Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-5437 - Inteligencia Artificial I
Abril - Julio 2021

Proyecto 4: SSS* y MDT(f)

Jesús De Aguiar 15-10360
Neil Villamizar 15-11523
Jesús Wahrman 15-11540

Problema

El objetivo del proyecto es el estudio de nuevos algoritmos de recorrido sobre árboles de juego. Se estudiará nuevamente la versión reducida del juego de Otellho utilizada en el segundo proyecto de la materia, pero con la introducción de dos nuevos algoritmos: SSS* y MDT(f). Estos algoritmos serán implementados y su rendimiento será medido con experimentos ejecutados sobre instancias del juego de Otellho. Los resultados serán comparados con aquellos obtenidos por los algoritmos implementados en el segundo proyecto.

Algoritmos estudiados

SSS*

*SSS** es un algoritmo de búsqueda sobre árboles de estados desarrollado por George C. Stockman en 1977, aunque a través del tiempo fue modificado y mejorado por otros investigadores.

La idea del algoritmo es ejecutar un recorrido al estilo *best first search*, de manera similar a lo realizado por A*. El algoritmo busca recortar las ramas del árbol de estados en cada nodo MAX.

Stockman fue capaz de demostrar que el algoritmo SSS* no genera más nodos que algoritmos con Alpha-Beta prunning, y que además pueden existir nodos que se generan en Alpha-Beta prunning que no son considerados en una ejecución de SSS* lo que se traduce en un mejor rendimiento en tiempo para una misma entrada si se comparan directamente ambos algoritmos. Stockman especulaba con que este algoritmo sería mejor de manera general que algoritmoa Alpha-Beta. Pero existía un problema: para SSS*, la contraposición a esta mejora en el rendimiento se traduce en un aumento del uso de memoria, ya que el algoritmo debe mantener una cola de prioridad de la cual tomar la mejor opción al momento, de manera similar a A*, por lo que debe mantener en memoria una cantidad de estados que crece de manera exponencial en la profundidad del árbol.

Este *trade-off*, además de su complejidad para ser entendido y relacionado con otros algoritmos tradicionales, ocasionó que, aunque las ideas del algoritmo parecían ser muy buenas para mejorar

el rendimiento, no se tomara como una opción útil para resolver problemas de este estilo, debido al importante consumo de recursos de memoria.

Sin embargo, Plaat, Schaeffer, Pijls y Bruin (1995), un grupo de investigadores de distintas universidades, pudieron demostrar que SSS* equivale a una secuencia de ejecución de un algoritmo Alpha-Beta con rangos nulos (esto es, donde alpha y beta tienen el mismo valor o difieren en una unidad) utilizando tablas de transposición. De esta manera, propusieron el algoritmo MDT(f), con la intención de mejorar el rendimiento general del algoritmo.

MDT(f)

MDT(f) es un algoritmo de búsqueda sobre árboles de juego basado en búsquedas Alpha-Beta con tablas de transposición, donde inicialmente los valores de Alpha y Beta difieren a lo sumo en una unidad.

La idea del algoritmo es realizar llamadas de rango nulo del algoritmo Alpha-Beta prunning con una aproximación al valor minimax del estado final del problema (el parámetro f), el cual servirá como la cota inicial (beta) para el algoritmo Alpha-Beta, en lugar de escoger una cota arbitrariamente grande. Mientras más se acerque el valor f al valor minimax del problema, mejor será el rendimiento del algoritmo. El resultado obtenido por el algoritmo es una aproximación cada vez más cercana al valor minimax. De esta manera, si se ejecuta al estilo de *iterative deeping*, de podrá llegar a la solución exacta después de una serie de iteraciones. Además, si se acompaña con tablas de transposición, se evita repetir búsquedas, eliminando el *overhead* de realizar búsquedas sobre partes del árbol que ya han sido exploradas.

El algoritmo fue descrito por los investigadores Aske Plaat, Jonathan Schaeffer, Wim Pijls, y Arie de Bruin de la Universidad de Alberta en 1995. El planteamiento surgió debido a dos problemas principales que los investigadores notaban sobre SSS*:

- SSS* es un algoritmo díficil de entender y muy complejo para realizar análisis sobre su comportamiento. La intención principal de los investigadores era reducir esta dificultad y fue logrado escribiendo un algoritmo basado en llamadas a un algoritmo Alpha-Beta.
- SSS* es un algoritmo con un uso de recursos importante. La reducción de estos se alcanzó al eliminar el requerimiento de SS* de mantener el conjunto ordenado de estados generados en memoria.

