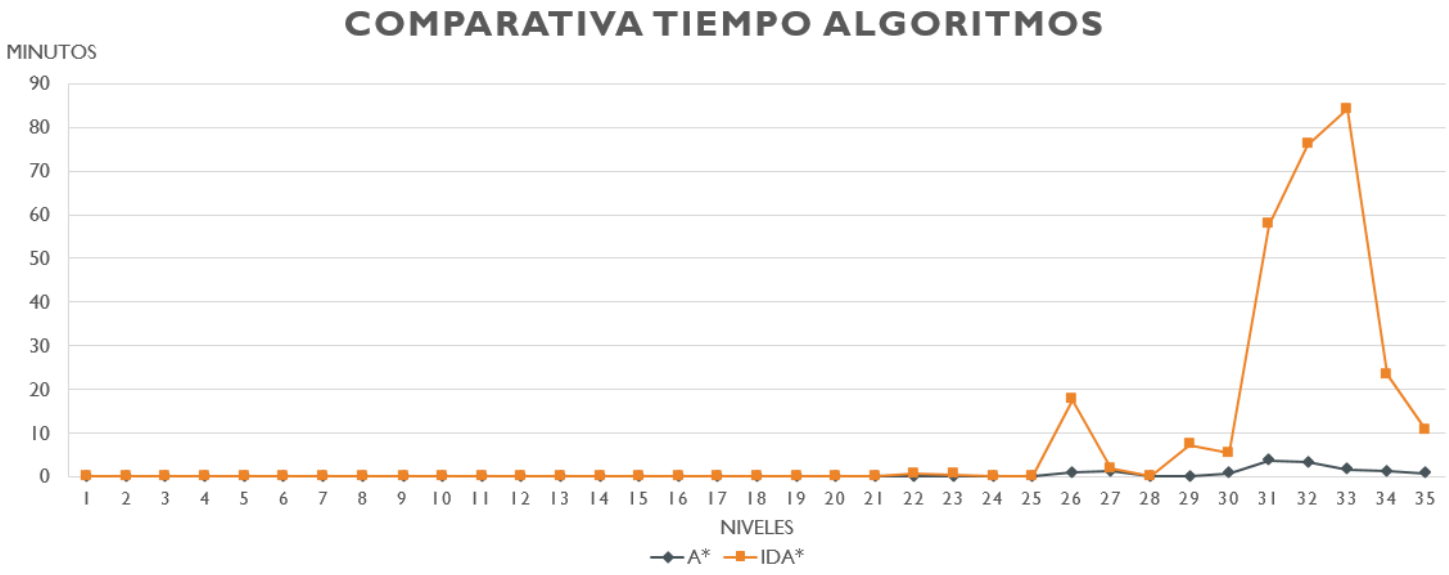
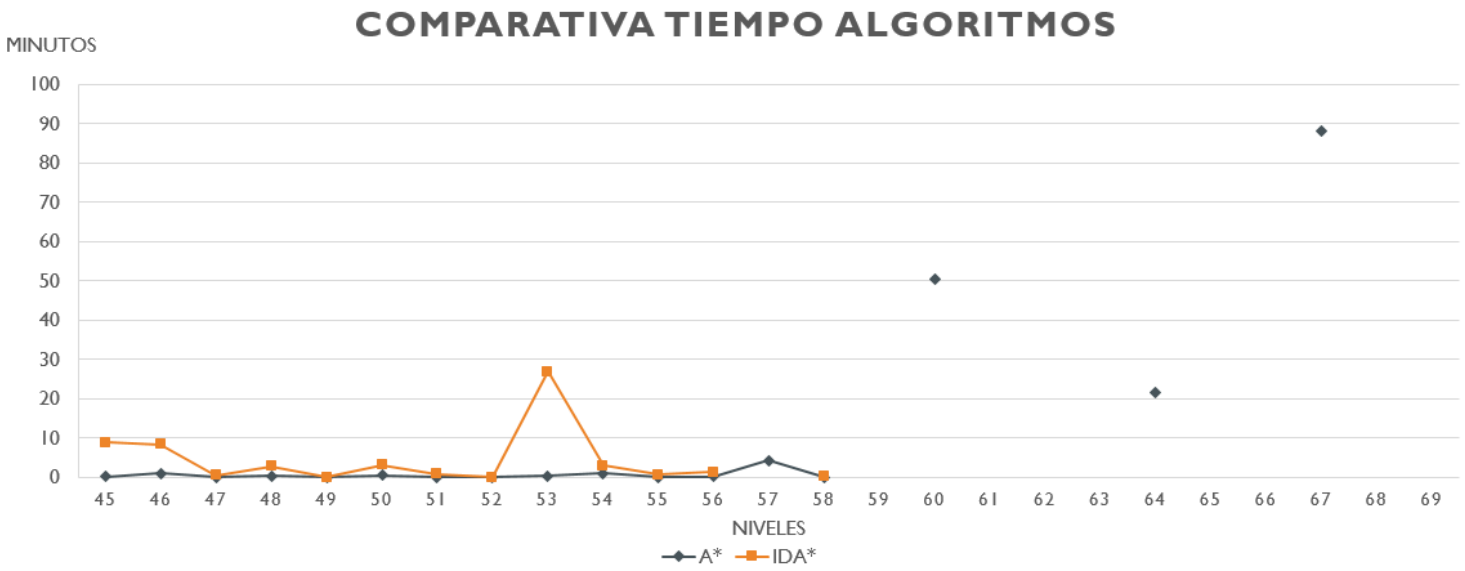


Ilustración 42: Comparativa tiempo algoritmos niveles de 2 cajas

La gráfica anterior refleja el tiempo invertido por cada uno de los algoritmos para resolver los niveles de dos cajas. El tiempo invertido en la solución de los niveles se refleja en minutos



para una mejor comparación.

Ilustración 43: Comparativa tiempo algoritmos niveles de 3 cajas

La gráfica anterior refleja el tiempo invertido por cada uno de los algoritmos para resolver los niveles de tres cajas. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

Se puede observar en la gráfica anterior como para ambos algoritmos en algunos niveles no aparece reflejado el tiempo de ejecución, esto se debe a que se ha fijado un tiempo máximo de ejecución de dos horas para cada nivel y si en ese tiempo no se encontraba una solución se saltaba al siguiente nivel. De esta manera se pretende obtener un buen rendimiento en el algoritmo elegido. Este mismo comportamiento se verá reflejado en las gráficas siguientes de mayor número de cajas por nivel.

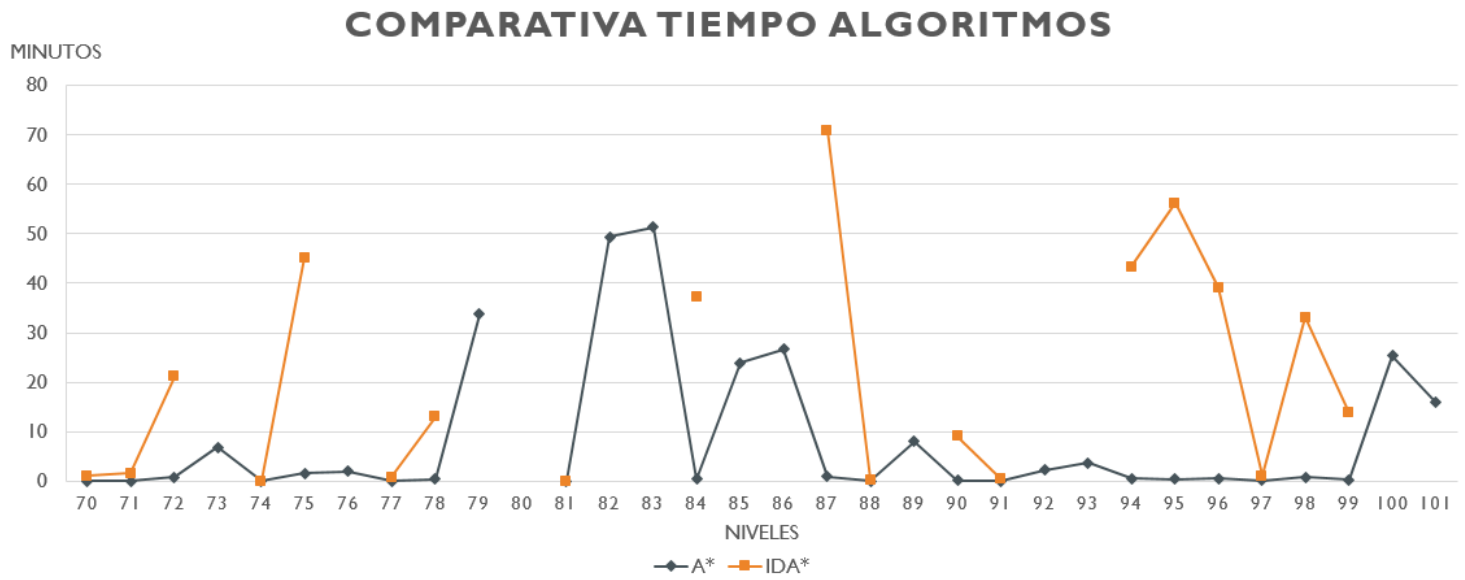


Ilustración 44: Comparativa tiempo algoritmos niveles de 4 cajas

La gráfica anterior refleja el tiempo invertido por cada uno de los algoritmos para resolver los niveles de cuatro. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

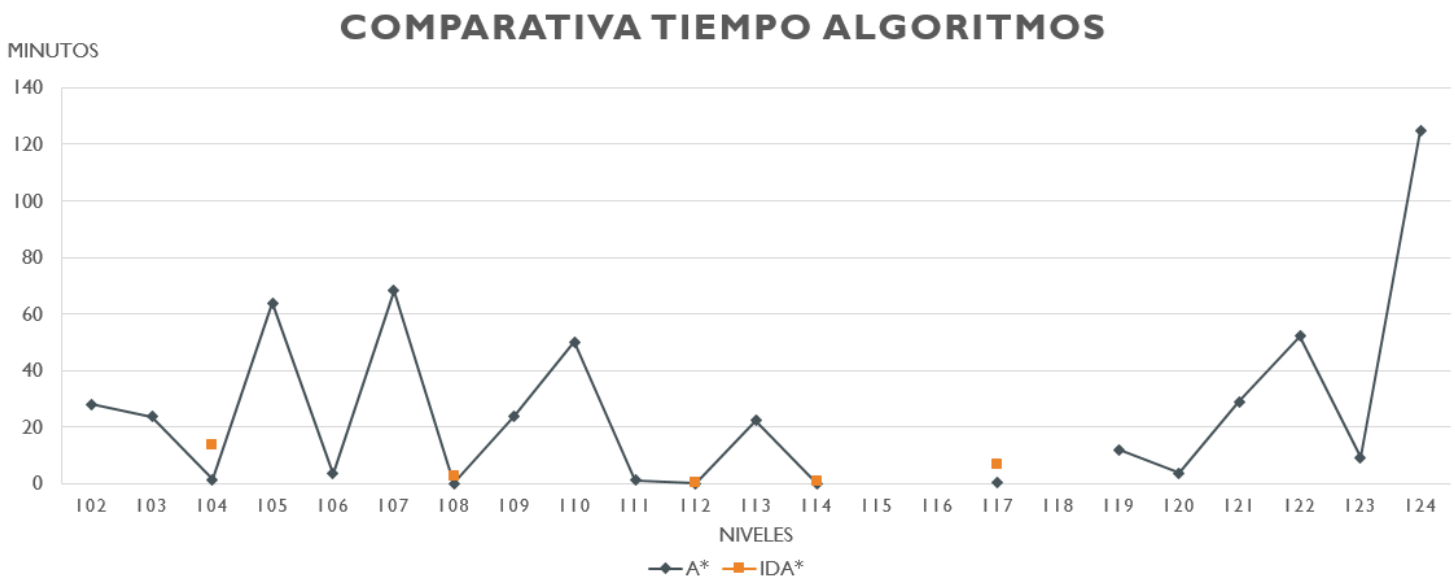


Ilustración 45: Comparativa tiempo algoritmos niveles de 5 cajas

La gráfica anterior refleja el tiempo invertido por cada uno de los algoritmos para resolver los niveles de cinco. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

