PONTIFICIA
UNIVERSIDAD
CATÓLICA
DE CHILE

Problemas de Búsqueda

Manuel Espinoza Quintero

IIC2613 — Inteligencia Artificial
Profesor Jorge Baier Aranda
Mayo de 2024

1. Teoría

1. La ventaja práctica de Rushed-A* se puede observar en escenarios donde la función heurística h utilizada es muy precisa, es decir, $h(s) \approx h^*(s)$, donde $h^*(s)$ es el costo heurístico real. En tales casos, es probable que muchos nodos sucesores v tengan un valor f similar al del nodo actual u. Este comportamiento puede ocurrir frecuentemente en problemas donde los costos de transición entre estados son uniformes o casi uniformes.

   Por ejemplo, la búsqueda en un Gráfico de Costo Uniforme, donde todas las aristas tienen el mismo costo. En tal gráfico, el costo real desde cualquier nodo s al objetivo puede aproximarse bien por una heurística $h(s)$ que mide la distancia en términos de número de aristas. Supongamos además que el gráfico es muy grande, pero de forma regular, similar a una cuadrícula.

   El algoritmo A* tradicional, se requiere expandir todos los nodos con el menor valor hasta encontrar el objetivo. Esto puede resultar en un gran número de expansiones si el objetivo está a varias aristas de distancia.

   En Rushed-A*, gracias a la línea adicional, tan pronto como se evalúe un nodo sucesor v que es objetivo y cumple $f(v) = f(u)$, el algoritmo puede terminar inmediatamente, evitando expansiones adicionales. En un gráfico de costo uniforme, esta condición se puede cumplir frecuentemente ya que la heurística precisa puede hacer que $f(v) \approx f(u)$.

   Finalmente, Rushed-A* puede reducir significativamente el tiempo de ejecución en escenarios donde la heurística es precisa y los costos de transición son uniformes. La reducción proviene de evitar expansiones innecesarias una vez que un nodo objetivo con el valor f adecuado es encontrado.

2. Para garantizar que Rushed-A* retorna soluciones óptimas, es necesario que la heurística h cumpla con dos condiciones:

   a) **Admisibilidad:** La heurística h nunca sobreestima el costo real desde cualquier nodo s hasta el objetivo s_{goal}, es decir, $h(s) \leq \delta(s, s_{goal})$ para todo estado s.

   b) **Consistencia (Monótona):** Para cada nodo s y cada sucesor s' generado por una acción a, la heurística satisface $h(s) \leq c(s, a, s') + h(s')$, donde $c(s, a, s')$ es el costo del paso desde s a s'.

Demostración de Óptimalidad

Teorema: Rushed-A* es óptimo si la heurística h es admisible y consistente.

Demostración:
Admisibilidad y Consistencia
Una heurística admisible y consistente garantiza que todos los caminos generados por el algoritmo no sobreestimen el costo real al objetivo.

Propiedad de la Función f

Para Rushed-A*, la función $f(s) = g(s) + h(s)$ sigue siendo válida. La propiedad de consistencia de la heurística asegura que el valor de f nunca decrece a lo largo de un camino expandido, lo que implica que si $f(v) = f(u)$, entonces el nodo v podría ser tan prometedor como u en términos de llegar al objetivo con un costo óptimo.

Propiedad de f en el Nodo Objetivo
Consideremos un nodo objetivo s_{goal} alcanzado por Rushed-A*. Si s_{goal} se encuentra en el conjunto de sucesores v tal que $f(v) = f(u)$ para algún nodo u extraído de la frontera (conjunto Open), entonces:

$$f(s_{goal}) = f(u) = g(u) + h(u)$$

Dado que u es un nodo expandido en el camino óptimo hacia s_{goal} y h es consistente, el valor $g(u)+h(u)$ es una estimación precisa del costo total desde el nodo inicial hasta el objetivo.

No Subestimación del Costo Real
La condición adicional de Rushed-A* verifica si v es objetivo y $f(v) = f(u)$ antes de insertar v en Open. Si esta condición se cumple y v es el nodo objetivo, Rushed-A* termina y retorna v. Debido a la admisibilidad y consistencia de h, podemos asegurar que $f(v)$ es el costo real óptimo.

