Algoritmos
Time-Bounded A*
Real-Time Bounded A*
Time-Bounded Adaptive A*
Learning Real-Time A*
Real-Time Adaptive A*

Ulisses Rodrigues Afonseca [1]
Marcos Fagundes Caetano [1]

Departamento de Computação
Universidade de Brasília

Conteúdo

Conteúdo

Algoritmo A*

A*

Busca heurística que avalia os nós a partir da combinação do custo de chegar ao nó e o custo do nó ao objetivo.

- Apresentado por Hart el all em 1968 no artigo *"A formal basis for the heuristic determination of minimum cost paths"*;
- É uma busca *best-first search*;
- É ótimo (resultado);

Algoritmo A*

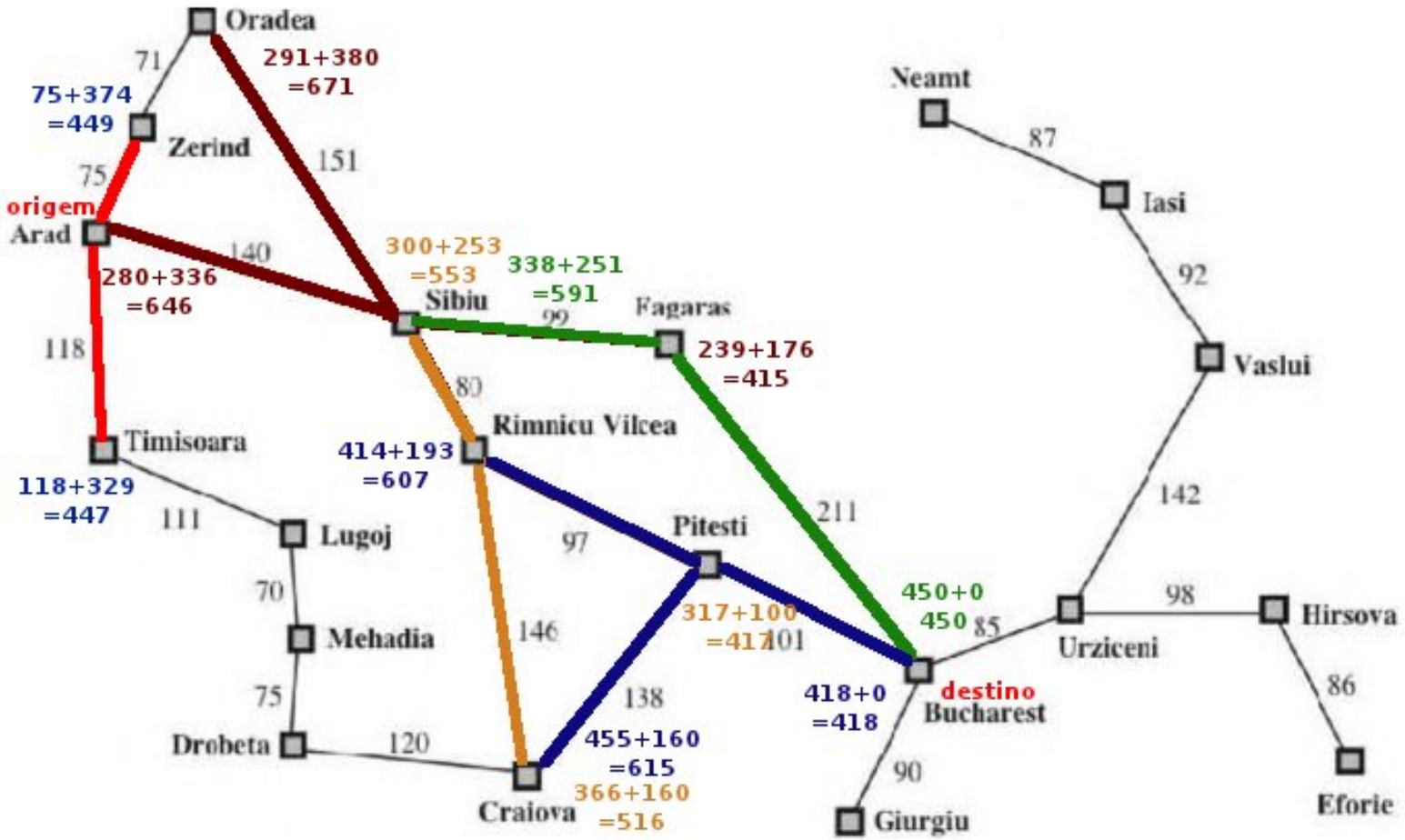
Avaliação de custo

$$f(n) = g(n) + h(n)$$

Onde:

- $f(n)$ é o custo estimado da solução mais barata através de n;
- $g(n)$ fornece o custo do caminho do nó inicial ao nó n;
- $h(n)$ é o custo estimado do caminho mais barato de r até o objetivo.