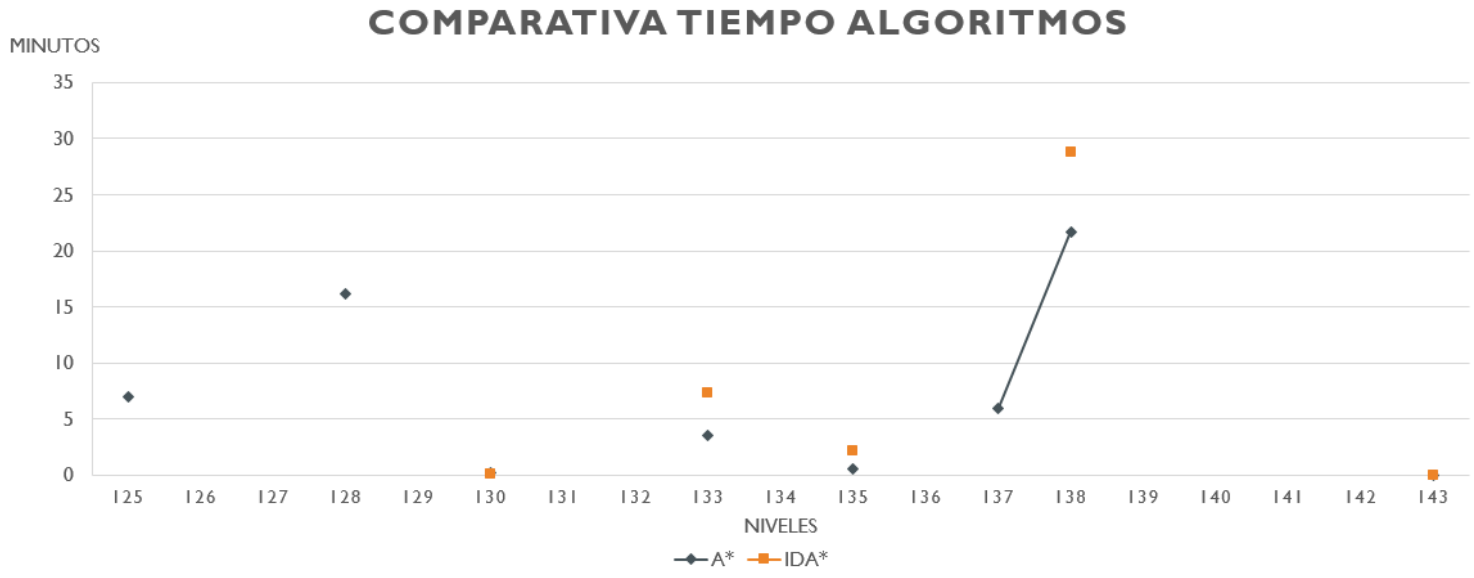


Ilustración 46: Comparativa tiempo algoritmos niveles de 6 cajas

La gráfica anterior refleja el tiempo invertido por cada uno de los algoritmos para resolver los niveles de seis. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

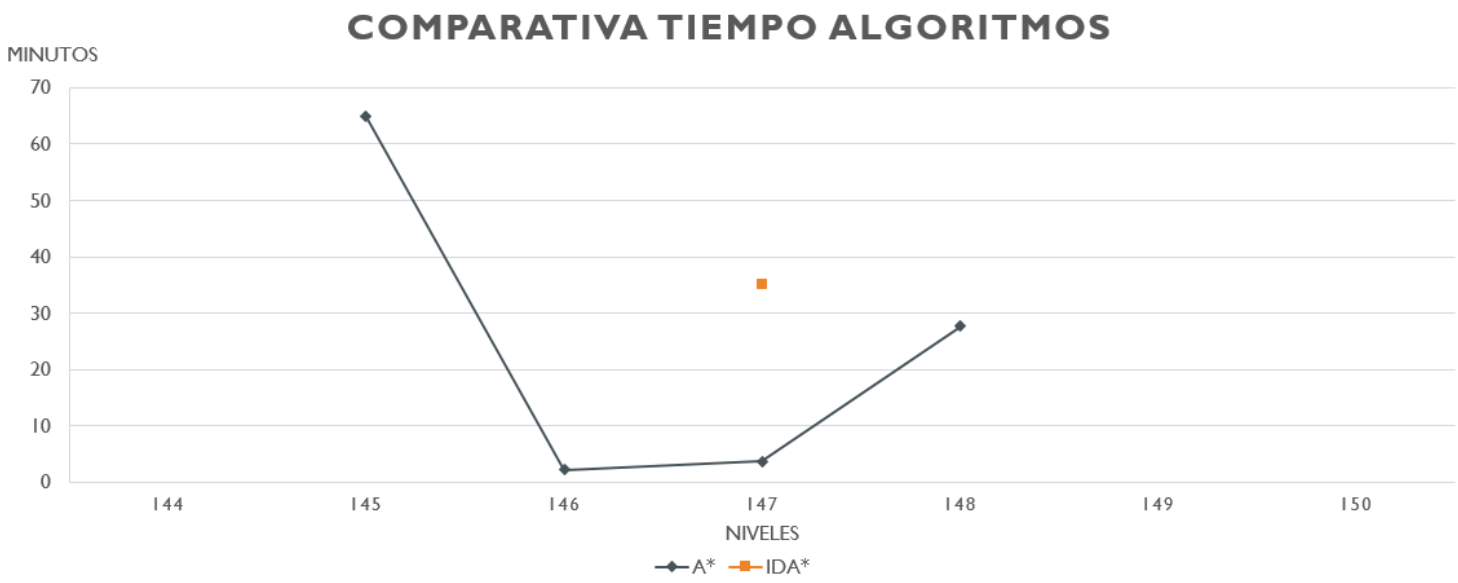


Ilustración 47: Comparativa tiempo algoritmos niveles de 7 cajas

La gráfica anterior refleja el tiempo invertido por cada uno de los algoritmos para resolver los niveles de siete cajas. El tiempo invertido en la solución de los niveles se refleja en minutos para una mejor comparación.

Tras la realización de un análisis de los resultados mostrados en las gráficas anteriores y que se pueden observar en forma de tabla en el *ANEXO II: Tablas de datos*, se puede observar como en los niveles que se completan el algoritmo A* es mejor con su función "Admisible" que el IDA* con esta misma función de evaluación.

En el apartado *5.7. Conclusiones de algoritmos* se puede observar la comparativa general y por categorías de los diferentes algoritmos para la suite de 150 niveles.

5.6. Nodos por algoritmo

En este punto se mostrarán los resultados de los nodos expandidos que se han obtenido para cada uno de los niveles utilizando el algoritmo A* y el algoritmo IDA* con la función de evaluación "Admisible".

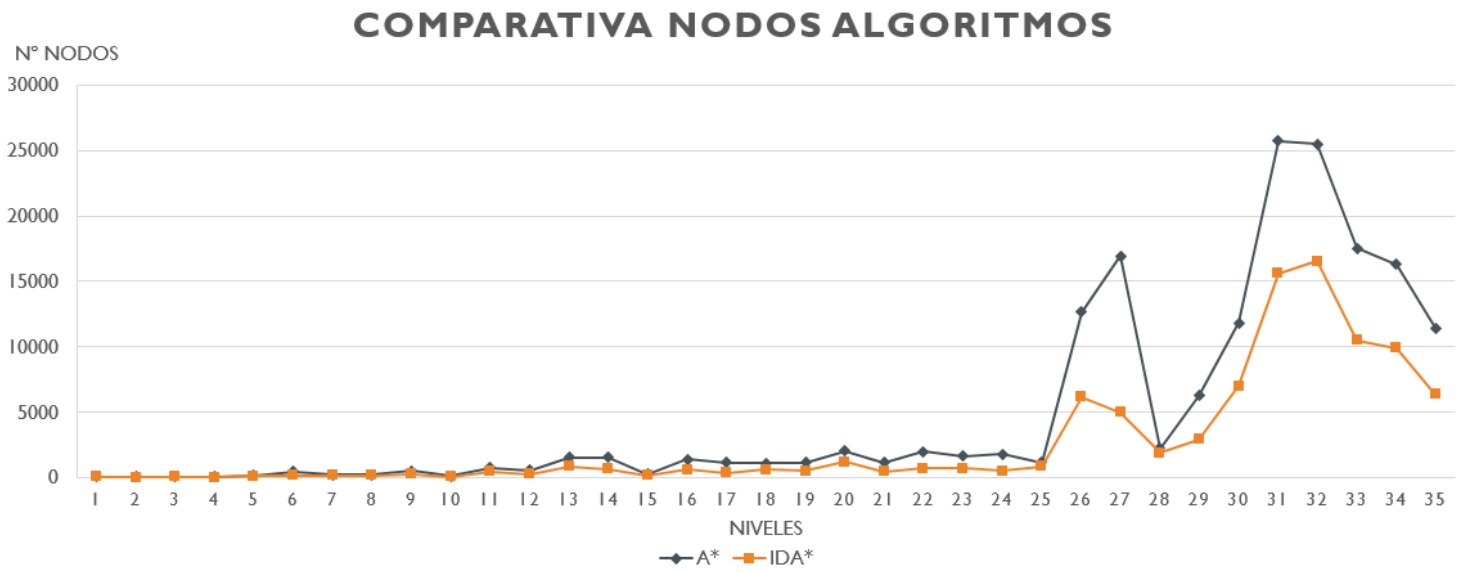


Ilustración 48: Comparativa nodos algoritmos niveles de 2 cajas

La gráfica anterior refleja número de nodos expandidos por cada uno de los algoritmos para resolver los niveles de dos cajas.