Implementación

La implementación de los algoritmos se encuentra en el repositorio de GitHub del proyecto. Ambos algoritmos se encuentran implementados en el archivo *main.cc*.

La implementación de SSS* se realizó realizando una traducción del algoritmo tradicional encontrado en la literatura. El pseudocódigo del algoritmo es el siguiente:

```
int SSS* (node n; int bound)
{
   push (n, LIVE, bound);
   while ( true ) {
      pop (node);
      switch ( node.status ) {
      case LIVE:
          if (node == LEAF)
             insert (node, SOLVED, min(eval(node),h));
          if (node == MIN_NODE)
             push (node.1, LIVE, h);
          if (node == MAX_NODE)
             for (j=w; j; j--)
                push (node.j, LIVE, h);
          break;
      case SOLVED:
          if (node == ROOT_NODE)
             return (h);
          if (node == MIN_NODE) {
              purge (parent(node));
              push (parent(node), SOLVED, h);
          }
          if (node == MAX_NODE) {
             if (node has an unexamined brother)
                push (brother(node), LIVE, h);
             else
                push (parent(node), SOLVED, h);
          }
          break;
      }
   }
}
```

Por otro lado, la implementación de MDT(f) se realizó con la implementación de negamax realizada en el proyecto anterior.

De la misma manera, la representación del juego de Othello y los demás algoritmos que serán utilizados en la experimentación son aquellas implementadas en la entrega del proyecto 2.

Experimentos y Resultados

Para poner a prueba los algoritmos, se ejecutó cada uno de la misma manera que en la entrega anterior de este proyecto: Evaluando el algoritmo en una instancia perteneciente a la variación principal del problema que se encuentra a una distancia cada vez más lejana de la meta del problema en cada iteración. El tiempo límite establecido por el grupo para la ejecución de cada uno de los algoritmos es de diez minutos, donde se permitió que el algoritmo se ejecutara en la mayor cantidad de nodos iniciales posibles pertenecientes a la variación principal hasta que este tiempo se agotara.

Los algoritmos ejecutados son:

- Negamax con poda Alpha-Beta
- Negamax con poda Alpha-Beta y Tablas de Transposición
- SSS*
- mdt con parámetro f=-4 (valor minimax real)
- mdt con parámetro f=16 (valor minimax cercano)
- mdt con parámetro f=50 (valor minimax lejano)
- negascout

Los resultados se exponen a continuación, en las distintas tablas y gráficas:

Algoritmo	Profundidad del nodo inicial
Negamax con poda alpha-beta	13
Negamax con poda alpha-beta y tablas de transposición	13
SSS*	13 (Abortada por consumo de memoria)
mdt(-4)	11
mdt(16)	12
mdt(50)	12
Negascout	12

Table 1: Versión más compleja del problema resuelta por cada algoritmo en el tiempo establecido

Esta primera capa nos muestra el problema real de SSS*, donde la ejecución del algoritmo fue abortada antes de los diez minutos por consumir la totalidad de la memoria de la máquina donde se realizaban los experimentos. Se puede observar como MDT es una mejora real de este algoritmo porque permitió un uso de memoria más eficiente lo que causó que los algoritmos llegaran más lejos en el árbol de estados.
Podemos compara un poco más a profundidad el rendimiento de cada una de estas ejecuciones utilizando la salida de la ejecución de los algoritmos. En primer lugar, veamos las comparaciones de resultados en las capas con profundidad 12 y 13:

Algoritmo	Nodos Generados	Nodos Expandidos	Tiempo de Ejecución (s)	Nodos Generados por Segundo
Negamax AB	415909956	315074162	129.014	3223760
Negamax AB & TT	91347571	61340613	89.4008	1021780
SSS*	163152895	141112133	111.397	1464610
MDT(-4)	26667290	18485815	18.7164	1424810
MDT(16)	44402795	30506877	39.9056	1112700
MDT(50)	49373925	33793435	43.66145	1130840
Negascout	242589301	185296093	80.6729	3007070

Table 2: Resultados obtenidos por los algoritmos tras recorrer el árbol del juego desde la profundidad 13

Finalmente, podemos ver la comparación del rendimiento de las distintas ejecuciones en las siguientes gráficas:

Figure 1: Nodos generados por corrida.