Exemplo de aplicação do Algoritmo A*



Limitações do Algoritmo A*

O algoritmo A* realiza a busca. Posteriormente o caminho é utilizado para realizar a navegação. O tempo de execução do algoritmo depende do do tamanho da árvore de busca. Assim,

- Não pode ser utilizado em ambientes de tempo real;
- Não permite que a busca e a navegação sejam realizados paralelamente;
- Não permite que o custo varie com o tempo;
- O ambiente deve ser conhecido totalmente;
- Não é possível utilizar valores localizados.

Variações do Algoritmo A*

Tabela: Variações do Algoritmo A*

Algoritmo	Real-time	Ambiente desconhecido	Multiplos destinos	h variável	h decrescente
Adaptative A*	não	não	sim	sim	não
Real-time AA*	sim	não	sim	sim	não
D*	sim	sim	não	não	não
Focused D*	sim	sim	não	sim	?
Path-Adaptive A*	não	sim	não	sim	não
Tree-Adaptive A*	sim	sim	não	sim	?
Lifelong Planning A*	?	sim	não	sim	?
Anytime Dynamic A*	sim	sim	?	?	?

Conteúdo

Game Time Model

Características:

- Tempo é particionado em intervalos uniformes
- Agente pode executar um movimento durante cada slot de tempo
- Busca e Movimentação podem ser realizados em paralelo

Objetivo: mover o agente da posição inicial para o objetivo no menor número de intervalos

Game Time Model

Jogos

- Ciclos de jogos de alguns milisegundos
- Cada personagem executa um movimento ao final de cada ciclo
- Aparência de continuidade

Game Time Model e Algoritmo A*

Algoritmo A* não é adequado para este modelo, pois necessita de um tempo longo para realizar busca e depois movimentação, i.e:

- Necessário alguns intervalos de tempo para encontrar o caminho de menor custo;
- Implica em um atraso para o início do movimento;
- Aumenta o tempo para chegar ao destino;
- Sem paralelismo na busca e movimento.

Conteúdo

Notação

Busca:

- (S, A): grafo não orientado onde S são estados e A são arestas
- $Succ(s) = \{t|(s,t) \in A\}$: sucessores de $s \in S$ (vizinhos)
- $c : A \mapsto \mathbb{R}^+$: função de custo que associa um custo a cada aresta
- $s_{start} \in S$: estado inicial
- $s_{goal} \in S$: estado objetivo

Utilização da Distância de Manhattan

Notação

A*:

- OPEN: lista de prioridades de estados
- $g(s)$: custo de s_{start} até s atual
- $h(s)$: valor aproximado do custo de s até s_{goal}
- $f(s) = g(s) + h(s)$
- $parent(s)$: aponta para o pai de s na áreade de busca
- s_{root}: raiz da árvore de busca ou s_{start}
- $H(s)$: user-provided $h - value$
- $h - value$ consistende bi-implica que $h(s_{goal}) = 0$ e $h(s) \leq c(s,t) + h(t)$ para todos os estados $s \in S$ e $t \in Succ(s)$

Idéia

- Tempo real: realiza uma ou várias buscas A* limitadas por intervalo de tempo;
- Uso da grade com quatro vizinhos e células bloqueadas e desbloqueadas;
- Ao final de cada intervalo de tempo, um movimento é executado de $s_{current}$ para $s_{best} \in OPEN$ com o menor valor de f.

TBA*: Time-Bounded A*

Algorithm 1 TBA*

```
1: procedure InitializeState(s)
2: if search(s) = 0 then
3:     h(s) := H(s);
4:     g(s) := ∞;
5: search(s) := searchnumber;

6: procedure InitializeSearch()
7: searchnumber := searchnumber + 1;
8: s_root := s_current;
9: InitializeState(s_root);
10: g(s_root) := 0;
11: OPEN := ∅;
12: Insert s_root into OPEN;
13: InitializeState(s_goal);
14: goalFoundFlag := 0;
```