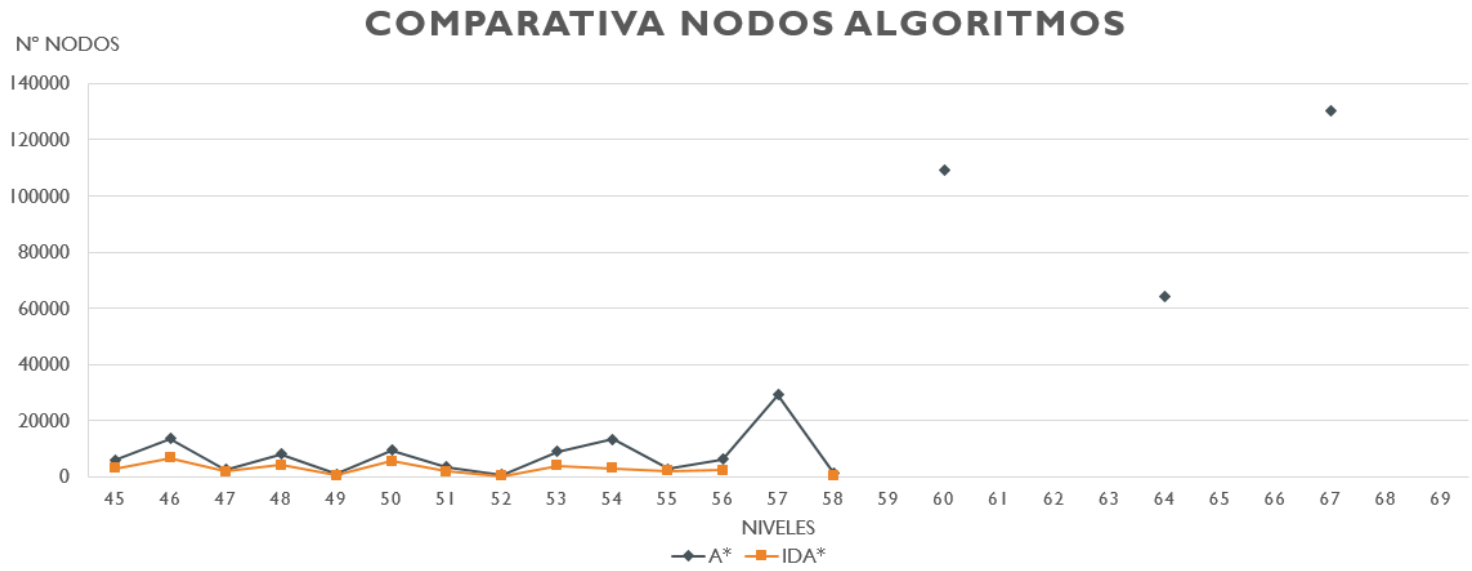


Ilustración 49: Comparativa nodos algoritmos niveles de 3 cajas

La gráfica anterior refleja número de nodos expandidos por cada uno de los algoritmos para resolver los niveles de tres cajas.

Se puede observar en la gráfica anterior que para ambos algoritmos en algunos niveles no aparece ningún número de nodos expandidos. Esto es debido a que como se ha comentado anteriormente se ha puesto un límite máximo de tiempo de dos horas de ejecución, y todos aquellos niveles que no han obtenido una solución en esas dos horas de tiempo no tendrán un numero de nodos expandidos.

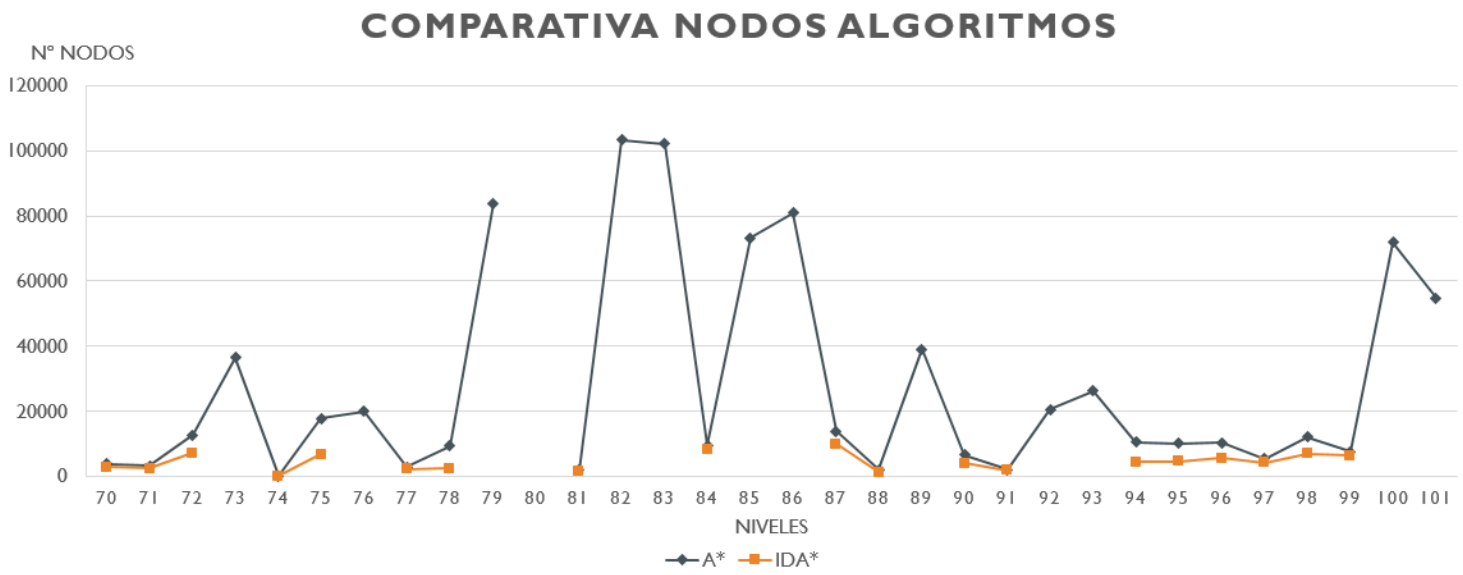


Ilustración 50: Comparativa nodos algoritmos niveles de 4 cajas

La gráfica anterior refleja número de nodos expandidos por cada uno de los algoritmos para resolver los niveles de cuatro cajas.

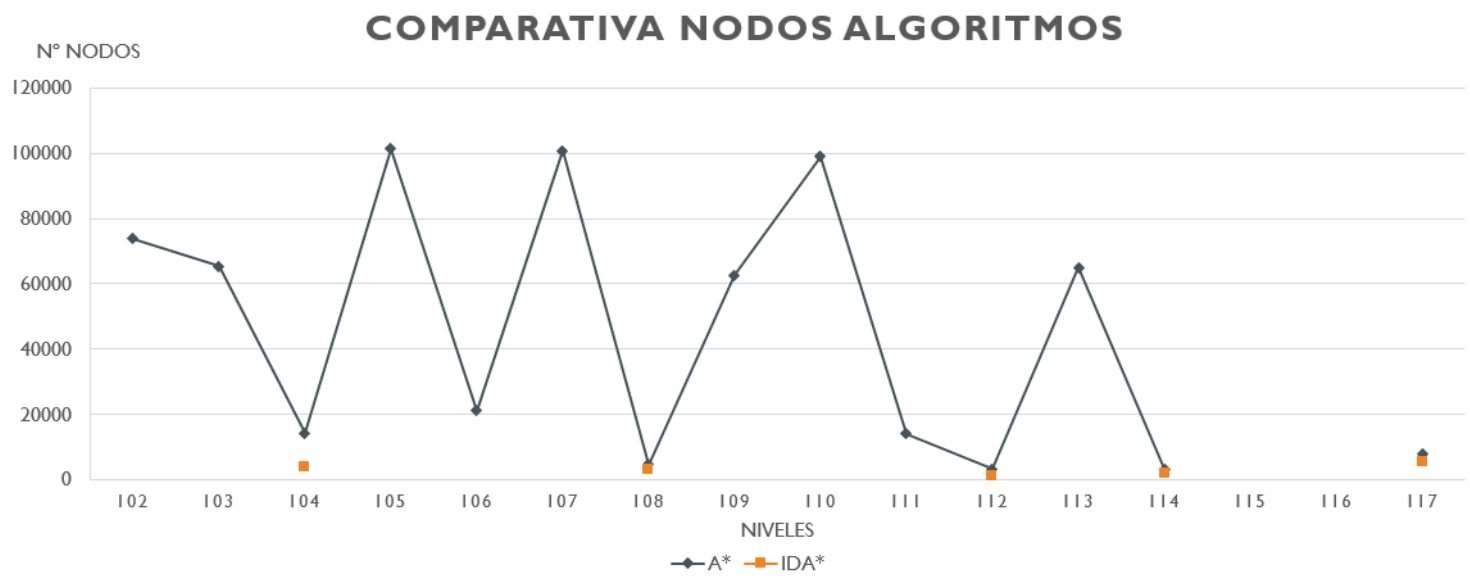


Ilustración 51: Comparativa nodos algoritmos niveles de 5 cajas

La gráfica anterior refleja número de nodos expandidos por cada uno de los algoritmos para resolver los niveles de cinco cajas.

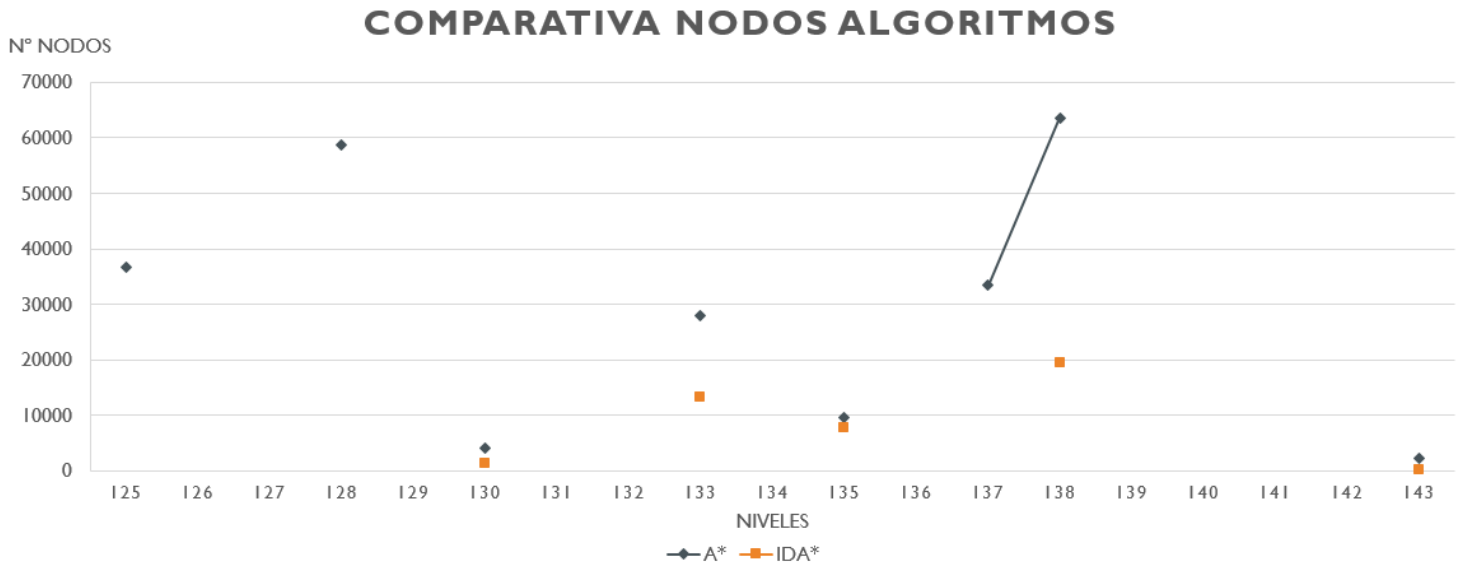


Ilustración 52: Comparativa nodos algoritmos niveles de 6 cajas

La gráfica anterior refleja número de nodos expandidos por cada uno de los algoritmos para resolver los niveles de seis cajas.

