PLANEJAMENTO DE CAMINHOS EM TEMPO REAL APLICADO A JOGOS

THIAGO SILVA VILELA

PLANEJAMENTO DE CAMINHOS EM TEMPO REAL APLICADO A JOGOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais - Departamento de Ciência da Computação como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: LUIZ CHAIMOWICZ

Belo Horizonte

Março de 2014

© 2014, Thiago Silva Vilela.
Todos os direitos reservados.

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Vilela, Thiago Silva

V699p Planejamento de Caminhos em Tempo Real Aplicado a Jogos / Thiago Silva Vilela. — Belo Horizonte, 2014
xviii, 91 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação
Orientador: Luiz Chaimowicz

1. Computação - Teses. 2. Inteligência artificial. 3. Jogos por computador. I. Orientador. II. Título.

CDU 519.6*82(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Planejamento de caminhos em tempo real aplicado a jogos digitais

THIAGO SILVA VILELA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ CHAIMOWICZ - Orientador
Departamento de Ciência da Computação - UFMG

PROF. RENATO ANTÔNIO CELSO FERREIRA
Departamento de Ciência da Computação - UFMG

PROF. THIAGO FERREIRA DE NORONHA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 28 de março de 2014.

Resumo

Nos últimos anos, jogos digitais têm se tornando uma área maior e mais importante na indústria de computação e, com isso, a Inteligência Artificial (IA) também tem ganhado destaque no desenvolvimento destes jogos.

Planejamento de caminhos é, dentre todos os problemas envolvidos na inteligência artificial de um jogo, provavelmente o mais comum. Ele existe em jogos de todo gênero, onde um computador precisa mover entidades.

Planejamento de caminhos em ambientes estáticos e completamente observáveis é uma área já bastante explorada. No entanto, em situações onde o ambiente é parcialmente observável e o agente possui restrições de tempo real para se locomover, o planejamento de caminhos é um problema bem mais difícil. A lista de jogos com tais características é extensa, mas o tipo mais famoso são os jogos RTS (*Real-Time Strategy*).

Muitos estudos têm sido realizados, e vários algoritmos foram propostos para permitir o planejamento de caminhos em ambientes parcialmente observáveis e com restrições de tempo real. No entanto, escolher qual desses algoritmos utilizar pode ser uma tarefa difícil. Diferentes estudos apresentam os algoritmos e resultados de formas divergentes, pois utilizam diferentes conjuntos de testes e formas de avaliação.

Nesse trabalho fazemos uma comparação entre os principais algoritmos existentes para resolver o problema do planejamento de caminhos em ambientes parcialmente observáveis e de tempo real.

Palavras-chave: planejamento de caminhos, jogos, inteligência artificial.

Abstract

Digital games are increasingly becoming a very important area in the computing industry. Thus, game's artifical intelligence (AI) is also growing in importance.

Pathplanning is, among all problems involved in a game's artificial intelligence, probably the most common. We have to deal with it in every game where the computer needs to move things around independently.

When we have static and completely observable environments, pathplanning is relatively an easy problem, which has been widely studied. However, when the agent needs to take decisions fast (move in real time) and the environment is unknown, the problem can be much harder. The list of games with such characteristics is extensive, but the most famous are the Real-Time Strategy (RTS) games.

Many studies have been conducted, and several algorithms have been proposed to allow pathplanning in partially observable environments with real-time constraints. However, choosing which of these algorithms to use can be a difficult task. Different studies show algorithms and results in different ways, using different benchmarks and evaluation metrics.

In this work we make a comparison between the major existing algorithms to solve the pathplanning problem in partially observable environments with real-time constraints.

Keywords: pathplanning, games, artifical intelligence.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

Acrônimo	*Significado*
IA	*Inteligência Artificial*
NPC	*Non-player character*
RTS	*Real-Time Strategy*
A*	*A estrela*
RTA*	*Real-time A**
LRTA*	*Learning real-time A**
D*	*Dynamic A**
LPA*	*Lifelong Planning A**
AA*	*Adaptive A**
LRTA*	*Learning Real-Time A**
LSSLRTA*	*Local Search Space Learning Real-Time A**
RTAA*	*Real-Time Adaptive A**
TBAA*	*Time-Bounded Adaptive A**
PRTA*	*Path Real-Time A**
LSS	*Local Search Space*
LRTA*(k)	*Learning Real-Time A*(k)*
P-LRTA*	*Prioritized Learning Real-Time A**
TBA*	*Time-Bounded A**
RTBA*	*Restarting Time-Bounded A**
kNN LRTA*	*k Nearest Neighbors LRTA**
RIBS	*Real-time Iterative-deepening Best-first Search*
f-LRTA*	*f-cost Learning Real-Time A**

Sumário

Capítulo 1

Introdução

Nesse capítulo discutimos, na Seção 1.1 a motivação para esse trabalho. Em seguida, na Seção 1.2 apresentamos nossos objetivos e, finalmente, na Seção 1.3 explicamos como nosso trabalho está organizado.

1.1 Motivação

O aumento do poder de processamento impactou a maneira como jogos digitais são desenvolvidos. Esse aumento de poder computacional permitiu que os desenvolvedores melhorassem gráficos, sons e *gameplay*. O próprio comportamento dos agentes pôde ser otimizado, permitindo por exemplo a tomada de decisões cada vez mais complexas. A área da Ciência da Computação responsável por modelar o comportamento dos jogadores não jogáveis (*non-player characters*, ou NPCs) é conhecida como Inteligência Artificial (IA). De acordo com Woodcock [2001] a quantidade de CPU reservada para realizar funções de IA cresceu 250% entre 1999 e 2000. Hoje, jogos digitais são uma das áreas onde a Inteligência Artificial (IA) é mais utilizada [Botea et al., 2009]. Essa indústria está em constante crescimento, com as vendas em 2013 alcançado 76 bilhões de dólares, o que marca um crescimento de 13 bilhões de dólares em relação a 2012 [Galarneau, 2014].

Um dos principais problemas da IA na área de jogos é a criação de agentes que possam interagir com o jogador. Esses agentes podem ter diferentes papéis, como oponentes ou aliados do jogador, e devem ser capazes de realizar uma série de tarefas. Uma das tarefas mais básicas e importantes é a navegação. Técnicas usadas para realizar o planejamento de trajetórias estão em jogos de todo gênero, onde um computador precisa mover entidades. Em um jogo de estratégia em tempo real (*Real Time Strategy*, ou RTS), por exemplo, o jogador precisa direcionar suas unidades para certo local

do mapa. As unidades devem ser capazes de se mover pelo mapa e chegar ao local pedido de forma autônoma, sem que o jogador precise traçar para elas toda a rota a ser seguida.

O planejamento de caminhos em ambientes estáticos e completamente observáveis é uma área já bastante explorada. Em jogos, geralmente é utilizado um mapa na forma de *grid*. Esse *grid* pode ser representado como um grafo, e soluções para o planejamento de caminhos podem ser encontradas através do uso de algoritmos de busca em grafos [Millington & Funge, 2009; Bourg & Seemann, 2004]. O algoritmo desse tipo mais utilizado na área de jogos digitais é o A* [Hart et al., 1968].

Algoritmos como o A*, no entanto, nem sempre são adequados. Quando um agente precisa, por exemplo, navegar em um mapa com restrições de visibilidade, algoritmos de busca em grafos tradicionais não podem ser utilizados. Jogos que apresentam ambientes parcialmente observáveis são muito comuns, como os jogos RTS.

Outra situação onde algoritmos de busca em grafos tradicionais não são adequados é na presença de restrições de tempo real. Ao comandar uma tropa para certa posição do mapa em um jogo RTS, por exemplo, o jogador espera que o movimento comece imediatamente. O A* precisa realizar o planejamento completo antes do agente iniciar seu movimento. Em mapas grandes isso pode causar um comportamento indesejado, onde as tropas ficam um tempo paradas antes de seguir o comando do jogador. Além disso, para que a movimentação em jogos seja fluida, decisões de navegação devem ser tomadas a cada ciclo do jogo, que é, geralmente, muito curto [Bulitko et al., 2010]. É desejável, portanto, que os agentes possam planejar durante curtos intervalos de tempo, intercalando planejamento e execução.

Uma possível solução para os problemas citados é o uso de algoritmos de busca heurística de tempo real para realizar o planejamento de trajetórias. Esses algoritmos podem ser utilizados em situações onde o tempo de planejamento é limitado. Além disso, grande parte dos algoritmos desse tipo também podem ser usados em ambientes parcialmente observáveis. Eles usam uma abordagem diferente dos algoritmos de busca em grafo convencionais: ao invés de encontrar uma solução completa antes de começar a executar ações, eles planejam somente os primeiros passos que o agente deve tomar e começam a agir. O algoritmo executa vários ciclos de planejamento-ação até alcançar seu objetivo. Dessa forma é possível definir um limite no tempo de planejamento, de forma que ele seja independente do tamanho do problema.

Apesar de vários algoritmos de busca heurística de tempo real terem sido propostos, não existe ainda um estudo abrangente sobre eles e sua aplicação em jogos digitais.

1.2 Objetivos

A cada nova geração de jogos o problema de planejamento de caminhos se torna mais desafiador, já que os mapas ficam cada vez maiores e mais complexos, além de possuírem mais agentes se locomovendo simultaneamente. Isso tem levado a comunidade científica a estudar e propor algoritmos de busca que tenham um tempo de planejamento por ação independente do tamanho do mapa usado.

Vários algoritmos de busca heurística de tempo real foram propostos e, no entanto, escolher qual desses algoritmos utilizar pode ser uma tarefa difícil. Cada algoritmo possui peculiaridades e características que podem fazer deles mais propícios ou piores em certos tipos de mapas ou sob certas condições. Além disso não existe um estudo centralizado e abrangente desses algoritmos.

O objetivo central desse trabalho é realizar um estudo aprofundado comparando os principais algoritmos de busca heurística de tempo real existentes. Nosso objetivo é avaliar os diferentes algoritmos, verificando seus pontos positivos e negativos, utilizando diversos conjuntos de testes e métricas de comparação. O foco desse trabalho é no problema de planejamento de caminhos. No entanto, como todo algoritmo de busca heurística, os algoritmos estudados e comparados podem ser utilizados na resolução de vários problemas diferentes. Dessa forma, os resultados apresentados podem ser úteis mesmo para um leitor com foco em um tipo diferente de problema.

Portanto, as duas principais contribuições deste trabalho podem ser listadas como:

- **Um estudo detalhado dos principais algoritmos de busca heurística de tempo real**. Os principais algoritmos desse tipo são analisados e explicados. Vários aspectos desses algoritmos são mostrados através de diferentes testes. Mostramos, por exemplo, como os diferentes algoritmos se comportam durante variações na visibilidade de mapas e como a solução obtida por eles melhora ou piora com o aumento ou diminuição das restrições de tempo real.

- **Uma comparação entre algoritmos de busca heurística de tempo real**. Os resultados obtidos pelos diversos algoritmos são comparados, e podemos perceber vantagens e desvantagens de cada um em relação aos demais em diferentes situações. Analisamos, por exemplo, quais algoritmos apresentam melhores e piores resultados em relação à qualidade da solução obtida em mapas com restrição de visibilidade.

1.3 Organização deste trabalho

O restante deste trabalho está organizada da seguinte forma:

- **Capítulo 2, Referencial Teórico**: este capítulo apresenta uma revisão da literatura existente sobre planejamento de caminhos, especialmente o planejamento de caminhos em ambientes parcialmente observáveis e com restrições de tempo real. Além disso, apresentamos também trabalhos relacionados.
- **Capítulo 3, Metodologia**: este capítulo apresenta a metodologia usada neste trabalho. São apresentadas discussões sobre detalhes de implementação, conjuntos de testes e métricas de comparação utilizadas.
- **Capítulo 4, Resultados**: este capítulo apresenta os resultados dos diversos testes utilizados para avaliar e comparar os algoritmos escolhidos.
- **Capítulo 5, Considerações Finais**: este capítulo conclui o trabalho realizado, apresentando algumas discussões e trabalhos futuros.

Capítulo 2

Referencial Teórico

Este capítulo busca introduzir o leitor às diversas pesquisas já realizadas e necessárias na compreensão deste trabalho. Inicialmente, na Seção 2.1, introduzimos o leitor à busca heurística, com foco em seu uso no planejamento de caminhos. Também nessa seção falamos um pouco sobre a busca heurística incremental e a busca heurística de tempo real, que são as classes da busca heurística às quais pertencem os algoritmos implementados em nossa pesquisa. Em seguida, na Seção 2.2, introduzimos o leitor ao modelo computacional que será utilizado no restante do texto. Na seção 2.3 explicamos o A* e o *D* Lite*, dois algoritmos tradicionais de busca heurística que são importantes para este trabalho. Na Seção 2.4 explicamos e discutimos uma série de algoritmos de busca heurística de tempo real. Esses algoritmos são, na maioria dos casos, aqueles que serão utilizados em nossa comparação. Finalmente, na Seção 2.5, apresentamos alguns trabalhos relacionados que comparam algoritmos de busca heurística ou apresentam novas ideias para melhorar algoritmos desse tipo.

2.1 Busca heurística e planejamento de caminhos

O planejamento de caminhos em jogos digitais apresenta um conjunto de características que possibilitam sua representação como um problema de busca. Essas características são [Nilsson, 1998; Russell & Norvig, 2002]: um estado inicial, um estado final, um conjunto de operadores que transformam os estados (função sucessora) e uma função de custo (opcional) que provê o custo de produzir um estado a partir de outro. A aplicação da função sucessora a partir do estado inicial define o espaço de estados [Korf, 1996]. A intenção da busca é transformar o estado inicial no final através da aplicação das regras disponíveis.

A forma como o espaço de estados é explorado pode variar, levando a diferentes

algoritmos. Existem, no entanto, duas categorias básicas de algoritmos de busca, que se diferenciam pelo uso ou não de conhecimento específico do domínio [Rios & Chaimowicz, 2010]. Aqueles que não aplicam esse tipo de informação são conhecidos como estratégias de busca sem informação. Exemplos de algoritmos dessa categoria são: busca em largura, busca em profundidade e busca uniforme (algoritmo de Dijkstra). A segunda categoria é conhecida como busca com informação ou busca heurística. Essa estratégia de busca assume a presença de conhecimento específico do problema e, dessa forma, permite que sejam encontradas soluções de forma mais eficiente. O A* [Hart et al., 1968] é, provavelmente, o algoritmo de busca heurística mais conhecido e um dos mais estudados em Inteligência Artificial. O funcionamento e as propriedades desse algoritmo são discutidos em detalhes na Seção 2.3.1.

Em algumas aplicações da busca heurística em um espaço de estados, pode haver alterações nos componentes do problema de busca enquanto o agente executa seu plano. Em um jogo RTS, por exemplo, uma nova construção pode obstruir o caminho previamente planejado por uma unidade que estava se locomovendo, ou um novo caminho pode se tornar disponível pela eliminação de algum objeto do mapa. Em alguns casos, como no primeiro citado, a solução que está sendo executada pelo agente pode ser invalidada, o que exige um novo planejamento. A busca heurística incremental tenta resolver esse problema de forma eficiente, aproveitando informações de planejamentos anteriores para calcular um novo caminho caso seja necessário [Koenig et al., 2004b].

Algoritmos de busca heurística incremental são convenientes em aplicações onde o ambiente é dinâmico e parcialmente observável. No primeiro caso, mudanças no ambiente podem ocorrer durante o planejamento e / ou durante a execução do resultado do planejamento. Já em ambientes parcialmente observáveis não ocorrem mudanças, ocorrem divergências entre o mundo real e o modelo de mundo do agente que, geralmente, planeja usando a suposição do espaço livre, ou *freespace assumption* [Koenig & Smirnov, 1997; Zelinsky, 1992]. No planejamento de caminhos essa suposição diz que estados desconhecidos do espaço de estados devem ser tratados como estados livres durante o planejamento. Como já citado, vários jogos possuem ambientes dinâmicos e parcialmente observáveis, o que torna interessante o uso de algoritmos de busca heurística incremental em seu planejamento de caminhos. Alguns algoritmos desse tipo são: *Adaptive A** (AA*) [Koenig & Likhachev, 2006], *Lifelong Planning A** (LPA*) [Koenig et al., 2004a], *Dynamic A** (D*) [Stentz, 1994] e *D* Lite* [Koenig & Likhachev, 2002]. O *D* Lite* e o AA* são explicados em detalhes nas seções 2.3.2 e 2.4.4.

Existe ainda uma outra classe importante de algoritmos de busca heurística para o planejamento de caminhos em jogos digitais. Algoritmos de busca heurística de tempo real, também conhecidos na literatura como algoritmos de busca heurística local ou

busca centrada no agente [Koenig, 2001], são convenientes na presença de restrições de tempo.

Algoritmos de busca tradicionais, como o A*, primeiro determinam um plano de custo mínimo (como tempo, ou distância) e depois o executam. Dessa forma, eles são métodos de busca *offline*. Algoritmos de busca de tempo real, por outro lado, intercalam planejamento e execução, e portanto são algoritmos de planejamento *online*. Eles podem apresentar duas vantagens imediatas em relação aos algoritmos de busca *offline*: é possível executar ações na presença de restrições de tempo e reduzir a soma dos tempos de planejamento e execução. No primeiro caso o objetivo do planejamento passa a ser tentar minimizar o custo da execução respeitando uma restrição de tempo em cada episódio de planejamento. Chamamos de um episódio de planejamento cada intervalo entre ações que o algoritmo usa para planejar a próxima ação ou o próximo conjunto de ações. Em um jogo, por exemplo, não é interessante que um personagem fique um tempo parado, esperando o cálculo de um caminho ótimo, e depois inicie seu movimento. Ele deve sempre iniciar sua movimentação imediatamente mesmo que, no fim do planejamento, o caminho percorrido não tenha sido aquele de custo mínimo. A segunda vantagem citada da busca centrada no agente, a redução da soma dos tempos de planejamento e execução, é interessante quando não nos importamos com o custo da solução obtida, mas sim com o tempo total necessário para chegar à solução. Mesmo que a solução obtida tenha um custo maior, se a soma do tempo gasto pelos pequenos intervalos de ação e planejamento de um algoritmo *online* for inferior à soma do tempo de ação e planejamento de um algoritmo *offline*, então seu uso é vantajoso. A busca tradicional e a busca centrada no agente são ilustradas na Figura 2.1.

Para atender as restrições de tempo, algoritmos de busca de tempo real reduzem a busca a uma porção restrita do grafo em torno do agente. A primeira implicação dessa característica é que tais algoritmos não garantem a otimalidade das soluções. A segunda é que, caso algum cuidado não seja tomado, é fácil o agente entrar em um *loop*, nunca chegando ao estado objetivo. Dessa forma, algoritmos desse tipo devem possuir um mecanismo para evitar *loops*. A maioria dos algoritmos de busca centrada no agente evitam *loops* através do aprendizado de *h-values*. *H-values* são valores associados a vértices do grafo que representa o espaço de estados. O *h-value* de certo vértice é obtido pela aplicação de uma função heurística, e corresponde a uma estimativa do custo de alcançar o estado objetivo a partir do vértice em questão. Conforme o agente caminha pelo grafo ele utiliza as informações obtidas para melhorar a heurística de cada vértice, de forma que a busca pode explorar diferentes áreas do espaço de estados, já que ela é direcionada pelo valor da heurística. Exemplos de algoritmos que usam esse tipo de estratégia são o *Learning Real-Time A** (LRTA*) [Korf, 1990], *Local Search*

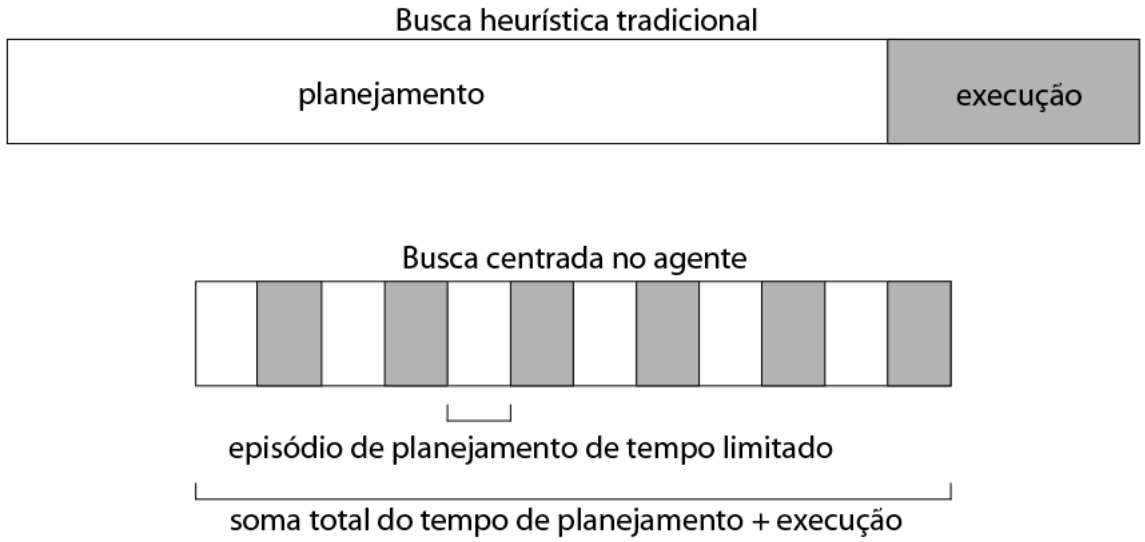

Figura 2.1. Busca tradicional e busca centrada no agente [Koenig, 2001].

*Space Learning Real-Time A** (LSSLRTA*) [Koenig, 2004], *Real-Time Adaptive A** (RTAA*) [Sun et al., 2008] e *Time-Bounded Adaptive A** (TBAA*) [Hernández et al., 2012]. Todos esses algoritmos são explicados de forma detalhada nesse trabalho, nas seções 2.4.1, 2.4.3, 2.4.4 e 2.4.8, respectivamente. Existem, no entanto, algoritmos que não fazem uso de aprendizado. Um exemplo é o *Path Real-Time A** (PRTA*) [Hernández & Baier, 2010] que, ao invés de atualizar *h-values*, marca estados já visitados. Esse algoritmo é explicado em detalhes na Seção 2.4.2.

2.2 Modelo Computacional

No restante deste texto serão usadas as seguintes notações: um problema de busca é definido pela tupla $P = (S, A, c, s_0, s_{goal})$, onde (S, A) é um grafo que representa o espaço de busca. O conjunto S são os vértices do grafo, que representam estados, enquanto A, as arestas, representam todas as ações possíveis. $c : A \mapsto \mathbb{R}^+$ é uma função de custo que associa um valor positivo a cada uma das ações possíveis, ou seja, uma ação que move um agente do vértice x para o y tem custo $c(x, y)$ positivo. $s_0 \in S$ é o estado inicial do agente e $s_{goal} \in S$ é o estado objetivo. A não ser que o contrário seja especificado, assumimos que o grafo é não-direcionado, ou seja, existe uma aresta do vértice x para o vértice y se e somente se existe uma aresta de y para x. Consideramos também que o custo associado a essas duas arestas é sempre o mesmo. Os sucessores, ou vizinhos, de um estado s são denotados por $Succ(s) = \{v|(s, v) \in A\}$. De forma similar,

$Pred(s) = \{v|(v, s) \in A\}$ denota os predecessores de um estado. Vale notar que, em um grafo não-direcionado, $Succ(s) = Pred(s)$. Usamos $g^*(s)$ para denotar a distância do menor caminho de s_0 a s. Além disso, uma função heurística $h : S \mapsto [0, \infty)$ associa, a cada estado s, uma aproximação $h(s)$ do custo de um caminho de s até s_{goal}.

Na literatura de Inteligência Artificial, e também nesse texto, um algoritmo de busca é classificado como completo caso ele sempre encontre uma solução (considerando que existe um caminho entre s_0 e s_{goal}). Se o algoritmo assegura encontrar sempre a solução ótima, então ele é classificado como admissível.

2.3 Algoritmos Tradicionais de Busca Heurística

Nesta seção são descritos dois algoritmos de busca heurística bastante conhecidos: o A* e o *D* Lite*. O entendimento do A* é importante para este trabalho, uma vez que a maioria dos algoritmos de busca heurística de tempo real utilizados o possuem como base. O *D* Lite* é importante uma vez que ele é bastante conhecido e utilizado, e apresenta soluções com boa qualidade. Utilizamos esse algoritmo como *baseline* em nossos experimentos.

2.3.1 A*

O A* [Hart et al., 1968] é um dos algoritmos de busca heurística mais conhecidos em Inteligência Artificial e provavelmente um dos mais estudados. Vários dos algoritmos utilizados neste trabalho são baseados no A* ou fazem uso dele. Nessa seção explicamos em detalhes o funcionamento e as propriedades desse algoritmo.

O A* é um algoritmo de busca heurística do tipo *best-first*, ou seja, ele escolhe um vértice para expansão (aplicação da função sucessora) com base em uma função de avaliação $f(s)$. Essa função de avaliação é construída como uma estimativa de custo, de forma que o vértice com menor valor $f(s)$ é sempre expandido primeiro. No A*, a avaliação dos vértices é feita através da combinação de dois valores: $g(s)$, o custo de alcançar certo vértice s a partir de s_0, e $h(s)$, uma estimativa do menor custo de alcançar s_{goal} a partir de s. O valor $f(s)$ utilizado é:

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

Como $g(s)$ é o custo do caminho de s_0 até s e $h(s)$ é a estimativa do menor custo de s até s_{goal}, então $f(s)$ nada mais é do que o custo estimado do menor caminho de s_0 até s_{goal} passando por s. Dessa forma, como nosso objetivo é encontrar a solução

de menor custo, o A* expande sempre o vértice com menor valor $f(s)$. Chamamos de *f-value* o valor $f(s)$ de um vértice s. Apesar de utilizar uma ideia bastante simples, esse algoritmo é completo e admissível caso algumas propriedades sejam satisfeitas.

Uma heurística é dita admissível quando ela nunca superestima o custo real de atingir o objetivo. Essas heurísticas são, naturalmente, otimistas, uma vez que sempre acreditam que o custo de resolver certo problema é menor ou igual ao custo real. Já heurísticas consistentes (também chamadas de monotônicas) respeitam a seguinte desigualdade: $h(s) \leq c(s, s') + h(s')$, onde $c(s, s')$ é o custo da aresta que liga o estado s ao estado s'. Dessa forma, uma heurística é consistente se, para cada vizinho s' de s, uma estimativa do custo de alcançar s_{goal} a partir de s não é maior que o custo de se mover para s' somado ao custo estimado de alcançar s_{goal} a partir de s'. Essa restrição pode ser vista como uma desigualdade triangular, que diz que cada lado do triângulo não pode ser maior que a soma dos outros dois. Toda heurística consistente é também admissível [Russell & Norvig, 2002], de forma que a consistência é uma restrição mais forte que a admissibilidade.

O A* é completo se as seguintes condições forem respeitadas [Pearl, 1984]: os vértices devem ter um número finito de sucessores e o peso associado às arestas deve ser positivo. Além disso, ele é admissível caso a heurística utilizada seja também admissível. Impondo uma restrição ainda mais forte à heurística utilizada, a consistência, é possível utilizar uma versão mais simples e eficiente do A*, que expande cada vértice no máximo uma vez [Nilsson, 1998].

O Algoritmo 1 apresenta o A* que assume o uso de uma heurística consistente. Para controlar a expansão de vértices são usadas duas estruturas de dados especiais: a lista *open* e a lista *closed*. A primeira, geralmente implementada como um *heap* binário, ajuda o algoritmo na seleção do próximo vértice a ser expandido. Ela fica constantemente ordenada em relação aos *f-values* dos vértices que possui, de forma que aquele com menor *f-value* pode ser facilmente obtido. A segunda lista é utilizada para armazenar nós já expandidos. Geralmente ela é implementada como uma tabela *hash*, de forma que é fácil consultar se certo vértice já foi ou não expandido, impedindo expansões repetidas.