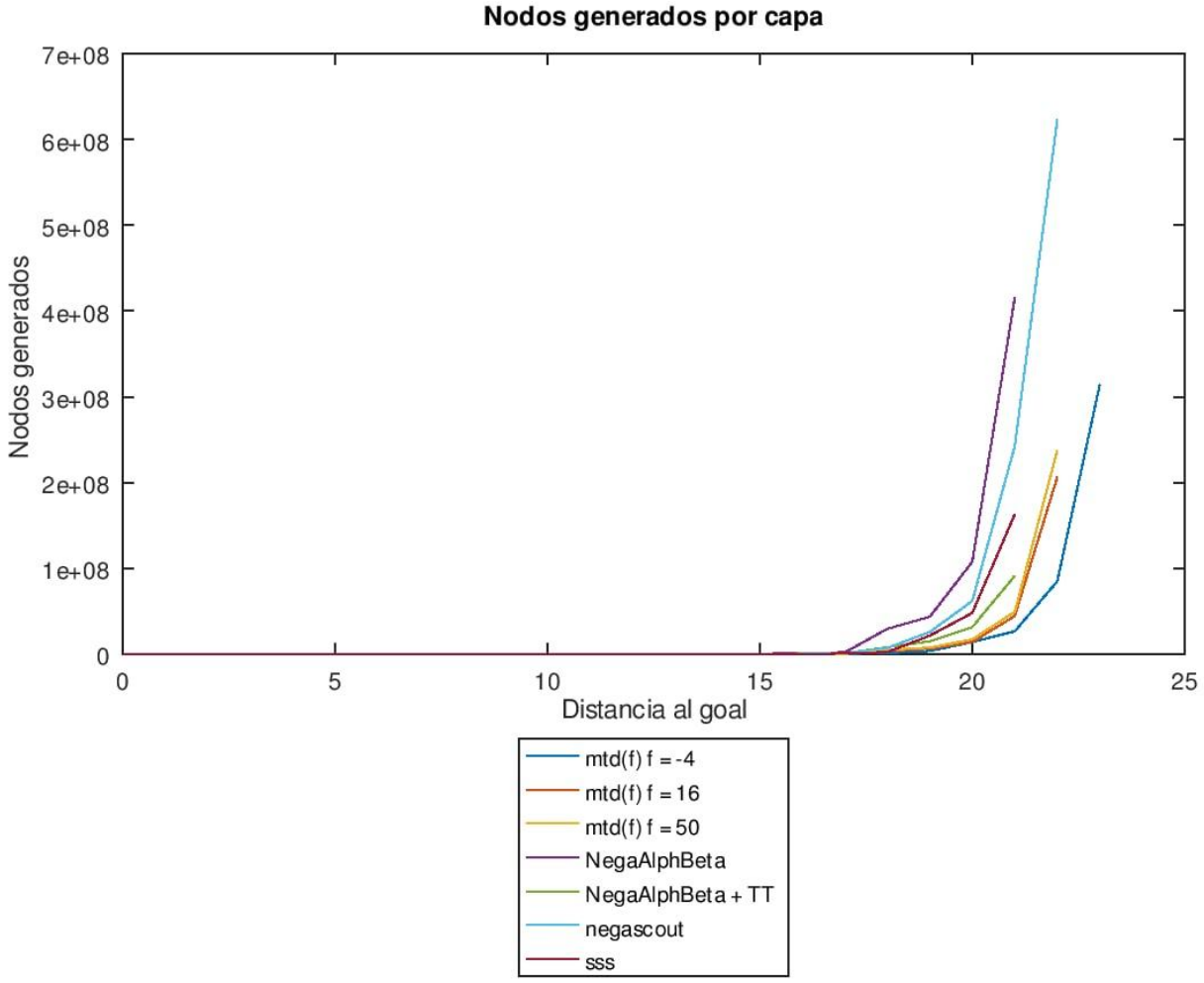


Figure 2: Nodos expandidos por corrida.