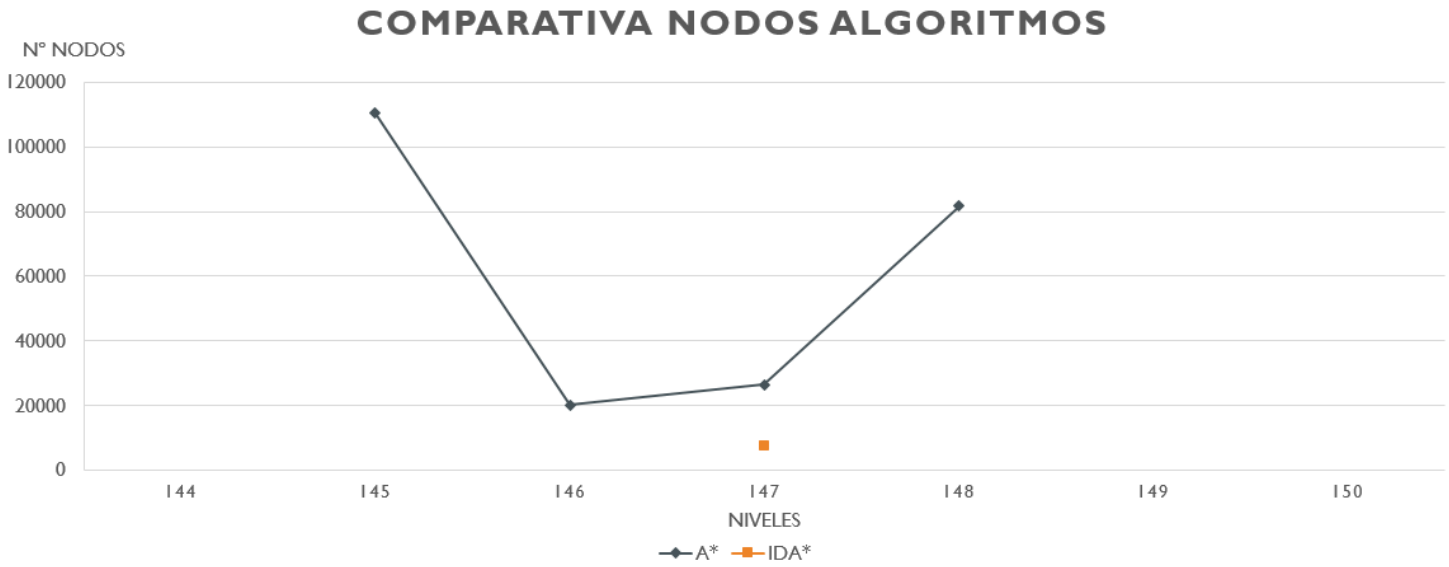


Ilustración 53: Comparativa nodos algoritmos niveles de 7 cajas

La gráfica anterior refleja número de nodos expandidos por cada uno de los algoritmos para resolver los niveles de siete cajas.

Tras la realización de un análisis de los resultados mostrados en las gráficas anteriores y que se pueden observar en forma de tabla en el *ANEXO II: Tablas de datos*, se puede observar como el algoritmo que tiene un menos número de nodos expandidos para todos los niveles que ha encontrado solución es el IDA*.

En el apartado *5.7. Conclusiones de algoritmos* se puede observar la comparativa general de los diferentes algoritmos para la suite de 150 niveles.

5.7. Conclusiones de algoritmos

Después de realizar un análisis de los resultados obtenidos para los diferentes algoritmos de manera desglosada por niveles en este punto se incluye la evaluación global y por categoría de los diferentes algoritmos que permita elegir uno de ellos como mejor algoritmo de búsqueda para este problema.

Para evaluar cuál es el mejor algoritmo, en primer lugar, se compara el número de niveles completados por cada uno de ellos:

Ilustración 54: Niveles completados por algoritmo

Se puede observar en la gráfica anterior como ningún algoritmo ha conseguido completar la suite de 150 niveles en un tiempo inferior a dos horas, pero el algoritmo que más resuelto ha sido el A* con un total de 124 mientras que el IDA* se ha quedado en 87.

Después de conocer el número de niveles resueltos por cada función es necesario conocer el tiempo total que han invertido para resolverlos.

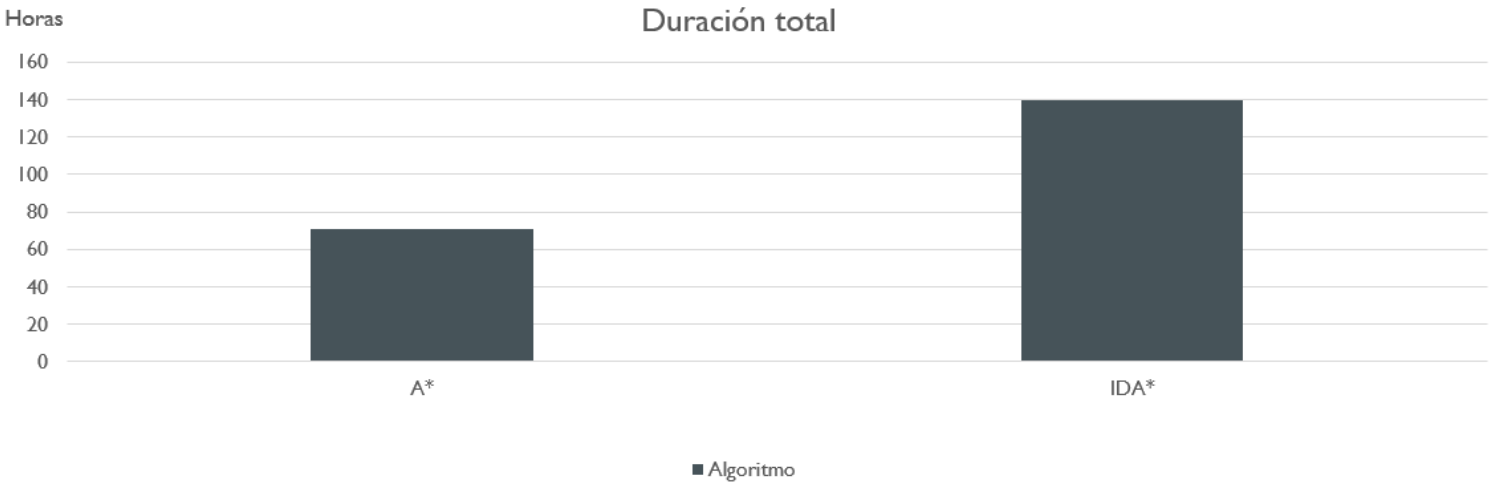


Ilustración 55: Tiempo total por algoritmo

La gráfica anterior refleja el tiempo que ha estado en ejecución cada uno de los algoritmos para intentar encontrar la solución de todos los niveles realizando el cambio de nivel a las dos horas de ejecución en caso de no haber encontrado la solución en ese tiempo.

Se puede observar que el algoritmo con un menor tiempo de evaluación es A* cuyo tiempo total es de 70 horas mientras el IDA* tiene un tiempo de 140 horas.

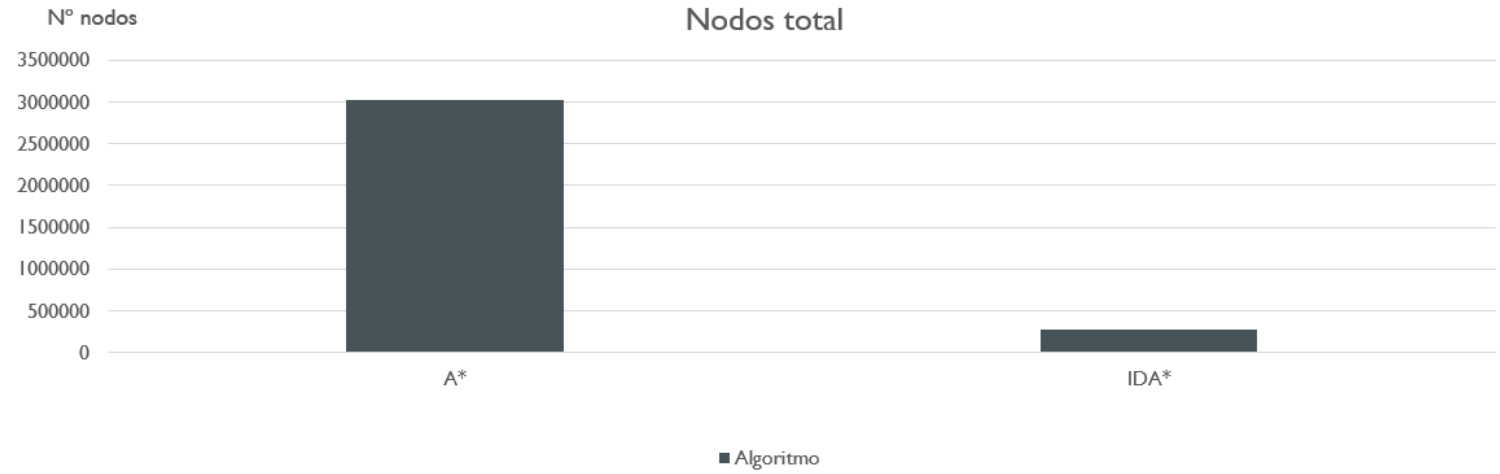


Ilustración 56: Nodos totales por algoritmo

La gráfica anterior refleja el número de nodos totales que ha expandido cada algoritmo para los niveles que ha encontrado solución. A pesar de que el algoritmo A* ha resuelto un mayor número de niveles esto no influye en que su número sea más elevado, debido a que para todos los niveles que ambos algoritmos han encontrado solución el A* ha necesitado expandir un mayor número de nodos.

Una vez se conocen los datos de manera total a continuación se muestran los datos diferenciando por la categoría del número de cajas a la cual pertenecen.