No início da execução, o A* faz as inicializações necessárias (linhas 2 a 6): $g(s_0)$ é inicializado como 0 e seu $parent()$ como $NULL$, uma vez que s_0 é o vértice inicial da busca. O atributo *parent* é utilizado para que seja possível conhecer as transformações que levaram à solução do problema. O *parent* de certo vértice x corresponde a outro vértice que, expandido durante a busca, alcançou x com menor custo. O *f-value* de s_0 também é inicializado, e o vértice s_0 é inserido na lista *open*. A lista *closed* é, inicialmente, vazia.

```
1  Procedure A*
2      g(s_0) ← 0;
3      f(s_0) ← g(s_0) + h(s_0);
4      parent(s_0) ← NULL;
5      open ← {s_0};
6      closed ← ∅;
7      while open ≠ ∅ do
8          current ← argmin_{n∈open}(f(n));
9          if current = s_goal then
10             return solução;
11         end
12         open ← open \ {current};
13         closed ← closed ∪ {current};
14         foreach n ∈ Succ(n) do
15             if n ∉ closed then
16                 if n ∉ open then
17                     g(n) ← g(current) + c(current, n);
18                     f(n) ← g(n) + h(n);
19                     parent(n) ← current;
20                     open ← open ∪ {n};
21                 else if g(current) + c(current, n) < g(n) then
22                     g(n) ← g(current) + c(current, n);
23                     f(n) ← g(n) + h(n);
24                     parent(n) ← current;
25                 end
26             end
27         end
28     end
29     return não existe solução;
30 end
```

Algoritmo 1: Algoritmo A* fazendo uso de uma heurística consistente.

O *loop* principal do algoritmo (linhas 7 a 28) é executado até que o vértice s_{goal} esteja para ser expandido (o que acontece quando a solução ótima é encontrada) ou até que a lista *open* esteja vazia (o que ocorre quando todos os vértices do problema já foram expandidos). No caso da lista *open* estar vazia, o problema não possui solução. A cada execução do *loop* principal do algoritmo é selecionado, da lista *open*, um novo vértice para expansão (linha 8). Ele é o vértice em *open* com menor *f-value*. Esse vértice é retirado de *open* (linha 12) e inserido em *closed* (linha 13). Em seguida os sucessores ainda não expandidos do vértice selecionado são gerados e inseridos em *open* (linhas 17 a 20). Caso algum desses vértices já esteja em *open*, isso significa que foi

encontrado um novo caminho a partir de s_0. Nesse caso, se o novo caminho for menor que o antigo, o *g-value*, *f-value* e *parent* de tal vértice são atualizados (linhas 22 a 24).

A principal desvantagem do A* é que tanto sua complexidade de tempo quanto a de espaço são exponenciais em função do número de arestas do caminho ótimo [Pearl, 1984]. Essa característica é indesejável, especialmente se é necessário realizar o planejamento de caminhos entre dois vértices distantes de um ambiente muito grande.

2.3.2 *D* Lite*

O *D* Lite* [Koenig & Likhachev, 2002] é um algoritmo de busca heurística incremental, ou seja, ele assume a existência de uma série de problemas de busca semelhantes e reutiliza informações de buscas passadas para computar a solução do problema atual de forma mais rápida que efetuar um novo cálculo completo. Ele é amplamente utilizado na robótica, principalmente para a navegação em ambientes parcialmente observáveis, uma vez que pode planejar usando a suposição do espaço livre. Diferentemente dos algoritmos de busca heurística de tempo real, não é possível definir um tempo máximo de planejamento para cada episódio de busca do *D* Lite*. Em geral, o primeiro episódio de busca é mais demorado, já que realiza uma busca completa entre o estado inicial e o objetivo, e os demais episódios são mais rápidos, uma vez que aproveitam o caminho anteriormente computado.

A elaboração do *D* Lite* teve como base o algoritmo de busca heurística incremental LPA* [Koenig & Likhachev, 2001]. No entanto, ao contrário do LPA* e da maioria dos algoritmos de busca heurística, o *D* Lite* realiza o planejamento a partir do estado final, s_{goal}, em direção ao estado inicial s_0. Dessa forma denotaremos $g_2^*(s)$ como a menor distância entre s_{goal} e s.

Para realizar o planejamento, o *D* Lite* mantém algumas variáveis. A primeira delas é uma estimativa $g_2(s)$ da distância $g_2^*(s)$ para cada vértice s. Essa estimativa, que chamaremos de g_2-*values*, é armazenada e reutilizada em cada episódio de busca do algoritmo. A segunda, os *rhs-values*, são valores obtidos através de um *lookahead* de um passo com base nos g_2-*values* e, portanto, são valores potencialmente mais bem informados que os g_2-*values*. Esses valores sempre satisfazem a seguinte relação:

$$rhs(s) = \begin{cases} 0 & \text{se } s = s_{goal} \\ min_{s' \in Succ(s)}(g_2(s') + c(s, s')) & \text{caso contrário.} \end{cases}$$

Um vértice s do espaço de busca é localmente consistente se e somente se $g_2(s) = rhs(s)$, e localmente inconsistente caso contrário. Se todos os vértices $s \in S$ forem localmente consistentes, um caminho de custo mínimo pode ser encontrado movendo

sempre o agente do vértice s, começando de s_0, para o sucessor que minimiza $c(s, s') + g_2(s')$, até que s_{goal} seja alcançado. O algoritmo, no entanto, não faz de todos os vértices localmente consistentes. Ele usa uma função heurística para focar a busca e atualizar somente os g_2-*values* relevantes para computar o menor caminho. Para isso, o *D* Lite* faz uso de uma fila de prioridade (*priority queue*). Essa fila de prioridade possui sempre os vértices localmente inconsistentes do espaço de busca. A ideia principal do algoritmo é expandir esses nós localmente inconsistentes, deixando-os consistentes. A chave de um vértice s nessa fila é $k(s) = [k_1(s), k_2(s)]$, onde $k_1(s) = min(g_2(s), rhs(s) + h(s_0, s))$ e $k_2(s) = min(g_2(s), rhs(s))$. Note que $k_1(s)$ corresponde aos *f-values* utilizados pelo A*, enquanto $k_2(s)$ corresponde aos *g-values*. Uma chave $k(s)$ é menor ou igual a outra $k'(s)$ se $k_1(s) < k'_1(s)$ ou $k_1(s) = k'_1(s)$ e $k_2(s) \leq k'_2(s)$. O algoritmo expande sempre o nó com menor chave, ou seja, ele é similar ao A*, que expande sempre o nó com menor *f-value*. Uma vez que um caminho entre os vértices inicial e final tenha sido encontrado, o algoritmo para de expandir os nós localmente inconsistentes da fila de prioridades.

O Algoritmo 2 mostra os dois principais métodos do *D* Lite*.

```
1  Procedure UpdateVertex(u)
2      if u ≠ s_goal then rhs(u) = min_{s'∈Succ(u)}(c(u, s') + g_2(s'));
3      if u ∈ U then U.remove(u);
4      if g_2(u) ≠ rhs(u) then U.insert(u, CalculateKey(u));
5  end
6  Procedure ComputeShortestPath()
7      while U.TopKey() < CalculateKey(s_0) OR rhs(s_0) ≠ g_2(s_0) do
8          k_old = U.TopKey();
9          u = U.Pop();
10         if k_old < CalculateKey(u) then
11             U.insert(u, CalculateKey(u));
12         else if g_2(u) > rhs(u) then
13             g_2(u) = rhs(u);
14             foreach s ∈ Pred(u) do UpdateVertex(s) ;
15         else
16             g_2(u) = ∞;
17             foreach s ∈ Pred(u) ∪ {u} do UpdateVertex(s) ;
18         end
19     end
20 end
```

Algoritmo 2: Procedimentos principais do D* Lite

No início da execução do algoritmo, os g_2-*values* e *rhs-values* de todos os vértices

são inicializados como infinito. O *rhs-value* do estado objetivo é inicializado com 0 e, portanto, somente ele é localmente inconsistente e é o único vértice na fila de prioridade U.

Em seguida, o método *ComputeShortestPath()* é utilizado para encontrar um caminho entre o vértice inicial e o final. Isso é feito expandido-se sempre o nó de menor chave na fila de prioridades, que é um nó localmente inconsistente. O nó expandido pode ter seu g_2-*value* atualizado, de forma a torná-lo localmente consistente. Além disso, o método *UpdateVertex()* é usado para atualizar o *rhs-value* dos vizinhos do nó expandido e inserí-los ou removê-los da fila de prioridade caso seja necessário.

O loop das linhas 7 até 19 executa enquanto o vértice inicial não for localmente consistente e a chave do próximo vértice a ser expandido não é menor que a chave do vértice inicial. Esse funcionamento é similar ao do A*, que expande vértices até que o nó objetivo seja expandido. De fato, a primeira execução do *D* Lite* expande exatamente os mesmos estados que o A* expandiria, se realizasse a busca do estado objetivo para o inicial [Koenig & Likhachev, 2002].

Depois do primeiro *ComputeShortestPath()*, o agente inicia a execução do seu planejamento e verifica alterações nos custos das arestas. Caso não hajam alterações, o agente executará todo o planejamento e alcançará o estado objetivo. Caso alguma alteração seja detectada no caminho previamente planejado, os vértices ligados pelas arestas alteradas terão seus *rhs-values* alterados caso necessário, e o método *UpdateVertex()* será chamado para cada um desses vértices. Isso colocará os novos vértices localmente inconsistente na fila de prioridades. Em seguida, uma nova chamada ao *ComputeShortestPath()* atualizará todos os valores necessários para que seja encontrado um novo caminho do vértice corrente ao objetivo.

É importante citar também que o algoritmo usa uma estratégia para não precisar reordenar a fila de prioridades a cada detecção de mudança nas arestas (essa alteração seria necessária, uma vez que as chaves dos vértices na fila de prioridade foram calculadas com base em uma heurística de um vértice antigo do agente). Koenig & Likhachev [2002] apresentam uma discussões mais completa e detalhada do *D* Lite*, que inclui a prova de admissibilidade do algoritmo.

Um exemplo do D* Lite em funcionamento pode ser visto em `https://vimeo.com/88994255`.

2.4 Algoritmos de Busca Heurística de Tempo Real

Nesta seção são descritos vários algoritmos de busca heurística de tempo real. Além disso, na sessão 2.4.9, mostramos resumidamente como esses algoritmos se relacionam. A maioria deles são baseados no A*, e utilizados em nossa comparação. Um estudo de outros algoritmos de busca heurística baseados no A* pode ser visto em Rios & Chaimowicz [2010].

2.4.1 *Learning Real-Time A**

O *Learning Real-Time A** (LRTA*) [Korf, 1990] é um método de busca centrada no agente que pode ser utilizado para o planejamento de caminhos em ambientes parcialmente observáveis e com restrições de tempo real. Ele é um dos algoritmo mais conhecidos de busca heurística de tempo real, e seu funcionamento é bastante simples. A ideia principal do algoritmo é mover o agente em uma direção promissora com base em uma heurística. Uma função heurística define, para cada estado, um valor que corresponde à estimativa de custo de alcançar o objetivo a partir desse estado. Conforme o agente caminha, o LRTA* atualiza o valor dessa heurística para que ela se torne cada vez mais informada. O Algoritmo 3 ilustra o LRTA*.

```
1 s ← s_0;
2 while s ≠ s_goal do
3 |   foreach w ∈ Succ(s) do
4 |   |   atualize c(s, w);
5 |   end
6 |   y ← argmin_{w∈Succ(s)}(c(s, w) + h(w));
7 |   h(s) ← max{h(s), c(s, y) + h(y)};
8 |   Mova o agente para y e faça s ← y;
9 end
```

Algoritmo 3: Pseudo código para o Learning Real-Time A*

Esse algoritmo executa um ciclo de observação-*lookahead*-atualização-ação de forma iterativa, até o agente atingir seu objetivo [Hernández & Baier, 2010]. Na fase de observação (linhas 3 a 5) o LRTA* observa os estados vizinhos em relação à posição atual do agente e atualiza o custo de visitar cada um deles. Isso consiste, normalmente, em atualizar o valor $c(s, w)$ para infinito caso um estado w sucessor de s esteja bloqueado. Na fase de *lookahead* (linha 6) é determinado qual deverá ser o próximo estado a ser visitado. Essa escolha é feita com base no valor da heurística calculada para os estados vizinhos e do custo de chegar a cada um deles a partir do estado atual. O

estado escolhido é aquele com menor valor $c(s, w) + h(w)$, onde w é um estado sucessor de s, ou seja, o estado mais promissor que levará o agente ao seu objetivo. Na terceira fase, a de atualização (linha 7), o estado corrente s do agente tem seu custo estimado de alcançar o objetivo ($h(s)$) atualizado. O novo valor de $h(s)$ é o maior valor escolhido entre $h(s)$ e $c(s, y)+h(y)$, onde y é o estado para o qual o agente irá se locomover. Essa atualização funciona como uma forma de aprendizado, aproximando cada vez mais o valor da heurística ao custo real de locomoção do estado s ao objetivo. Vale notar que, caso a heurística usada seja consistente, essa atualização preservará tal propriedade: o valor de h é atualizado para o maior valor possível tal que $h(s)$ continua consistente. Finalmente, na última fase, a de ação (linha 8), o agente de fato se locomove para o estado previamente escolhido.

A Figura 2.2 ilustra o comportamento do LRTA* em um mapa na forma de *grid* simples. Nesse exemplo o agente pode se mover nas direções norte, sul, leste e oeste, a não ser que a direção esteja bloqueada. O custo das ações é uniforme e igual a 1. Células pretas estão bloqueadas e brancas estão livres. Os estados foram inicializados com a distância de *Manhattan*, ou seja, a distância até o estado objetivo caso os obstáculos sejam desconsiderados. Na primeira fase de *lookahead* do algoritmo é decidido que o próximo estado a ser visitado deverá ser C2, uma vez que ele possui o menor *h-value* entre os vizinhos do estado atual e as todas as ações possuem custo uniforme. Em seguida, na fase de atualização, o valor de C1 é atualizado. O novo valor é $1 + 2 = 3$, o custo de se mover para C2 adicionado de seu *h-value*. Depois disso o agente se move para C2. Em C2, o procedimento é repetido: o próximo estado a ser visitado será C1, que é o único vizinho livre de C2. O valor de C2 será atualizado para $1 + 3 = 4$, o custo de se mover para C1 adicionado de seu *h-value*, e o agente se move para C1. O procedimento é repetido até que o agente alcance o objetivo.

Pelo exemplo é possível perceber a importância do aprendizado de *h-values* nesse algoritmo: sem ele o agente ficaria em *loop* entre os estados C1 e C2, nunca atingindo o estado objetivo.

O LRTA* (e todos os outros algoritmos de busca heurística de tempo real apresentados nesse trabalho) garantidamente alcança o estado objetivo se o espaço de estados for finito e puder ser explorado de forma segura, ou seja, caso seja sempre possível voltar a um estado anterior. Além disso, o algoritmo tende a ser mais eficiente quanto mais informados forem os *h-values* iniciais [Koenig, 2001], ou seja, quanto melhor for a estimativa do custo de alcançar o estado objetivo. Vale citar também que o tempo de planejamento do LRTA* é muito pequeno, de forma que, em problemas onde o interesse está apenas em alcançar rapidamente o estado objetivo, ele pode ser mais rápido que algoritmos de busca *offline*, como o A*. No entanto, em problemas como o planeja-

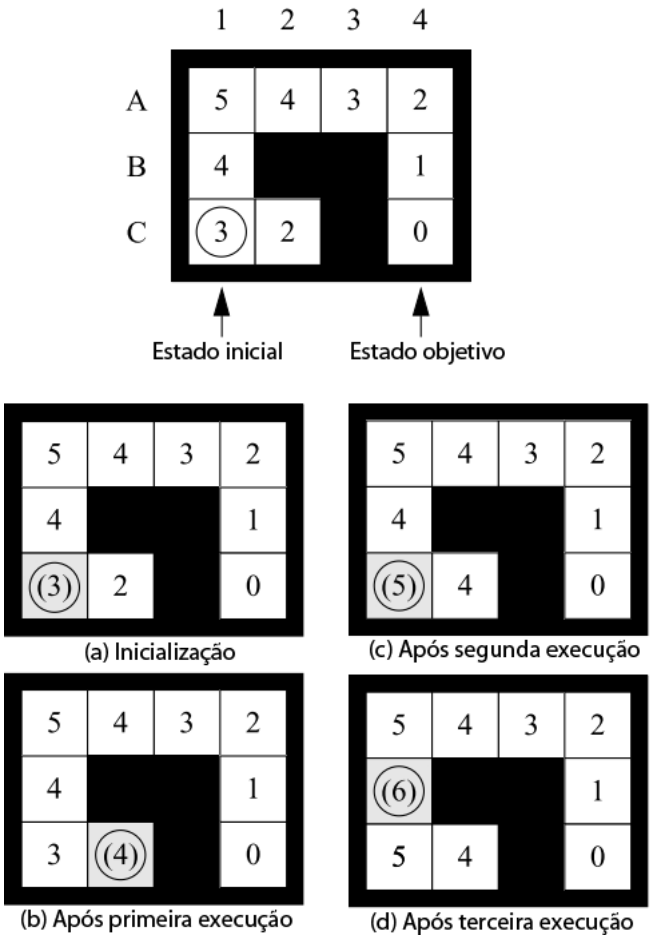

Figura 2.2. Exemplo de funcionamento do LRTA*.

mento de caminhos, onde estamos interessados no número de movimentos necessários para chegar ao estado objetivo, o LRTA* pode encontrar soluções bastante ruins.

Uma outra característica importante do LRTA*, compartilhada por outros algoritmos de busca de tempo real que aprendem *h-values*, é a possibilidade de encontrar menores caminhos através de repetidas execuções. Se o mesmo problema for repetido diversas vezes, em algum momento o caminho encontrado convergirá para o caminho de custo mínimo desde que duas condições sejam satisfeitas: (1) a heurística inicial utilizada é admissível e (2) os *h-values* dos estados são mantidos entre as execuções. Essa característica é particularmente interessante para o planejamento de caminho de agentes usados para coletar recursos em jogos RTS, uma vez que eles caminham entre os mesmos dois pontos repetidas vezes. Além disso, esse processo de convergência pode ser melhorado pelo uso de computação paralela, como mostrado em Marques et al. [2011], onde o processo de convergência é otimizado para a arquitetura paralela do *Playstation 3*.

Um exemplo de funcionamento do LRTA* pode ser visto em `https://vimeo.com/88902096`.

2.4.1.1 Depressões heurísticas

Em problemas de busca de tempo real a heurística utilizada geralmente possui "depressões". Intuitivamente, uma depressão heurística é uma região limitada do espaço de busca que possui estados cujo valor da heurística é muito baixo em relação ao valor da heurística dos estados que estão na borda da depressão. Depressões existem naturalmente em heurísticas utilizadas com algoritmos de busca heurística de tempo real. Como acabamos de ver, o LRTA* constrói uma solução de forma incremental, atualizando o valor da heurística associado a certo estado conforme mais informações do ambiente são coletadas. Além disso, vários outros algoritmos de busca de tempo real seguem esse mesmo princípio, especialmente aqueles baseados no LRTA*. Esses algoritmos, em geral, se comportam de forma ruim em depressões heurísticas e, por isso, é importante compreender esse conceito.

Hernández & Baier [2012] utilizam a seguinte definição: um componente conectado de estados D é uma depressão heurística de uma heurística h se e somente se para cada estado $s \in D$ e todo estado $s' \notin D$ que é vizinho de algum estado em D, $h(s) < k(s, s') + h(s')$, onde $k(s, s')$ é o custo do caminho de menor custo que começa em s, atravessa somente estados em D, e termina em s'.

A Figura 2.3 ilustra essa definição. Essa figura representa um mapa na forma de *grid* de conectividade 4 (um agente só pode se locomover para norte, sul, leste e oeste). O estado objetivo é o estado **C9**, marcado pela letra G. Os números em cada estado representam o valor da função heurística para cada um deles (a distância de Manhattan até o estado objetivo). Os estados coloridos de cinza representam uma depressão heurística: eles formam um componente conectado de estados onde, para todos os estados s da área cinza, $h(s) < k(s, s') + h(s')$, onde s' pode ser qualquer um dos estados coloridos de branco.

É conhecido que algoritmos como o LRTA* se comportam mal em depressões heurísticas [Ishida, 1992]. Vamos assumir, por exemplo, que o LRTA* descrito anteriormente é executado em um mapa com uma depressão heurística. Suponha que o agente visita um estado nessa depressão, e que o estado objetivo está fora dela. Para sair da depressão heurística o agente deve seguir um caminho dentro dela, por exemplo, $s_1, \ldots, s_n$, até escolher um estado na borda da depressão, s_e. Enquanto em s_n, portanto, o agente deve escolher s_e como o próximo estado a ser visitado, o que significa que s_e deve possuir o menor custo estimado de alcançar o objetivo entre todos

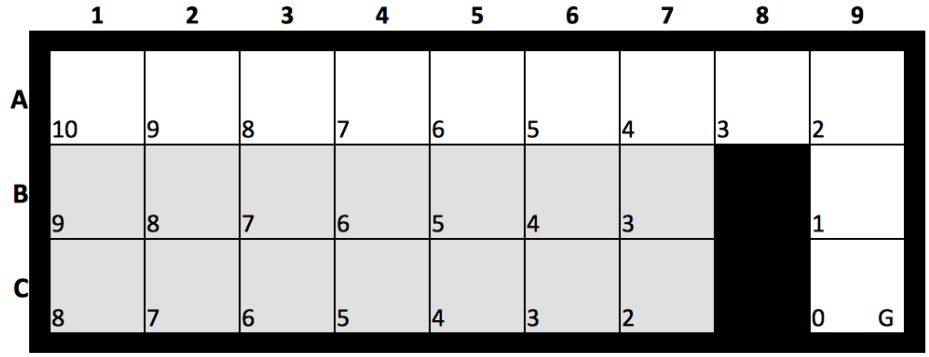

Figura 2.3. Exemplo de uma depressão heurística.

os vizinhos de s_n. Em um problema com custos de ação uniformes, isso só pode acontecer se $h(s_e)$ é menor ou igual ao valor da heurística de todos os outros vizinhos de s_n. Isso significa que a depressão no espaço de estados não existe mais, ou seja, que os *h-values* dos estados na depressão heurística já foram atualizados (incrementados). Para o LRTA*, no pior caso, todos os estados da depressão heurística precisam ser atualizados, possivelmente diversas vezes.

2.4.2 *Path Real-Time A**

Um dos grandes problemas do LRTA* e de grande parte dos algoritmos de busca heurística de tempo real é o comportamento em depressões heurísticas. Em geral, algoritmos mais complexos apresentam um tempo maior de planejamento para que seja possível obter um melhor comportamento em tais situações. Exemplos de algoritmos desse tipo são o LSSLRTA*, LRTA*(k) e RTAA*. O *Path Real-Time A** (PRTA*) [Hernández & Baier, 2010] procura sair de depressões heurísticas rapidamente sem um aumento significativo no tempo de planejamento, ou seja, ele tenta ser tão rápido quanto o LRTA*, evitando os problemas das depressões heurísticas.

Existem duas diferenças principais entre o PRTA* e o LRTA*. A primeira é que, quando um estado é identificado como sendo parte de uma depressão heurística, ele é marcado como não visitável. Isso significa que o agente não irá mais visitar tal estado no futuro. A segunda diferença principal é que, apesar de fazer uso de uma função heurística, o PRTA* não atualiza o valor associado aos estados. Ele faz uso de um mecanismo diferente para evitar *loops*. Esse mecanismo consiste em armazenar a sequência de estados visitados pelo agente em uma pilha. Se um estado sem vizinhos visitáveis é alcançado, então o agente volta para o último estado visitado. Uma desvantagem do PRTA* é que, como ele não possui um processo de aprendizado, não é

possível encontrar a solução ótima de um determinado problema através de repetidas execuções.

```
1  Procedure PRTA*
2      s ← s_0;
3      path.push(s);
4      seen ← {};
5      foreach s ∈ S do s.updaterule ← false;
6      while s ∉ G do
7          s ← seen ∪ {s};
8          foreach w ∈ Succ(s) do updatec(s, w);
9          N ← {w ∈ Succ(s)|w.updaterule = false};
10         y ← argmin_{w∈N} c(s, w) + h(w);
11         if y ≠ Null then
12             if h(s) < c(s, y) + h(y) then s.updaterule ← true;
13             Mova o agente para y;
14             s ← y;
15             path.push(s);
16         else
17             s.updaterule ← true;
18             path.pop();
19             if path = EmptyStack then return no-solution;
20             Mova o agente para path.top();
21             s ← path.top();
22         end
23     end
24 end
```

Algoritmo 4: Pseudo código do PRTA*.

O Algoritmo 4 mostra o pseudo código do PRTA*. Para cada estado s, $s.updaterule$ pode passar a valer $true$ se s for detectado como estando em uma depressão heurística. Dessa forma, $s.updaterule$ vale $true$ quando um estado s é marcado como não visitável. Inicialmente, o valor é inicializado como $false$ para todos os estados (linha 5). Depois de uma movimentação do agente (linhas 2 e 14), o estado para o qual o agente se move é adicionado à variável $path$. A seleção do próximo vértice a ser visitado é feita de forma análoga ao LRTA*, mas somente estados visitáveis (que ainda não foram detectados como sendo parte de uma depressão heurística) são considerados. Se o estado corrente está em uma depressão heurística o algoritmo faz a marcação na linha 12. Um estado s é considerado como pertencente a uma depressão heurística se $h(s) < c(s, y) + h(y)$, onde y é o sucessor w de s com menor valor $c(s, w) + h(w)$. Quando não existe um sucessor válido para o estado corrente o agente retorna para

o último vértice visitado, marcando o estado corrente para evitar revisitá-lo(linhas 17 a 21).

Um exemplo de funcionamento desse algoritmo pode ser visto em `https://vimeo.com/88902342`.

2.4.3 *Local Search Space Learning Real-Time A**

O *Local Search Space Learning Real-Time A** (LSSLRTA*) [Koenig, 2004] é uma variação do LRTA*. Ele faz melhorias em duas partes do LRTA* básico. A primeira consiste em como definir o espaço de busca local do algoritmo. O LRTA* utiliza, geralmente, um espaço de busca local que consiste somente no estado corrente do agente. A segunda melhoria consiste na decisão de quais estados do espaço de busca local terão seus *h-values* atualizados. Novamente, o LRTA* original atualiza somente o *h-value* do estado corrente.

O espaço de busca local (*local search space*, ou LSS) consiste nos estados que serão utilizados pelo algoritmo de busca em um episódio de planejamento. Várias versões do LRTA* com um espaço de busca local maior foram propostas [Ishida, 1992; Bulitko et al., 2007; Russell & Wefald, 1991; Pemberton & Korf, 1992]. No entanto, tipicamente, elas não satisfazem restrições de tempo real, uma vez que consideram fatores como o tamanho de uma depressão heurística ou a área conhecida do mapa para definir o LSS. A forma mais fácil de limitar o tempo de planejamento de forma a satisfazer tais restrições é fazendo com que o LSS seja de um tamanho fixo. Uma opção é, por exemplo, definir o LSS com uma busca em largura de profundidade limitada [Korf, 1990]. Os estados expandidos pela busca em largura formam o LSS. O uso da busca em profundidade definirá espaços de busca em formato de disco o que, em alguns casos, pode ser sub-ótimo. O LSSLRTA* define o espaço de busca fazendo uso do A*, e atualiza os *h-values* de todos os estados no espaço de busca para acelerar o processo de aprendizado. Dessa forma, o LSSLRTA* pode ser dividido nos seguintes passos:

1. Use o A* para buscar do estado corrente do agente em direção ao estado objetivo até que $l > 0$ estados tenham sido expandidos ou o estado objetivo esteja para ser expandido. Os estados expandidos pelo A* (aqueles na lista *closed*) formam o espaço de busca local.