Dado que en cada paso del algoritmo, la heurística y el costo acumulado se mantienen consistentes y admisibles, Rushed-A* nunca sobreestima el costo al objetivo. Así, cuando el algoritmo retorna un nodo objetivo, el costo asociado es óptimo. Finalmente, bajo las condiciones de admisibilidad y consistencia de la heurística, Rushed-A* es óptimo.

2. DCComilon

2.1. Implementación BFS y DFS invertido

Los resultados obtenidos son los siguientes para los algoritmos de DFS, DFS invertido y BFS:

Prueba	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
DFS 1	0.1099000000	41	39	39
DFS 2	0.1018000000	41	39	49.5
DFS 3	0.7789000000	197	155	155
DFS 4	0.7770000000	197	155	163.5
DFS 5	8.0915000000	360	124	124
DFS 6	10.8664000000	360	124	144.5
DFS 7	96.8171000000	1696	433	481.0
DFS 8	81.1145000000	1668	343	414.0

Cuadro 1: Resultados del Algoritmo DFS

Prueba	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
Inverted DFS 1	0.1580000000	55	25	25
Inverted DFS 2	0.2236000000	55	25	20.5
Inverted DFS 3	2.0982000000	330	107	107
Inverted DFS 4	2.2058000000	330	107	107.0
Inverted DFS 5	9.3002000000	330	174	174
Inverted DFS 6	8.1272000000	330	174	196.0
Inverted DFS 7	33.2251000000	1245	661	779.5
Inverted DFS 8	22.2682000000	914	391	415.0

Cuadro 2: Resultados del Algoritmo Inverted DFS

Prueba	Tiempo de Ejecución (ms)	Número de Expansiones	Largo del Camino	Costo Total del Camino
BFS 1	0.1387000000	58	19	19
BFS 2	0.1343000000	58	19	22
BFS 3	2.4620000000	381	93	93
BFS 4	2.4297000000	381	93	106
BFS 5	24.5708000000	628	94	94
BFS 6	21.8449000000	628	94	109.5
BFS 7	124.5110000000	1792	261	353
BFS 8	42.4207000000	1597	223	291

Cuadro 3: Resultados del Algoritmo BFS

2.2. Implementación de heurísticas

Los resultados de A* con la heurística de Manhattan son:

Prueba	Tiempo de Ejecución (ms)	Número de Expansiones	Largo del Camino	Costo Total del Camino
A* 1	0.1794000000	56	19	19
A* 2	0.1403000000	45	25	20.5
A* 3	0.9209000000	378	93	93
A* 4	0.9506000000	378	97	102.5
A* 5	2.0774000000	563	94	94
A* 6	1.8590000000	545	110	103.5
A* 7	6.3295000000	1606	321	284.5
A* 8	18.6506000000	1811	347	260.0

Cuadro 4: Resultados del Algoritmo A* utilizando la heurística de Manhattan

Por otro lado, los resultados de A* con la heurística Euclidiana son:

Prueba	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
A* 1	0.4060000000	59	19	19
A* 2	0.3596000000	48	25	20.5
A* 3	2.5070000000	380	93	93
A* 4	2.5857000000	378	97	102.5
A* 5	3.9935000000	582	94	94
A* 6	5.1637000000	556	110	103.5
A* 7	14.6340000000	1611	321	284.5
A* 8	16.2128000000	1817	347	260.0

Cuadro 5: Resultados del Algoritmo A* utilizando la heurística euclidiana

Se utiliza la mitad del valor original de las heurísticas en algoritmos como A* asegura que la heurística sea admisible, es decir, que nunca sobreestime el costo real para llegar al objetivo. Esto garantiza que A* encontrará el camino óptimo, ya que la estimación conservadora (h(n)) hace que el algoritmo dependa más del costo real acumulado (g(n)), resultando en una búsqueda más exhaustiva pero confiable en términos de optimalidad. Sin embargo, esta estrategia puede incrementar el número de nodos expandidos y el tiempo de búsqueda.

Por otro lado, usar el valor completo de las heurísticas puede hacer que A* sea más eficiente al priorizar nodos más cercanos al objetivo, reduciendo el número de expansiones y el tiempo de ejecución. Sin embargo, esto aumenta el riesgo de que la heurística no sea admisible, lo que puede llevar a soluciones subóptimas. El balance entre eficiencia y optimalidad debe ajustarse según las necesidades específicas del problema, ya que una heurística más alta puede sesgar la búsqueda excesivamente hacia el objetivo, ignorando caminos potencialmente óptimos.