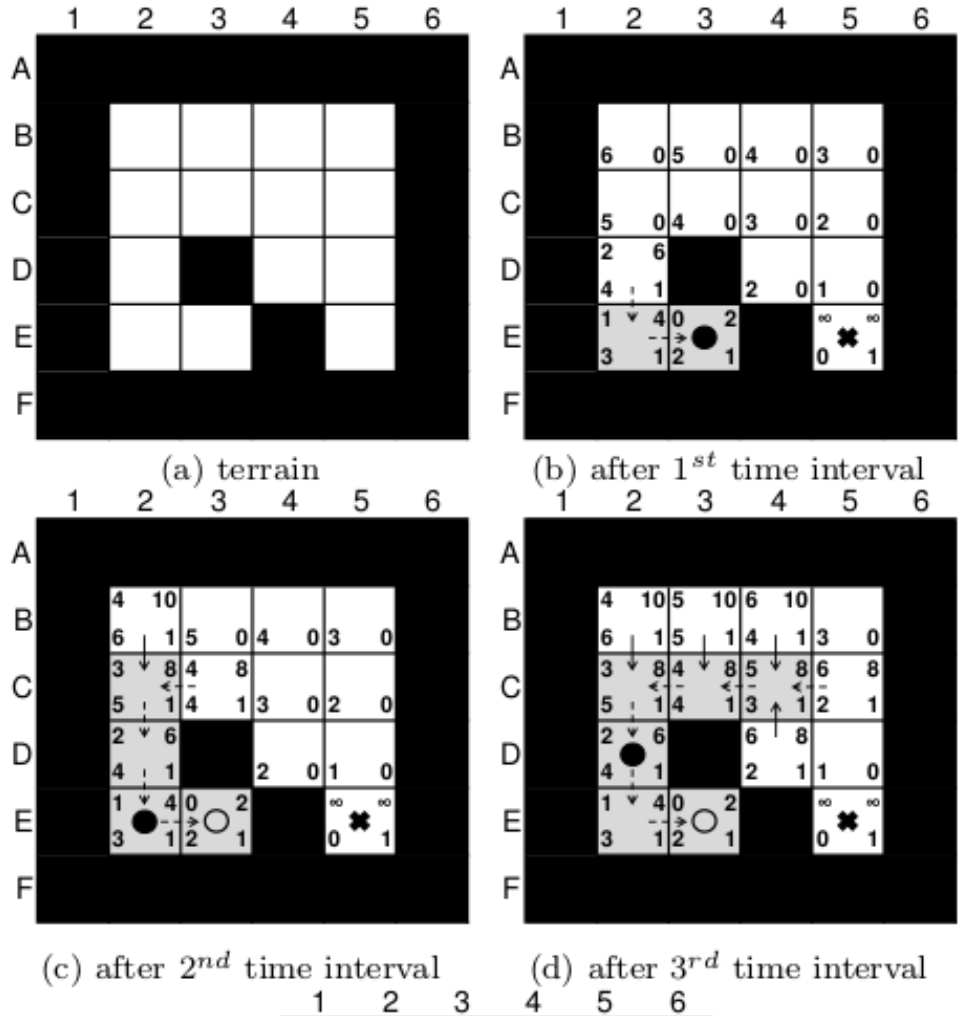
TBA*: Time-Bounded A*

```
15: function Search()
16: expansions := 0;
17: while OPEN ≠ ∅ AND expansions < k AND
18:           g(s_goal) + h(s_goal) > min_{t∈OPEN}(g(t) + h(t)) do
19:     s := arg min_{t∈OPEN}(g(t) + h(t));
20:     Remove s from OPEN;
21:     for all t ∈ Succ(s) do
22:         InitializeState(t)
23:         if g(t) > g(s) + c(s, t) then
24:             g(t) := g(s) + c(s, t);
25:             parent(t) := s;
26:             Insert t into OPEN;
27:     expansions := expansions + 1;
28: if OPEN = ∅ then
29:     return false;
30: s_best := arg min_{t∈OPEN}(g(t) + h(t));
31: if s_best = s_goal then
32:     goalFoundFlag := 1;
33: path := path from s_root to s_best;
34: return true;
```

TBA*: Time-Bounded A*

```
35: function MoveToGoal()
36: s_current := s_start;
37: InitializeSearch();
38: while s_current ≠ s_goal do
39:     print("a new time interval starts now");
40:     if goalFoundFlag = 0 then
41:         if Search() = false then
42:             return false;
43:     if s_current is on path then
44:         s_current := state after s_current on path;
45:     else
46:         s_current := parent(s_current);
47:     Execute movement to s_current;
48: return true;

49: procedure Main()
50: searchnumber := 0;
51: for all s ∈ S do
52:     search(s) := 0;
53: if MoveToGoal() = true then
54:     print("the agent is now at the goal state");
55: else
56:     print("the agent cannot reach the goal state");
```