2. Use o algoritmo de Dijkstra para substituir os *h-values* de todos os estados no espaço de busca local pela soma da distância do estado até um estado s e o *h-value* do estado s, minimizado entre todos os estados $s \in S$ que estão na borda do espaço de busca local (os estados na lista *open* do A* executado).

3. Mova o agente pelo caminho encontrado pelo A* até que ele chegue ao final do caminho (saia do espaço de busca local) ou encontre um obstáculo previamente desconhecido que obstrua seu caminho.

4. Se o estado corrente do agente é diferente do estado objetivo, volte para o passo 1. Caso contrário o objetivo foi alcançado.

No passo 1, o A* é executado do estado corrente do agente em direção ao estado objetivo, expandindo estados até que um número $l > 0$ pré-definido de estados tenham sido expandidos ou o estado objetivo seja o próximo a ser expandido. O valor da variável l é o *lookahead* do LSSLRTA*. Os estados expandidos pela execução do A* formam o LSS. Como o *lookahead* do algoritmo é previamente definido, é possível alterá-lo para satisfazer as condições necessárias de tempo de planejamento. No passo 2, o algoritmo de Dijkstra é usado para atualizar o valor de todos os estados no LSS. Essa atualização é feita com base nos estados que formam a borda do espaço de busca local. Tais estados são obtidos facilmente: após a execução do A*, eles correspondem à lista *open* gerada. O novo *h-value* de cada estado s será a menor soma da distância de s até um estado w da borda com o *h-value* de w. No passo 3 o agente se move até sair do espaço de busca ou descobrir informações sobre o mapa que prejudiquem seu planejamento anterior, como um estado bloqueado (o que pode acontecer, por exemplo, em ambientes parcialmente observáveis). No passo 4 o agente repete o ciclo caso ainda não tenha alcançado seu objetivo.

A ideia de usar o A* no passo 1 e atualizar o valor de todos os estados do espaço de busca no passo 2 é focar a busca e o aprendizado na direção do estado objetivo. Além disso, a atualização de todos os estados permite uma maior propagação de informações, fazendo com que o algoritmo aprenda mais rápido.

Um exemplo de funcionamento do LSSLRTA* pode ser encontrado em `https://vimeo.com/88902102`.

2.4.4 *Real-Time Adaptive A**

O *Real-Time Adaptive A** (RTAA*) [Sun et al., 2008] é um algoritmo de busca heurística de tempo real baseado no *Adaptive A** (AA*) que apresenta várias semelhanças ao LSSLRTA*. Assim como o LSSLRTA*, o RTAA* também faz uso de um espaço de busca local, e esse espaço de busca também é escolhido através da execução do A* com um número limitado de expansões. Além disso, o RTAA* também atualiza os *h-values* de todos os estados no LSS. Como a maioria dos algoritmos de busca de tempo real que fazem uso de aprendizado de *h-values*, o RTAA* também encontra a

solução ótima para certo problema se ele for repetido um número suficiente de vezes. A principal diferença entre os dois algoritmos encontra-se na forma como esses estados são atualizados.

Para compreender o RTAA*, primeiro é necessário entender o funcionamento do AA*. A ideia principal desse algoritmo é simples. Assuma que seja necessário executar o A* diversas vezes com uma heurística consistente no mesmo espaço de estados e com o mesmo estado objetivo, mas, possivelmente, com diferentes estados iniciais. O AA* torna a heurística associada a cada estado mais informada depois de cada execução do A*, de forma a melhorar as buscas posteriores.

Como detalhado na Seção 2.3.1, o A* mantém, para cada estado s encontrado durante a busca, um valor $g(s)$, que corresponde ao custo do menor caminho encontrado de s_0 até s e um valor $f(s) = g(s) + h(s)$, que corresponde a uma estimativa do custo de atingir s_{goal} passando por s. Assuma agora que s foi um estado expandido durante uma execução do A*. É possível obter uma estimativa admissível da distância de s até s_{goal} (que chamaremos de $gd(s)$) da seguinte forma: a distância do estado inicial s_0 para o estado objetivo s_{goal} passando por s é igual à distância de s_0 até s somada à distância de s ao estado objetivo, ou seja, $gd(s)$. Essa soma claramente não pode ser menor que $gd(s_0)$ já que $gd(s_0)$ é a menor distância entre s_0 e s_{goal}, encontrada pelo A*. Dessa forma, temos que:

$$\begin{aligned} g(s) + gd(s) &\geq gd(s_0) \\ gd(s) &\geq gd(s_0) - g(s) \end{aligned}$$

Note que $gd(s_0)$ nada mais é que o *f-value* do estado objetivo, $f(s_{goal})$, computado previamente pelo A*. Então:

$$gd(s) \geq f(s_{goal}) - g(s)$$

Dessa forma, $f(s_{goal}) - g(s)$ é uma estimativa admissível da distância entre o estado s e o estado objetivo. Além disso, essa estimativa pode ser calculada rapidamente após a execução do A*, uma vez que, nesse momento, teremos o valor $f(s_{goal})$ e os *g-values* de cada estado expandido. O AA*, então, encontra heurísticas mais informadas após cada execução de um A*, calculando e atribuindo essa diferença a cada estado da lista *closed*. Uma análise mais detalhada do AA*, assim como a avaliação experimental desse algoritmo, pode ser encontrada em Koenig & Likhachev [2005].

O RTAA* usa a mesma estratégia do AA* para tornar as heurísticas utilizadas mais informadas. No entanto, diferentemente do AA*, o RTAA* não pode executar sempre o A* do estado corrente até o objetivo, uma vez que deve satisfazer restrições

de tempo real. O Algoritmo 5 mostra o pseudo código do RTAA*.

```
1  Procedure RTAA*
2      while s ≠ s_goal do
3          lookahead ← valor previamente definido;
4          a_star(lookahead);
5          foreach s ∈ closed do h(s) = g(s_next) + h(s_next) − g(s);
6          while s ≠ s_next do
7              Mova o agente na direção de s_next;
8              if houve algum aumento de custo na trajetória de s até s_next then
9                  break;
10             end
11         end
12     end
13 end
```

Algoritmo 5: Pseudo código do RTAA*.

Inicialmente é definido um valor para a variável *lookahead* (linha 3). Esse valor define quantos estados serão expandidos pelo A*. Ele é relacionado diretamente com o tempo de planejamento do algoritmo. Se seu valor for infinito, o algoritmo se comportará da mesmo maneira que o AA*, e o algoritmo dificilmente ira satisfazer restrições de tempo real. Se seu valor for 1, o algoritmo se comportará da mesma forma que o LRTA* [Sun et al., 2008]. Após definido o *lookahead*, é executado o A* com profundidade limitada (linha 4). Sua execução irá gerar duas variáveis importantes: a primeira, *closed*, corresponde à lista *closed* utilizada pelo A*. Ela contém, como já citado, todos os estados expandidos durante a busca. A segunda, s_{next}, corresponde ao próximo estado que seria expandido pelo A*. Quando *lookahead* é usado como infinito, $s_{next} = s_{goal}$. Em seguida, os *h-values* dos estados na lista *closed* são atualizados (linha 5). Essa etapa corresponde ao processo de aprendizado do algoritmo. A atualização ocorre da mesma forma que aquela utilizada no AA*. Note que, no entanto, a atualização do AA* é sempre baseada no s_{goal}, enquanto a atualização do RTAA* é baseada no s_{next}, uma vez que nem sempre o A* conseguirá chegar até o s_{goal} utilizando um *lookahead* limitado. Finalmente, após o processo de aprendizado, o agente executa ações para se locomover em direção ao estado s_{next}. Ele executa tais ações até que s_{next} seja atingido ou até que o custo de alguma ação no caminho até s_{next} aumente (linhas 6 a 11). Em geral, o custo aumenta quando o agente encontra um obstáculo previamente desconhecido em seu caminho. O procedimento é repetido até que o agente chegue ao estado objetivo.

É importante perceber que a heurística de um estado nunca decresce durante a

execução do algoritmo, de forma que ela sempre se torna mais informada. Além disso ela também nunca se torna inconsistente (considerando que usamos, desde o princípio, uma heurística consistente) [Sun et al., 2008].

O RTAA* e o LSSLRTA* apresentam muitas semelhanças. A maior diferença entre os dois algoritmos está na forma como é feita a atualização dos *h-values* dos estados no espaço de busca local. Uma observação interessante é que, normalmente, a atualização das heurísticas feita pelo LSSLRTA* é mais informada que aquela feita pelo RTAA* quando os dois algoritmos utilizam o mesmo valor de *lookahead* e, no entanto, como será mostrado posteriormente, o RTAA* encontra, na maioria das vezes, soluções melhores. Uma das razões disso é que a atualização de valores do RTAA* é muito simples: basta percorrer a lista *closed* realizando a atualização. Dessa forma o processo de atualização é muito mais rápido, o que permite que, no mesmo tempo, esse algoritmo utilize sempre um *lookahead* maior, levando a soluções melhores.

Um exemplo do RTAA* em funcionamento pode ser visto em `https://vimeo.com/88902104`.

2.4.5 *Learning Real-Time A*(k)*

O LRTA* é um algoritmo simples, que dá espaço para várias otimizações. O *Learning Real-Time A*(k)* (LRTA*(k)) [Hernández & Meseguer, 2005] é um algoritmo baseado no LRTA*. Os dois algoritmos diferem somente na forma como fazem a atualização de *h-values*: enquanto o LRTA* atualiza somente o valor do estado corrente do agente, o LRTA*(k) propaga a atualização para k outros estados. Essa alteração, apesar de bastante simples, proporciona diversos benefícios, como melhora de qualidade da primeira solução e diminuição do custo de convergência (quando o algoritmo é executado diversas vezes para o mesmo problema).

A propagação de valores realizada pelo LRTA*(k) é uma propagação limitada, uma vez que o algoritmo deve satisfazer restrições de tempo real. No entanto, por facilidade de compreensão, vamos primeiro explicar como funcionaria uma propagação ilimitada e, em seguida, será explicado como o LRTA*(k) restringe tal propagação.

Caso o *h-value* do estado corrente do agente sofra uma alteração depois de um *lookahead*, o LRTA* escolhe o próximo estado e executa uma ação para alcançá-lo. Ele não realiza mais atualizações. Uma propagação ilimitada propagaria essa mudança de *h-value* para todos os sucessores do estado corrente. Cada sucessor seria considerado como base de um novo *lookahead* e, se necessário, seus *h-values* seriam atualizados. Se a atualização ocorresse, os sucessores de cada um desses estados atualizados seriam, novamente, considerados bases para um novo *lookahead*, e assim sucessivamente, até

que não ocorressem mais atualizações.

A atualização limitada do LRTA*(k) funciona como a ilimitada descrita, porém com duas alterações:

1. É definido um número máximo k de estados que podem ser atualizados durante uma propagação.

2. Somente estados previamente expandidos são considerados para propagação, ou seja, somente estados que pertencem ao caminho do estado inicial ao estado corrente podem ser atualizados. Essa condição tenta prevenir que a atualização de *h-values* se espalhe muito pelo espaço de estados.

Dessa forma é possível controlar o tempo de planejamento do algoritmo e realizar atualizações de *h-values* focadas. Os resultados encontrados melhoram conforme k aumenta, ao custo de um aumento também no tempo de planejamento. No entanto, como estudado por Hernández & Meseguer [2005], o uso de valores pequenos de k proporcionam aumentos pequenos de tempo de execução e grandes melhorias nos resultados encontrados.

Um exemplo de execução do LRTA*(k) pode ser visto em `https://vimeo.com/88902099`.

2.4.6 *Prioritized Learning Real-Time A**

Vários algoritmos de busca de tempo real apresentam um processo de aprendizado que permite o refinamento da solução ao longo de execuções repetidas. O foco do *Prioritized Learning Real-Time A** (P-LRTA*) [Rayner et al., 2007] é na melhoria desse processo. Esse algoritmo torna o aprendizado mais eficiente, priorizando as atualizações de *h-values*. Estados considerados importantes são atualizados antes dos outros, onde a importância de um estado é medida pela magnitude das atualizações feitas em seus vizinhos.

O P-LRTA* utiliza ideias do Prioritized Sweeping [Moore & Atkeson, 1993], um algoritmo de aprendizado por reforço que faz atualizações por uma ordem de prioridades. Ele combina as atualizações com prioridades do Prioritized Sweeping com a busca em tempo real do LRTA*.

Como na maioria dos algoritmos baseados no LRTA*, o P-LRTA* funciona em duas partes principais, que são repetidas até que o agente atinja seu objetivo. Na primeira, a de planejamento, o agente ganha conhecimento de como navegar no ambiente atualizando a heurística de alguns estados. Na segunda, a de ação, o agente escolhe o próximo estado a visitar e se locomove para ele.

O Algoritmo 6 mostra o pseudo código do P-LRTA*. Ele faz uso de uma lista de prioridades, que chamamos de *queue*. Além disso é preciso definir, previamente, um valor N, que será a quantidade máxima de estados atualizados por episódio de planejamento. Essa variável definirá o tempo de cada episódio de planejamento: quanto maiores as restrições de tempo, menor deve ser N.

```
1  Procedure StateUpdate(s)
2      s' ← argmin_{w∈Succ(s)}(c(s, w) + h(w));
3      Δ ← c(s, s') + h(s') − h(s);
4      if Δ > 0 then
5          h(s) ← c(s, s') + h(s');
6          foreach n ∈ Succ(s) do
7              AddToQueue(n, Δ);
8          end
9      end
10 end
11 Procedure P-LRTA*
12     s ← s_0;
13     queue ← ∅;
14     N ← valor previamente definido;
15     while s ≠ s_goal do
16         StateUpdate(s);
17         repeat
18             p ← queue.pop();
19             if p ≠ s_goal then
20                 StateUpdate(p);
21             end
22         until N estados forem atualizados ou queue = ∅;
23         s ← argmin_{w∈Succ(s)}(c(s, w) + h(w));
24     end
25 end
```

Algoritmo 6: Pseudo código do P-LRTA*.

No começo da fase de planejamento o algoritmo atualiza o *h-value* do estado corrente (linha 16) através do método *StateUpdate()*. Nesse método, a quantidade pela qual o valor foi atualizado é armazenada (linha 3) em uma variável, Δ, antes que ocorra a atualização (linha 5). Em seguida, cada sucessor do estado corrente é inserido na lista de prioridades (linhas 6 a 8). A prioridade de cada estado na lista será Δ. Um estado será inserido na lista pelo método *AddToQueue()* somente se ele ainda não estiver na lista de prioridades. Além disso, é comum limitar o tamanho dessa lista. Nesse caso, caso a lista esteja cheia no momento de inserção, o estado com menor prioridade é

retirado da lista para a inserção do novo.

Após a atualização do estado corrente e as novas inserções na lista de prioridades, o algoritmo inicia uma série de atualizações priorizadas. No máximo N estados são retirados do topo da lista de prioridades e seus *h-values* são atualizados da mesmo forma que o estado corrente, usando o método *StateUpdate()* (linhas 17 a 22). O número de estados atualizados pode ser menor que N, caso todos os estados disponíveis tenham sido atualizados e a lista de prioridades esteja vazia. Se, após a atualização de N estados, a fila ainda possuir elementos, eles são mantidos para uso na próxima iteração do algoritmo.

Completada a etapa de planejamento, o algoritmos seleciona o próximo estado a visitar. Ele escolhe somente uma ação, e de forma gulosa: o próximo estado a ser visitado é o vizinho s' do estado corrente s com menor valor $c(s, s') + h(s')$ (linha 23).

Uma característica importante do P-LRTA* e que o diferencia da maioria dos outros algoritmos baseados no LRTA* é que a atualização de *h-values* não ocorre, necessariamente, em uma área ao redor do agente, uma vez que a fila de prioridades é mantida entre as várias iterações do algoritmo. Como citado por Rayner et al. [2007] essa característica pode ser benéfica, uma vez que alterações na heurística de certo estado podem influenciar mesmo os valores de estados distantes.

Um exemplo da execução desse algoritmo pode ser visto em `https://vimeo.com/88902343`.

2.4.7 *Time-Bounded A**

O *Time-Bounded A** (TBA*) [Björnsson et al., 2009] é uma variação do A* que pode ser utilizada em ambientes com restrições de tempo real.

Um dos maiores problemas de algoritmos como o LSSLRTA* é o replanejamento: após executar uma ação (ou um pequeno conjunto de ações) e voltar à fase de planejamento, apesar do algoritmo possuir mais informações devido às atualizações de *h-values*, ele ainda assim expande vários vértice já expandidos em buscas anteriores. O A* com uma heurística consistente, por outro lado, nunca expande um estado mais de uma vez. No entanto, o agente não pode iniciar sua movimentação até que a busca seja totalmente completada.

A ideia principal do TBA* é fazer o planejamento em tempo real evitando a expansão de estados repetidos. O algoritmo expande vértices do estado inicial em direção ao final como o A*, até que o estado final seja expandido. No entanto, enquanto o A* planeja o caminho completo antes de iniciar a execução de ações, o TBA* interrompe a busca depois de um número pré-definido de expansões para agir. O caminho do es-

tado mais promissor na lista *open* do A* (aquele que seria o próximo expandido) até o estado inicial é computado. Essa computação termina antes que o estado inicial seja atingido caso o agente se encontre nesse caminho: nesse caso a próxima ação do agente será tomar mais um passo em direção ao estado mais promissor. Caso o agente não seja encontrado, o caminho é traçado do estado mais promissor até o estado inicial. O caminho no qual o agente se encontra é considerado obsoleto, e ele começa então a se locomover na direção do novo caminho gerado. A forma mais simples de obter esse comportamento é fazer com que o agente faça um *backtracking* em direção ao estado inicial, até que ele cruze o novo caminho traçado. Depois disso, no próximo episódio de planejamento, a busca continua de onde foi interrompida, com as mesmas listas *open* e *closed*. A Figura 2.4 mostra a ideia básica do TBA*. S é o estado inicial e G o estado objetivo. As curvas representam a lista *open* do A* após cada interrupção da busca. Os pontos (a), (b) e (c) são estados da lista *open* com o menor *f-value*. As linhas tracejadas são os menores caminhos até esses pontos. Os primeiros três passos do agente são: $S \rightarrow 1 \rightarrow 2 \rightarrow 1$. O agente realiza um *backtrack* no último passo porque o caminho até o novo estado mais promissor não passa por 2, o estado em que o agente se encontrava. É importante notar que o TBA* mantém as listas *closed* e *open* entre cada episódio de planejamento. Isso significa que ele não começa a busca do zero a cada planejamento, como o LSSLRTA*.

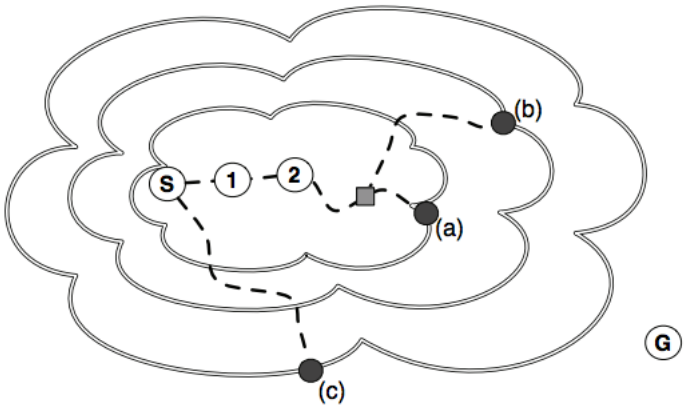

Figura 2.4. Exemplo do TBA* em funcionamento [Björnsson et al., 2009].

O TBA* pode, portanto, ser dividido em 4 passos:

1. Use o A* para buscar do estado corrente do agente em direção ao estado objetivo até que $l > 0$ estados tenham sido expandidos ou o estado objetivo esteja para

ser expandido.

2. Compute o caminho do estado mais promissor na lista *open* até o estado inicial.

3. Se o agente se encontra no caminho computado, mova-o para o próximo estado desse caminho. Caso contrário mova o agente para o estado em que estava anteriormente.

4. Se o estado corrente do agente é diferente do estado objetivo, volte para o passo 1. Caso contrário o objetivo foi alcançado.

A variável l é o *lookahead* do algoritmo, e define a quantidade de tempo que pode ser utilizada por cada episódio de planejamento. Quanto maiores as restrições de tempo, menor deverá ser o valor de l.

Apesar de evitar a expansão de estados repetidos, o TBA* possui uma desvantagem em relação aos algoritmos baseados no LRTA* (como o RTAA* e o LSSLRTA*): ele funciona somente quando não existem alterações no custo das arestas do grafo. Isso significa que esse algoritmo funciona somente em ambientes completamente observáveis. Essa restrição é necessária para que o algoritmo possa manter as listas *open* e *closed* entre cada episódio de planejamento.

Um exemplo do TBA* em funcionamento pode ser visto em `https://vimeo.com/88994254`.

2.4.8 *Time-Bounded Adaptive A**

O *Time-Bounded Adaptive A** (TBAA*) [Hernández et al., 2012] é um algoritmo de busca de tempo real que estende o TBA* para o funcionamento em ambientes parcialmente observáveis, fazendo uso da suposição do espaço livre. Isso é feito combinando o TBA* e o Adaptive A*.

Explicaremos o TBAA* em duas etapas. Inicialmente, será explicado um algoritmo intermediário, o *Restarting Time-Bounded A** RTBA* [Hernández et al., 2012], um algoritmo que faz somente as alterações mínimas necessárias para que o TBA* funcione em ambientes parcialmente observáveis. Em seguida explicaremos como adicionar ao RTBA* algumas características do AA*, o que resulta no TBAA*.

O RTBA* funciona da seguinte forma: inicialmente é executado o TBA* com a suposição do espaço livre. O algoritmo expande um certo número de estados, calcula a rota até o melhor vértice da lista *open* e o agente se movimenta na direção desse vértice. A diferença em relação ao TBA* ocorre quando, após a movimentação, é percebida uma mudança no custo da rota até o vértice mais promissor de *open*. Em um

ambiente parcialmente observável isso significa que um obstáculo previamente desconhecido foi encontrado, e a rota previamente calculada é, agora, inválida. Ao detectar esse problema o RTBA* recomeça a busca, usando o estado corrente como novo ponto de partida. As listas *open* e *closed* são esvaziadas e um novo TBA* começa do estado corrente em direção ao estado objetivo. Esse algoritmo é, como mostrado por Hernández et al. [2012], completo. Além disso, ele eventualmente encontra a solução ótima de um problema de busca caso seja repetido um número suficiente de vezes.

Cada vez que o RTBA* começa um novo A*, todas as informações obtidas pela execução do A* anterior são perdidas. O TBAA* funciona como o RTBA*, e se diferencia dele aproveitando as informações das buscas anteriores, atualizando *h-values* da mesma forma que o AA* e o RTAA*. Sempre que o TBAA* precisa iniciar um novo A*, ele atualiza os *h-values* de todos os vértices vistos durante a busca anterior atribuindo $h(s) = f(s_{best}) - g(s)$ caso isso aumente $h(s)$. s_{best} é, nesse caso, o vértice da lista *open* com menor *f-value*. Essa operação mantém a consistência da heurística utilizada e a torna mais informada. Assim como o RTBA*, esse algoritmo é completo e converge para a solução ótima com um número suficiente de repetições.

Um exemplo do TBAA* em funcionamento pode ser visto em `https://vimeo.com/88902106`.

2.4.9 Relação entre os algoritmos

Finalizamos nossa discussão sobre algoritmos de busca heurística de tempo real resumindo o relacionamento entre eles. A Figura 2.5 ilustra a relação entre os algoritmos desse tipo discutidos nesse trabalho. Na figura, uma seta entre dois algoritmos indica que o algoritmo apontado pela seta foi derivado do algoritmo do qual a seta sai.

Podemos ver que a maioria dos algoritmos discutidos são derivados do LRTA*. O LRTA*(k), P-LRTA*, LSSLRTA* e RTAA* alteram um ou mais aspectos do LRTA*, como a forma de atualização de *h-values* e a definição do espaço de busca local.

O PRTA*, apesar de apresentar certa semelhança com o LRTA*, movendo o agente sempre na direção do vizinho mais promissor, não é derivado dele. Como já citado, esse algoritmo é importante por apresentar ideias diferentes de como desviar de obstáculos e sair de depressões heurísticas.

Finalmente, os algoritmos TBA*, RTBA* e TBAA* estão relacionados entre si. O RTBA* apenas estende o funcionamento do TBA* para ambientes parcialmente observáveis. O TBAA*, por sua vez, adiciona ideias do AA* ao RTBA* para realizar o aprendizado de *h-values* visto em outros algoritmos.

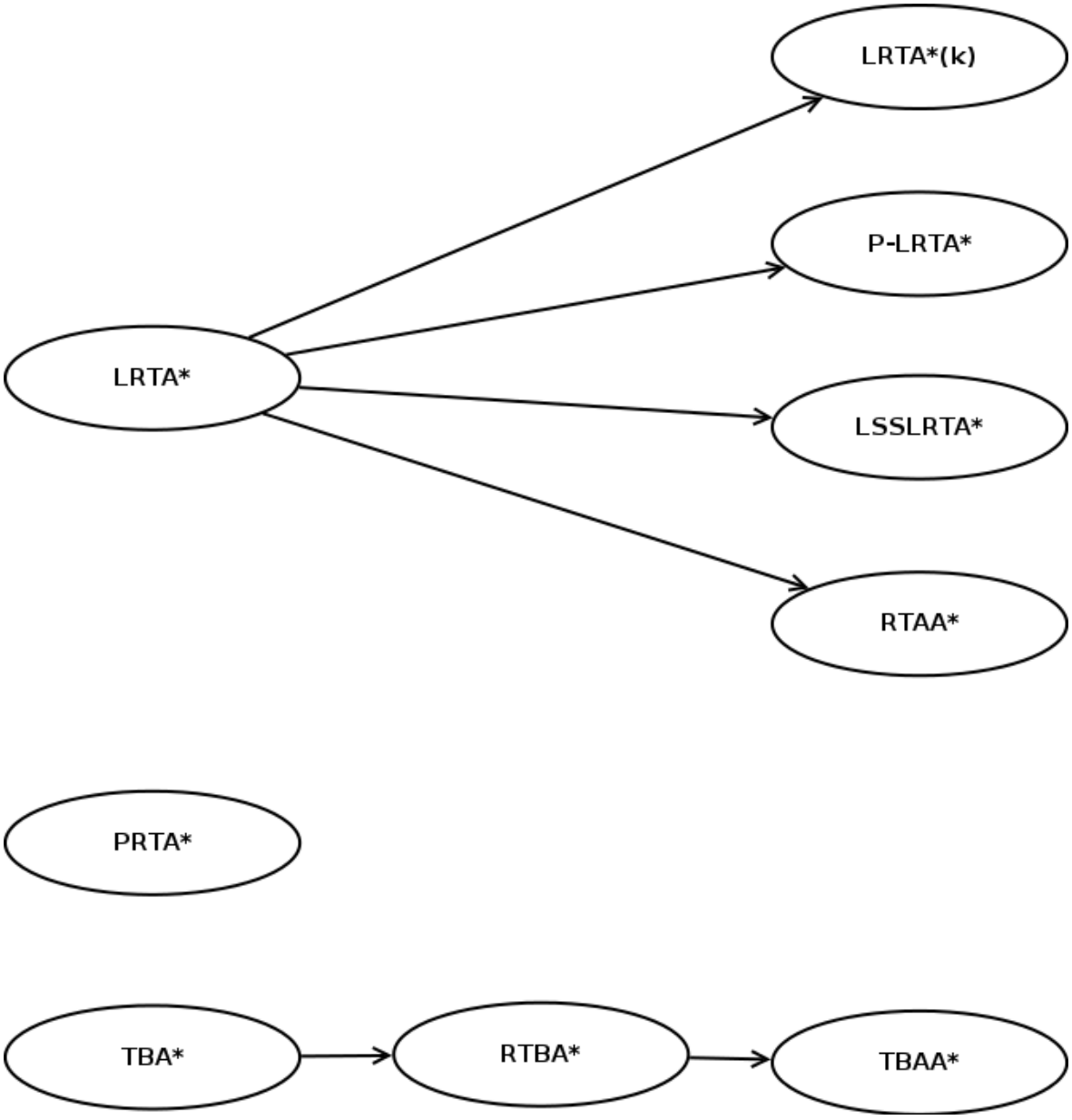

Figura 2.5. Relacionamento entre os algoritmos de busca heurística de tempo real discutidos nesse trabalho.