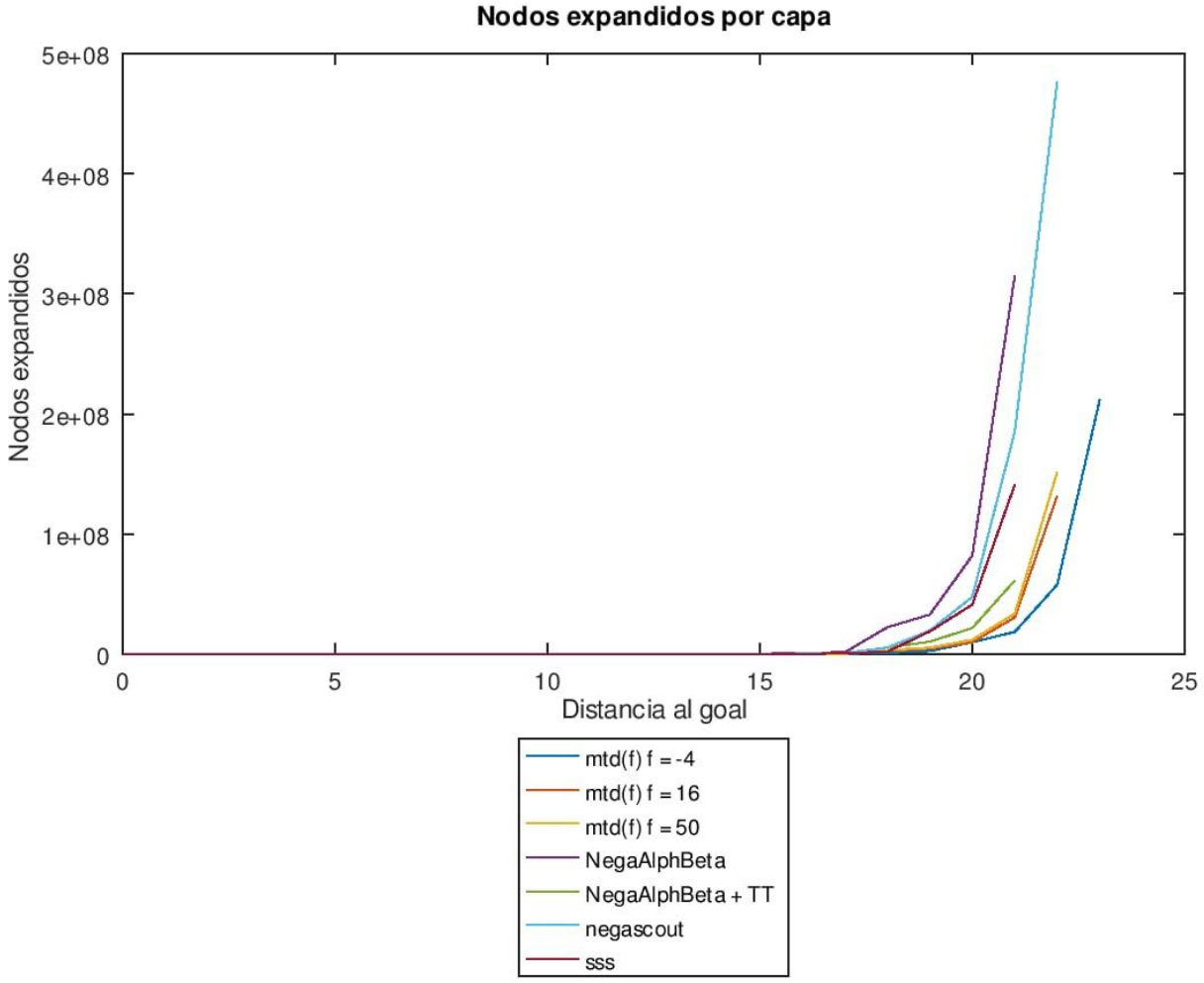
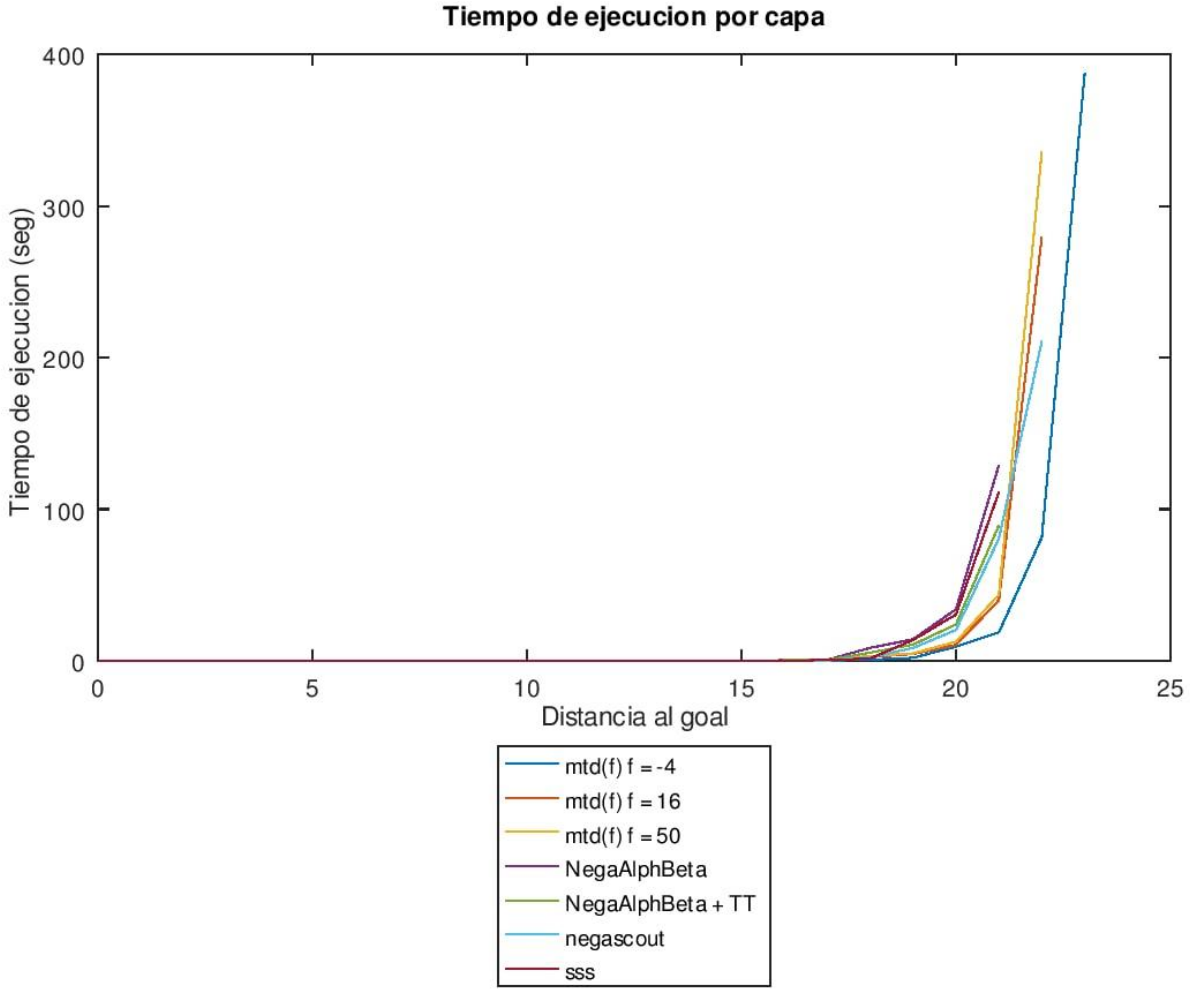