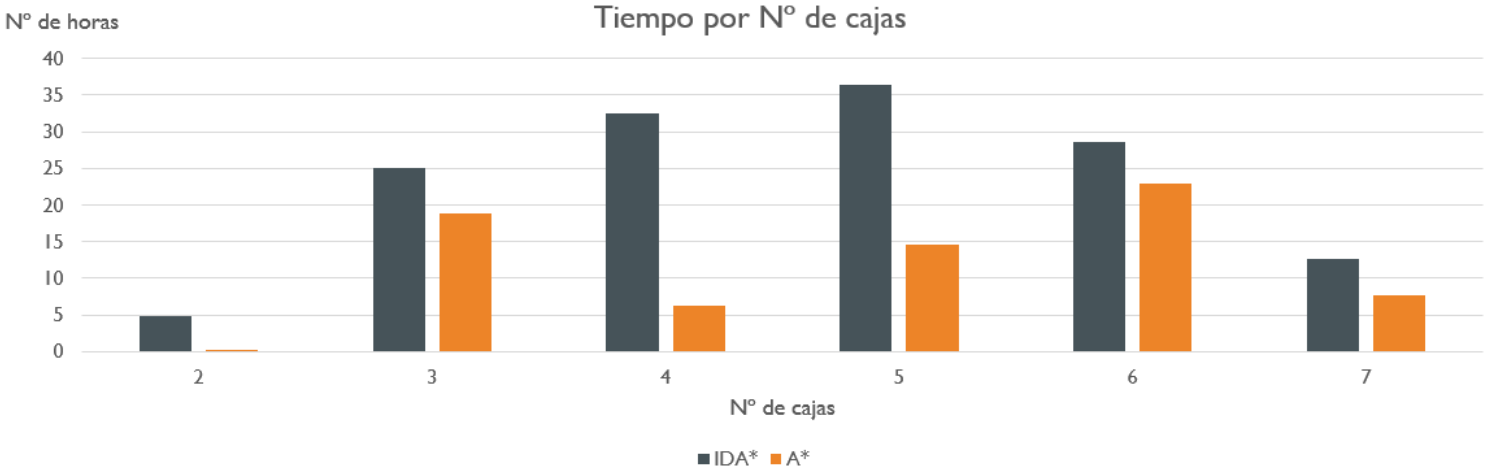


Ilustración 57: Tiempo por categoría para algoritmos

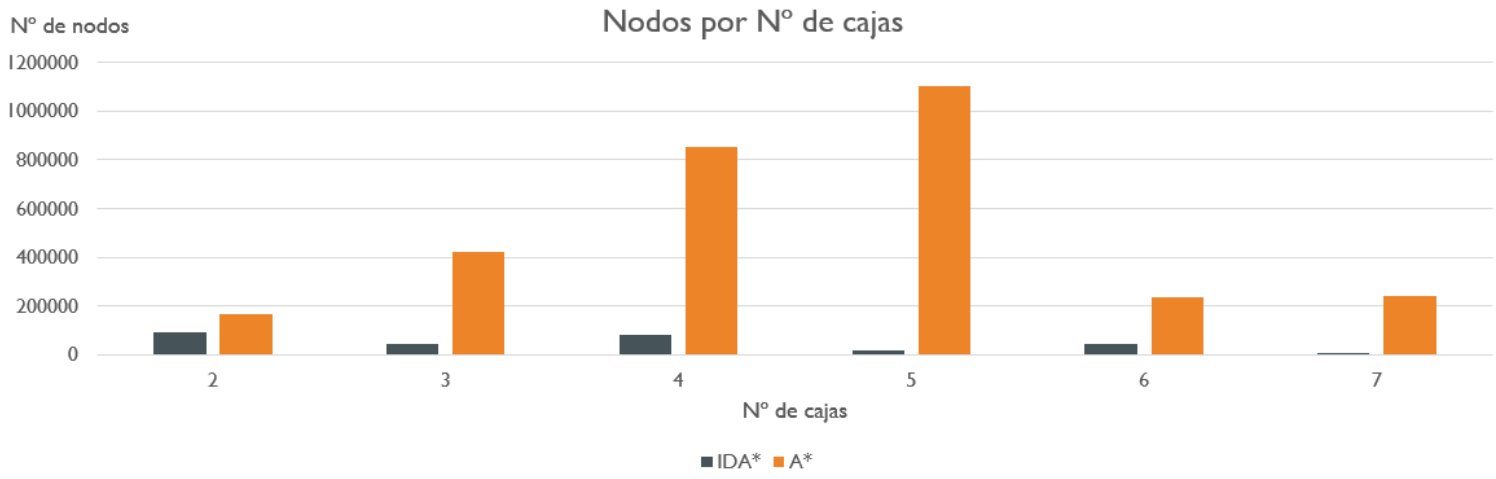


Ilustración 58: Nodos por categoría para algoritmos

Una vez mostrados los datos de comparativa en los gráficos, los cuales demuestran que con el IDA* se consigue un ahorro de memoria sacrificando el tiempo, se muestra una tabla que recoge los ratios de manera similar a lo ocurrido para la comparativa de funciones de evaluación:

Algoritmo	Nºpasos/tiempo	Nodos/tiempo
A*	136,78	42782,18
IDA*	76,09	2005,48

Tabla 2: Ratios por algoritmo

Para la resolución de este problema ofrece mejor rendimiento el algoritmo A* en relación nº de pasos /tiempo y nodos expandidos/tiempo, a pesar de ello cuando nos enfrentáramos a este problema en máquinas que pudieran tener un problema de memoria debido a sus bajas prestaciones, situación que no se ha dado en las máquinas utilizadas para las pruebas, sería más recomendable utilizar el algoritmo IDA* debido a que su consumo de memoria es inferior como se ha podido observar en la cantidad de nodos expandidos y para niveles de pocas cajas el tiempo no se disparaba en exceso.

6. Conclusiones

A nivel global se ha cumplido con el diseño inicial propuesto en la práctica corta 1 salvo el módulo de aprendizaje automático por falta de tiempo y datos.

Por otro lado, se ha conseguido hacer la aplicación del Sokoban con tres niveles de juego para así hacerla más atractivo. Estos modos de juego son la resolución manual, completa o parcial, esta última para jugadores que no quieran que el solver les complete todo el nivel, pero estén atascados en ciertos puntos del escenario y necesiten avanzar.

Finalmente se ha justificado la elección del algoritmo de búsqueda heurística, así como su función de evaluación, en base a los resultados en cuanto a tiempo, nodos y pasos mostrados y analizados a lo largo de esta memoria.

7. Trabajos futuros

En este punto se incluyen los trabajos futuros que se pueden realizar sobre el proyecto con el fin de completar algunos aspectos o incluir ciertas mejoras.

En primer lugar, orientado a los usuarios de nuestra versión del juego Sokoban, el primer trabajo futuro a realizar es la inclusión de nuevos niveles que atraigan a mas jugadores ya que en la actualidad el juego cuenta únicamente con 150 niveles. Estos nuevos niveles pendientes de inclusión en la versión del juego deben aumentar la dificultad de los niveles ya incluidos para suponer nuevos desafíos a los jugadores. Por ello se pretenden incluir niveles de 7 o más que supongan retos difíciles para los jugadores.

En segundo lugar, en base a la IA del juego se incluyen dos posibles trabajos futuros que permitan mejorar el solver actual del Sokoban.

- Estudio de algoritmos alternativos para resolver el problema debido a que como se ha comentado a lo largo de este documento el problema del Sokoban se trata de un problema NP-hard y PSPACE-Completo en términos de complejidad algorítmica y es necesario el estudio de más algoritmo existente o posibles nuevos algoritmos que puedan mejorar el rendimiento de los actuales.
- Realizar el módulo de aprendizaje automático que fue incluido en el diseño de la arquitectura del sistema y no ha sido posible realizar debido a la falta de tiempo. En la actualidad se guarda toda la información de cada una de las jugadas con el fin de disponer de una base de conocimiento sobre la que posteriormente poder realizar el aprendizaje automático. El uso de aprendizaje automático puede estar orientado a los siguientes aspectos:
    - Mejorar la heurística h(x) para de esta manera mejorar la función de evaluación y permitir a los algoritmos existentes obtener un mejor rendimiento en cuanto a tiempo al estar la heurística más informada.
    - Elegir entre diferentes funciones de evaluación en base a las características del nivel, ya que se ha podido observar que en base a las características del nivel como pueden ser ancho, alto, número de casillas colocadas o paredes en el interior del mapa puede funcionar mejor una función de evaluación u otra. Por lo tanto, en base al conocimiento almacenado sobre las jugadas, los resultados expuestos, y cuando se disponga de un mayor número de niveles se podrían obtener patrones para elegir entre diferentes funciones de evaluación en base a las características del nivel.
    - Uso de clasificadores en lugar de búsqueda heurística para la resolución de los niveles y comparar los resultados obtenidos de cada método.

8. Referencias

A continuación, se listan las referencias usadas para la realización de este trabajo sobre un solver del juego Sokoban:

[1]. Wikipedia. *Sokoban* [online]. Abril, 2016. Disponible en: https://en.wikipedia.org/wiki/Sokoban

[2]. J. García Alvarado. *"MSAMA Sokoban Solver."* [online]. Enero, 2011. Disponible en: http://sokobano.de/wiki/index.php?title=MSAMA_Sokoban_Solver#4.3_La_heur.C3.ADstica_elegida

[3]. J. Taylor & I. Parberry. *"Procedural Generation of Sokoban Levels."* [online]. 2011. Disponible en: http://larc.unt.edu/ian/research/sokoban/

[4]. D. Skinner. *Sokoban*. [online]. 2010. Disponible en: http://web.archive.org/web/20120918183734/http://users.bentonrea.com/~sasquatch/sokoban/

[5]. Athena: MIT's academic computing environment. *XSokoban*. [online]. 1994. Disponible en: https://stuff.mit.edu/afs/athena/contrib/games/lib/xsokoban/screens/

[6]. S. Lindhurst. *Sokoban for the Macintosh*. [online]. Julio, 2002. Disponible en: http://www.sneezingtiger.com/sokoban/levels.html

[7]. Y. Chao. *Sokoban.org*. [online]. Julio, 2013. Disponible en: http://sokoban.org/list.php

[8]. A. Martín. *My Sokoban collections.* [online]. 2013. Disponible en: http://www.abelmartin.com/rj/sokoban_colecciones.html

[9]. *@1001com*. *Sokoban pack*. [online]. Septiembre, 2014. Disponible en: http://opengameart.org/content/sokoban-pack

[10]. M. Meger. *Sokoban: Level format.* [online]. Enero, 2011. Disponible en: http://sokobano.de/wiki/index.php?title=Level_format

[11]. M. González. *"Sokoban: sencillo de aprender, difícil de dominar."* [online]. Junio, 2006. Disponible en: http://www.pdatungsteno.com/2006/06/29/sokoban-sencillo-de-aprender-dificil-de-dominar/

[12]. Wikipedia. *NP-Hard.* [online]. Abril, 2016. Disponible en: https://en.wikipedia.org/wiki/NP-hardness

[13]. Wikipedia. *NP (complexity).* [online]. Abril, 2016. Disponible en: https://en.wikipedia.org/wiki/NP_(complexity)

[14]. Wikipedia. *PSPACE-complete.* [online]. Febrero, 2016. Disponible en: https://en.wikipedia.org/wiki/PSPACE-complete

ANEXO I: Guía de instalación

En este anexo se explica la instalación del proyecto en los sistemas operativos Windows 10 y Ubuntu 14.04.

Instalación en Windows 10

Para poder ejecutar el JAR del juego proporcionado debe tener instalado Java en su versión 7 o superior. Si no dispone de ello puede descargarlo de https://www.java.com/es/download/.

Ilustración 59 Ejecutable del juego

Por otro lado, se necesita tener instalado MongoDB en su versión 3.0 o superior. Si no dispone de ello descárguelo y siga la guía del link proporcionado a continuación para configurarlo https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/. Las colecciones necesarias se generarán automáticamente la primera vez que se ejecute el juego, si se desea se pueden importar las proporcionadas ya con resultados de niveles, aunque no es necesario.

Si se desea manipular las colecciones para elegir el nivel en el que se quiere empezar con un jugador o borrar las soluciones ya calculadas para cierto nivel recomendamos usar la herramienta de trabajo MongoChef si no se dispone de los conocimientos necesarios para realizar estas tareas por un terminal. Puede descargarla de http://3t.io/mongochef/.