2.5 Trabalhos Relacionados

Além dos trabalhos já citados nesse capítulo, alguns outros também merecem destaque. Bulitko et al. [2010] fazem uma revisão de três algoritmos de busca em tempo real que podem ser utilizados para o planejamento de caminhos em jogos digitais: o *k Nearest Neighbors LRTA** (kNN LRTA*) [Bulitko & Björnsson, 2009], o TBA* e o *Real-time Iterative-deepening Best-first Search* (RIBS) [Sturtevant et al., 2010]. O TBA* já foi explicado nesse trabalho. O kNN LRTA* faz uso de um banco de dados pré computado

de caminhos para otimizar o LRTA*, e o RIBS funciona como uma versão centrada no agente do TBA*. O RIBS, assim como TBA*, não funciona em ambientes parcialmente observáveis. Os autores concluem que, dentre os três algoritmos comparados, o kNN LRTA* apresenta as soluções de melhor qualidade. No entanto sua desvantagem é a necessidade de gerar um banco de dados previamente, um procedimento bastante custoso.

Burns et al. [2013] introduzem um *benchmark* para avaliar algoritmos de busca de tempo real. Esse *benchmark* consiste em um jogo de plataforma simples. A maior diferença dele para os *benchmarks* mais utilizados na literatura, que consistem em mapas na forma de *grid*, é que o grafo representante do problema é direcionado ao invés de não-direcionado. Os autores testam o LSSLRTA* nesse novo *benchmark* e mostram experimentalmente que não é difícil obter com ele uma performance superior à do A* nos testes feitos, uma vez que o A* não pode planejar e executar ações de forma intercalada, o que é penalizado no *benchmark* proposto.

As métricas mais utilizadas na literatura para avaliação de algoritmos de busca heurística são o tempo gasto para resolver um problema e o custo da solução obtida. Hernández et al. [2012] sugere uma outra métrica: o número de intervalos de tempo gastos até alcançar o objetivo. Considera-se o tempo como sendo dividido em vários intervalos com duração t. Durante cada intervalo o algoritmo deve planejar e pode escolher uma ação que será executada. Esse modelo é fortemente motivado pelo uso em jogos, onde o tempo é dividido em ciclos, geralmente muito curtos (da ordem de alguns milissegundos) [Bulitko et al., 2010]. Essa métrica é particularmente útil para comparar algoritmos de busca de tempo real com algoritmos *offline*, como o A*, já que tais algoritmos serão penalizados por deixarem o agente parado nos primeiros intervalos de tempo. Note que, para algoritmos de busca de tempo real, que escolhem sempre uma ação a cada intervalo de tempo, essa métrica equivale ao custo da solução encontrada caso as ações executadas pelo agente possuam todas o mesmo custo.

Hernández & Baier [2012] sugerem uma melhoria que pode ser utilizada na maioria dos algoritmos de busca tempo real. Essa melhoria consiste em evitar depressões heurísticas, áreas do espaço de busca na qual a maioria desses algoritmos se comporta de forma bastante ruim. Além disso, os autores mostram a aplicação desse princípio em dois algoritmos, o LSSLRTA* e o RTAA*. Eles mostram que os resultados obtidos após a aplicação dessa técnica são superiores, e que os algoritmos continuam respeitando um tempo máximo de planejamento pré-definido.

Finalmente, Sturtevant & Bulitko [2011] propõem um algoritmo de busca de tempo real bastante interessante, o *f-cost Learning Real-Time A** (f-LRTA*). Esse algoritmo utiliza um aprendizado tanto de *h-values* (como a maioria dos algoritmos

descritos nesse capítulo) como de *g-values*. O aprendizado de *g-values* é particularmente útil para sair de depressões heurísticas. O algoritmo proposto, no entanto, funciona somente em ambientes completamente observáveis.

Na literatura, não existe nenhum trabalho que compare detalhadamente os vários algoritmos de busca heurística de tempo real. Trabalhos relacionados somente comparam alguns desses algoritmos em termos de qualidade da solução. Dessa forma, o objetivo de nosso trabalho é realizar uma comparação em um nível mais aprofundado desses importantes algoritmos, analisando-os de acordo com diferentes métricas.

Capítulo 3

Metodologia

No capítulo anterior apresentamos vários algoritmos de busca heurística. Entre eles, estão os principais algoritmos desse tipo que podem ser utilizados em situações com restrições de tempo real e em ambientes parcialmente observáveis. Neste capítulo discutimos, na Seção 3.1, a escolha dos algoritmos utilizados em nossa análise, assim como alguns detalhes importantes de implementação. Em seguida, na Seção 3.2, apresentamos alguns detalhes sobre a forma como foram realizadas as avaliações dos algoritmos. Finalmente, na Seção 3.3, apresentamos o conjunto de testes e as condições utilizadas em sua execução.

3.1 Algoritmos utilizados

Escolhemos utilizar, em nossa análise, os seguintes algoritmos de busca heurística de tempo real:

- **LRTA***: o LRTA* foi escolhido por ser um algoritmo de busca heurística de tempo real bastante conhecido, além de ser a base de diversos algoritmos mais complexos.
- **LRTA*(k)**: esse algoritmo foi escolhido por ser uma variação simples e bastante conhecida do LRTA*, que tem foco em tempo de convergência da solução.
- **LSSLRTA***: esse algoritmo foi escolhido por ser a variação mais popular do LRTA*. Ele apresenta, em geral, bons resultados, e é facilmente configurável para executar com diferentes tempos máximos de episódio de planejamento.
- **RTAA***: outro algoritmo bastante conhecido, o RTAA* foi escolhido por utilizar ideias do AA* no LRTA*.

- **TBAA***: esse algoritmo foi escolhido por não ser, como a maioria dos algoritmos de busca heurística de tempo real, uma variação do LRTA*. Além disso ele também possui um processo de convergência de solução.
- **PRTA***: esse algoritmo bastante simples foi escolhido pela sua diferença em relação aos demais. Enquanto a maioria dos algoritmos utiliza um processo de aprendizado, o PRTA* utiliza episódios de planejamento muito curtos e sem aprendizado para chegar ao estado objetivo.
- **P-LRTA***: esse algoritmo foi escolhido pelo modo não convencional que utiliza em seu processo de aprendizado. Enquanto a maioria dos algoritmos foca o aprendizado em uma área ao redor do agente, o P-LRTA* usa uma lista de prioridades que pode atualizar valores de estados remotos.

Além disso, utilizamos também, em nossa comparação, dois algoritmos que não satisfazem todas as condições satisfeitas pelos anteriores (funcionamento em ambientes parcialmente observáveis e com restrições de tempo real). São eles:

- **D* Lite**: o D* Lite funciona em ambientes parcialmente observáveis, mas não satisfaz restrições de tempo real. No entanto, como ele é um algoritmo amplamente utilizado e que apresenta, geralmente, bons resultados, é interessante compará-lo com os demais algoritmos escolhidos.
- **TBA***: o TBA*, ao contrário do D* Lite, funciona com restrições de tempo real, mas não em ambientes parcialmente observáveis. Dessa forma, pelas mesmas razões citadas acima, utilizamos o TBA* em algumas de nossas comparações.

É importante citar agora alguns detalhes de implementação. Para que nossa comparação fosse justa, a implementação dos algoritmos foi feita de forma bastante parecida. Inicialmente, foi implementado um *framework* para problemas de planejamento de caminhos. Para cada algoritmo implementado bastava implementar um método desse *framework*, que se encarregava de escolher a próxima ação a ser tomada por um agente. As medições utilizadas na comparação foram todas feitas diretamente pelo *framework*. Além disso, utilizamos as mesmas estruturas de dados para os diferentes algoritmos. Todos os algoritmos que faziam uso de uma lista *open*, por exemplo, tiveram tal lista implementada como um *heap* de *fibonacci*. A heurística inicial utilizada por cada algoritmo também foi a mesma, a distância de *Chebyshev*. A distância de *Chebyshev* d entre dois pontos p e q de coordenadas (p_i, p_j) e (q_i, q_j) é dada por:

$$d = max(|p_i - q_i|, |p_j - q_j|)$$

Essa heurística é consistente para o tipo de mapa utilizado nos testes, que são mapas na forma de *grid* com conectividade oito. Mais detalhes sobre esses mapas são fornecidos na Seção 3.3.

3.2 Modelo de comparação

Como já explicado anteriormente, algoritmos de busca heurística de tempo real podem, geralmente, ter o tempo máximo de cada episódio de planejamento ajustado através de uma variável, na maioria das vezes chamada de *lookahead*. O primeiro passo para realizar uma boa comparação entre algoritmos desse tipo é configurar tal variável de forma que o tempo de planejamento máximo dos diversos algoritmos implementados seja o mesmo, ou seja, garantir que cada algoritmo sempre tomará uma decisão em, no máximo, um tempo t pré-definido. Esse tempo é a restrição de tempo real do problema. Vale notar que, no entanto, vários dos algoritmos implementados não vão gastar o tempo máximo para cada decisão de ação. Enquanto alguns algoritmos gastam praticamente o mesmo tempo planejando antes de cada tomada de decisão, como o TBAA*, outros, como o LSSLRTA*, por exemplo, armazenam uma sequência de ações após um episódio de planejamento e, em seguida, somente executam essa sequência de ações, até que todas tenham sido executadas ou seja encontrado um obstáculo previamente desconhecido. Dessa forma, o tempo gasto por ação executada fica menor, apesar de todos os algoritmos satisfazerem as mesmas restrições de tempo real. Isso pode causar diferenças, por exemplo, no tempo total de planejamento de cada algoritmo, como mostramos no Capítulo 4, que discute os resultados obtidos. Em nossas comparações, cada algoritmo de tempo real tem o *lookahead* configurado para que seu episódio de planejamento dure, no máximo, um tempo pré-definido t. Essa configuração é feita experimentalmente, e precisa ser feita somente em um mapa, uma vez que o tempo de planejamento dos algoritmos de tempo real não depende do mapa utilizado.

Métricas padrão para a avaliação de algoritmos de busca heurística incluem o tempo de CPU gasto para resolver um problema e o custo da solução encontrada. Essas duas métricas são utilizadas em nossa comparação. Além disso, utilizamos uma terceira métrica para avaliar os algoritmos implementados: o número necessário de execuções até a convergência. Como já citamos, alguns algoritmos de busca de tempo real podem encontrar a solução ótima para determinado problema caso o problema seja resolvido um número suficiente de vezes, mantendo os *h-values* aprendidos. Dos algoritmos implementados, a maioria (seis) apresenta tal propriedade, o que torna interessante a avaliação dessa métrica.

3.3 Conjunto de testes

Como o foco da nossa análise é no planejamento de caminhos aplicado a jogos, o conjunto de testes escolhido consiste em uma série de problemas de planejamento de caminhos em mapas de jogos comerciais na forma de *grid*.

O *grid* é uma representação discreta de um espaço bidimensional. É formado por células de mesmo tamanho, que possuem associadas a elas atributos relacionados com a porção do ambiente que representam. A variação na resolução do *grid* permite controlar a quantidade de detalhes e de informações necessárias para representá-lo. Vários jogos comerciais, como *Dragon Age: Origins* e *Baldur's Gate* utilizam *grids* como o formato interno dos mapas [Sturtevant, 2012].

No contexto de planejamento de caminhos em *grids*, algumas células geralmente estão bloqueadas e o agente não pode se deslocar através delas. O restante possui um custo associado que pode ser uniforme (igual para todas as células) ou não uniforme. A partir de uma célula, o agente pode se locomover para as células vizinhas não bloqueadas. As células vizinhas são definidas pela conectividade do *grid*. Em um *grid* com conectividade quatro os vizinhos permitidos estão à direita, à esquerda, acima e abaixo da célula corrente. Em um *grid* de conectividade oito também é possível se locomover para os vizinhos nas direções diagonais. Dessa forma, o *grid* define um grafo onde as células são nós e a conexão entre elas as arestas. Esse grafo representa o espaço de estados, como citado na Seção 2.2.

Os mapas na forma de *grid* utilizados neste trabalho estão disponíveis em `http://movingai.com/benchmarks/`. Esse repositório, proposto por Sturtevant [2012], possui diversos mapas de jogos comerciais que podem ser utilizados como *benchmarks* para algoritmos de planejamento de trajetórias. Cada mapa fornecido no repositório vem também acompanhado de um arquivo de testes, que possui um conjunto de problemas. Esses problemas consistem, basicamente, em uma posição inicial, onde o agente deve iniciar o planejamento, e uma posição final, onde o agente deve chegar. Além disso, no próprio arquivo de testes, é fornecido o custo da solução ótima para cada problema. O conjunto de problemas fornecidos para cada mapa é dividido em *buckets*. Problemas que estão em um mesmo *bucket* possuem o custo do caminho ótimo parecido. Mais especificamente, um problema cujo custo do caminho é l está no *bucket* de número $l/4$. Cada *bucket* em cada mapa tem 10 problemas. A utilização desse conjunto de problemas é interessante para que diferentes pesquisas possam comparar seus resultados facilmente.

Para nossos testes foram escolhidos cinco mapas do jogo *Dragon Age: Origins* e um mapa do jogo *Warcraft III*. Os mapas escolhidos são mostrados na Figura 3.1. A

dimensão, número de células livres e número de problemas de busca utilizados em cada mapa são mostrados na Tabela 3.1.

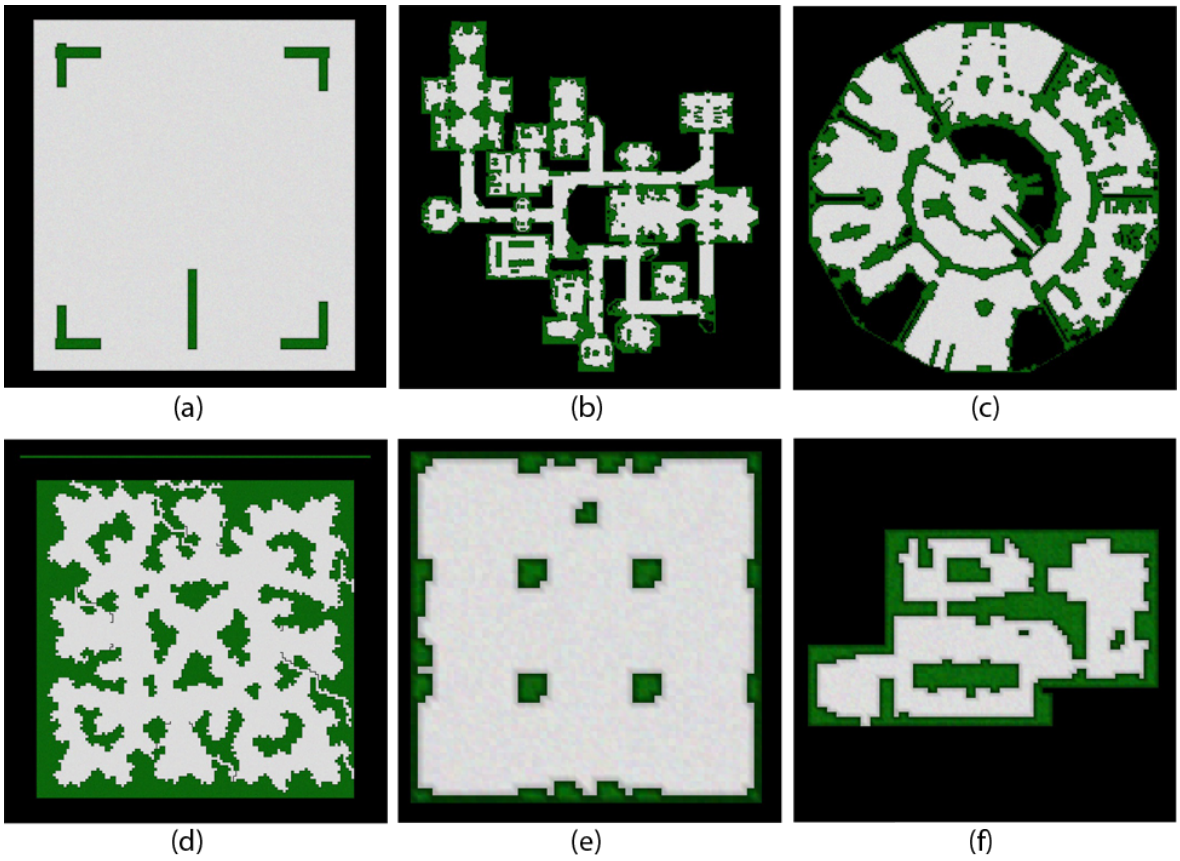

Figura 3.1. Mapas utilizados: (a) combat; (b) hrt201n; (c) lak503d; (d) duskwood; (e) arena; (f) den101d. Os mapas (a), (b), (c), (e) e (f) são do jogo *Dragon Age: Origins*, enquanto o mapa (d) pertence ao jogo *Warcraft III*.

Mapa	Dimensão	Número de células livres	Número de problemas
combat	177x193	32967	130
duskwood	512x512	127229	260
hrt201n	294x305	23652	250
lak503d	194x194	17953	250
arena	49x49	2054	40
den101d	73x41	1360	40

Tabela 3.1. Características dos mapas utilizados nos testes.

Os mapas são utilizados como *grids* de conectividade oito, ou seja, o agente pode se locomover na diagonal. O custo de locomoção na diagonal é $\sqrt{2}$, enquanto o custo de locomoção no restante das direções é 1. O agente só pode se locomover para uma

diagonal livre caso os dois vizinhos nas direções que levam a esta diagonal também estejam livres. Por exemplo, o agente só pode se locomover para o vizinho na direção noroeste caso esse vizinho esteja livre e os vizinhos a oeste e norte também estejam livres.

Os problemas de cada mapa foram resolvidos por cada algoritmo em diferentes condições de ambiente: desconhecidos, parcialmente observáveis e completamente observáveis. Isso foi feito para que pudéssemos avaliar o quanto a visibilidade do agente influencia cada algoritmo. Quando o ambiente é considerado como desconhecido, o agente tem visão, inicialmente, somente de seus vizinhos imediatos. O restante dos estados são desconhecidos, e considerados livres pela suposição do espaço livre. Uma vez que o agente adquire visão de certo estado ele armazena sua condição na memória, passando assim a ter mais informação do ambiente. Ambientes parcialmente observáveis funcionam como os desconhecidos, com a única diferença de que o agente tem uma visão de todos os estados que estão a certa distância da sua posição corrente. Nos testes realizados o raio de visão utilizado foi de 10 células do *grid*, ou seja, o agente tem um campo de visão de $21x21$ células, que fica ao redor dele. Nos testes considerando ambientes completamente observáveis o agente sabe as condições de todas as células do *grid* do mapa durante todo o tempo.

Como citado na seção anterior, é necessário configurar o tempo máximo de planejamento t de cada algoritmo para a execução dos testes. Em nossos experimentos configuramos os algoritmos para 4 valores de t: $0,025$s, $0,050$s, $0,075$,s e $0,100$s. A utilização de diferentes valores é útil para que possamos analisar o comportamento de cada algoritmo para diferentes restrições de tempo. Algoritmos que não podem ter seu tempo de planejamento configurado, como o LRTA* e o PRTA*, foram executados com seu tempo padrão de planejamento que é, em geral, bem menor que os valores utilizados. Os valores escolhidos foram valores bastante baixos, uma vez que o foco do nosso estudo é o planejamento de caminho em jogos, onde o tempo de computação em cada ciclo é bastante reduzido.

Capítulo 4

Resultados

Neste capítulo são apresentados os resultados obtidos pelos algoritmos nos diversos testes realizados. Iniciamos a apresentação, na Seção 4.1, mostrando os tempos médios dos episódios de planejamento dos algoritmos implementados. Em seguida, na Seção 4.2, mostramos a otimalidade das soluções encontradas nos diversos mapas utilizados. Na Seção 4.3, analisamos o tempo total de busca gasto nesses testes, de forma que é possível verificar o *tradeoff* entre a qualidade da solução obtida e o tempo total de busca. Na Seção 4.4, analisamos o número de execuções necessárias até a convergência para cada algoritmo. Por fim, na Seção 4.5, sumarizamos as informações obtidas durante todo o capítulo.

4.1 Tempo do episódio de planejamento

Como citado na Seção 3.2, o tempo do episódio de planejamento de grande parte dos algoritmos de busca heurística de tempo real implementados pode ser configurado. Essa configuração foi feita para os tempos $0,025$s, $0,050$s, $0,075$s e $0,100$s. Nessa seção mostramos o tempo médio do episódio de planejamento de cada um dos algoritmos durante a solução dos problemas do conjunto de testes escolhido. Nosso objetivo é mostrar a diferença do tempo médio de planejamento entre os algoritmos configuráveis e os não configuráveis, além de verificar se os algoritmos configuráveis gastam, em média, o mesmo tempo em cada episódio de planejamento.

A Figura 4.1 mostra o tempo médio do episódio de planejamento dos algoritmos de busca de tempo real no mapa **lak503d**, usando visibilidade parcial, para cada tempo máximo definido. O gráfico está em escala logarítmica, uma vez que o tempo do episódio de planejamento do LRTA* e PRTA* é muito baixo em relação aos demais algoritmos. Mostramos aqui somente os resultados para um mapa, uma vez que o

tempo do episódio do planejamento é independentemente do mapa utilizado. Além disso são mostrados somente os resultados somente para visibilidade parcial, uma vez que os tempos obtidos foram bastante parecidos para as outras duas visibilidades.

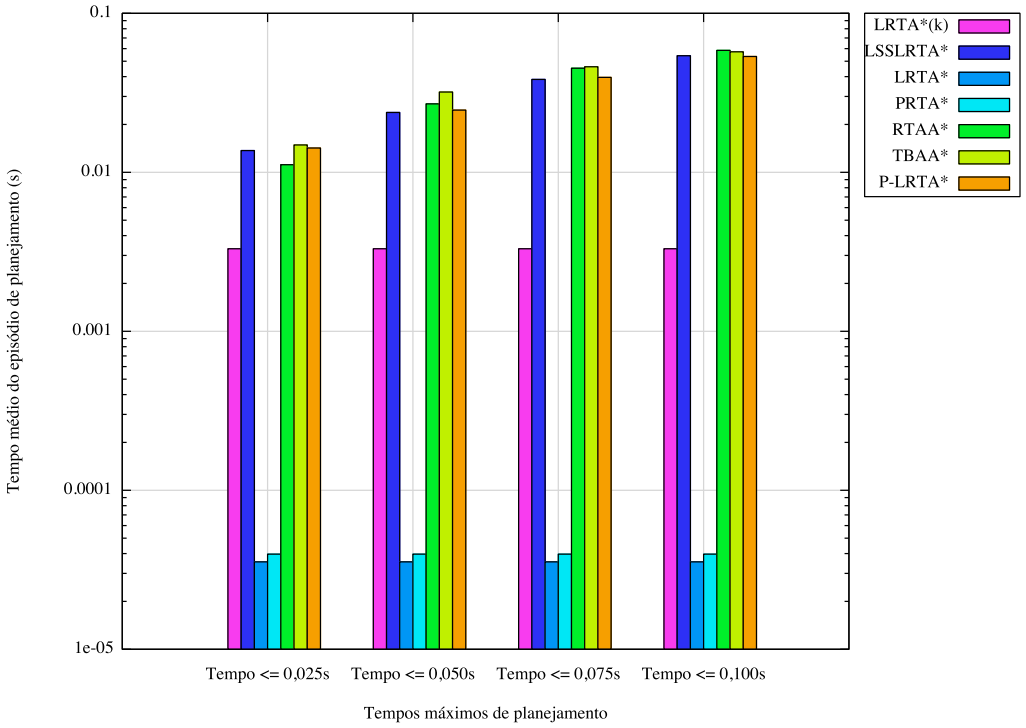

Figura 4.1. Tempo médio do episódio de planejamento no mapa **lak503d** com visibilidade parcial.

Observando a figura é fácil notar que, como esperado, o LRTA*, PRTA* e LRTA*(k) apresentam praticamente o mesmo tempo médio de episódio de planejamento para qualquer tempo limite estabelecido, além de apresentarem os tempos médios de planejamento mais curtos, especialmente os dois primeiros. O LRTA* e PRTA* se comportam dessa forma pois não possuem um parâmetro de ajuste. O LRTA*(k) apresentou esse comportamento pois, como explicaremos na Seção 4.2, foi utilizado o valor máximo para k (o número de estados do mapa) em todos os testes e, ainda assim, os episódios de planejamento desse algoritmo foram bastante curtos.

Os demais algoritmos sempre possuem o tempo médio do episódio de planejamento menor que o tempo máximo permitido. Isso ocorre pois nem todos os episódios de planejamento usarão o tempo limite. Para algoritmos como o TBAA* e LSSLRTA*, por exemplo, o número de nós expandidos durante um episódio de planejamento dificilmente será o máximo permitido quando o agente se encontra próximo ao objetivo. Além disso, podemos verificar que, em média, os tempos dos episódios de planejamento dos

algoritmos configuráveis são bastante parecidos, o que indica que nossa configuração realizada experimentalmente foi boa.

4.2 Otimalidade da solução encontrada

Nessa seção analisamos a qualidade da solução de cada algoritmo implementado. Ela corresponde ao custo do caminho obtido por um algoritmo ao resolver um problema de planejamento de caminhos. Essa métrica é importante para qualquer algoritmo de busca heurística, tanto em jogos digitais como em diversas outras aplicações.

As figuras 4.3 a 4.14 mostram a otimalidade da solução encontrada pelos algoritmos implementados nos mapas **combat**, **duskwood**, **hrt201n** e **lak503d**. Cada gráfico apresenta os resultados referentes a um mapa com certa visibilidade. Os resultados mostrados para certo algoritmo em certo mapa são a média dos resultados obtidos pelo algoritmo na resolução de todos os problemas do mapa em questão. Além disso, os resultados são exibidos em relação ao custo da solução ótima. Por exemplo: um algoritmo que apresente, para certo mapa, um custo proporcional de valor 10 encontra caminhos, em geral, dez vezes maiores que os ótimos. Um algoritmo que apresenta custo de valor proporcional 1 encontra a solução ótima.

Os resultados mostrados pelo LRTA* estão, na maioria dos casos, saturados nos gráficos. Essa decisão foi tomada pois os resultados do LRTA* são bastante inferiores, e a exibição completa prejudicaria a visualização dos gráficos. Apresentamos, no entanto, o valor numérico dos resultados obtidos pelo LRTA* ao lado de cada barra que não foi exibida de forma completa. As tabelas utilizadas na geração dos gráficos são exibidas no Apêndice A.

Vale citar também que o *D* Lite*, apesar de ser mostrado nos gráficos, é diferente dos demais algoritmos, uma vez que ele não atende restrições de tempo real. Dessa forma, o tempo do episódio de planejamento desse algoritmo não respeita os limites impostos. Exibimos os resultados do *D* Lite* para que o leitor possa ter uma base de comparação da qualidade das soluções obtidas, uma vez que esse algoritmo é largamente conhecido e utilizado. Por fim, os resultados obtidos pelo TBA* são exibidos somente nos gráficos que mostram os resultados em ambientes completamente observáveis, uma vez que tal algoritmo funciona somente nesse tipo de ambiente. Novamente, o TBA* é também utilizado principalmente para dar ao leitor uma base de comparação.