TBA*: Time-Bounded A*



(a) terrain (b) after 1^{st} time interval

(c) after 2^{nd} time interval (d) after 3^{rd} time interval

Conteúdo

Idéia geral

- Agente conhece apenas origem e destino e não conhece os bloqueados;
- Sempre olha o estado de seus vizinhos (C,B,D,E);
- Para as células bloqueadas: a) ele adiciona 1 para todos os caminhos dele para o vizinho (não bloqueado) e b) infinito das células que chegam ao bloqueado;
- Executa o TBA* sempre que os valores mudam;

RTBA*: Real Time-Bounded A*

Algorithm 2 RTBA*

```
1: procedure InitializeState(s)
2: if search(s) = 0 then
3:     h(s) := H(s);
4:     g(s) := ∞;
5: else if search(s) ≠ searchnumber then
6:     g(s) := ∞;
7: search(s) := searchnumber;

8: procedure StartNewSearch?()
9: if edge costs on path have increased
10:            since the last call to InitializeSearch then
11:    InitializeSearch();
12:    path := empty;
```

RTBA*: Real Time-Bounded A*

```
13: function MoveToGoal()
14: s_current := s_start;
15: InitializeSearch();
16: Observe edge cost increases (if any);
17: while s_current ≠ s_goal do
18:     print("a new time interval starts now");
19:     if goalFoundFlag = 0 then
20:         if Search() = false then
21:             return false;
22:         StartNewSearch?();
23:     if path ≠ empty then
24:         if s_current is on path then
25:             s_current := state after s_current on path;
26:         else
27:             s_current := parent(s_current);
28:         Execute movement to s_current;
29:         Observe edge cost increases (if any);
30:         StartNewSearch?();
31: return true;
```

Conteúdo

Motivação e Proposta

Motivação:

- No RTBA, a cada nova busca, todas as informações de buscas anteriores são perdidas.
- Porém, algoritmo de tempo real atualizam o valor de h.

Solução proposta:

- TBAA* trabalha de forma semelhanta ao RTBA* porém atualia o $h - value$ das células de forma semelhante ao que acontece com AA* e RTAA*.
- A cada vez que TBAA* inicia um nova busca, ele ataliza o $h - value$ de todos os estados gerados por buscas anteriores atribuindo $h(s) := f(sb_{best}) - g(s)$
- Atualização acontece quando o h-value é necessário pela busca atual pela primeira vez (para evitar a computação do que não será necessário posteriormente).

TBAA*: Time-Bounded Adaptive A*

Algorithm 3 TBAA*

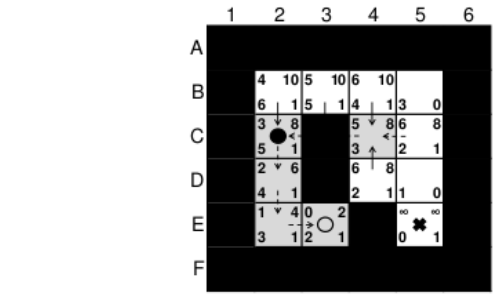
```
1: procedure InitializeState(s)
2: if search(s) = 0 then
3:     h(s) := H(s);
4:     g(s) := ∞;
5: else if search(s) ≠ searchnumber then
6:     if h(s) < pathcost(search(s)) − g(s) then
7:         h(s) := pathcost(search(s)) − g(s);
8:     g(s) := ∞;
9: search(s) := searchnumber;

10: procedure StartNewSearch?()
11: if edge costs on path have increased
12:             since the last call to InitializeSearch then
13:     pathcost(searchnumber) := min_{s∈OPEN}(g(s) + h(s));
14:     InitializeSearch();
15:     path := empty;
```

TBAA*: Time-Bounded Adaptive A*

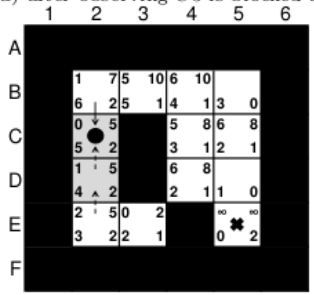
Algorithm 4 Repeated Runs of RTBA* and TBAA*

```
1:  procedure Main()
2:  searchnumber := 0;
3:  for all s ∈ S do
4:      search(s) := 0;
5:  repeat
6:      if MoveToGoal() = false then
7:          print("the agent cannot reach the goal state");
8:  until s_root = s_start;
9:  print("cost-minimal path from start state to goal state: ");
10: print(path);
```