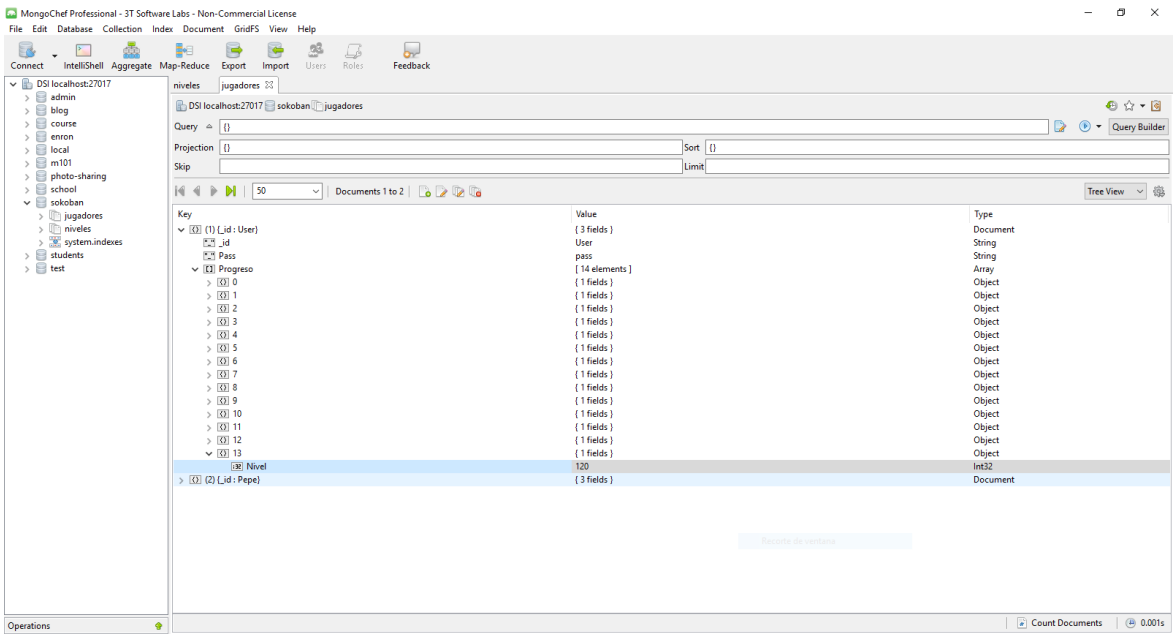


Ilustración 60 MongoChef en Windows 10

Si se desea consultar el código del proyecto proporcionado o ejecutar el juego sin usar el JAR puede utilizarse cualquier IDE que permita abrir archivos con extensión *.java*. Si se quiere realizar algún cambio compruebe antes los comentarios sobre ese bloque de código.

Instalación en Ubuntu 14.04

Para poder ejecutar el JAR del juego proporcionado debe tener instalado Java en su versión 7 o superior. Si no dispone de ello puede descargarlo y configurarlo siguiendo el siguiente link https://www.java.com/en/download/linux_manual.jsp.

Ilustración 59 Ejecutable del juego

Por otro lado, se necesita tener instalado MongoDB en su versión 3.0 o superior. Si no dispone de ello descárguelo y siga la guía del link proporcionado a continuación para configurarlo https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/. Las colecciones necesarias se generarán automáticamente la primera vez que se ejecute el juego, si se desea se pueden importar las proporcionadas con resultados de niveles, aunque no es necesario.

Si se desea manipular las colecciones para elegir el nivel en el que se quiere empezar con un jugador o borrar las soluciones ya calculadas para cierto nivel recomendamos usar la herramienta de trabajo MongoChef si no se dispone de los conocimientos necesarios para realizar estas tareas por un terminal. Puede descargarla e instalarla siguiendo los siguientes links que se proporcionan a continuación: http://3t.io/mongochef/download/platform/#tab-id-3 y http://3t.io/blog/install-mongochef-mongodb-linux/.

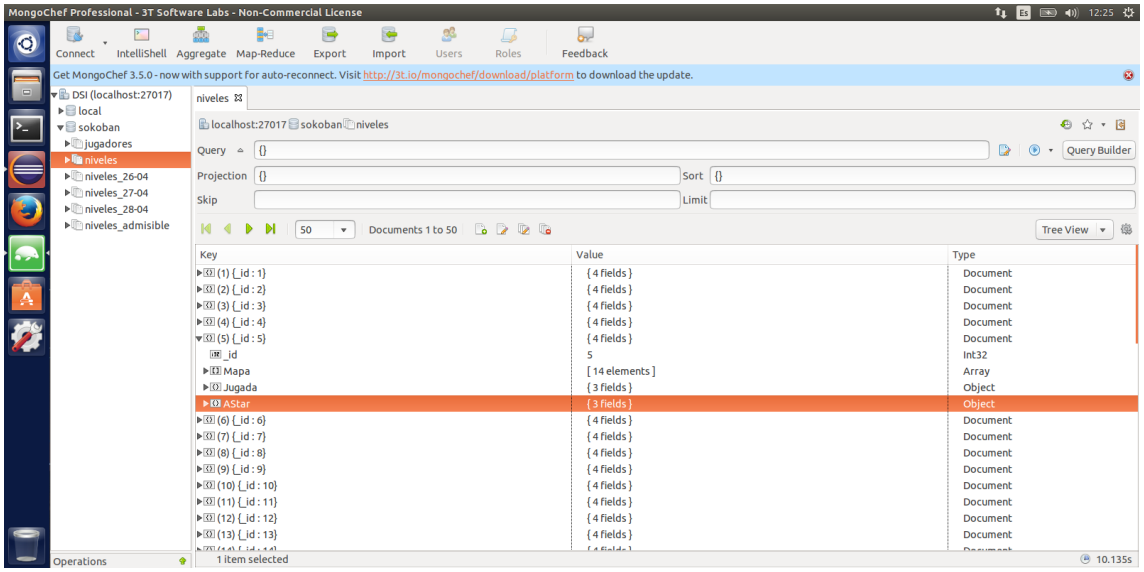


Ilustración 61 MongoChef en Ubuntu 14.04

Si se desea consultar el código del proyecto proporcionado o ejecutar el juego sin usar el JAR puede utilizarse cualquier IDE que permita abrir archivos con extensión *.java*. Si se quiere realizar algún cambio compruebe antes los comentarios sobre ese bloque de código.

ANEXO II: Tablas de datos

En este anexo se incluyen las tablas con los datos obtenidos para los diferentes algoritmos y funciones de evaluación. Los cuales han sido mostrados en gráficas en el apartado 5. Experimentos realizados.

Algoritmo A* con función de evaluación "Elegida"

ID	Steps	Time(ms)	Nodes	Time (min)
1	15	65	72	0.001083333
2	11	6	30	0.0001
3	10	37	25	0.000616667
4	10	1	16	1.67E-05
5	29	69	159	0.00115
6	41	104	219	0.001733333
7	32	35	203	0.000583333
8	30	62	188	0.001033333
9	39	23	140	0.000383333
10	20	2	49	3.33E-05
11	36	380	510	0.006333333
12	34	195	447	0.00325
13	44	59	289	0.000983333
14	39	61	242	0.001016667
15	45	31	200	0.000516667
16	43	131	322	0.002183333
17	41	334	665	0.005566667
18	42	100	381	0.001666667
19	43	353	845	0.005883333
20	43	88	304	0.001466667
21	45	340	485	0.005666667
22	66	3537	1861	0.05895
23	51	1275	1585	0.02125
24	46	293	563	0.004883333
25	61	402	859	0.0067
26	69	1996	2306	0.033266667
27	71	10572	4605	0.1762
28	60	1219	2050	0.020316667
29	66	11824	5798	0.197066667
30	58	943	1748	0.015716667
31	76	5619	3634	0.09365
32	99	4499	3541	0.074983333
33	97	9814	5554	0.163566667
34	82	1988	2720	0.033133333
35	77	657	1500	0.01095
36	19	1	36	1.67E-05
37	30	4	70	6.67E-05
38	36	69	328	0.00115
39	43	11	197	0.000183333
40	55	128	505	0.002133333
41	51	667	1172	0.011116667
42	48	32	383	0.000533333
43	47	40	291	0.000666667
44	62	773	1492	0.012883333
45	81	11191	5305	0.186516667
46	92	54036	11704	0.9006
47	77	603	1239	0.01005
48	72	2865	3185	0.04775
49	64	120	668	0.002
50	59	353	790	0.005883333
51	67	1384	2326	0.023066667
52	48	89	558	0.001483333
53	116	7029	4828	0.11715
54	58	1148	1967	0.019133333
55	76	977	2182	0.016283333
56	66	549	1520	0.00915
57	228	282393	27829	4.70655
58	55	110	553	0.001833333
59	130	1423210	62904	23.72016667
60	116	38244	11909	0.6374
61	91	11936	5863	0.198933333
62	141	2260838	78109	37.68063333
63	169	476048	37775	7.934133333
64	134	1401175	61734	23.35291667
65	149	68857	13062	1.147616667
66	111	20897	8805	0.348283333
67	113	426	1398	0.0071
68	115	15875	6804	0.264583333
69	125	65371	13184	1.089516667
70	64	2283	3003	0.03805
71	76	1606	2534	0.026766667
72	93	853	2095	0.014216667
73	148	458492	34481	7.641533333

74	17	1	29	1.67E-05
75	102	1526	2333	0.025433333
76	133	142871	19538	2.381183333
77	80	862	2108	0.014366667
78	98	3415	3565	0.056916667
79	160	59461	12276	0.991016667
80	117	82521	15680	1.37535
81	59	83	701	0.001383333
82	135	8669	5817	0.144483333
83	113	124647	15631	2.07745
84	188	24084	9035	0.4014
85	113	433445	34842	7.224083333
86	148	19804	8476	0.330066667
87	81	19124	7541	0.318733333
88	74	553	1781	0.009216667
89	129	66745	13455	1.112416667
90	107	8847	5265	0.14745
91	63	199	839	0.003316667
92	103	104127	15221	1.73545
93	113	1889	2620	0.031483333
94	115	10524	5589	0.1754
95	109	8461	5663	0.141016667
96	113	2768	3080	0.046133333
97	75	2880	3383	0.048
98	134	9139	5360	0.152316667
99	103	11803	6259	0.196716667
100	152	358928	32315	5.982133333
101	172	283180	28150	4.719666667
102	130	163745	21095	2.729083333
103	269	379418	36011	6.323633333
104	70	602	1609	0.010033333
105	143	652046	46290	10.86743333
106	104	29910	9619	0.4985
107	139	1018352	57021	16.97253333
108	73	3090	3653	0.0515
109	150	625841	43869	10.43068333
110	183	737805	49360	12.29675
111	142	34039	10635	0.567316667
112	50	320	963	0.005333333
113	196	842698	54134	14.04496667
114	82	1184	2258	0.019733333
115	228	3953416	106977	65.89026667
116	281	1217061	66767	20.28435
117	94	2827	3161	0.047116667
118	208	2492941	91602	41.54901667
119	149	258148	28623	4.302466667
120	201	150968	23467	2.516133333
121	170	957367	57751	15.95611667
122	121	372220	36528	6.203666667
123	169	376186	37097	6.269766667
124	127	337009	33397	5.616816667
125	117	358123	34431	5.968716667
126	246	49391	12688	0.823183333
127	208	2348674	89448	39.14456667
128	240	711620	50838	11.86033333
129	372	4109298	123886	68.4883
130	56	415	1327	0.006916667
131	197	191270	25008	3.187833333
132	85	303475	29314	5.057916667
133	58	213	868	0.00355
134	178	1703504	75315	28.39173333
135	33	92	442	0.001533333
136	90	136865	21963	2.281083333
137	95	53914	13258	0.898566667
138	68	1995	2241	0.03325
139	192	4310860	123913	71.84766667
140	98	97960	15795	1.632666667
141	185	6699881	148322	111.6646833
142	97	59770	14708	0.996166667
143	40	17	176	0.000283333
144	158	1390775	60670	23.17958333
145	58	17859	7715	0.29765
146	80	2264335	79460	37.73891667
147	88	215514	23746	3.5919
148	121	177426	24410	2.9571
149	217	138692	19939	2.311533333
150	145	158936	20387	2.648933333