Antes de iniciar nossa análise vamos fazer uma observação sobre os mapas. Observando os gráficos de otimalidade da solução e as imagens dos mapas mostradas na Figura 3.1, podemos perceber que é possível dividir os mapas utilizados de acordo com

sua dificuldade: mapas fáceis, intermediários e difíceis. O mapa **combat** é um mapa fácil. Ele é um mapa aberto e com poucos obstáculos, onde a heurística inicial utilizada é bem informada. O mapa **duskwood** é um mapa intermediário. Ele possui muitos obstáculos, mas potencialmente poucas depressões heurísticas. Como já citado, depressões heurísticas influenciam bastante no comportamento de algoritmos de busca de tempo real. Finalmente, os mapas **hrt201n** e **lak503d** são mapas difíceis. Além da grande quantidade de obstáculos também existe uma grande quantidade de possíveis depressões heurísticas.

Vamos iniciar nossa análise comparando a qualidade das soluções obtidas pelos diversos algoritmos, a fim de verificar quais apresentam os melhores resultados e quais mostram resultados ruins.

Em relação aos algoritmos com resultados ruins, podemos ver que o LRTA* encontra as soluções de pior qualidade entre todos os algoritmos testados, para qualquer visibilidade usada e em qualquer um dos mapas. Ele chega a encontrar soluções 450 vezes piores que a ótima no mapa **lak503d** e 190 vezes no **hrt201n**, como pode ser visto nas figuras 4.9 e 4.12. Esse resultado era esperado, uma vez que o LRTA* é o algoritmo mais simples entre todos os implementados. Vale notar, no entanto, que, para mapas fáceis (como o **combat**), os resultados obtidos pelos LRTA* não são muito piores que aqueles obtidos pelos demais algoritmos. Nesse tipo de mapa os resultados encontrados por todos os algoritmos são basante parecidos. O LRTA* encontra, em média, resultados $1,5$ vezes piores que os ótimos, enquanto outros algoritmos encontram resultados em geral $1,1$ vezes piores, como mostrado na Figura 4.4. Os outros algoritmos que obtiveram soluções de pior qualidade foram o PRTA* (outro algoritmo bastante simples), o P-LRTA* e o LRTA*(k). No mapa **hrt201n**, por exemplo, esses algoritmos chegaram a encontrar resultados 23, 15 e 10 vezes piores que os ótimos, respectivamente. Os dois últimos algoritmos citados são focados em melhorar o tempo de convergência da solução e não em encontrar uma boa primeira solução, o que explica os resultados obtidos. Utilizando um tempo máximo de episódio de planejamento baixo, o P-LRTA* é pior que o LRTA*(k) e o PRTA*. No entanto, com um aumento da visibilidade e do tempo do episódio de planejamento, ele acaba superando os outros dois, que não oferecem melhorias com tais alterações. Esse comportamento pode ser verificado, por exemplo, na Figura 4.12. Quando o tempo máximo do episódio de planejamento é de $0,025$ segundos, o P-LRTA* é pior que o LRTA*(k) e o PRTA*. Já quando esse tempo é de $0,100$ segundos, o P-LRTA apresenta resultados melhores que os outros dois algoritmos. Além disso, no mapa **combat**, o TBAA* apresentou, especialmente para tempos baixos do episódio de planejamento, resultados piores que os demais algoritmos. Esse resultado mostra que, para mapas simples, algoritmos menos

complexos podem exibir soluções melhores que algoritmos mais complexos.

É interessante notar também como as soluções do PRTA* foram superiores às do LRTA*, mesmo com os dois algoritmos sendo bastante simples e executando cada episódio de planejamento muito rapidamente. A causa desse resultado é que o PRTA* é muito mais rápido que o LRTA* para sair de depressões heurísticas, uma vez que ele visita cada estado da depressão somente uma vez. Sua desvantagem, no entanto, é não possuir um processo de convergência até a solução ótima depois de várias execuções.

Analisando agora os algoritmos que apresentaram os melhores resultados podemos ver que o LSSLRTA*, RTAA* e TBAA* mostraram, no geral, melhor qualidade de solução. Em ambientes desconhecidos (com visibilidade 1) o RTAA* apresenta os melhores resultados para quase todos os mapas, independentemente do tempo máximo de planejamento usado, como mostrado, por exemplo, na Figura 4.12. Somente no mapa **lak503d** o TBAA* apresenta resultados melhores (Figura 4.9). O LSSLRTA*, apesar de apresentar soluções com custo próximo ao dos outros dois algoritmos, se mostrou pior que eles na maioria dos testes.

Nos mapas parcialmente observáveis, novamente, o RTAA* foi melhor na maioria dos testes. O TBAA* se mostrou melhor somente em dois mapas (**lak503d**, Figura 4.10 e **combat**, Figura 4.4), e somente quando o tempo máximo dos episódios de planejamento foi maior ou igual a $0,075$s. Nesses mapas, com o tempo em $0,025$s e $0,050$s, o LSSLRTA* também foi superior ao TBAA* em parte dos testes. É interessante notar que isso ocorreu pois, aumentado a visibilidade do mapa de 1 para 10, o TBAA* teve uma visível perda nos resultados para tempos baixos de episódio de planejamento. Esse comportamento é discutido em detalhes posteriormente.

Em mapas completamente observáveis, o TBAA* apresentou os melhores resultados, como mostrado, por exemplo, na Figura 4.8. Podemos ver também que o LSSLRTA* e o RTAA* não se beneficiaram muito do conhecimento completo do ambiente. Na maioria dos casos a qualidade da solução obtida piorou passando de mapas parcialmente observáveis para completamente observáveis. Intuitivamente, por reiniciar um A* com número de expansões limitadas a cada episódio de planejamento, esses algoritmos não exploram áreas muito distantes da posição corrente do agente, de forma que uma visão completa do ambiente, muitas vezes, não fornece mais informações que uma visão parcial. Além disso, como mostraremos a seguir, a informação extra obtida pelo aumento da visibilidade pode prejudicar o LSSLRTA* e o RTAA*.

Vamos analisar agora o comportamento dos algoritmos com o aumento na visibilidade dos mapas. O LRTA*, PRTA* e LRTA*(k) não apresentam melhoria na qualidade da solução conforme a visibilidade dos mapas aumenta. Isso ocorre pois esses algoritmos escolhem o próximo estado a ser visitado usando como base somente

os vizinhos imediatos do estado corrente do agente. Dessa forma, sua escolha independente da visibilidade, uma vez que os vizinhos imediatos do estado corrente são sempre visíveis.

O P-LRTA* se beneficia do aumento na visibilidade, especialmente em mapas difíceis. No mapa **hrt201n**, por exemplo, a qualidade da solução melhorou em média 18% na passagem de ambiente parcialmente observável para completamente observável. Nesse mesmo mapa, a melhoria da solução na passagem do ambiente desconhecido para parcialmente observável foi, em média, de 19%. Em alguns casos mais incomuns, no entanto, a qualidade da solução piorou um pouco. Por exemplo, no mapa **duskwood**, aumentado a visibilidade de 10 (parcialmente observável) para infinito (completamente observável), usando o tempo máximo do episódio de planejamento como $0,025$s, a qualidade média da solução piorou. Essa alteração foi, no entanto, pouco significativa (a qualidade da solução piorou menos que 1%).

O LSSLRTA*, RTAA*, TBAA* apresentaram os comportamentos mais interessantes e, algumas vezes, não intuitivos. O LSSLRTA*, na mudança de visibilidade de 10 para infinito, piorou a qualidade da solução em todos os mapas, para todos os tempos máximos de episódio de planejamento. Os mapas que apresentaram maior decréscimo na qualidade da solução foram **hrt201n** e **duskwood** com, em média, 10% de piora na qualidade das soluções. Além disso, na mudança de visibilidade de 1 (mapas completamente desconhecidos) para 10, a solução piorou usando $0,025$s como tempo máximo do episódio de planejamento em alguns mapas, como **duskwood**. Nesse caso, a qualidade da solução piorou cerca de 26%. O RTAA* apresentou o mesmo comportamento, porém em menor escala. Passando a visibilidade de 10 para infinito não houve uma piora em todos os casos, mas a qualidade da solução mostrou decréscimos, especialmente para tempos máximos de episódio de planejamento baixos. No mapa **lak503d**, por exemplo, a qualidade da solução caiu em 20% usando $0,050$s como tempo máximo do episódio de planejamento. Finalmente, o TBAA* apresentou resultados bastante diferentes dos demais algoritmos. Passando de mapas sem nenhuma visibilidade para mapas parcialmente observáveis, é possível notar uma clara queda na qualidade das soluções para tempos máximos de episódio de planejamento baixos. No mapa **duskwood**, usando $0,025$s como tempo máximo do episódio de planejamento, por exemplo, a qualidade da solução piorou 226%. Passando de mapas parcialmente observáveis para completamente observáveis, no entanto, a qualidade das soluções aumenta drasticamente. Para todos os mapas é possível observar um aumento de pelo menos 80% na qualidade das soluções.

Realizando alguns estudos sobre o comportamento apresentado pelo LSSLRTA* e RTAA*, foi possível concluir que ele é causado, em ambos os algoritmos, pela mesma

razão: o uso de *lookaheads*, que definem o tempo máximo do episódio de planejamento, pequenos com visibilidades grandes pioram o comportamento dos algoritmos na maioria dos casos (em mapas completamente observáveis o *lookahead* é quase sempre bastante pequeno em relação à visibilidade). Isso acontece porque, nesses casos, o agente controlado por tais algoritmos apresenta maior dificuldade para sair de depressões heurísticas. A Figura 4.2 apresenta um exemplo onde um aumento de visibilidade prejudicará o LS-SLRTA* e o RTAA*. Nessa figura o agente é representado pelo ponto vermelho, e seu objetivo pelo ponto amarelo. A área clara ao redor do agente representa sua visibilidade, enquanto a área escura é desconhecida. Os obstáculos no mapa são representados pela cor verde. O agente considera que todos os estados na área escura estão livres. No ambiente apresentado, utilizando um *lookahead* pequeno, o agente se locomoverá nas bordas da depressão heurística e sairá dela com facilidade. Caso o ambiente se torne totalmente observável, utilizando o mesmo *lookahead* pequeno, o agente ficará preso na depressão heurística. Será necessário atualizar o valor de todos os estados na depressão antes que o agente saia e alcance o estado objetivo. Um exemplo desse comportamento pode ser visto em `https://vimeo.com/88695267`.

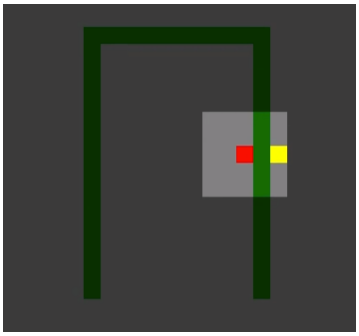

Figura 4.2. Problema de planejamento de caminhos em um ambiente parcialmente observável. O agente (quadrado vermelho) deve caminhar até seu objetivo (quadrado amarelo). A área escura é desconhecida pelo agente.

O comportamento do TBAA* na passagem de mapas desconhecidos para parcialmente observáveis com baixos tempos máximos de episódio de planejamento também foi estudado em detalhes. Foi possível concluir que, nesses casos, o agente controlado pelo TBAA* tem maior dificuldade para desviar de obstáculos. Um aumento na visibilidade com um *lookahead* pequeno faz com que sejam necessários vários episódios de planejamento até que o A* executado pelo TBAA* consiga contornar certo obstáculo. O obstáculo será visível, e o A* não expandirá nós suficientes para contorná-lo. A busca acaba concentrada atrás do obstáculo, prejudicando o agente. Por outro lado, com uma visibilidade muito baixa e um *lookahead* baixo, a suposição do espaço vazio

fará com que o A* executado contorne o obstáculo, mesmo que erroneamente (passando por um estado potencialmente ocupado). Isso fará com que o agente acabe andando próximo ao obstáculo até se desviar. Um exemplo desse comportamento pode ser visto em `https://vimeo.com/88695549`.

Terminaremos nossa análise verificando o comportamento de cada algoritmo em diferentes mapas com o aumento do tempo máximo do episódio de planejamento. Podemos observar que:

- Por não possuírem um parâmetro de ajuste do tempo de planejamento, o PRTA* e o LRTA* apresentam os mesmos resultados para todos os tempos máximos de episódio de planejamento. Em compensação, o tempo utilizado por eles em cada episódio de planejamento é sempre muito inferior ao limite, como mostrado na Seção 4.1.

- Como já citado na Seção 2.4.5, o LRTA*(k) melhora a solução encontrada e aumenta seu tempo de planejamento quanto maior é o número de atualizações feitas. Como o maior mapa utilizado possui tamanho 512×512, o número máximo de atualizações seria, no pior caso, $512 * 512 = 262144$, o que poderia ocorrer se os valores de todos os estados do mapa fossem atualizados. No entanto, utilizando $k = 262144$, o tempo máximo do episódio de planejamento do LRTA*(k) não ultrapassou o limite de tempo mínimo que utilizamos, que foi $0,025$s. Dessa forma, os resultados mostrados pelo LRTA*(k) são iguais para todos os tempos utilizados, uma vez que o valor de k utilizado foi o maior possível em todos os testes. O tempo do episódio de planejamento desse algoritmo, no entanto, foi sempre bastante baixo.

- O LSSLRTA*, RTAA*, TBAA* e P-LRTA* apresentaram, na maioria das vezes, o comportamento esperado: quanto maior é o tempo máximo de planejamento (e, consequentemente, o numero de operações realizadas) melhor é a qualidade da solução obtida. No entanto, em alguns casos, esse aumento causou uma piora na qualidade da solução encontrada. Por exemplo, no mapa **lak503d** com visibilidade 1, ao passar o tempo máximo do episódio de planejamento de $0,075$s para $0,100$s a solução encontrada pelo P-LRTA* piorou, como pode ser visto na Figura 4.9. Apesar de inesperado, esse resultado não é incorreto: nenhum desses algoritmos garante que um aumento no número de operações realizadas irá necessariamente causar um aumento na qualidade da solução encontrada (embora isso aconteça na maioria das vezes). É possível que as atualizações extras possam, em

algum momento, guiar o agente para uma área do mapa que parece promissora mas que acaba se mostrando ruim.

Sumarizando as discussões dessa seção, temos que o LRTA* apresentou as soluções de pior qualidade, para qualquer tipo de mapa. Quanto mais difícil é o mapa utilizado e maior é o tempo máximo do episódio de planejamento permitido, maior é a diferença da qualidade obtida por ele em relação aos demais algoritmos. Apresentando resultados bastante competitivos na maioria dos testes, o RTAA* e o TBAA* foram, no geral, os algoritmos com melhor qualidade de solução. Em mapas desconhecidos e parcialmente observáveis o RTAA* foi melhor na maioria dos testes. Em mapas completamente observáveis, no entanto, o TBAA* apresentou os melhores resultados. No mapa fácil utilizado a maioria dos algoritmos funciona bem e, em vários casos, algoritmos mais complexos, como o TBAA*, apresentam resultados piores que os demais. Em mapas com poucos obstáculos, portanto, é interessante o uso de um algoritmo mais simples. Verificamos também que alguns algoritmos nem sempre se beneficiam do aumento da visibilidade dos mapas, e que esse fator impacta os diferentes algoritmos de diferentes formas. Por fim, vimos que a maioria dos algoritmos de busca de tempo real se beneficia de um aumento do tempo máximo do episódio de planejamento, apesar de que, em alguns casos, é possível verificar quedas na qualidade da solução.

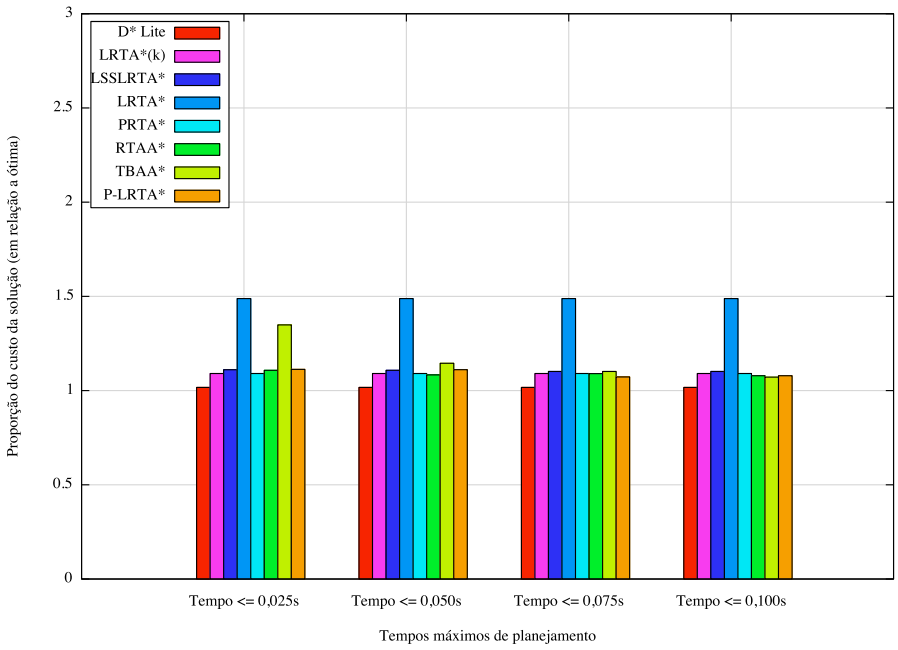

Figura 4.3. Qualidade da solução para o mapa **combat** com visibilidade = 1.

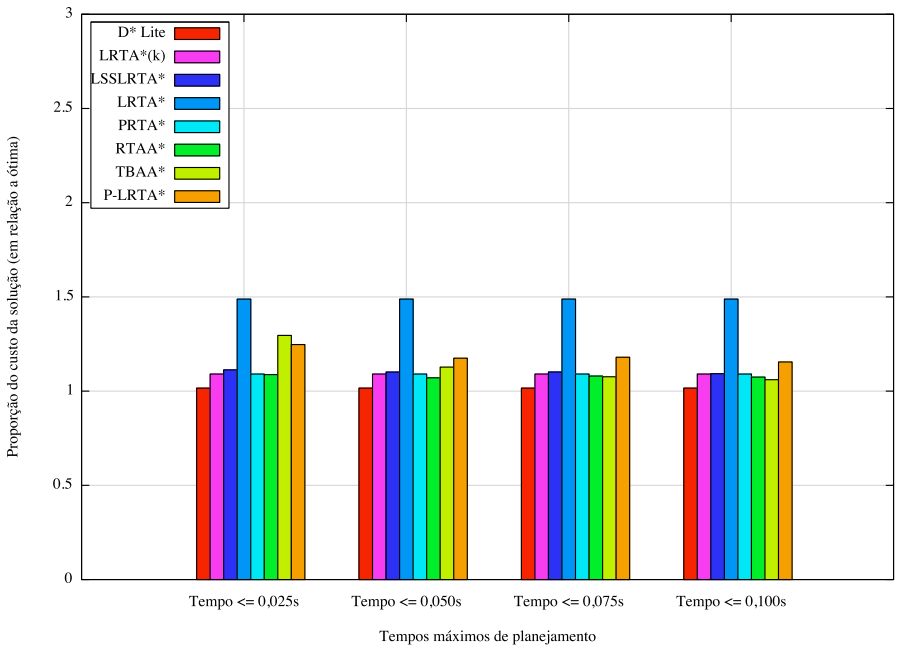

Figura 4.4. Qualidade da solução para o mapa **combat** com visibilidade = 10.

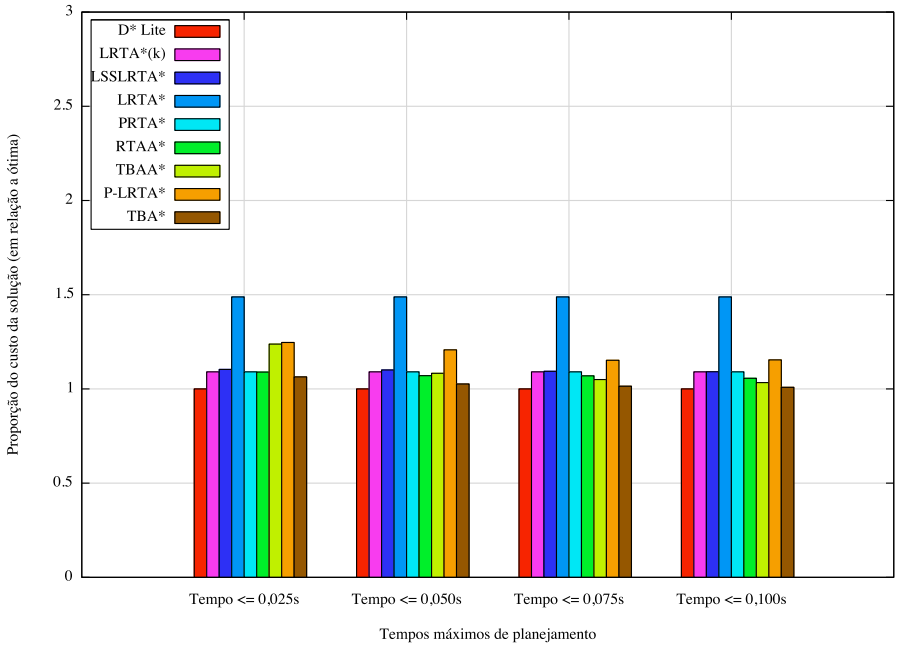

Figura 4.5. Qualidade da solução para o mapa **combat** com visibilidade = ∞.

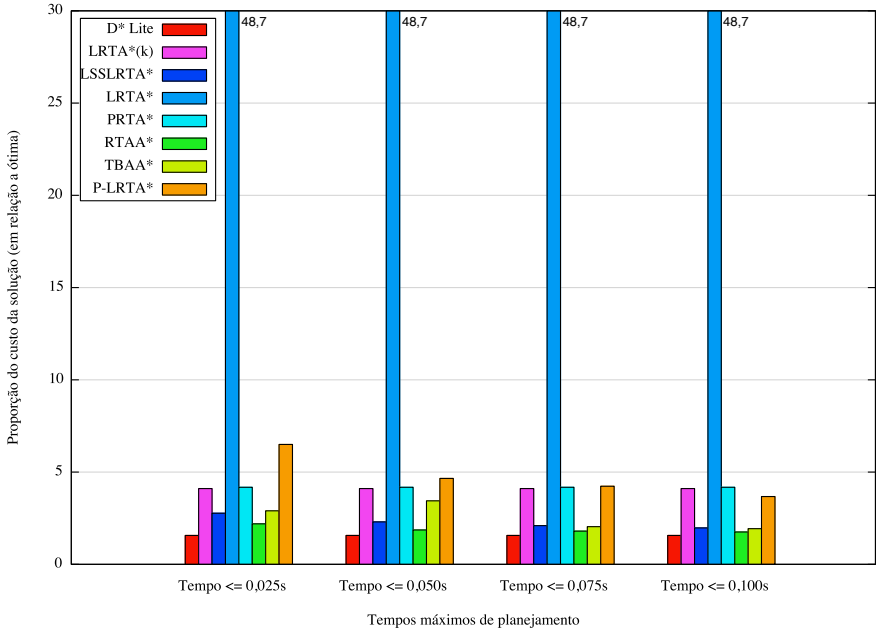

Figura 4.6. Qualidade da solução para o mapa **duskwood** com visibilidade = 1.

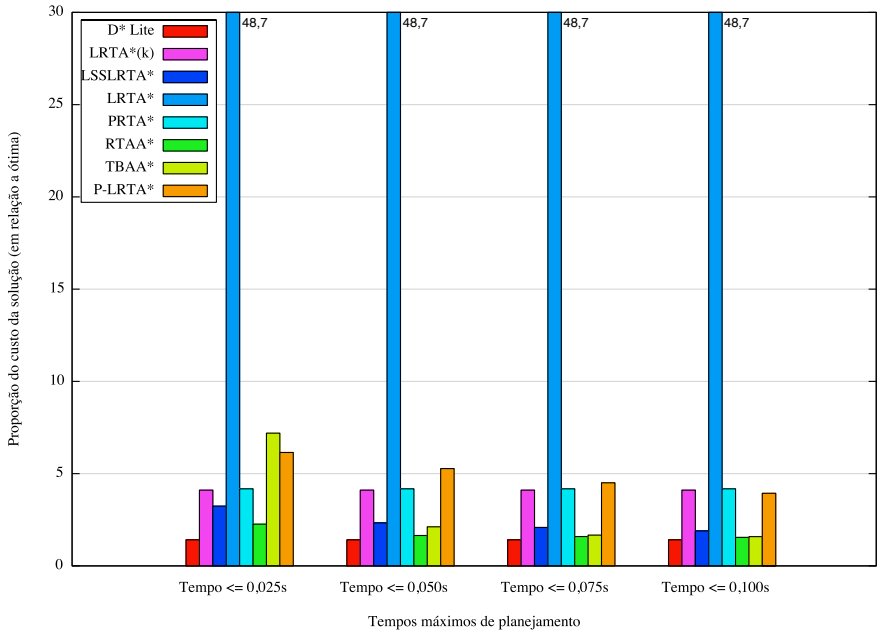

Figura 4.7. Qualidade da solução para o mapa **duskwood** com visibilidade = 10.

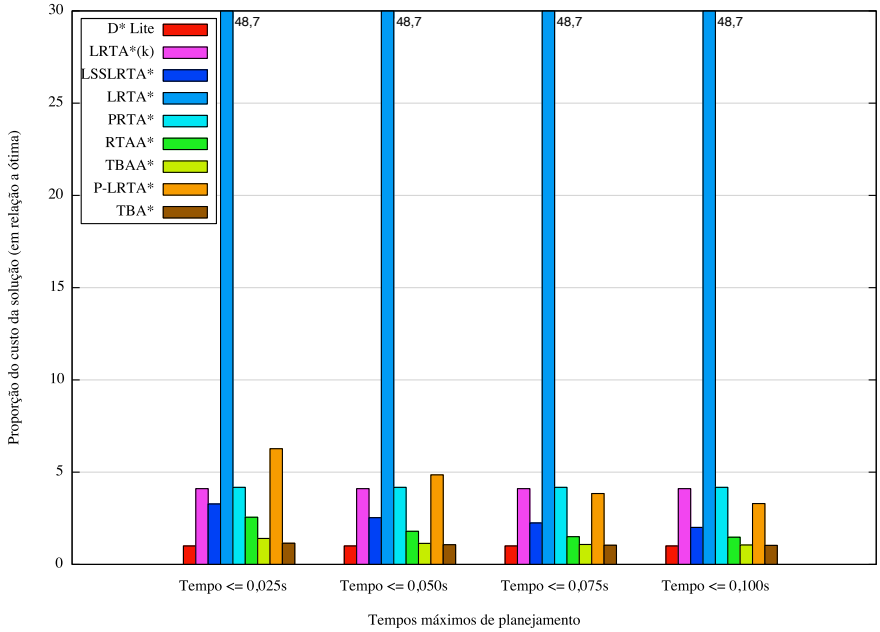

Figura 4.8. Qualidade da solução para o mapa **duskwood** com visibilidade = ∞.