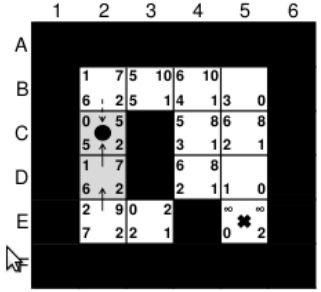
TBAA*: Time-Bounded Adaptive A*



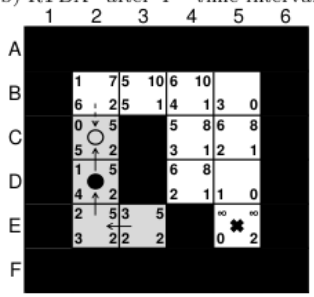
(a) after observing $C3$ is blocked after 3^{rd} time interval and movement



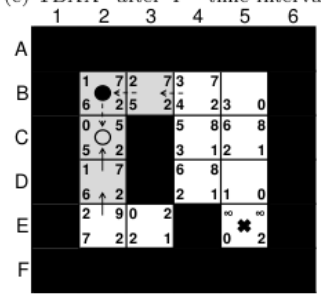
(b) RTBA* after 4^{th} time interval



(c) TBAA* after 4^{th} time interval



(d) RTBA* after 5^{th} time interval



(e) TBAA* after 5^{th} time interval

TBAA*: Time-Bounded Adaptive A*

Table 1: Known Terrain

	RTAA*		daRTAA*		TBA*		A*	
Length of Time Intervals (ms)	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments
0.3	2,193	2,192	1,729	1,728	**568**	567	584	545
0.6	1,541	1,540	1,303	1,302	**556**	555	565	545
0.9	1,362	1,361	1,151	1,150	**552**	551	558	545
1.2	1,196	1,195	1,057	1,056	**550**	549	555	545
1.5	1,087	1,086	970	969	**549**	548	553	545

Table 2: Initially Completely Unknown Terrain

	RTAA*		daRTAA*		RTBA*		TBAA*		Repeated A*		Adaptive A*		D* Lite	
Length of Time Intervals (ms)	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments
0.3	3,245	3,244	2,879	2,878	4,613	4,604	2,290	2,286	7,155	2,004	3,230	2,010	**2,203**	2,027
0.6	2,598	2,597	2,472	2,471	3,368	3,360	2,147	2,144	4,487	2,004	2,572	2,010	**2,090**	2,027
0.9	2,451	2,450	2,418	2,417	2,918	2,910	2,101	2,099	3,611	2,004	2,361	2,010	**2,062**	2,027
1.2	2,310	2,309	2,305	2,304	2,695	2,688	2,086	2,083	3,178	2,004	2,260	2,010	**2,051**	2,027
1.5	2,281	2,280	2,272	2,271	2,560	2,553	2,070	2,068	2,920	2,004	2,202	2,010	**2,045**	2,027

Table 3: Initially Partially Unknown Terrain

	RTAA*		daRTAA*		RTBA*		TBAA*		Repeated A*		Adaptive A*		D* Lite	
Length of Time Intervals (ms)	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments	# Time Intervals	# Move-ments
0.3	2,694	2,693	2,460	2,459	2,734	2,730	**1,505**	1,504	6,324	1,409	2,430	1,399	1,659	1,418
0.6	2,039	2,038	1,863	1,862	2,037	2,034	**1,442**	1,441	3,812	1,409	1,875	1,399	1,532	1,418
0.9	1,840	1,839	1,779	1,778	1,860	1,857	**1,431**	1,430	2,979	1,409	1,695	1,399	1,490	1,418
1.2	1,707	1,706	1,643	1,642	1,726	1,724	**1,421**	1,420	2,564	1,409	1,608	1,399	1,470	1,418
1.5	1,620	1,619	1,642	1,641	1,668	1,666	**1,415**	1,414	2,316	1,409	1,556	1,399	1,458	1,418

Conteúdo

Motivação

- Algoritmo aplicado em situações em que o tempo gasto em processamento é crítico;
  - Aplicações de tempo real;
- Jogos: diversos componentes movimentam-se ao mesmo tempo;

O que muda no algoritmo?