2.3. Implementación de Recursive Best-First Search (RBFS)

Los resultados obtenidos por RBFS son los siguientes: [1]

Prueba	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
test 1	1.3126000000	19	19	19
test 2	1.5543000000	25	25	20.5
test 3	186.7181000000	107	107	107
test 4	183.6332000000	107	107	107.0
test 5	260.7066000000	94	94	94
test 6	273.8994000000	110	110	103.5
test 7	-	-	-	-
test 8	25378.6640000000	231	231	299

Cuadro 6: Resultados del Algoritmo RBFS

2.4. Comparación

Resultados Test 1:

Algoritmo	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
DFS	0.2010000000	41	39	39
Inverted DFS	0.2173000000	55	25	25
BFS	0.2337000000	58	19	19
RBFS	2.1967000000	19	19	19
A* Manhattan Heuristic	0.2679000000	56	19	19
A* Euclidian Heuristic	0.2771000000	59	19	19

Cuadro 7: Comparación de Algoritmos - Test 1

Resultados Test 2:

Algoritmo	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
DFS	0.1963000000	41	39	49.5
Inverted DFS	0.1932000000	55	25	20.5
BFS	0.2266000000	58	19	22
RBFS	2.0102000000	25	25	20.5
A* Manhattan Heuristic	0.1888000000	45	25	20.5
A* Euclidian Heuristic	0.2122000000	48	25	20.5

Cuadro 8: Comparación de Algoritmos - Test 2

Resultados Test 3:

Algoritmo	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
DFS	1.0023000000	197	155	155
Inverted DFS	2.8567000000	330	107	107
BFS	2.9263000000	381	93	93
RBFS	340.8365000000	107	107	107
A* Manhattan Heuristic	1.5648000000	378	93	93
A* Euclidian Heuristic	1.5147000000	380	93	93

Cuadro 9: Comparación de Algoritmos - Test 3

Resultados Test 4:

Algoritmo	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
DFS	1.1821000000	197	155	163.5
Inverted DFS	2.0290000000	330	107	107.0
BFS	2.2667000000	381	93	106
RBFS	257.4659000000	107	107	107.0
A* Manhattan Heuristic	1.5556000000	378	97	102.5
A* Euclidian Heuristic	1.5091000000	378	97	102.5

Cuadro 10: Comparación de Algoritmos - Test 4

Resultados Test 5:

Algoritmo	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
DFS	3.8397000000	360	124	124
Inverted DFS	3.2743000000	330	174	174
BFS	11.9877000000	628	94	94
RBFS	433.1879000000	94	94	94
A* Manhattan Heuristic	3.0179000000	563	94	94
A* Euclidian Heuristic	3.1272000000	582	94	94

Cuadro 11: Comparación de Algoritmos - Test 5

Resultados Test 6:

Algoritmo	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
DFS	5.8423000000	360	124	144.5
Inverted DFS	10.7948000000	330	174	196.0
BFS	26.0287000000	628	94	109.5
RBFS	327.7467000000	110	110	103.5
A* Manhattan Heuristic	3.4650000000	545	110	103.5
A* Euclidian Heuristic	2.8809000000	556	110	103.5

Cuadro 12: Comparación de Algoritmos - Test 6

Resultados Test 7:

Algoritmo	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
DFS	89.6735000000	1696	433	481.0
Inverted DFS	70.5688000000	1245	661	779.5
BFS	93.1907000000	1792	261	353
RBFS	-	-	-	-
A* Manhattan Heuristic	11.7520000000	1606	321	284.5
A* Euclidian Heuristic	12.2431000000	1611	321	284.5

Cuadro 13: Comparación de Algoritmos - Test 7

Resultados Test 8:

Algoritmo	**Tiempo de Ejecución (ms)**	**Número de Expansiones**	**Largo del Camino**	**Costo Total del Camino**
DFS	73.0796000000	1668	343	414.0
Inverted DFS	25.2121000000	914	391	415.0
BFS	69.2516000000	1597	223	291
RBFS	36464.1848000000	231	231	299
A* Manhattan Heuristic	24.2954000000	1811	347	260.0
A* Euclidian Heuristic	18.6767000000	1817	347	260.0