Figure 3: Tiemplo empleado en cada corrida.



De los experimentos realizados, podemos realizar las siguientes observaciones:

- SSS* representa una mejora en el tiempo de ejecución de con respecto a negamax con poda y negamax con poda y tablas de transposición.
- SSS* fue capaz de reducir también la cantidad de nodos generados y nodos expandidos por negamax alpha-beta. No sucedió lo mismo con el algoritmo negamax utilizando tablas de transposición (ya que la intención de estas es reducir la generación y expansión), pero aún así fue capaz de superar a los algoritmos en tiempo.
- Se evidenció el problema principal de SSS* al consumir la memoria de la máquina donde el algoritmo se ejecutaba. Con más recursos, podríamos haber estudiado si llegaba a una mayor profundidad.
- Las distintas versiones de MDT fueron capaces de superar el rendimiento de SSS* al resolver el problema del uso de la memoria. Esto le permitió avanzar a una profundidad mayor que los algoritmos anteriores.
- Al mismo tiempo, el algoritmo fue capaz de reducir la cantidad de nodos generados y expandidos en la ejecución de los algoritmos.
- Si realizamos una comparación con negascout, para los casos triviales las distintas versiones de MDT superan su rendimiento, pero a partir de una profundidad más alta, negascout es mucho más veloz que las versiones de MDT(16) y MDT(50).
- Por otro lado, MDT(-4) es capaz de superar a negascout ya que posee información precisa del valor minimax y es capaz de resolver el problema de manera eficiente.

Conclusiones

- Aunque SSS* representa una mejora en tiempo para Negamax Alpha-Beta, el uso de memoria representa un impedimento importante para la utilización del algoritmo en una tarea real.
- MDT representa una mejora, tanto en la facilidad para escribir el algoritmo, como en el rendimiento del algoritmo. Aún así, depende la escogencia del los parámetros Alpha - Beta (que por defecto pueden utilizarse unos valores lo suficientemente grande).
- La versión de MDT con el valor minimax total fue capaz de superar a Negascout, pero en la práctica esto no es tan fácil de realizar, ya que es muy complicado en algunos casos estimar este valor.
- Debido a lo expuesto en el último punto, se considera MDT(-4) como un buen algoritmo para resolver Othello. Esto no es aplicable para los problemas en general.