- Assumindo que s é um estado que foi expandido durante a pesquisa;
- Uma estimativa razoável da distância objetivo ($gd[s]$) pode ser calculada:
  - A distância a partir do estado inicial S_{curr} para qualquer estado de destino, passando pelo estado s é igual a distância a partir do estado de origem S_{curr} até o estado s mais a distância objetivo $gd[s]$, partindo do estado s;
- "**Por trivialidade**":

$$g[s] + gd[s] \geq gd[s_{curr}]$$
$$gd[s] \geq gd[s_{curr}] - g[s]$$
$$gd[s] \geq f[s^*] - g[s]$$

Algoritmo RTAA

procedure realtime_adaptive_astar():
{01} while ($s_{\text{curr}} \notin GOAL$) do
{02} $lookahead :=$ any desired integer greater than zero;
{03} astar();
{04} if $\bar{s} = FAILURE$ then
{05} return $FAILURE$;
{06} for all $s \in CLOSED$ do
{07} $h[s] := g[\bar{s}] + h[\bar{s}] - g[s]$;
{08} $movements :=$ any desired integer greater than zero;
{09} while ($s_{\text{curr}} \neq \bar{s}$ AND $movements > 0$) do
{10} $a :=$ the action in $A(s_{\text{curr}})$ on the cost-minimal trajectory from s_{curr} to $\bar{s}$;
{11} $s_{\text{curr}} := succ(s_{\text{curr}}, a)$;
{12} $movements := movements - 1$;
{13} for any desired number of times (including zero) do
{14} increase any desired $c[s, a]$ where $s \in S$ and $a \in A(s)$;
{15} if any increased $c[s, a]$ is on the cost-minimal trajectory from s_{curr} to $\bar{s}$ then
{16} break;
{17} return $SUCCESS$;

LRTA e RTAA, o que muda?

- LRTA significa *Learning Real-Time A**.
- **Diferença básica**: maneira como é feita a atualização da heurística após uma consulta;
- LRTA substitui cada estado expandido com a soma da distância a partir do estado gerado porem não expandido s e a heurística do estado s, minimizando sobre todos os estados gerados porem não expandidos.

$$h'[s] = \min_{a \in A(s)} (c[s,a] + h'[succ(s,a)]) \quad (1)$$

Algoritmo RTAA: Informações Importantes

Algumas definições importantes para o entendimento do funcionamento do algoritmo (exemplo):

- os seguintes valores distribuídos:
  - **g[s]**: canto-superior-esquerdo;
  - **f[s]**: canto-superior-direito;
  - **h[s]**: canto-inferior-esquerdo (antes de aplicar a heurística);
  - **h'[s]**: canto-interior-direito (depois de aplicar a heurística);
- os quadrados em preto encontram-se bloqueados para o agente;