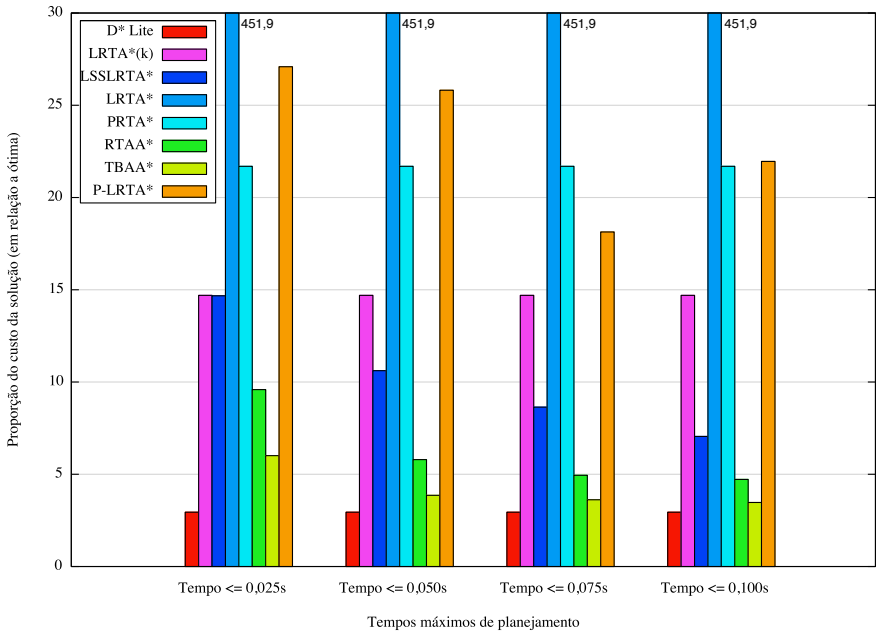

Figura 4.9. Qualidade da solução para o mapa **lak503d** com visibilidade = 1.

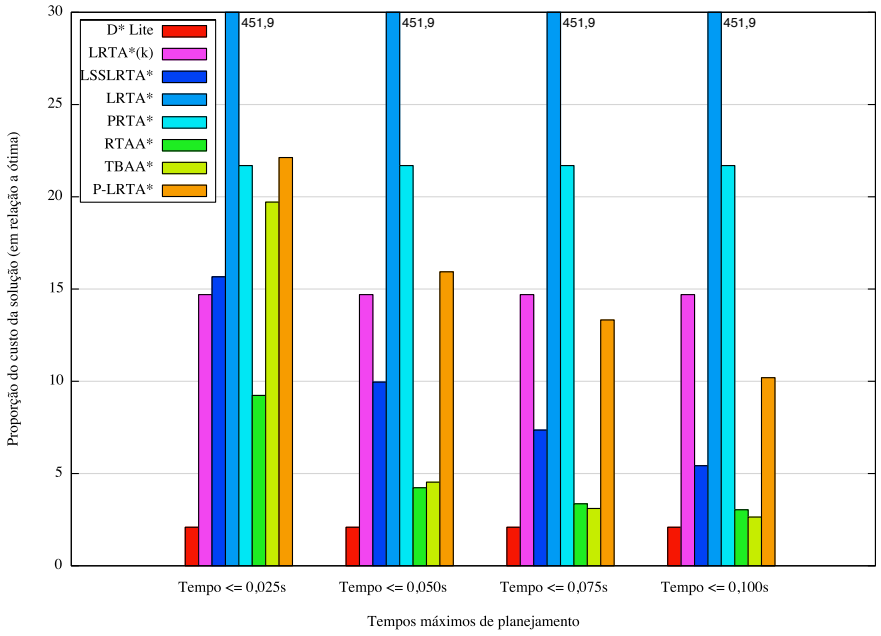

Figura 4.10. Qualidade da solução para o mapa **lak503d** com visibilidade = 10.

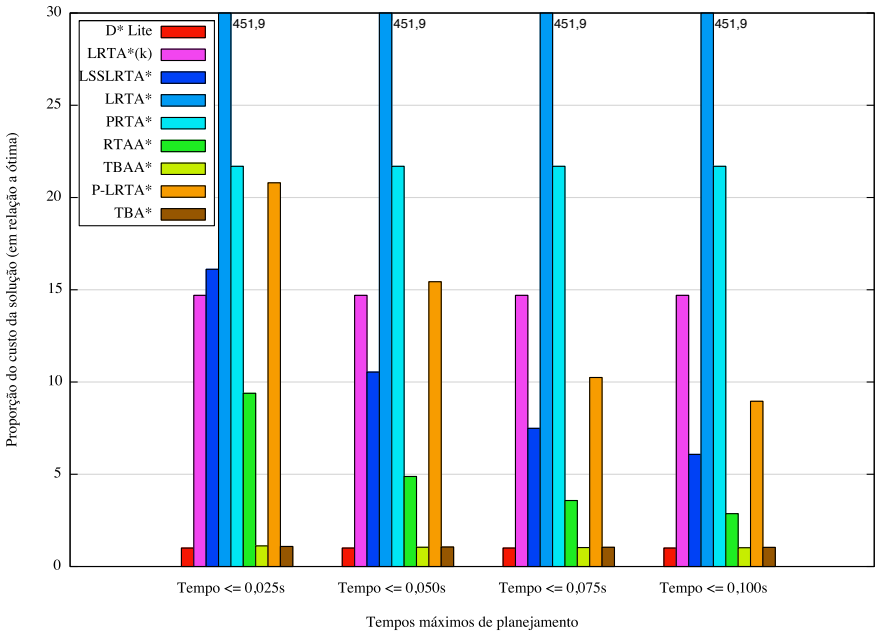

Figura 4.11. Qualidade da solução para o mapa **lak503d** com visibilidade = ∞.

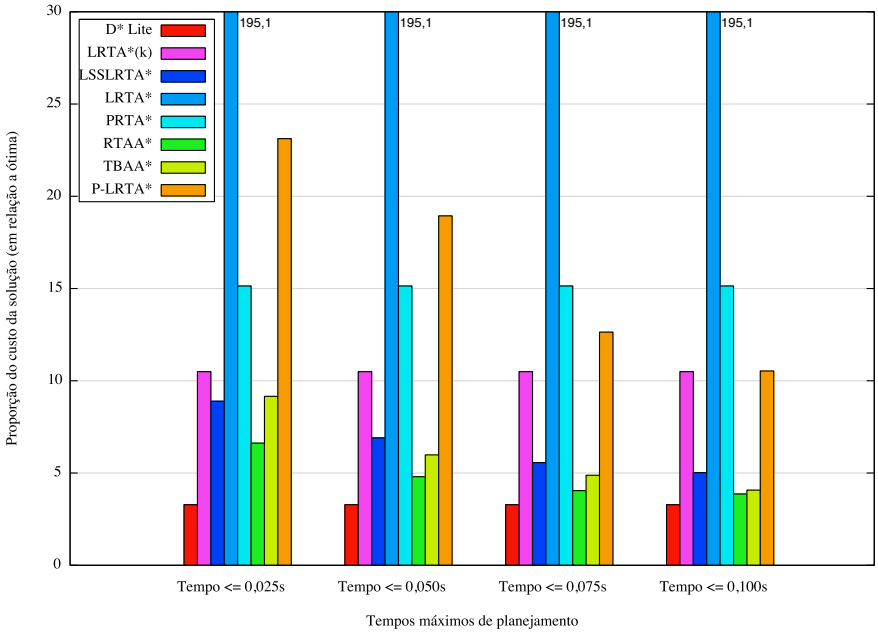

Figura 4.12. Qualidade da solução para o mapa **hrt201n** com visibilidade = 1.

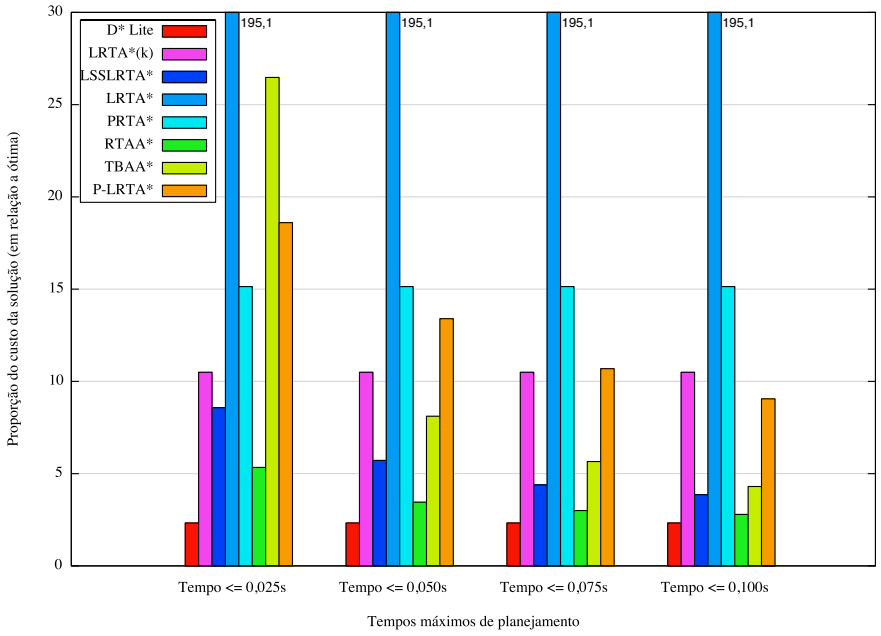

Figura 4.13. Qualidade da solução para o mapa **hrt201n** com visibilidade = 10.

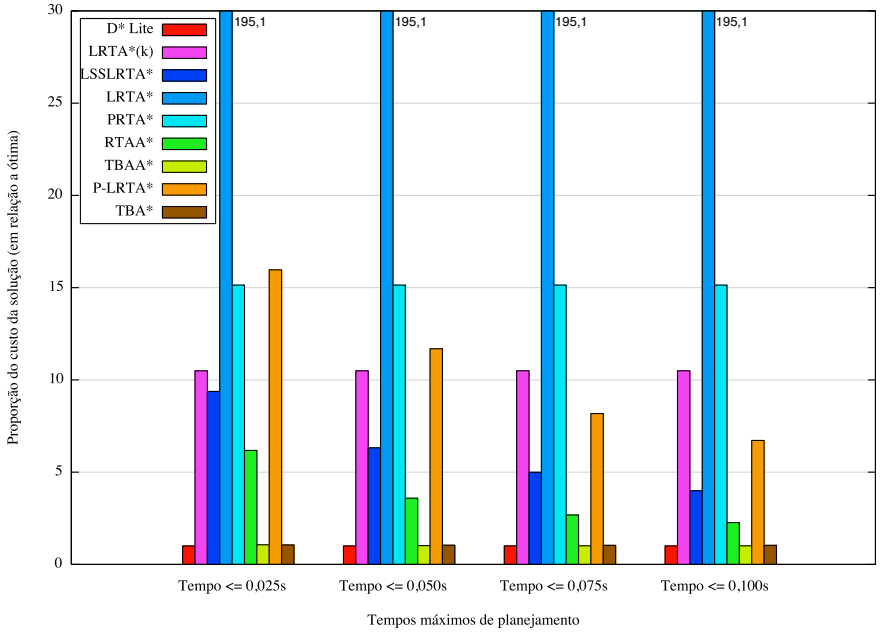

Figura 4.14. Qualidade da solução para o mapa **hrt201n** com visibilidade = ∞.

4.3 Tempo total de planejamento

Nessa seção analisamos o tempo total de planejamento (ou tempo total de busca) de cada algoritmo implementado. Esse tempo consiste na soma do tempo de todos os episódios de planejamento de um certo algoritmo. Como já citamos, nem todos os algoritmos utilizam o tempo máximo do episódio de planejamento a cada iteração. Além disso, cada um deles gasta um número diferente de episódios de planejamento até alcançar o estado objetivo. Essa métrica, portanto, mostra qual algoritmo utiliza mais tempo de CPU para chegar do estado inicial ao estado objetivo pela primeira vez. Em jogos digitais, pode ser interessante optar por um algoritmo com menor tempo total de busca em situações onde temos, por exemplo, que realizar o planejamento de caminhos para um número muito grande de agentes. Essa análise também é importante para que seja possível verificar o *tradeoff* entre o tempo total de busca e a qualidade da solução obtida.

As figuras 4.15 a 4.26 mostram o tempo total de busca de cada algoritmo nos mapas **combat**, **duskwood**, **hrt201n** e **lak503d**. Cada gráfico apresenta os resultados referentes a um mapa com certa visibilidade. Os resultados mostrados para certo algoritmo com certo tempo máximo de episódio de planejamento são a média dos resultados obtidos pelo algoritmo na resolução de todos os problemas do mapa em questão. Os tempos mostrados estão em segundos. Os testes foram feitos em um computador com processador Intel Core i3-550 e 4 Gigabytes de memória RAM.

Os resultados mostrados pelo P-LRTA* estão, em alguns casos, saturados nos gráficos, como os resultados mostrados pelo LRTA* na seção anterior. O valor dos resultados saturados é, novamente, exibido nos gráficos. As tabelas utilizadas na geração dos gráficos estão no Apêndice A.

Iniciaremos nossa análise pelas figuras 4.15, 4.18, 4.21 e 4.24, que mostram os resultados para mapas completamente desconhecidos.

Comparando o tempo total de busca de todos os algoritmos em ambientes completamente desconhecidos podemos perceber que, em mapas fáceis (**combat**, Figura 4.15) o *D* Lite* e o P-LRTA* são os algoritmos que gastam o maior tempo. O P-LRTA* chega a gastar, em média, $5,6$ segundos para encontrar soluções, um número alto, uma vez que os outros algoritmos encontram soluções em menos de 1 segundo na maioria das vezes. Isso acontece porque, nesse tipo de mapa, existem poucos obstáculos e, geralmente, se mover na direção mais promissora depois de um curto planejamento é um bom movimento. Dessa forma, algoritmos que gastam mais tempo realizando um planejamento mais complexo (como o *D* Lite*) ou atualizando muitos estados (como o P-LRTA*) acabam se prejudicando. Em mapas intermediários (**duskwood**, Fi-

gura 4.18) os algoritmos com maior tempo total de busca são o P-LRTA*, gastando 74 segundos para o maior tempo máximo de episódio de planejamento, o *D* Lite*, gastado 18 segundos, e o TBAA*, com 20 segundos para o maior tempo máximo de episódio de planejamento. O *D* Lite* e o TBAA* são os algoritmos que mais realizam operações, o que explica o maior tempo. Já o P-LRTA*, além de realizar um grande número de operações, também gasta uma grande quantidade de episódios de planejamento para alcançar o objetivo (a qualidade da solução desse algoritmo é, geralmente, ruim). Já em mapas difíceis (**hrt201n**, Figura 4.24 e **lak503d**, Figura 4.21) o P-LRTA* e TBAA*, novamente, apresentam o maior tempo total de busca na maioria das vezes, com o P-LRTA* chegando a apresentar 482 segundos como tempo total de busca no mapa **hrt201n**. Nesse tipo de mapa é possível perceber que algoritmos como o LRTA* e o LRTA*(k) também apresentam tempos bastante altos. Isso ocorre pois, em mapas difíceis, as soluções encontradas por eles são ruins. O número extra de episódios de planejamento necessários para encontrar a solução acabam superando o tempo maior de computação gasto por algoritmos como o *D* Lite* e o TBAA*.

O algoritmo que gastou o menor tempo total de busca foi, em todos os mapas, o PRTA*. Esse tempo foi, em média, de 1 segundo. Seu funcionamento simples permite que os episódios de planejamento sejam muito rápidos. Além disso, esse algoritmo não sofre tanto com problemas de depressões heurísticas, uma vez que usa uma estratégia diferente do aprendizado de *h-values*. O LRTA*, por exemplo, aumenta consideravelmente seu tempo total de busca em mapas difíceis, mesmo executando cada episódio de planejamento tão rápido quanto o PRTA*. Isso ocorre porque, nesse tipo de mapa, o agente fica constantemente preso em depressões heurísticas, aumentando muito o número de episódios de planejamento utilizados e, consequentemente, o tempo total de busca.

É interessante notar, observando também os gráficos mostrados na seção anterior, que os algoritmos que apresentam soluções de melhor qualidade possuem também, em geral, maior tempo total de busca. O TBAA* e o *D* Lite* são dois bons exemplos. Um algoritmo, no entanto, claramente não se comporta dessa forma: o P-LRTA* apresentou soluções de qualidade bastante ruins, e também apresentou alguns dos piores tempos totais de busca. Pode-se perceber que as atualizações prioritárias de áreas possivelmente remotas do espaço de busca não se mostram muito boas para obter uma primeira solução de qualidade. Outro algoritmo que se mostrou bastante interessante nos testes foi o RTAA*. Ele apresentou tempos totais de busca intermediários (na maioria das vezes inferiores aos do *D* Lite* e *TBAA**) e soluções de boa qualidade. Para mapas completamente desconhecidos esse algoritmo apresentou um excelente *tradeoff* de tempo total de busca e qualidade da solução.

Ainda em mapas completamente desconhecidos, a respeito do comportamento de cada algoritmo com o aumento do tempo máximo do episódio de planejamento, podemos observar que:

- O LRTA*, *D* Lite*, LRTA*(k) e PRTA* não apresentam variação no tempo total de planejamento para diferentes tempos máximos de episódio de planejamento. Esse resultado era esperado, uma vez que esses algoritmos se comportam da mesma forma para todos os tempos utilizados, já que não possuem um parâmetro de configuração (com exceção do LRTA*(k), que usa o mesmo parâmetro em todos os tempos).

- O LSSLRTA*, o RTAA* e o P-LRTA* têm o tempo total de planejamento aumentado para cada incremento no tempo máximo do episódio de planejamento. Isso significa que, para mapas completamente desconhecidos, a melhoria na qualidade da solução observada pelo aumento do *lookahead* dos algoritmos vem em troca de um aumento do tempo total de planejamento. Esse comportamento pode parecer óbvio mas, como será mostrado adiante, é possível que um aumento de *lookahead* ocasione uma diminuição do tempo total de busca. Isso ocorre se o novo caminho encontrado é suficientemente menor, a ponto de diminuir a soma total dos episódios de planejamento.

- O TBAA* apresentou o comportamento mais irregular entre os algoritmos testados. Na maioria dos casos houve um aumento no tempo total de busca para cada aumento de *lookahead*. No entanto, em alguns casos, o tempo diminuiu. Por exemplo, no mapa **duskwood**, quando o tempo máximo do episódio de planejamento foi de $0,05$s para $0,075$s, o tempo de total de busca caiu cerca de 48%. Esse tempo ainda foi, no entanto, maior que aquele observado quando usamos $0,025$s como tempo máximo do episódio de planejamento. O custo da solução encontrada foi, como mostrado anteriormente, sempre menor. Isso mostra que, em alguns casos, a melhoria da solução foi grande o suficiente para diminuir o tempo total de busca.

Vamos observar agora as figuras 4.16, 4.19, 4.22 e 4.25, que mostram os resultados para mapas parcialmente observáveis.

Comparando o tempo total de busca de todos os algoritmos em ambientes parcialmente observáveis podemos perceber que o comportamento foi parecido com aquele obtido em mapas desconhecidos: em mapas fáceis (**combat**, Figura 4.16), o *D* Lite* e o P-LRTA* são, novamente, os algoritmos que gastam maior tempo. Eles gastam, respectivamente, $3,4$s e $6,5$s em média para encontrar soluções usando $0,100$s como tempo

máximo do episódio de planejamento, enquanto os demais algoritmos gastam menos de $1,5$s na mesma situação. Em mapas intermediários (**duskwood**, Figura 4.19) os algoritmos com maior tempo total de busca são o P-LRTA*, o *D* Lite* e o TBAA*. Eles apresentam tempos médios de $78,6$s, $18,3$s e $20,7$s, respectivamente, para encontrar soluções usando $0,100$s como tempo máximo do episódio de planejamento. Na mesma situação, os demais algoritmos gastam, em média, menos de 6 segundos para encontrar soluções. Já em mapas difíceis (**hrt201n**, Figura 4.25 e **lak503d**, Figura 4.22) o P-LRTA* foi, novamente, o algoritmo de maior tempo total de busca, chegando a apresentar o tempo total de busca médio superior a 154 segundos no mapa **lak503d**, um valor cerca de 5 vezes maior que o segundo algoritmo mais lento.

O menor tempo total de busca foi, novamente, do PRTA* para todos os mapas. A diferença desse tempo em relação ao dos outros algoritmos, no entanto, diminuiu. A razão disso é que o PRTA* não se beneficia do aumento de visibilidade.

Com relação à qualidade da solução em relação ao tempo total de busca, as mesmas obervações feitas anteriormente permanecem válidas para mapas parcialmente observáveis. O TBAA* e o *D* Lite* apresentam bons resultados, e possuem um tempo total de busca relativamente alto. O RTAA* continuou apresentando um excelente *tradeoff* de qualidade da solução em relação ao tempo total de busca.

A respeito do comportamento de cada algoritmo em diferentes mapas parcialmente observáveis com o aumento do tempo máximo do episódio de planejamento, podemos observar que:

- O LRTA*, *D* Lite*, LRTA*(k) e PRTA* se comportam da mesma forma que para mapas completamente desconhecidos: não apresentam variação no tempo total de planejamento.

- O P-LRTA*, novamente, teve o tempo total de busca aumentado para cada incremento no tempo máximo de episódio de planejamento.

- O TBAA*, RTAA* e LSSLRTA* apresentaram, dessa vez, um comportamento irregular. Em vários casos um aumento no tempo máximo de planejamento (que permite um aumento no *lookahead* desses algoritmos) causou uma diminuição no tempo total de busca. No mapa **lak503d**, por exemplo, os três algoritmos tiveram uma diminuição no tempo total de busca ao passar o tempo máximo do episódio de planejamento de $0,025$s para $0,05$s. Podemos perceber que, para mapas parcialmente observáveis, um aumento no *lookahead* parece ser mais vantajoso do que para mapas desconhecidos. Intuitivamente, o planejamento com um *lookahead* maior será mais correto caso o agente tenha conhecimento de parte

do ambiente, o que poderá levar a soluções melhores e, algumas vezes, a um tempo total de busca menor.

Finalmente, vamos observar agora as figuras 4.17, 4.20, 4.23 e 4.26, que mostram os resultados para mapas completamente observáveis.

Comparando o tempo total de busca de todos os algoritmos em ambientes completamente observáveis, podemos perceber uma diferença em relação às demais visibilidades: o TBAA* não é mais um dos algoritmos com maior tempo total de busca. O P-LRTA* e *D* Lite* apresentam o maior tempo total de busca para mapas fáceis (Figura 4.17) e intermediários (Figura 4.20). Em mapas difíceis (figuras 4.23 e 4.26), os maiores tempos são do P-LRTA*, LRTA* e LRTA*(k).

Para o menor tempo total de busca também houve uma mudança em relação aos outros tipos de mapa. O PRTA* foi o algoritmo com menor tempo total de busca somente para os mapas fáceis e intermediários. Nos mapas difíceis, o TBAA* e o TBA* apresentaram menores tempos. Isso ocorre pois o TBAA* se beneficia muito do conhecimento completo do ambiente, enquanto o TBA* é um algoritmo bastante eficiente para ambientes completamente observáveis.

Em mapas completamente observáveis, a relação da qualidade da solução e do tempo total de busca é bastante interessante: o TBA* e TBAA*, alguns dos algoritmos com melhores soluções, são aqueles com menor tempo total de busca. Esse comportamento mostra como o TBAA* se beneficia da visibilidade completa do ambiente.

Em mapas completamente observáveis, a respeito do comportamento de cada algoritmo em diferentes com o aumento do tempo máximo do episódio de planejamento, podemos observar que:

- Novamente, o LRTA*, *D* Lite*, LRTA*(k) e PRTA* se comportam da mesma forma: não apresentam variação no tempo total de planejamento.

- O P-LRTA* teve o tempo total de busca aumentado para cada incremento no tempo máximo de episódio de planejamento.

- O TBAA*, RTAA* e LSSLRTA* apresentaram, em vários casos, uma diminuição no tempo total de busca com um aumento no tempo máximo do episódio de planejamento. A ocorrência dessa melhoria foi ainda mais frequente do que a observada em mapas parcialmente observáveis. É possível concluir que, para esses algoritmos, o uso de *lookaheads* é mais aproveitado em mapas com maior visibilidade.

Vamos finalizar nossa análise verificando o comportamento do tempo total de busca dos algoritmos em relação às alterações de visibilidade. Podemos observar que:

- O LRTA*, LRTA*(k) e PRTA* possuem o mesmo tempo total de busca para todas as visibilidades. Esse resultado era esperado, uma vez que esses algoritmos se movem somente com base nos estados vizinhos ao corrente.

- O LSSLRTA* e o RTAA* melhoraram seu tempo total de busca quando passam de mapas que não possuem nenhuma visibilidade para mapas parcialmente observáveis, principalmente devido às melhorias na qualidade da solução. As maiores melhorias foram observadas no mapa **hrt201n**, onde o LSSLRTA* diminuiu o tempo total de busca em cerca de 62%, enquanto o RTAA* obteve diminuições de, em média, 76%. Na passagem para mapas completamente observáveis, no entanto, o tempo se alterou de maneira menos significativa. Também no mapa **hrt201n**, o LSSLRTA* apresentou, em média, um aumento de 3.5% no tempo total de busca, enquanto o RTAA* apresentou diminuições de, em média, 27%. Esse comportamento pode ser explicado pela mesma razão desses algoritmos não apresentarem grandes melhorias na qualidade de solução em mapas completamente observáveis, como explicado na Seção 4.2. Eles utilizam somente uma área ao redor do agente, de forma que a visibilidade completa do ambiente não é muito superior à visibilidade parcial.

- O D* Lite, ao contrário do LSSLRTA* e RTAA*, continua com o tempo praticamente igual na mudança de mapas desconhecidos para parcialmente observáveis. No mapa **hrt201n**, por exemplo, o tempo total de busca diminuiu somente em 0.5% nesse caso. No entanto, em mapas completamente observáveis, seu tempo melhora consideravelmente. Também no mapa **hrt201n**, observam-se diminuições de 80% no tempo total de busca na passagem de mapas parcialmente observáveis para completamente observáveis. Esse algoritmo se beneficia mais de uma maior visibilidade uma vez que, dessa forma, são necessários menos ajustes à rota inicial traçada por ele.

- O P-LRTA*, em geral, melhorou o tempo total de busca para cada aumento de visibilidade. Essa melhoria é mais perceptível em mapas difíceis. No mapa **lak503d**, por exemplo, o tempo total de busca diminuiu, em média, 48% na passagem de visibilidade nula para parcial, e 21% na passagem de visibilidade parcial para completa. Podemos ver que esse algoritmo, em geral, se beneficia do aumento na visibilidade, fazendo atualizações de *h-values* mais corretamente.

- O TBAA* apresentou um comportamento bastante irregular. Na passagem de ambientes desconhecidos para parcialmente observáveis, alguns tempos melhoram e outros pioraram. As pioras foram geralmente observadas na utilização de tempos máximos de episódio de planejamento baixos. Já em mapas completamente observáveis houve somente melhorias, e essas melhorias foram bastante significativas: nos mapas difíceis ela foi de, em média, 96%. Em geral, quanto maior a visibilidade, menos o TBAA* irá encontrar obstáculos inesperados. Dessa forma ele terá que reiniciar seu A* poucas vezes, e conseguirá rapidamente um caminho até o objetivo. No entanto, especialmente para tempos máximos de episódio de planejamento baixos, o TBAA* pode se prejudicar de um pequeno aumento na visibilidade, como explicado na seção anterior, passando a encontrar soluções piores e aumentando seu tempo total de busca.

Sumarizando as discussões dessa seção temos que, no geral, os maiores tempos totais de busca são obtidos pelo P-LRTA*. Especialmente em mapas difíceis, o tempo total de busca é bastante alto. Em mapas fáceis, no entanto, o tempo total de busca do P-LRTA* se equipara ao do D* Lite, sendo algumas vezes até mesmo inferior. É interessante citar também que o TBAA* apresenta, na maioria dos casos, um tempo total de busca bastante alto. Em mapas completamente observáveis, no entanto, o tempo utilizado por ele cai drasticamente. Essa melhora é ainda mais significativa nos mapas difíceis. O algoritmo de menor tempo de planejamento foi, na maioria dos testes, o PRTA*. Apesar de encontrar soluções de qualidade bastante ruim, ele executa cada ação muito rapidamente. Mesmo não sendo muito interessante para o planejamento de caminhos em jogos digitais, esse comportamento pode ser útil se nosso objetivo é chegar rapidamente a uma solução, não se preocupando com as ações tomadas. Em mapas difíceis e completamente observáveis, no entanto, o TBAA* teve o tempo total de busca menor que o do PRTA*. Verificamos também que a maioria dos algoritmos de busca de tempo real aumenta seu tempo total de busca com um aumento do tempo máximo do episódio de planejamento para mapas sem nenhuma visibilidade. Conforme a visibilidade aumenta, no entanto, o aumento no tempo máximo de planejamento passa a melhorar os tempos totais de busca. Foi possível observar também que os diferentes algoritmos apresentam diferentes variações no tempo total de busca conforme a visibilidade dos mapas é alterada.