Cuadro 14: Comparación de Algoritmos - Test 8

- **Test 1**:
  - Mejor tiempo de ejecución: DFS (0.201 ms)
  - Menor número de expansiones: RBFS (19)
- **Test 2**:
  - Mejor tiempo de ejecución: A* Manhattan Heuristic (0.1888 ms)
  - Menor número de expansiones: A* Manhattan Heuristic (45)
- **Test 3**:
  - Mejor tiempo de ejecución: A* Euclidian Heuristic (1.5147 ms)
  - Menor número de expansiones: DFS (197)
- **Test 4**:
  - Mejor tiempo de ejecución: A* Euclidian Heuristic (1.5091 ms)
  - Menor número de expansiones: DFS (197)
- **Test 5**:
  - Mejor tiempo de ejecución: Inverted DFS (3.2743 ms)
  - Menor número de expansiones: RBFS (94)
- **Test 6**:
  - Mejor tiempo de ejecución: A* Euclidian Heuristic (2.8809 ms)
  - Menor número de expansiones: Inverted DFS (330)
- **Test 7**:
  - Mejor tiempo de ejecución: Inverted DFS (70.5688 ms)
  - Menor número de expansiones: A* Manhattan Heuristic (1606)
- **Test 8**:
  - Mejor tiempo de ejecución: A* Euclidian Heuristic (18.6767 ms)
  - Menor número de expansiones: RBFS (231)

Se puede ver que A* Euclidian Heuristic y Inverted DFS destacan por su eficiencia en tiempo de ejecución. Por otro lado RBFS y DFS son mejores si consideramos un menor número de expansiones en varios tests.

3. DDCheckers

3.1. Actividad 1: Monte Carlo Tree Search

Resultados obtenidos:

Depth	Minimax Wins	MCTS Wins	Ties	Avg White Time (s)	Avg Black Time (s)
1	6	0	4	0.004774	3.045583
2	4	0	6	0.037439	3.021178
3	0	0	10	0.031274	3.203200

Cuadro 15: Resultados de partidas entre Minimax y MCTS con diferentes profundidades

Los resultados muestran que el rendimiento del algoritmo Minimax varía con la profundidad de búsqueda. Teóricamente, aumentar la profundidad de búsqueda en Minimax mejora la calidad de las decisiones al prever más movimientos futuros, pero incrementa el tiempo de cómputo. Experimentalmente, esto se traduce en más empates a mayores profundidades, ya que Minimax puede prever más movimientos pero a costa de un mayor tiempo de cálculo. MCTS, aunque toma más tiempo por movimiento, no mostró una mejora significativa en comparación con Minimax.

3.2. Actividad 2: Profundidades Distintas

Resultados obtenidos:

White Depth	Black Depth	White Wins	Black Wins	Ties	Avg White Time (s)	Avg Black Time (s)
1	3	3	0	0	0.006816	0.057607
3	5	0	0	3	0.044286	0.048436
5	1	0	0	3	0.052196	0.006162

Cuadro 16: Resultados de partidas entre Minimax con diferentes profundidades

Análisis del Cambio de Profundidad en el Algoritmo Minimax

Teóricamente:

Aumentar la profundidad de búsqueda en Minimax permite prever más movimientos futuros, mejorando la toma de decisiones estratégicas. Sin embargo, esto incrementa exponencialmente el número de nodos evaluados, aumentando significativamente el tiempo de cómputo. De igual forma, habría que hacer un análisis más profundo sobre la complejidad del juego debido a que parece ser que el algoritmo que empieza (blanco) tiene mucha ventaja sobre el otro.

Experimentalmente:

Profundidad 1 vs 3: Minimax a profundidad 1 ganó 3 partidas contra profundidad 3, con tiempos promedio de 0.006816 y 0.057607 segundos por movimiento, respectivamente.

Profundidad 3 vs 5: Todas las partidas terminaron en empate, con tiempos promedio de 0.044286 segundos

(profundidad 3) y 0.048436 segundos (profundidad 5).