Exemplo - Lookahead = infinito

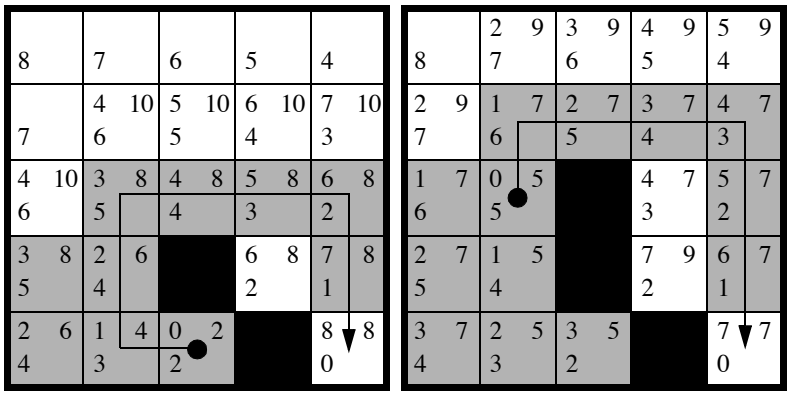


Figure 4: Forward A* Searches

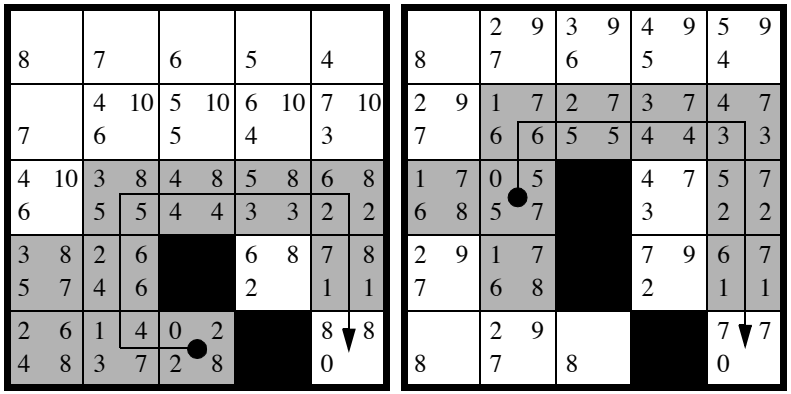


Figure 6: LRTA* with *lookahead* $= \infty$

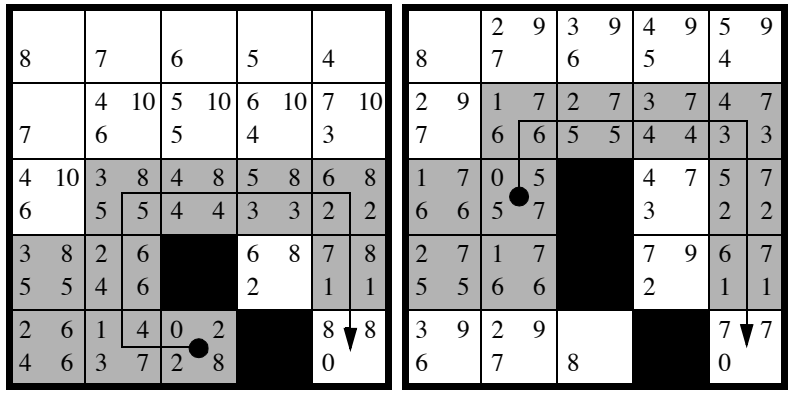
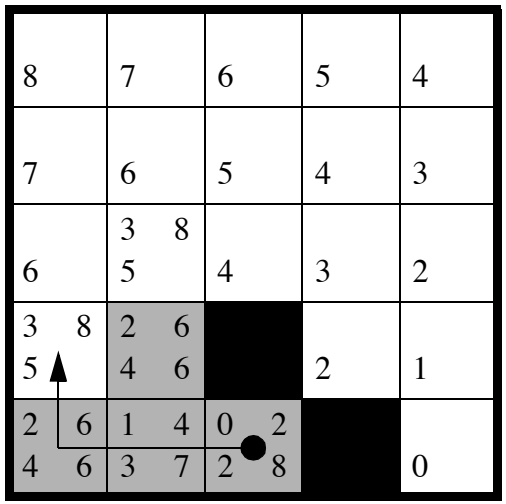
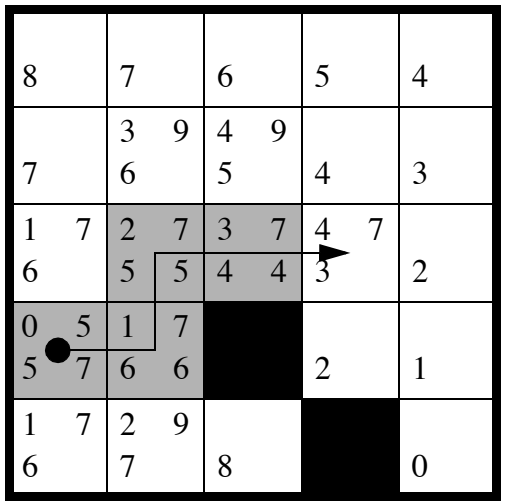
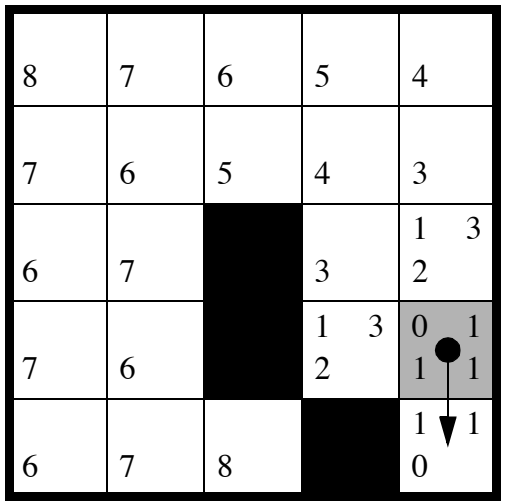


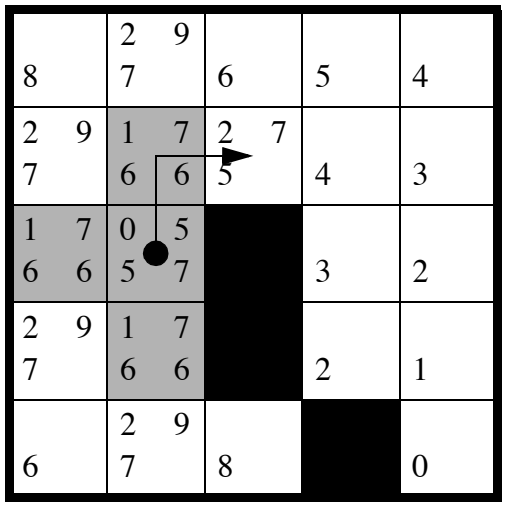
Figure 5: RTAA* with *lookahead* $= \infty$