Tabla 3: Datos A con función "Elegida"*

Tiempo total (ms)	Tiempo total (min)	Tiempo total (horas)	Tiempo ejecución (horas)	Nodos totales	Pasos totales	Niveles completados
48566420	809.44	13.49	13.49	2565610	15103	150

Tabla 4: Datos total A con función "Elegida"*

Algoritmo A* con función de evaluación “Admisible”

ID	Steps	Time (ms)	Nodes	Time (min)
1	15	143	80	0.00238333
2	11	21	74	0.00035000
3	10	12	58	0.00020000
4	10	9	40	0.00015000
5	29	60	183	0.00100000
6	37	178	452	0.00296667
7	32	36	212	0.00060000
8	30	28	230	0.00046667
9	36	111	489	0.00185000
10	18	15	103	0.00025000
11	35	202	775	0.00336667
12	32	255	559	0.00425000
13	44	642	1520	0.01070000
14	38	755	1559	0.01258333
15	45	17	252	0.00028333
16	39	605	1415	0.01008333
17	41	408	1133	0.00680000
18	40	382	1111	0.00636667
19	41	305	1142	0.00508333
20	41	1020	2054	0.01700000
21	45	417	1163	0.00695000
22	60	1955	1977	0.03258333
23	51	571	1657	0.00951667
24	45	652	1770	0.01086667
25	61	334	1140	0.00556667
26	67	54530	12674	0.90883333
27	63	86033	16954	1.43388333
28	55	1094	2231	0.01823333
29	66	10329	6263	0.17215000
30	58	46217	11815	0.77028333
31	76	225011	25715	3.75018333
32	99	206229	25472	3.43715000
33	97	88690	17515	1.47816667
34	78	85481	16301	1.42468333
35	73	41219	11401	0.68698333
36	19	5	77	0.00008330
37	30	41	382	0.00068333
38	31	91	485	0.00151667
39	42	24	273	0.00040000
40	55	426	1439	0.00710000
41	41	7068	4646	0.11780000
42	47	58	478	0.00096667
43	46	49	379	0.00081667
44	62	850	1995	0.01416667
45	73	14050	5965	0.23416667
46	90	58120	13576	0.96866667
47	73	1483	2493	0.02471667
48	68	20034	7973	0.33390000
49	52	283	970	0.00471667
50	55	30128	9339	0.50213333
51	67	3012	3467	0.05020000
52	47	137	673	0.00228333
53	116	23886	8946	0.39810000
54	58	63906	13346	1.06510000
55	66	1877	2752	0.03128333
56	60	13003	6124	0.21671667
57	224	254935	29069	4.24891667
58	55	584	1358	0.00973333
59				
60	100	3020811	109044	50.34685000
61				
62				
63				
64	134	1290250	64270	21.50416667
65				
66				
67	101	5293212	130423	88.22020000
68				
69				
70	63	4441	3764	0.07401667
71	76	2921	3177	0.04868333
72	87	49498	12667	0.82496667
73	146	412463	36378	6.87438333
74	15	4	76	0.00006670
75	102	97517	17807	1.62528333
76	125	119815	19954	1.99691667
77	80	1591	2801	0.02651667
78	96	25673	9308	0.42788333
79	154	2024922	83852	33.74870000

80				
81	59	991	1922	0.01651667
82	120	2961708	103354	49.36180000
83	99	3082117	102054	51.36861667
84	172	24848	9407	0.41413333
85	93	1434782	73308	23.91303333
86	138	1602174	80805	26.70290000
87	75	57569	13753	0.95948333
88	74	688	1992	0.01146667
89	123	483741	38982	8.06235000
90	101	12266	6631	0.20443333
91	63	891	2072	0.01485000
92	99	134365	20532	2.23941667
93	109	219734	26160	3.66223333
94	109	36091	10448	0.60151667
95	105	28986	10001	0.48310000
96	93	33673	10370	0.56121667
97	69	7164	5277	0.11940000
98	126	45795	12039	0.76325000
99	91	15304	7583	0.25506667
100	140	1516803	71926	25.28005000
101	170	957527	54751	15.95878333
102	98	1681558	73875	28.02596667
103	193	1418532	65268	23.64220000
104	57	87300	14173	1.45500000
105	103	3836615	101409	63.94358333
106	84	216569	21194	3.60948333
107	117	4097585	100747	68.29308333
108	69	6294	4661	0.10490000
109	126	1432545	62549	23.87575000
110	169	3008260	99089	50.13766667
111	142	66149	14018	1.10248333
112	49	3549	3234	0.05915000
113	180	1335366	64951	22.25610000
114	76	2641	3207	0.04401667
115				
116				
117	94	30337	7870	0.50561667
118				
119	97	719529	47879	11.99215000
120	177	229416	28931	3.82360000
121	122	1743017	73125	29.05028333
122	119	3129003	108354	52.15005000
123	145	560004	43982	9.33340000
124	87	7490761	162198	124.84601670
125	101	417152	36737	6.95253333
126				
127				
128	171	972012	58833	16.20020000
129				
130	40	10090	4191	0.16816667
131				
132				
133	40	216043	28071	3.60071667
134				
135	33	33501	9697	0.55835000
136				
137	81	358431	33373	5.97385000
138	40	1303306	63557	21.72176667
139				
140				
141				
142				
143	24	2044	2334	0.03406667
144				
145	47	3889514	110456	64.82523333
146	40	129618	20215	2.16030000
147	39	222082	26483	3.70136667
148	83	1661010	81592	27.68350000
149				
150				

Tabla 5: Datos A con función "Admisible"*

Tiempo total (ms)	Tiempo total (min)	Tiempo total (horas)	Tiempo ejecución (horas)	Nodos totales	Pasos totales	Niveles completados
66658184	1110.97	18.52	70.52	3016835	9645	124

Tabla 6: Datos total A con función "Admisible"*

Algoritmo A* con función de evaluación "Voraz"

ID	Steps	Time(ms)	Nodes	Time (min)
1	15	86	95	0.001433333
2	12	14	26	0.000233333
3	10	2	21	3.33E-05
4	10	1	16	1.67E-05
5	29	83	167	0.001383333
6	43	127	197	0.002116667
7	34	83	175	0.001383333
8	32	73	191	0.001216667
9	41	106	242	0.001766667
10	20	10	74	0.000166667
11	37	128	309	0.002133333
12	42	269	349	0.004483333
13	56	206	359	0.003433333
14	49	184	379	0.003066667
15	45	27	195	0.00045
16	43	78	262	0.0013
17	101	250	645	0.004166667
18	44	48	166	0.0008
19	55	309	794	0.00515
20	69	73	387	0.001216667
21	79	93	414	0.00155
22	86	1029	1706	0.01715
23	91	715	1528	0.011916667
24	83	219	789	0.00365
25	77	243	855	0.00405
26	147	1491	2063	0.02485
27	235	2554	2942	0.042566667
28	71	1178	2343	0.019633333
29	166	6305	3024	0.105083333
30	110	1756	1446	0.029266667
31	264	5606	3888	0.093433333
32	277	8583	4492	0.14305
33	319	4204	3788	0.070066667
34	231	10748	4977	0.179133333
35	163	691	1626	0.011516667
36	19	2	40	3.33E-05
37	30	5	88	8.33E-05
38	39	57	366	0.00095
39	43	12	207	0.0002
40	57	259	973	0.004316667
41	83	309	1108	0.00515
42	52	68	400	0.001133333
43	53	49	328	0.000816667
44	74	417	1375	0.00695
45	119	7296	4981	0.1216
46	186	14414	6796	0.240233333
47	85	338	1285	0.005633333
48	268	2420	3126	0.040333333
49	50	143	621	0.002383333
50	137	1485	2244	0.02475
51	83	433	1545	0.007216667
52	54	58	598	0.000966667
53	156	3254	3809	0.054233333
54	188	884	1883	0.014733333
55	90	721	2017	0.012016667
56	180	1040	1981	0.017333333
57	402	262284	28075	4.3714
58	55	25	319	0.000416667
59	636	1123045	58753	18.71741667
60	463	41187	11476	0.68645
61	397	6760	4245	0.112666667
62	363	308081	27559	5.134683333
63	385	725509	45620	12.09181667
64	1060	908214	48014	15.1369
65	658	138741	16368	2.31235
66	1219	85842	14667	1.4307
67	366	16476	6657	0.2746
68	371	13601	5721	0.226683333
69	393	2878607	91915	47.97678333
70	80	1652	2408	0.027533333
71	86	607	1640	0.010116667
72	137	1228	2199	0.020466667
73	188	275381	28121	4.589683333
74	17	1	33	1.67E-05
75	146	3912	3489	0.0652
76	173	130594	19517	2.176566667
77	88	1310	2342	0.021833333
78	162	3389	3459	0.056483333
79	302	28407	9111	0.47345
80	377	66775	14125	1.112916667
81	61	239	754	0.003983333
82	205	37683	11408	0.62805
83	213	134929	17761	2.248816667
84	222	20123	7291	0.335383333
85	415	342273	32428	5.70455
86	418	14661	7746	0.24435
87	103	15710	6354	0.261833333

88	84	579	1806	0.00965
89	225	72513	13213	1.20855
90	139	4295	3729	0.071583333
91	71	284	846	0.004733333
92	255	83492	14798	1.391533333
93	263	4384	3691	0.073066667
94	157	7486	4947	0.124766667
95	111	11589	6615	0.19315
96	135	1894	2661	0.031566667
97	89	5535	4623	0.09225
98	152	10020	5960	0.167
99	137	10678	6374	0.177966667
100	248	725765	46760	12.09608333
101	350	115167	17877	1.91945
102	236	133732	18569	2.228866667
103	277	311032	27664	5.183866667
104	60	8217	4850	0.13695
105	193	883668	51225	14.7278
106	118	79851	15077	1.33085
107	187	2411713	88127	40.19521667
108	81	2408	3357	0.040133333
109	194	581727	41663	9.69545
110	219	662621	46266	11.04368333
111	170	33411	10563	0.55685
112	56	302	1189	0.005033333
113	276	927432	53958	15.4572
114	84	1706	2201	0.028433333
115	680	4367459	104987	72.79098333
116	454	2994292	99455	49.90486667
117	129	3014	3430	0.050233333
118	244	5152601	135889	85.87668333
119	169	421364	37204	7.022733333
120	215	121973	22025	2.032883333