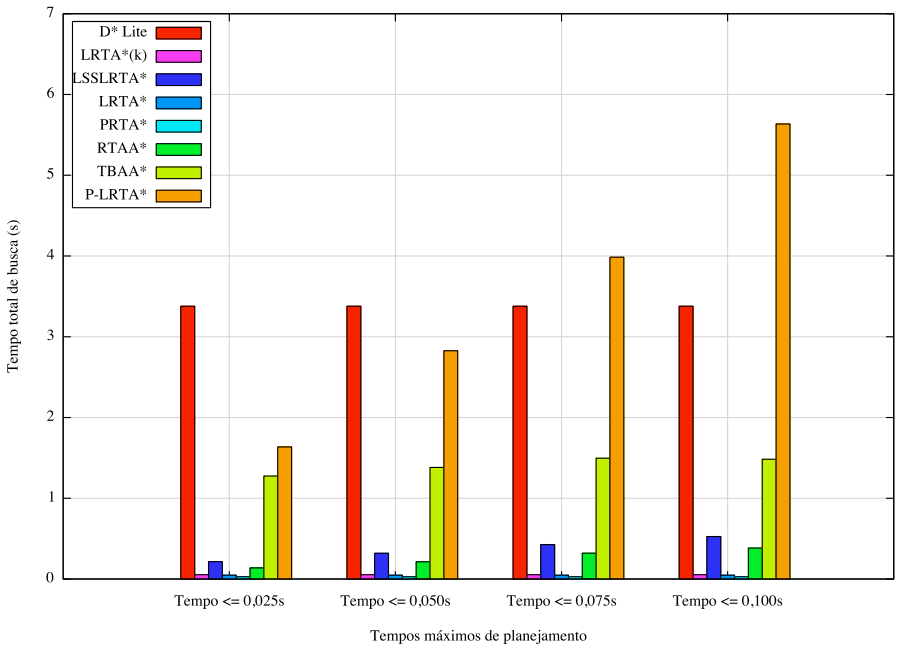

Figura 4.15. Tempo total de busca para o mapa **combat** com visibilidade = 1.

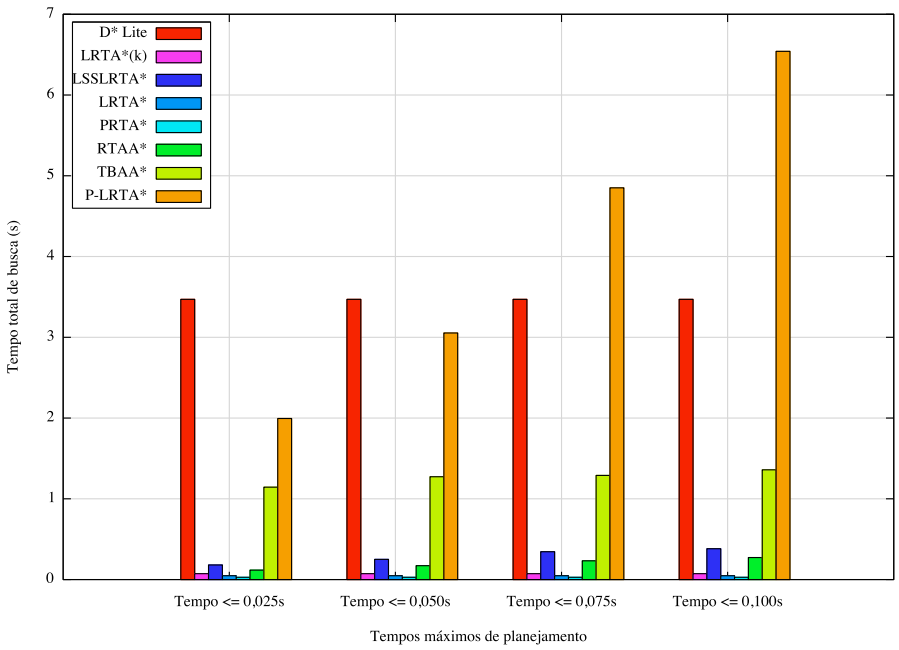

Figura 4.16. Tempo total de busca para o mapa **combat** com visibilidade = 10.

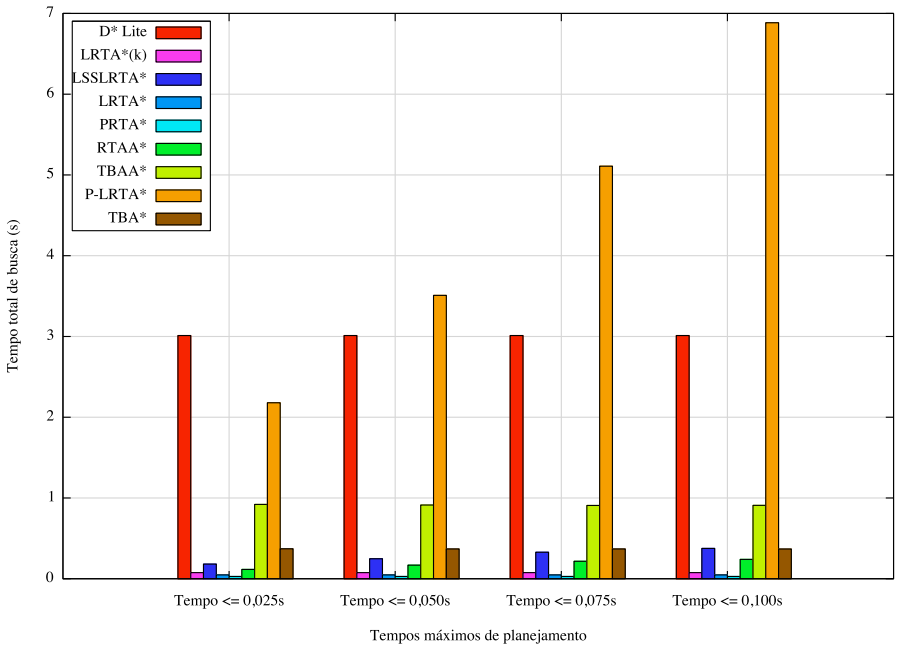

Figura 4.17. Tempo total de busca para o mapa **combat** com visibilidade = ∞.

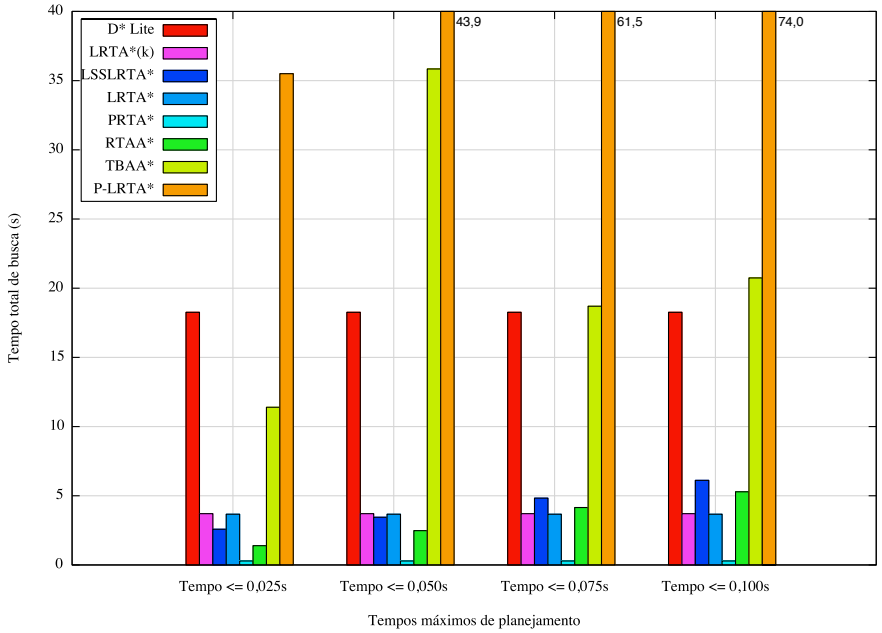

Figura 4.18. Tempo total de busca para o mapa **duskwood** com visibilidade = 1.

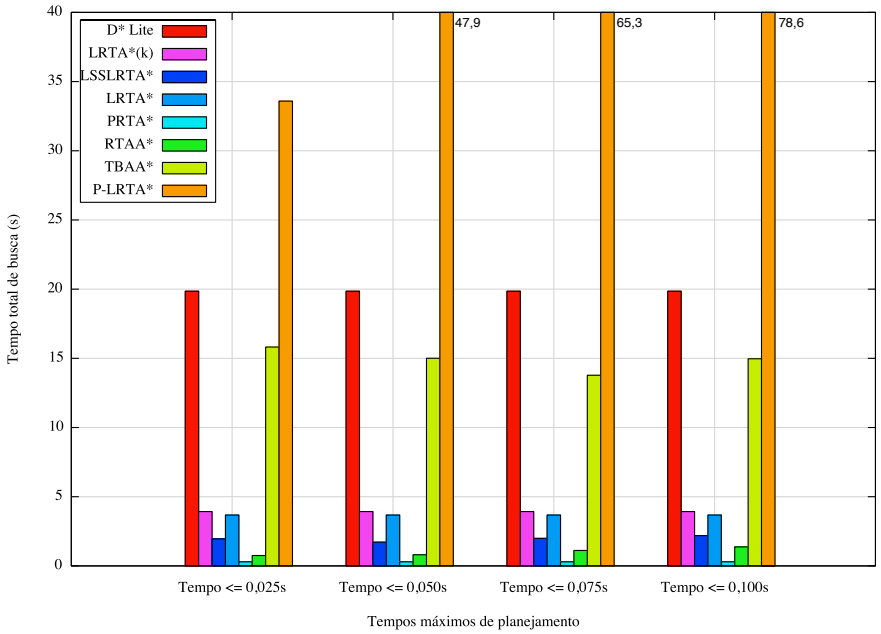

Figura 4.19. Tempo total de busca para o mapa **duskwood** com visibilidade = 10.

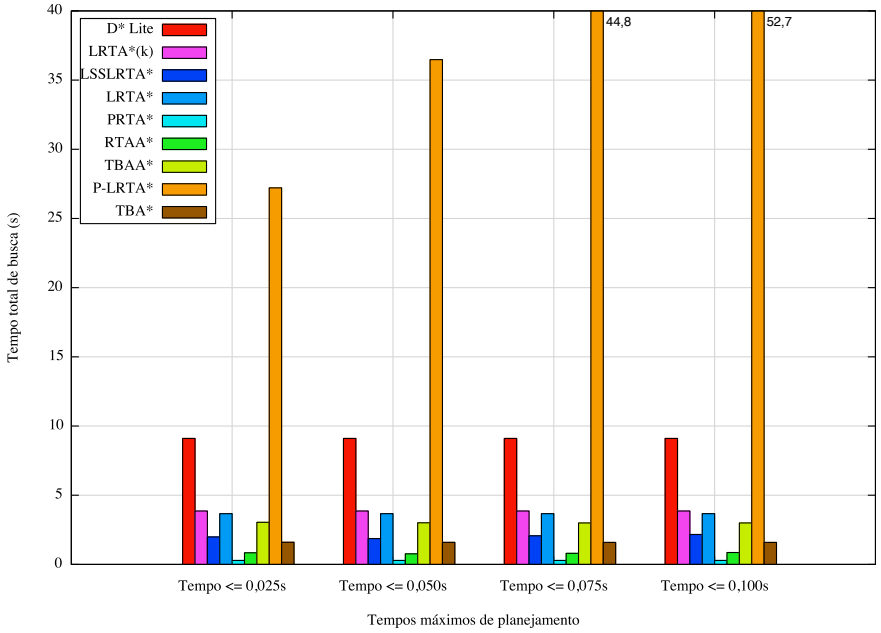

Figura 4.20. Tempo total de busca para o mapa **duskwood** com visibilidade = ∞.

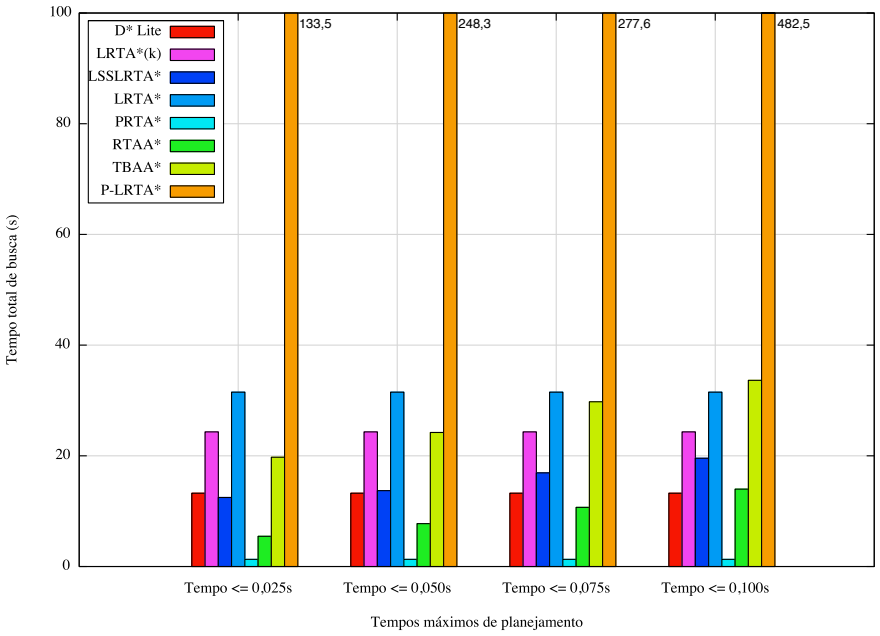

Figura 4.21. Tempo total de busca para o mapa **lak503d** com visibilidade = 1.

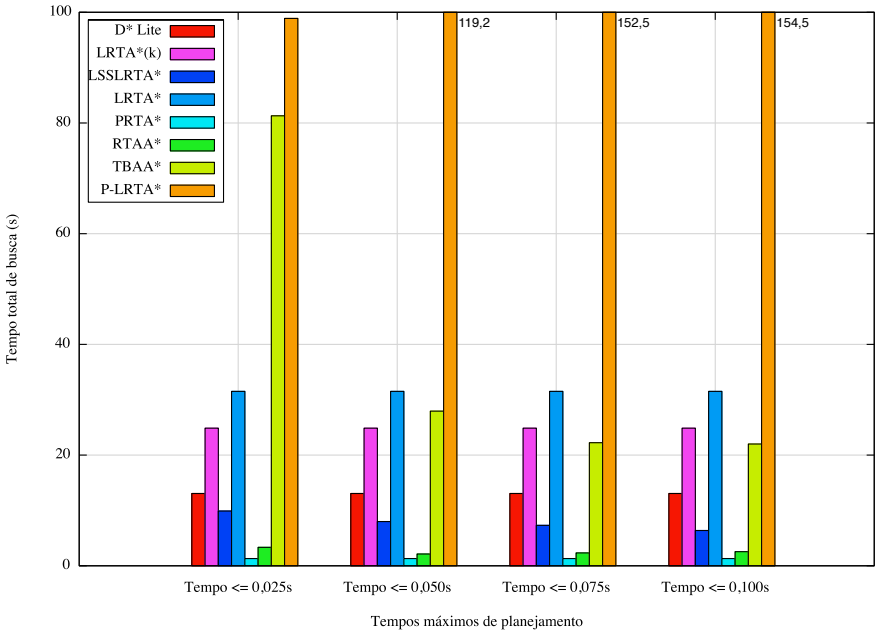

Figura 4.22. Tempo total de busca para o mapa **lak503d** com visibilidade = 10.

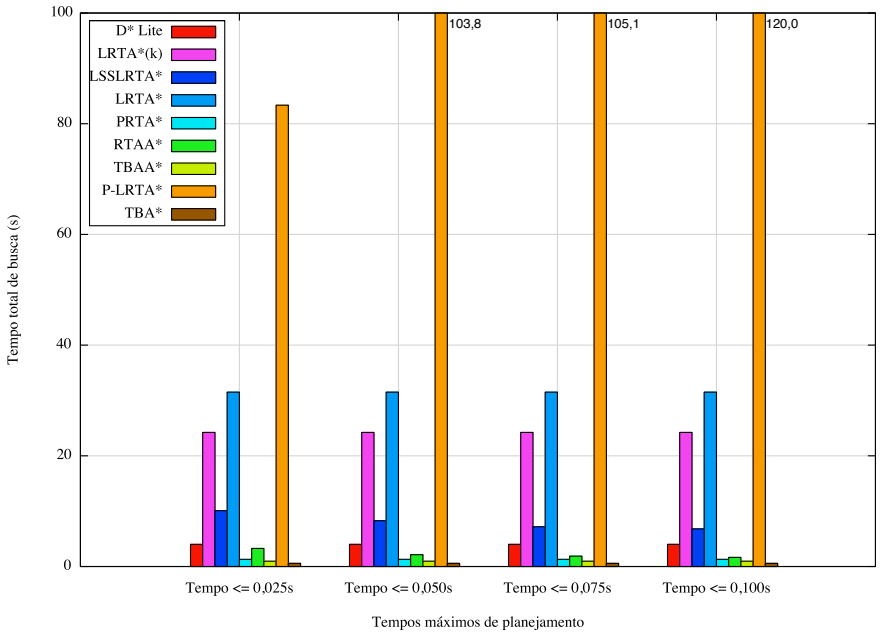

Figura 4.23. Tempo total de busca para o mapa **lak503d** com visibilidade = ∞.

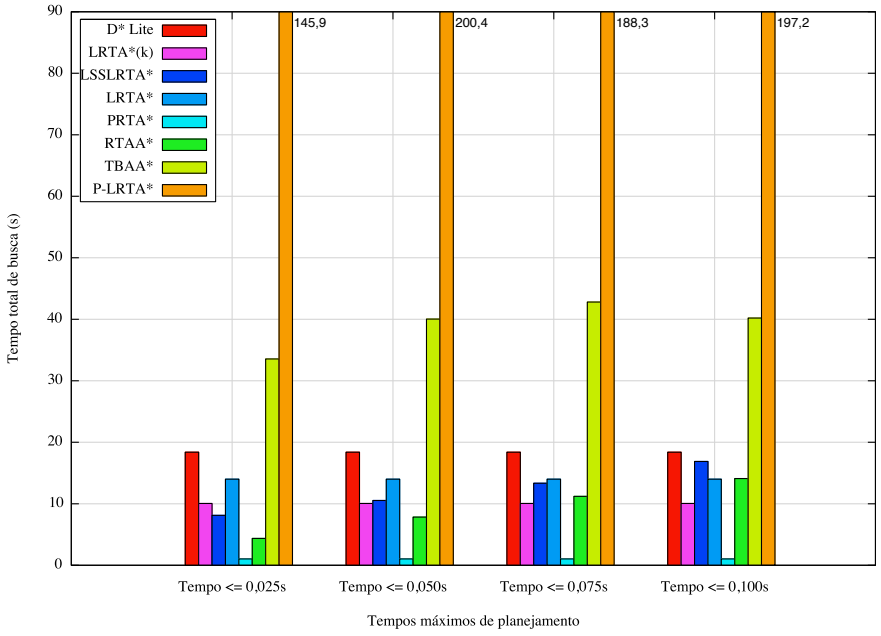

Figura 4.24. Tempo total de busca para o mapa **hrt201n** com visibilidade = 1.

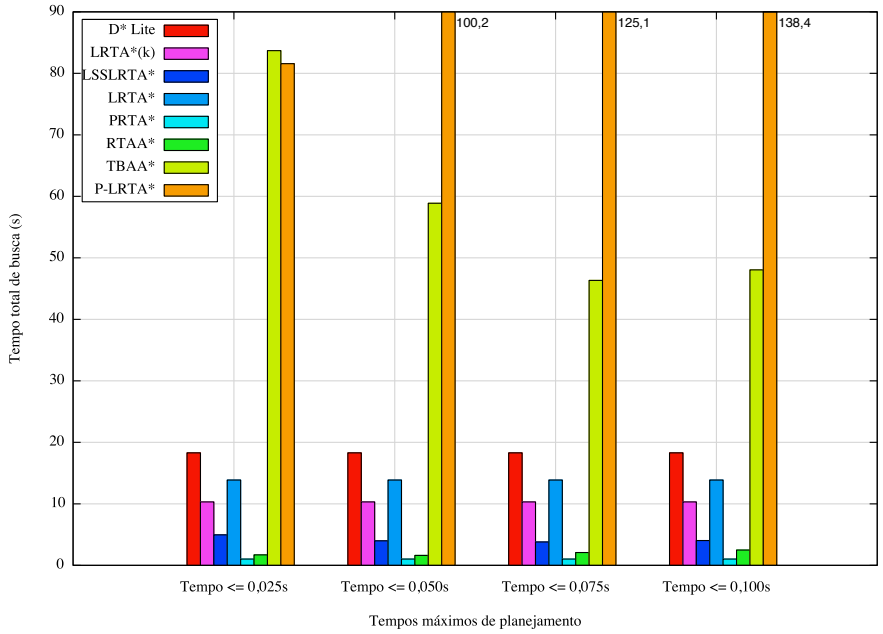

Figura 4.25. Tempo total de busca para o mapa **hrt201n** com visibilidade = 10.

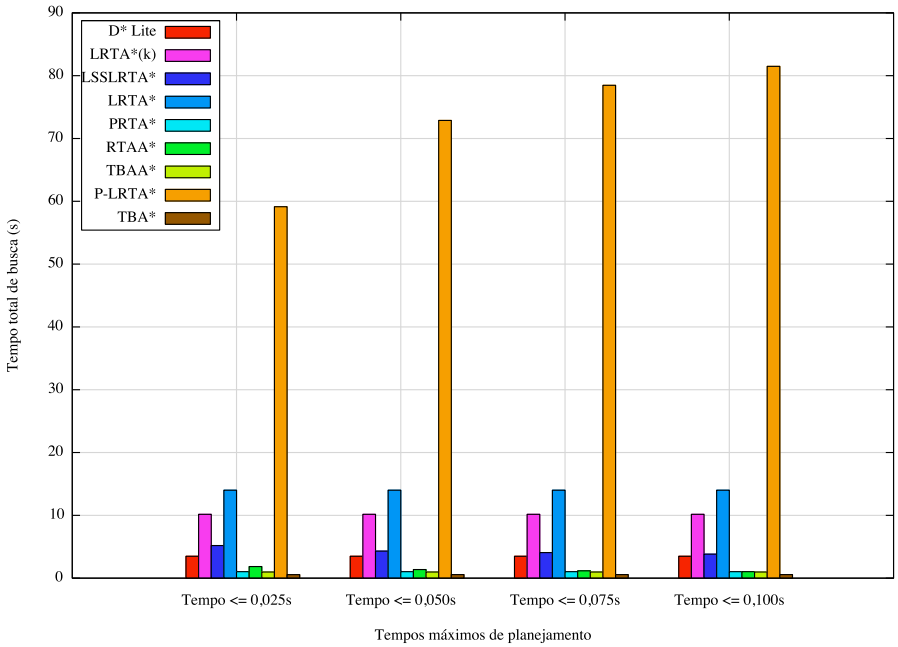

Figura 4.26. Tempo total de busca para o mapa **hrt201n** com visibilidade = ∞.

4.4 Convergência

Nessa seção analisamos a convergência dos algoritmos implementados. Um algoritmo de busca de tempo real converge quando, após repetidas execuções de um mesmo problema, sendo mantidos os *h-values* a cada execução, o algoritmo passa a encontrar sempre a solução ótima. Em jogos, esse comportamento pode ser importante em diversos cenários. Um deles é a coleta de recursos em jogos RTS.

Nem todos os algoritmos implementados nesse trabalho apresentam convergência. Aqueles que apresentam tal comportamento e são, portanto, comparados nessa seção, são: LRTA*, LRTA*(k), LSSLRTA*, RTAA*, TBAA*, P-LRTA*. Além disso, para realizar essa comparação, utilizamos mapas diferentes daqueles que usamos para comparar qualidade da solução e tempo total de busca. Isso foi feito pois, com mapas muito grandes (como os utilizados anteriormente), o tempo de convergência pode ser extremamente alto em problemas complexos. Os mapas usados foram **arena** e **den101d**, que são mostrados na Figura 3.1.

As figuras 4.27 e 4.28 mostram o número médio de execuções necessárias para cada algoritmo alcançar a convergência nos dois mapas testados. Cada figura mostra os resultados para todos os tempos máximos de episódio de planejamento usados. Novamente, o número mostrado para certo mapa é a média do resultado obtido para cada problema. Os resultados mostrados são aqueles obtidos em mapas parcialmente observáveis. Foram feitos testes também em mapas desconhecidos e completamente observáveis. No entanto, como os resultados obtidos em todas as situações foram muito parecidos, optamos por mostrar os gráficos somente para um tipo de visibilidade.

Em relação ao número de execuções até a convergência com o aumento do tempo máximo de planejamento (e, consequentemente, *lookahead* dos algoritmos) podemos perceber que o LRTA* e o LRTA*(k) exibem sempre o mesmo resultado. Novamente, um resultado esperado, já que o LRTA* não possui uma configuração de *lookahead* e o LRTA*(k) utiliza o mesmo valor de k em todos os casos. O restante dos algoritmos precisa, em geral, de menos execuções até a convergência quanto maior é o tempo de planejamento fornecido. Intuitivamente, quanto maior o *lookahead*, mais estados serão atualizados e, consequentemente, o número de execuções até a convergência será menor.

O algoritmo de pior convergência foi o LRTA*, seguido pelo LRTA*(k) em todos os testes realizados. Já o algoritmo com melhor convergência foi o P-LRTA* para o mapa **den101d** e o TBAA* para o mapa **arena**. Podemos concluir que o P-LRTA* parece melhor em mapas mais complexos (como o **den101d**), enquanto o TBAA* parece apresentar melhores resultados em mapas mais simples e com menos obstáculos

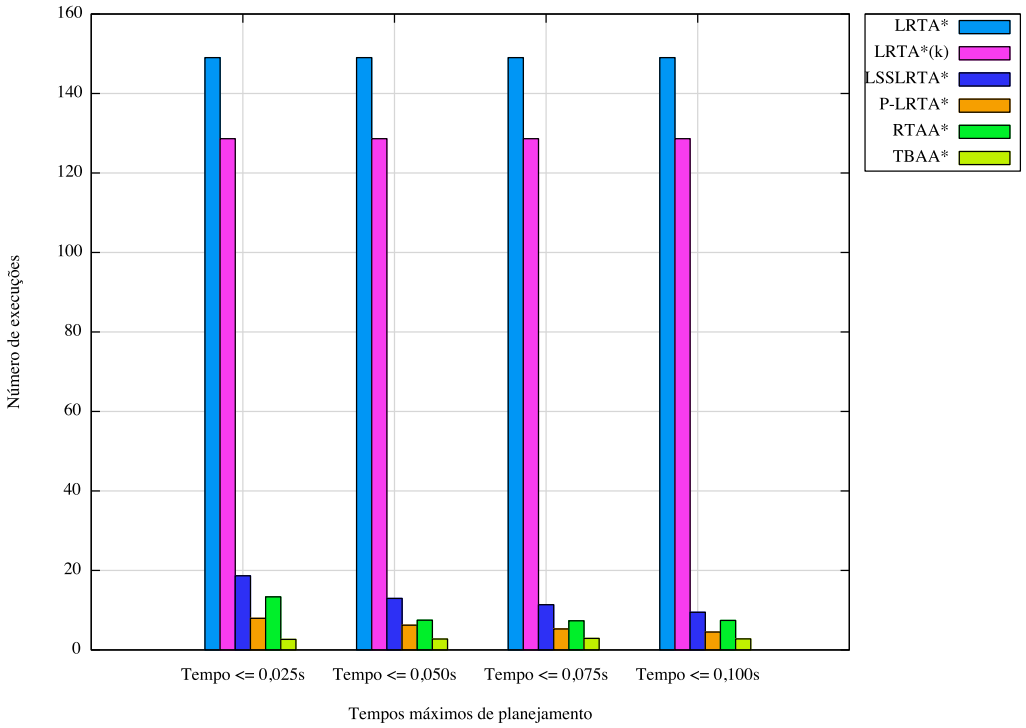

Figura 4.27. Número de execuções até a convergência no mapa **arena**.