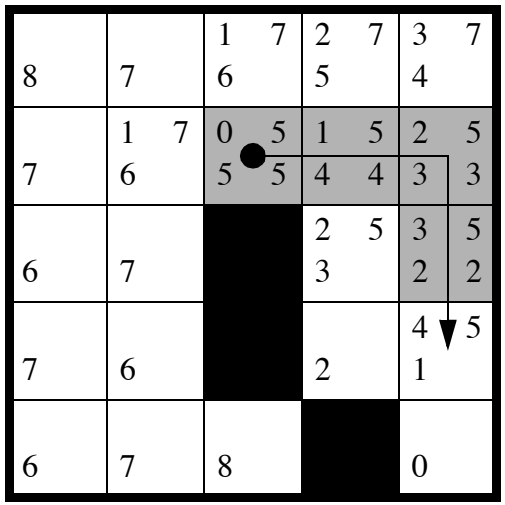
Exemplo - RTAA* - Lookahead = 4



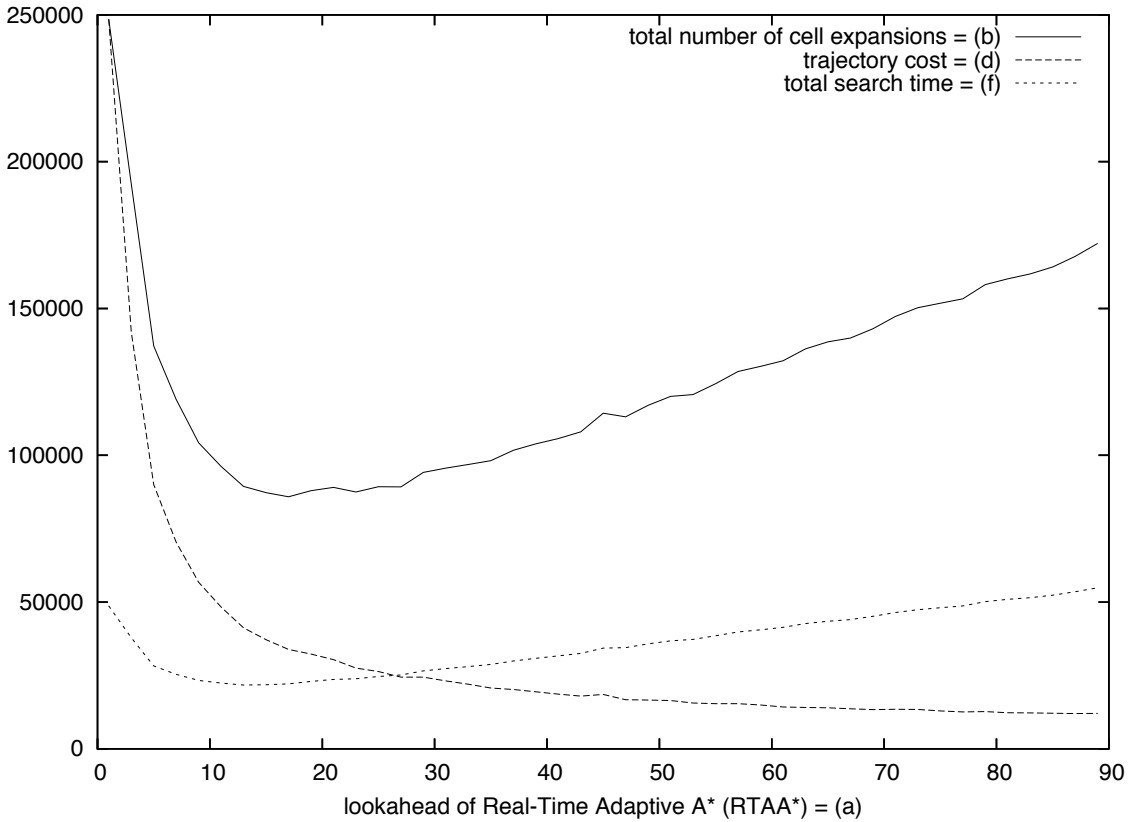








Performance - RTAA*



Conteúdo

Conclusão

- Uso de heurística possibilida o desenvolvimento de novas ferramentas que se adaptam facilmente ao problema
- O algoritmo A* é limitado em vários sentidos, variações resolvem problemas isolados
- Para cada novo problema, o A* pode ser facilmente adaptado
- A* é de 1968 é aparentemente sempre surgem variações interessantes (ver soluções atuais).