Profundidad 5 vs 1: Nuevamente, todas las partidas terminaron en empate, con tiempos promedio de 0.052196 segundos (profundidad 5) y 0.006162 segundos (profundidad 1).

Mayor profundidad en Minimax mejora la calidad de decisiones estratégicas al prever más movimientos, pero incrementa considerablemente el tiempo de cómputo, lo que puede resultar en más empates debido a una mejor anticipación del oponente. Este balance entre precisión y eficiencia es crucial en la implementación práctica del algoritmo.

3.3. Actividad 3: Poda Alfa-Beta

La poda alfa-beta es una técnica de optimización para el algoritmo Minimax que reduce el número de nodos que se evalúan en el árbol de búsqueda. La idea básica es eliminar ramas del árbol que no pueden afectar el resultado final, mejorando así la eficiencia del algoritmo sin cambiar su resultado. [2]

Donde:

- Alfa: El mejor valor que el jugador maximizador (Max) puede asegurar hasta ahora.
- Beta: El mejor valor que el jugador minimizador (Min) puede asegurar hasta ahora.

La poda alfa-beta es de la siguiente manera:

1. Durante la búsqueda, se mantienen los valores de alfa y beta.
2. En cada nodo, se actualizan estos valores según el valor actual del nodo.
3. Si en algún momento se detecta que el valor de un nodo es peor que la mejor opción que el jugador puede asegurar, se poda (es decir, no se evalúan) las ramas subsecuentes.

Al repetir el experimento con poda alfa-beta, podemos comparar los resultados con los obtenidos previamente sin poda. La poda alfa-beta debería mejorar la eficiencia del algoritmo, reduciendo el tiempo promedio de búsqueda por movimiento, sin afectar el resultado final.

Los resultados obtenidos al hacer nuevamente la Actividad 2:

White Depth	Black Depth	White Wins	Black Wins	Ties	Avg White Time (s)	Avg Black Time (s)
1	3	3	0	0	0.006633	0.052701
3	5	0	0	3	0.042054	0.045811
5	1	0	0	3	0.045060	0.005649

Cuadro 17: Resultados de partidas entre agentes Minimax con diferentes profundidades utilizando poda alfa-beta

- Profundidad 1 vs 3: Sin poda alfa-beta, el tiempo promedio de búsqueda para White es 0.004929 s y para Black es 0.042198 s. Con poda alfa-beta, los tiempos son 0.009644 s y 0.069762 s respectivamente. La poda alfa-beta no mejora el tiempo en esta configuración.

- Profundidad 3 vs 5: Sin poda alfa-beta, el tiempo promedio de búsqueda para White es 0.031527 s y para Black es 0.035025 s. Con poda alfa-beta, los tiempos son 0.042224 s y 0.045683 s. Nuevamente, la poda alfa-beta no reduce los tiempos de manera efectiva.
- Profundidad 5 vs 1: Sin poda alfa-beta, el tiempo promedio de búsqueda para White es 0.044770 s y para Black es 0.005557 s. Con poda alfa-beta, los tiempos son 0.048103 s y 0.005828 s. La diferencia es mínima, pero la poda alfa-beta no muestra una mejora significativa.

Los resultados experimentales demuestran que la poda alfa-beta contribuye a mejorar la eficiencia del algoritmo Minimax, especialmente en profundidades de búsqueda mayores y más complejas (caso de profundidad 5 por ejemplo). Vemos como de igual manera en todos los casos, ya sea jugador blanco o negro y profundidades 1, 3 y 5 se bajó el tiempo promedio. Sin embargo, la diferencia en tiempos no es siempre significativa, probablemente por la variabilidad en la complejidad de los juegos. Teóricamente si el juego fuera más complejo se podrian reducir de mayor manera los promedios.

Referencias

[1] Matthew Hatem, Scott Kiesel, Wheeler Ruml. *Recursive Best-First Search with Bounded Overhead*. University of New Hampshire. https://www.cs.unh.edu/~ruml/papers/rbfscr-aaai-15.pdf

[2] "¿Qué es la poda alfa-beta en inteligencia artificial?", *Ichi.pro*, 2021. [Online]. Available: https://ichi.pro/es/que-es-la-poda-alfa-beta-en-inteligencia-artificial-198457106108314.