121	204	952269	58333	15.87115
122	147	447535	41033	7.458916667
123	249	410515	38588	6.841916667
124	481	353315	33591	5.888583333
125	157	349432	34542	5.823866667
126	572	9226	5659	0.153766667
127	490	2225392	84862	37.08986667
128	280	732077	50825	12.20128333
129	420	4805904	137269	80.0984
130	106	317	1083	0.005283333
131	485	4516877	116758	75.28128333
132	111	1562247	65448	26.03745
133	60	3538	3219	0.058966667
134				
135	89	3840	3544	0.064
136	230	11916	6565	0.1986
137	105	106523	18845	1.775383333
138	66	445	1099	0.007416667
139				
140	180	146679	20877	2.44465
141				
142	239	67717	15258	1.128616667
143	48	38	194	0.000633333
144				
145	50	1899581	74658	31.65968333
146	194	3723903	106539	62.06505
147	108	231352	24666	3.855866667
148	119	310913	34093	5.181883333
149	254	52782	13200	0.8797
150	157	174941	23623	2.915683333

Tabla 7: Datos A con función "Voraz"*

Tiempo total (ms)	Tiempo total (min)	Tiempo total (horas)	Tiempo ejecución (horas)	Nodos totales	Pasos totales	Niveles completados
50919960	848.67	14.15	22.15	2522246	27452	146

Tabla 8: Datos total A con función "Voraz"*

Algoritmo A* con función de evaluación "Test"

ID	Steps	Time(ms)	Nodes	Time(min)
1	15	75	75	0.00125
2	15	2	25	3.33E-05
3	12	8	22	0.000133333
4	10	4	17	6.67E-05
5	37	124	164	0.002066667
6	41	89	142	0.001483333
7	34	92	158	0.001533333
8	30	163	156	0.002716667
9	39	190	153	0.003166667
10	20	12	47	0.0002
11	36	397	437	0.006616667
12	34	89	231	0.001483333
13	45	151	429	0.002516667
14	41	31	168	0.000516667
15	45	12	195	0.0002
16	53	51	282	0.00085
17	53	40	285	0.000666667
18	44	158	418	0.002633333
19	47	221	667	0.003683333
20	49	88	341	0.001466667
21	70	115	420	0.001916667
22	74	92	407	0.001533333
23	63	111	345	0.00185
24	54	286	672	0.004766667
25	61	180	506	0.003
26	71	378	865	0.0063
27	113	342	724	0.0057
28	64	364	984	0.006066667
29	90	4129	2616	0.068816667
30	92	207	551	0.00345
31	96	2431	1825	0.040516667
32	124	32454	9397	0.5409
33	97	9610	6356	0.160166667
34	82	4300	4175	0.071666667
35	95	1001	1721	0.016683333
36	19	2	30	3.33E-05
37	30	3	57	0.00005
38	36	69	227	0.00115
39	43	22	202	0.000366667
40	55	244	889	0.004066667
41	67	88	394	0.001466667
42	48	66	379	0.0011
43	47	22	199	0.000366667
44	72	549	1290	0.00915
45	85	578	1560	0.009633333
46	142	2659	3319	0.044316667
47	83	388	1240	0.006466667
48	88	148	721	0.002466667
49	74	218	676	0.003633333
50	78	109	528	0.001816667
51	109	851	1748	0.014183333
52	48	22	344	0.000366667
53	120	10103	5530	0.168383333
54	76	558	1517	0.0093
55	77	611	1889	0.010183333
56	78	59	511	0.000983333
57	308	75316	14748	1.255266667
58	55	34	398	0.000566667
59	240	1257854	60171	20.96423333
60	147	889	2293	0.014816667
61	103	180	906	0.003
62	197	169419	20900	2.82365
63	221	50206	13339	0.836766667
64	140	119329	17785	1.988816667
65	165	783	2212	0.01305
66	139	52679	12927	0.877983333
67	118	97	621	0.001616667
68	131	217	939	0.003616667
69	183	534	1663	0.0089
70	64	2848	3314	0.047466667
71	80	592	1415	0.009866667
72	119	7988	4815	0.133133333
73	186	373643	31580	6.227383333
74	17	2	27	3.33E-05
75	116	7401	3811	0.12335
76	157	134534	18622	2.242233333
77	90	402	1501	0.0067
78	164	5458	4304	0.090966667
79	224	18689	7791	0.311483333
80	179	54508	12164	0.908466667
81	59	13	253	0.000216667
82	165	15364	7509	0.256066667
83	139	31286	9179	0.521433333
84	204	14588	7087	0.243133333
85	137	184238	23456	3.070633333

86	180	628	2109	0.010466667
87	89	19067	7476	0.317783333
88	74	433	1622	0.007216667
89	181	14603	6586	0.243383333
90	169	6514	4794	0.108566667
91	67	260	1096	0.004333333
92	141	48545	11224	0.809083333
93	125	1611	2572	0.02685
94	153	11797	6230	0.196616667
95	119	4635	4327	0.07725
96	103	2872	3396	0.047866667
97	75	5120	4621	0.085333333
98	134	16224	7240	0.2704
99	103	10057	6143	0.167616667
100	152	3070	3592	0.051166667
101	210	159183	21868	2.65305
102	186	57354	13272	0.9559
103	285	191334	24701	3.1889
104	72	1710	2626	0.0285
105	164	734096	48098	12.23493333
106	100	31828	10411	0.530466667
107	151	1922651	78720	32.04418333
108	77	255	1153	0.00425
109	194	664306	45059	11.07176667
110	211	1651148	75051	27.51913333
111	176	45077	12175	0.751283333
112	50	1156	2329	0.019266667
113	210	656443	47620	10.94071667
114	80	792	1736	0.0132
115	238	5897455	122888	98.29091667
116	431	2646764	92384	44.11273333
117	157	2328	2878	0.0388
118	242	1318180	63758	21.96966667
119	157	106876	17596	1.781266667

120	201	144391	22341	2.406516667
121	215	588843	40196	9.81405
122	163	394754	33098	6.579233333
123	171	238459	26752	3.974316667
124	205	85244	16414	1.420733333
125	179	199958	25502	3.332633333
126	276	2721	3410	0.04535
127	206	268611	28162	4.47685
128	270	585194	42719	9.753233333
129				
130	71	331	490	0.005516667
131	347	1470342	61297	24.5057
132	93	208637	21296	3.477283333
133	96	1808	2362	0.030133333
134	190	106211	15979	1.770183333
135	33	65	149	0.001083333
136				
137	101	3499	2583	0.058316667
138	44	163293	14267	2.72155
139	236	4855401	135323	80.92335
140	114	15313	6846	0.255216667
141				
142	111	954	1891	0.0159
143	36	37	113	0.000616667
144	170	5285905	121432	88.09841667
145	62	51024	12496	0.8504
146	172	1749630	73182	29.1605
147	98	40625	10727	0.677083333
148	155	2458733	95837	40.97888333
149	327	565201	43327	9.420016667
150	137	2804	3287	0.046733333

Tabla 9: Datos A con función "Test"*

Tiempo total (ms)	Tiempo total (min)	Tiempo total Niveles(horas)	Tiempo ejecución (horas)	Nodos totales	Pasos totales	Niveles completados
38376817	639.61	10.66	16.66	1963785	17403	147

Tabla 10: Datos total A con función "Test"*

Algoritmo IDA*

ID	Steps	Time(ms)	Nodes	Time(min)
1	17	163	78	0.002716667
2	13	61	25	0.001016667
3	10	22	43	0.000366667
4	10	13	11	0.000216667
5	33	223	96	0.003716667
6	45	514	178	0.008566667
7	38	151	171	0.002516667
8	40	231	156	0.00385
9	44	1093	279	0.018216667
10	22	79	66	0.001316667
11	54	1608	478	0.0268
12	52	525	250	0.00875
13	78	3543	850	0.05905
14	87	5325	671	0.08875
15	57	276	177	0.0046
16	91	4790	597	0.079833333
17	65	2345	358	0.039083333
18	54	1486	611	0.024766667
19	45	1706	499	0.028433333
20	51	3287	1175	0.054783333
21	86	3293	468	0.054883333
22	160	33566	691	0.559433333
23	119	18661	690	0.311016667
24	85	9469	511	0.157816667
25	97	6116	826	0.101933333
26	243	1055500	6185	17.59166667
27	144	104551	4977	1.742516667
28	108	11936	1907	0.198933333
29	290	432668	2914	7.211133333
30	148	320734	7002	5.345566667
31	392	3465054	15621	57.7509
32	550	4557886	16526	75.96476667
33	533	5056394	10493	84.27323333
34	162	1405227	9898	23.42045
35	143	651860	6354	10.86433333
36	21	20	28	0.000333333
37	36	317	180	0.005283333
38	33	390	258	0.0065
39	46	330	189	0.0055
40	71	6084	790	0.1014
41	69	19789	2316	0.329816667
42	51	502	321	0.008366667
43	56	623	320	0.010383333
44	80	12179	1486	0.202983333
45	243	537170	2939	8.952833333
46	148	507447	6600	8.45745
47	151	26737	1887	0.445616667
48	120	173001	4185	2.88335
49	60	885	490	0.01475
50	120	191879	5616	3.197983333
51	141	54802	1805	0.913366667
52	51	729	333	0.01215
53	374	1610149	3891	26.83581667
54	114	178020	3066	2.967
55	97	43311	1994	0.72185
56	116	79722	2414	1.3287
57				
58	97	13946	354	0.232433333
59				
60				
61				
62				
63				
64				
65				
66				
67				
68				
69				
70	93	67524	2906	1.1254
71	188	97415	2396	1.623583333
72	135	1269562	7048	21.15936667
73				
74	15	86	41	0.001433333
75	203	2711143	6754	45.18571667
76				
77	108	50265	2274	0.83775
78	198	781159	2365	13.01931667
79				
80				
81	63	5813	1464	0.096883333
82				
83				
84	298	2228441	8070	37.14068333
85				
86				
87	231	4248216	9897	70.8036
88	94	20007	1263	0.33345
89				
90	319	541513	4080	9.025216667
91	103	26320	1825	0.438666667

92				
93				
94	235	2595959	4316	43.26598333
95	229	3371025	4662	56.18375
96	251	2344972	5502	39.08286667
97	77	67519	4196	1.125316667
98	308	1982106	6930	33.0351
99	221	825276	6378	13.7546
100				
101				
102				
103				
104	80	822827	3911	13.71378333
105				
106				
107				
108	115	164331	3003	2.73885
109				
110				
111				
112	53	29802	1087	0.4967
113				
114	104	40823	1821	0.680383333
115				
116				
117	158	413120	5473	6.885333333
118				
119				
120				
121				
122				

123				
124				
125				
126				
127				
128				
129				
130	51	3656	1255	0.060933333
131				
132				
133	50	441038	13350	7.350633333
134				
135	43	130628	7864	2.177133333
136				
137				
138	48	1724102	19396	28.73503333
139				
140				
141				
142				
143	26	2607	314	0.04345
144				
145				
146				
147	69	2100453	7257	35.00755
148				
149				
150				

*Tabla 11: Datos IDA**

Tiempo total (ms)	Tiempo total (min)	Tiempo total (horas)	Tiempo ejecución (horas)	Nodos totales	Pasos totales	Niveles completados
49726096	828.77	13.81	139.81	280392	10627	87

*Tabla 12: Datos total IDA**