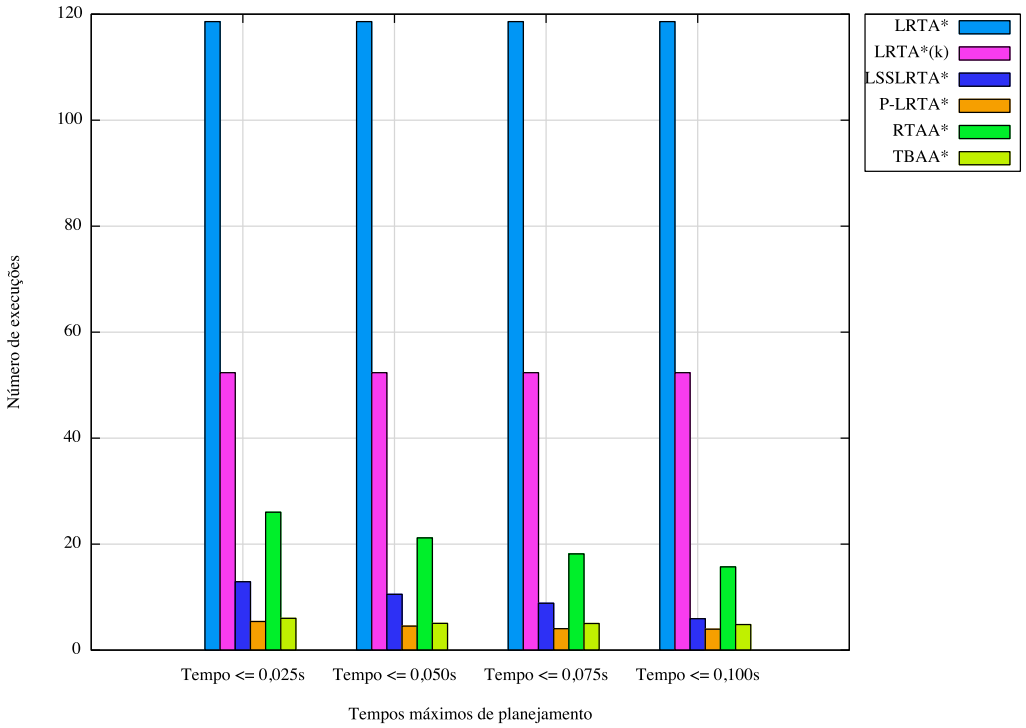

Figura 4.28. Número de execuções até a convergência no mapa **den101d**.

(como o **arena**). Apesar de inferiores aos dois já citados, o LSSLRTA* e o RTAA* apresentaram também uma convergência em um número razoável de execuções. Para o mapa **arena**, o RTAA* foi sempre o terceiro melhor algoritmo. No mapa **den101d**, o LSSLRTA* ocupou essa posição.

Sumarizando as discussões dessa seção temos que o LRTA* apresenta o maior número de execuções até a convergência. Ele precisa executar consideravelmente mais vezes que os demais algoritmos para convergir. Os melhores algoritmos nessa métrica são o TBAA* e o P-LRTA*. Além disso foi possível verificar que, em geral, um aumento de *lookahead* melhora o número de execuções necessárias para convergência.

4.5 Sumário dos resultados

Nessa seção, apresentamos, de forma resumida, os principais resultados observados anteriormente. Após a execução e análise de nossos testes, pudemos ver que:

- Em relação à qualidade da solução encontrada:
  - em mapas desconhecidos e parcialmente observáveis, o RTAA* apresenta, em geral, os melhores resultados;
  - em mapas completamente observáveis, o TBAA* é o algoritmo com melhor qualidade de solução;
  - quanto mais difícil o mapa maior a diferença de otimalidade dos melhores algoritmos (RTAA*, TBAA*) para os demais;
  - o P-LRTA*, em geral, melhora a qualidade da solução com aumentos de visibilidade e *lookahead*;
  - o LSSLRTA*, RTAA* e TBAA*, em geral, melhoram a qualidade da solução com aumentos de *lookahead*;
  - o LSSLRTA* e RTAA* apresentam quedas na qualidade da solução caso tenham um aumento de visibilidade e utilizem um *lookahead* pequeno. Com um *lookahead* grande o bastante, porém a qualidade da solução melhora. Além disso, esses dois algoritmos se beneficiam pouco do aumento de visibilidade parcial para completa;
  - o TBAA* apresenta quedas na qualidade de solução passando de mapas sem visibilidade para parcialmente observáveis com *lookaheads* pequenos. Em mapas completamente observáveis, no entanto, a qualidade das soluções aumenta drasticamente.

- Em relação ao tempo total de busca:
  - em mapas desconhecidos e parcialmente observáveis, o PRTA* apresentou os menores tempos totais de busca;
  - em mapas completamente observáveis, o PRTA* apresentou os melhores tempos para mapas fáceis e intermediários. Em mapas difíceis, no entanto, o TBAA* apresentou os melhores resultados;
  - em mapas desconhecidos e parcialmente observáveis o RTAA* apresentou, no geral, o melhor *tradeoff* entre tempo total de busca e qualidade de solução;
  - o melhor *tradeoff* entre tempo e qualidade de solução foi obtido pelo TBAA* em mapas completamente observáveis;
  - o LRTA*, LRTA*(k) e PRTA* não alteram o tempo total de busca para variações de visibilidade;
  - o P-LRTA*, em geral, melhora o tempo total de busca com aumentos de visibilidade e *lookahead*;
  - LSSLRTA*, RTAA* e TBAA* apresentam comportamento irregular com o aumento de *lookahead*. Em mapas desconhecidos, o tempo total de busca, em geral, aumenta com o aumento do *lookahead*. Quanto maior a visibilidade, porém, mais comum se tornam diminuições do tempo total de busca com o aumento dessa variável. Pode-se concluir que para esses algoritmos, um aumento de *lookahead* é mais vantajoso em mapas com maior visibilidade.
- Em relação à convergência:
  - o TBAA* foi o algoritmo com menor número de execuções até a convergência para mapas simples;
  - para mapas mais complexos, o P-LRTA* apresentou menor número de execuções até a convergência;
  - para todos os algoritmos testados nessa métrica, um aumento no *lookahead*, em geral, diminui o número de execuções necessárias até a convergência;
  - um aumento na visibilidade, em geral, não altera muito o número de execuções necessárias para a convergência.

Capítulo 5

Considerações Finais

Nesse capítulo concluímos o trabalho realizado. Na Seção 5.1 apresentamos a conclusão, assim como um breve resumo do que foi feito. Na Seção 5.2 discutimos trabalhos que ainda podem ser feitos.

5.1 Conclusão

Nesse trabalho estudamos algoritmos de busca heurística de tempo real, com foco em sua aplicação no problema do planejamento de trajetórias. O uso desses algoritmos nesse tipo de problema é interessante em jogos digitais, onde agentes devem mostrar sempre uma movimentação fluida, tomando decisões a cada ciclo de jogo (que é, geralmente, muito curto). Algoritmos de busca heurística de tempo real permitem que os agentes se movimentem após um pequeno episódio de planejamento, que pode ser configurado para não durar mais que um certo tempo pré-definido. Além disso, o planejamento é independente do tamanho do ambiente em que o agente se encontra, uma característica importante, uma vez que mapas de jogos digitais têm se tornado cada vez maiores.

Apesar da existência de diversos algoritmos de busca heurística de tempo real, um trabalho dedicado à comparação detalhada entre eles ainda não foi feito. Em nosso trabalho selecionamos, então, sete dos principais algoritmos desse tipo e realizamos um estudo detalhado sobre eles. Os algoritmos escolhidos para o nosso estudo foram: LRTA*, LRTA*(k), PRTA*, LSSLRTA*, RTAA*, TBAA* e P-LRTA*. Esses algoritmos foram escolhidos ou por serem bastante citados na literatura ou por apresentarem ideias diferentes da maioria dos outros.

Para testar e comparar os algoritmos escolhidos utilizamos vários mapas de jogos comerciais, especificamente dos jogos *Dragon Age: Origins* e *Warcraft III*. Foram esco-

lhidos mapas fáceis, intermediários e difíceis, classificados de acordo com a quantidade de obstáculos e possíveis depressões heurísticas. Cada um dos algoritmos implementados foi usado para resolver vários problemas de planejamento de caminhos em cada um dos mapas, sob diferentes condições de visibilidade e usando diferentes tempos máximos para cada episódio de planejamento. Além disso, analisamos três diferentes métricas para cada algoritmo: a qualidade da solução obtida, o tempo total necessário para resolver certo problema e o número de execuções necessárias até a convergência.

Em relação à otimalidade da solução, foi possível verificar que todos os algoritmos apresentam boas soluções em mapas fáceis. Em mapas intermediários e difíceis o RTAA* mostrou os melhores resultados, tanto em mapas sem visibilidade quanto em mapas com visibilidade parcial. Com o ambiente completamente observável, no entanto, o TBAA* foi o melhor algoritmo. O pior algoritmo foi o LRTA* em todas as situações. Observamos também os diferentes comportamentos dos algoritmos com o aumento do tempo máximo do episódio de planejamento e com a variação de visibilidade dos mapas.

Analisando o tempo total de busca foi possível ver que o PRTA*, um dos algoritmos com piores qualidades de solução, apresentou os melhores resultados. Seus resultados foram superados somente em mapas difíceis com ambientes completamente observáveis, onde o TBAA* se mostrou superior. Verificamos também que, na maioria dos casos, os algoritmos com soluções de melhor qualidade apresentam tempos de busca maiores, e que o RTAA* mostrou, no geral, o melhor *tradeoff* entre essas duas métricas. Novamente, fizemos uma análise, para cada algoritmo, de como o tempo total de busca é influenciado pela visibilidade dos mapas e pelo tempo máximo do episódio de planejamento.

Por fim, em relação ao número de execuções até a convergência, vimos que o TBAA* e o P-LRTA* apresentaram os melhores resultados, o primeiro para mapas mais simples e com poucos obstáculos, e o segundo para mapas um pouco mais complexos.

Os resultados obtidos atendem os objetivos desse trabalho. Foi possível mostrar vantagens, desvantagens e comparações de resultados de vários algoritmos de busca heurística de tempo real em diferentes situações. Acreditamos que nossos resultados sejam interessantes para vários outros trabalhos que estudam algoritmos desse tipo, mas especialmente úteis para um leitor interessado em aplicar algum deles em jogos digitais.

5.2 Trabalhos Futuros

A seguir mostramos uma lista de possíveis continuações desse trabalho.

- **Avaliar novos algoritmos**: apesar de termos escolhido os algoritmos mais conhecidos para utilizar em nossa comparação, vários outros também podem ser avaliados. Uma vez que já temos um *framework* para avaliação e comparação de algoritmos de busca de tempo real, uma possível continuação do nosso trabalho é implementar e avaliar novos algoritmos. Exemplos de algoritmos interessantes que podem ser avaliados são o daLSSLRTA* e o daRTAA* [Hernández & Baier, 2012], variações do LSSLRTA* e RTAA* que tentam evitar depressões heurísticas.

- **Estudo de diferentes heurísticas**: em nossa comparação utilizamos, para todos os algoritmos implementados, a mesma heurística, a distância de *Chebyshev*. Uma análise muito interessante seria avaliar os resultados de todos os algoritmos com diferentes heurísticas, especialmente com o uso de heurísticas não admissíveis. O uso de heurísticas não admissíveis em algoritmos de busca heurística de tempo real já foi proposto [Sadikov & Bratko, 2006; Shimbo & Ishida, 2000], mas ainda não há um trabalho que compare o comportamento de diferentes algoritmos desse tipo com o uso de várias dessas heurísticas.

- **Aplicação em jogos reais**: os experimentos realizados nesse trabalho foram feitos em mapas de jogos comerciais. No entanto, um experimento bastante interessante seria a comparação desses algoritmos em jogos reais. Poderíamos, por exemplo, utilizar uma *engine* de código aberto para jogos RTS e fazer o planejamento de caminho dos agentes com os algoritmos implementados nesse trabalho. Poderíamos, assim, avaliar os resultados na perspectiva do jogador.

- **Aplicação em situações de memória restrita**: uma situação onde o uso de algoritmos de busca heurística de tempo real para o planejamento de caminhos pode ser muito interessante é quando o mapa onde o agente deve planejar não cabe por completo em memória, uma vez que a maioria dos algoritmos desse tipo buscam somente em uma área ao redor do agente. Uma continuação desse trabalho é avaliar o comportamento dos algoritmos implementados em tal situação, utilizando diferentes limites de memória.

Referências Bibliográficas

Björnsson, Y.; Bulitko, V. & Sturtevant, N. R. (2009). TBA*: Time-bounded A*. Em Boutilier, C., editor, *IJCAI*, pp. 431–436.

Botea, A.; Herbrich, R. & Graepel, T. (2009). Video games and artificial intelligence. http://research.microsoft.com/en-us/projects/ijcaiigames/.

Bourg, D. M. & Seemann, G. (2004). *AI for Game Developers*. O'Reilly Media.

Bulitko, V.; Björnsson, Y.; Sturtevant, N. & Lawrence, R. (2010). Real-time heuristic search for pathfinding in video games. Em *Artificial Intelligence for Computer Games*. Springer.

Bulitko, V. & Björnsson, Y. (2009). knn lrta*: Simple subgoaling for real-time search. Em Darken, C. & Youngblood, G. M., editores, *AIIDE*. The AAAI Press.

Bulitko, V.; Björnsson, Y.; Lustrek, M.; Schaeffer, J. & Sigmundarson, S. (2007). Dynamic control in path-planning with real-time heuristic search. Em Boddy, M. S.; Fox, M. & Thiébaux, S., editores, *ICAPS*, pp. 49–56. AAAI.

Burns, E.; Kiesel, S. & Ruml, W. (2013). Experimental real-time heuristic search results in a video game. Em Helmert, M. & Röger, G., editores, *SOCS*. AAAI Press.

Galarneau, L. (2014). 2014 global gaming stats: Who's playing what, and why? http://www.bigfishgames.com/blog/2014-global-gaming-stats-whos-playing-what-and-why/.

Hart, P. E.; Nilsson, N. J. & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107.

Hernández, C.; Baier, J.; Uras, T. & Koenig, S. (2012). Time-bounded adaptive a*. Em *AAMAS*, pp. 997–1006. International Foundation for Autonomous Agents and Multiagent Systems.

Hernández, C. & Baier, J. A. (2012). Avoiding and escaping depressions in real-time heuristic search. *Jornal of Artificial Intelligence Research*, 43(1):523–570.

Hernández, C. & Baier, J. A. (2010). Escaping heuristic hollows in real-time search without learning. Em Ochoa, S. F.; Meza, F.; Mery, D. & Cubillos, C., editores, *SCCC*, pp. 172–177. IEEE Computer Society.

Hernández, C. & Meseguer, P. (2005). Lrta*(k). Em Kaelbling, L. P. & Saffiotti, A., editores, *IJCAI*, pp. 1238–1243. Professional Book Center.

Ishida, T. (1992). Moving target search with intelligence. Em Swartout, W. R., editor, *AAAI*, pp. 525–532. AAAI Press / The MIT Press.

Koenig, S. (2001). Agent-centered search. *AI Magazine*, 22(4):109–132.

Koenig, S. (2004). A comparison of fast search methods for real-time situated agents. Em *AAMAS*, pp. 864–871. IEEE Computer Society.

Koenig, S. & Likhachev, M. (2001). Incremental a*. Em *NIPS*, pp. 1539–1546.

Koenig, S. & Likhachev, M. (2002). Improved fast replanning for robot navigation in unknown terrain. Em *International Conference on Robotics and Automation*, pp. 968–975.

Koenig, S. & Likhachev, M. (2005). Adaptive a* [poster abstract]. Em *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 1311–1312.

Koenig, S. & Likhachev, M. (2006). A new principle for incremental heuristic search: Theoretical results. Em Long, D.; Smith, S. F.; Borrajo, D. & McCluskey, L., editores, *ICAPS*, pp. 402–405. AAAI.

Koenig, S.; Likhachev, M. & Furcy, D. (2004a). Lifelong planning a*. *Artif. Intell.*, 155(1-2):93–146.

Koenig, S.; Likhachev, M.; Liu, Y. & Furcy, D. (2004b). Incremental heuristic search in ai. *AI Magazine*, 25(2):99–112.

Koenig, S. & Smirnov, Y. V. (1997). Sensor-based planning with the freespace assumption. Em *ICRA*, pp. 3540–3545. IEEE.

Korf, R. E. (1990). Real-time heuristic search. *Artif. Intell.*, 42(2-3):189–211.

Korf, R. E. (1996). Artificial intelligence search algorithms. Em *In Algorithms and Theory of Computation Handbook*. CRC Press.

Marques, V.; Chaimowicz, L. & Ferreira, R. (2011). Speeding up learning in real-time search through parallel computing. *Symposium on Computer Architecture and High Performance Computing*, 0:176–182. ISSN 1550-6533.

Millington, I. & Funge, J. (2009). *Artificial Intelligence for Games, Second Edition*. Morgan Kaufmann.

Moore, A. W. & Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. Em *Machine Learning*, pp. 103–130.

Nilsson, N. J. (1998). *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann, San Francisco, CA, USA.

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

Pemberton, J. & Korf, R. E. (1992). Making locally optimal decisions on graphs with cycles. Relatório técnico, Computer Science Departament, University of California at Los Angeles.

Rayner, D. C.; Davison, K.; Bulitko, V.; Anderson, K. & Lu, J. (2007). Real-time heuristic search with a priority queue. Em *IJCAI*, pp. 2372–2377.

Rios, L. H. O. & Chaimowicz, L. (2010). A survey and classification of a* based best-first heuristic search algorithms. Em da Rocha Costa, A. C.; Vicari, R. M. & Tonidandel, F., editores, *SBIA*, volume 6404 of *Lecture Notes in Computer Science*, pp. 253–262. Springer.

Russell, S. J. & Norvig, P. (2002). *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall.

Russell, S. J. & Wefald, E. (1991). *Do the Right Thing: Studies in Limited Rationality*. MIT Press, Cambridge, MA.

Sadikov, A. & Bratko, I. (2006). Pessimistic heuristics beat optimistic ones in real-time search. Em *ECAI*, pp. 148–152, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Shimbo, M. & Ishida, T. (2000). Towards real-time search with inadmissible heuristics. Em *ECAI*, pp. 609–613.

Stentz, A. T. (1994). Optimal and efficient path planning for partially-known environments. Em *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '94)*, volume 4, pp. 3310–3317.

Sturtevant, N. (2012). Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games*, 4(2):144–148.

Sturtevant, N.; Bulitko, V. & Björnsson, Y. (2010). On learning in agent-centered search. Em *Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 333–340. International Foundation for Autonomous Agents and Multiagent Systems.

Sturtevant, N. R. & Bulitko, V. (2011). Learning where you are going and from whence you came: h- and g-cost learning in real-time heuristic search. Em Walsh, T., editor, *IJCAI*, pp. 365–370. IJCAI/AAAI.

Sun, X.; Koenig, S. & Yeoh, W. (2008). Real-time adaptive a*. Em *AAMAS*, pp. 281–288.

Woodcock, S. (2001). Game ai: The state of the industry. http://www.gamasutra.com/features/20001101/woodcock_pfv.htm.

Zelinsky, A. (1992). Mobile robot navigation exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6):707–717.

Apêndice A

Tabelas

Aqui são apresentadas as tabelas utilizadas na geração dos gráficos exibidos nesse trabalho. Essas tabelas mostram os resultados obtidos pelos algoritmos de busca heurística de tempo real nos diversos mapas usados como *benchmarks*. Na Seção A.1, mostramos as tabelas referentes à otimalidade da solução obtida pelos algoritmos. Na Seção A.2, exibimos as tabelas referentes ao tempo total de busca. Finalmente, na Seção A.3, apresentamos as tabelas referentes ao número de execuções até a convergência.

A.1 Otimalidade da solução

Nessa seção mostramos as tabelas que mostram a qualidade da solução obtida pelos algoritmos implementados nesse trabalho. Cada tabela mostra os resultados para um mapa em certa visibilidade, com os quatro tempos máximos de episódio de planejamento utilizados. Cada resultado mostrado é a média da solução obtida para todos os problemas do mapa em questão. Os resultados são mostrados em relação ao custo das soluções ótimas, e são o número de vezes que a solução obtida é pior que a ótima. Um resultado de valor 3, por exemplo, indica que as soluções obtidas são 3 vezes piores que a ótima. Um resultado de valor 1 indica que a solução ótima foi obtida.

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	1,017	1,017	1,017	1,017
LRTA*	1,488	1,488	1,488	1,488
LRTA*(k)	1,090	1,090	1,090	1,090
LSSLRTA*	1,111	1,108	1,102	1,101
P-LRTA*	1,113	1,111	1,072	1,079
PRTA*	1,090	1,090	1,090	1,090
RTAA*	1,108	1,083	1,090	1,079
TBAA*	1,349	1,145	1,101	1,071

Tabela A.1. Qualidade da solução para o mapa **combat** sem visibilidade (visibilidade = 1).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	1,016	1,016	1,016	1,016
LRTA*	1,488	1,488	1,488	1,488
LRTA*(k)	1,090	1,090	1,090	1,090
LSSLRTA*	1,113	1,101	1,101	1,092
P-LRTA*	1,247	1,175	1,180	1,155
PRTA*	1,090	1,090	1,090	1,090
RTAA*	1,088	1,071	1,080	1,074
TBAA*	1,296	1,127	1,076	1,060

Tabela A.2. Qualidade da solução para o mapa **combat** com visibilidade parcial (visibilidade = 10).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	1,000	1,000	1,000	1,000
LRTA*	1,488	1,488	1,488	1,488
LRTA*(k)	1,090	1,090	1,090	1,090
LSSLRTA*	1,104	1,101	1,093	1,091
P-LRTA*	1,247	1,207	1,152	1,154
PRTA*	1,090	1,090	1,090	1,090
RTAA*	1,089	1,070	1,069	1,057
TBA*	1,064	1,026	1,014	1,009
TBAA*	1,238	1,083	1,050	1,034

Tabela A.3. Qualidade da solução para o mapa **combat** com visibilidade total (visibilidade = ∞).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	1,566	1,566	1,566	1,566
LRTA*	48,728	48,728	48,728	48,728
LRTA*(k)	4,106	4,106	4,106	4,106
LSSLRTA*	2,770	2,301	2,098	1,975
P-LRTA*	6,496	4,657	4,234	3,674
PRTA*	4,175	4,175	4,175	4,175
RTAA*	2,194	1,862	1,799	1,754
TBAA*	2,899	3,442	2,040	1,931

Tabela A.4. Qualidade da solução para o mapa **duskwood** sem visibilidade (visibilidade = 1).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	1,411	1,411	1,411	1,411
LRTA*	48,728	48,728	48,728	48,728
LRTA*(k)	4,106	4,106	4,106	4,106
LSSLRTA*	3,239	2,331	2,083	1,894
P-LRTA*	6,147	5,273	4,502	3,935
PRTA*	4,175	4,175	4,175	4,175
RTAA*	2,261	1,647	1,591	1,545
TBAA*	7,195	2,118	1,667	1,587

Tabela A.5. Qualidade da solução para o mapa **duskwood** com visibilidade parcial (visibilidade = 10).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	1,000	1,000	1,000	1,000
LRTA*	48,728	48,728	48,728	48,728
LRTA*(k)	4,106	4,106	4,106	4,106
LSSLRTA*	3,275	2,532	2,246	2,003
P-LRTA*	6,270	4,854	3,840	3,298
PRTA*	4,175	4,175	4,175	4,175
RTAA*	2,557	1,796	1,504	1,473
TBA*	1,147	1,067	1,038	1,029
TBAA*	1,406	1,137	1,079	1,054

Tabela A.6. Qualidade da solução para o mapa **duskwood** com visibilidade total (visibilidade = ∞).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	2,943	2,943	2,943	2,943
LRTA*	451,948	451,948	451,948	451,948
LRTA*(k)	14,696	14,696	14,696	14,696
LSSLRTA*	14,673	10,615	8,640	7,051
P-LRTA*	27,089	25,815	18,132	21,957
PRTA*	21,692	21,692	21,692	21,692
RTAA*	9,588	5,795	4,945	4,721
TBAA*	6,008	3,859	3,619	3,470

Tabela A.7. Qualidade da solução para o mapa **lak503d** sem visibilidade (visibilidade = 1).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	2,089	2,089	2,089	2,089
LRTA*	451,948	451,948	451,948	451,948
LRTA*(k)	14,696	14,696	14,696	14,696
LSSLRTA*	15,662	9,957	7,355	5,424
P-LRTA*	22,127	15,932	13,323	10,196
PRTA*	21,692	21,692	21,692	21,692
RTAA*	9,232	4,228	3,360	3,034
TBAA*	19,712	4,533	3,106	2,641

Tabela A.8. Qualidade da solução para o mapa **lak503d** com visibilidade parcial (visibilidade = 10).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	1,000	1,000	1,000	1,000
LRTA*	451,948	451,948	451,948	451,948
LRTA*(k)	14,696	14,696	14,696	14,696
LSSLRTA*	16,109	10,538	7,489	6,077
P-LRTA*	20,797	15,436	10,244	8,956
PRTA*	21,692	21,692	21,692	21,692
RTAA*	9,388	4,880	3,577	2,864
TBA*	1,084	1,055	1,040	1,036
TBAA*	1,119	1,045	1,027	1,019

Tabela A.9. Qualidade da solução para o mapa **lak503d** com visibilidade total (visibilidade = ∞).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	3,284	3,284	3,284	3,284
LRTA*	195,139	195,139	195,139	195,139
LRTA*(k)	10,495	10,495	10,495	10,495
LSSLRTA*	8,891	6,906	5,565	5,019
P-LRTA*	23,124	18,939	12,636	10,529
PRTA*	15,138	15,138	15,138	15,138
RTAA*	6,621	4,800	4,048	3,865
TBAA*	9,157	5,982	4,878	4,072

Tabela A.10. Qualidade da solução para o mapa **hrt201n** sem visibilidade (visibilidade = 1).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	2,326	2,326	2,326	2,326
LRTA*	195,139	195,139	195,139	195,139
LRTA*(k)	10,495	10,495	10,495	10,495
LSSLRTA*	8,574	5,714	4,389	3,858
P-LRTA*	18,603	13,398	10,688	9,053
PRTA*	15,138	15,138	15,138	15,138
RTAA*	5,334	3,452	3,000	2,787
TBAA*	26,474	8,112	5,656	4,302

Tabela A.11. Qualidade da solução para o mapa **hrt201n** com visibilidade parcial (visibilidade = 10).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	1,000	1,000	1,000	1,000
LRTA*	195,139	195,139	195,139	195,139
LRTA*(k)	10,495	10,495	10,495	10,495
LSSLRTA*	9,374	6,317	4,991	3,994
P-LRTA*	15,965	11,688	8,176	6,718
PRTA*	15,138	15,138	15,138	15,138
RTAA*	6,176	3,588	2,679	2,268
TBA*	1,052	1,038	1,032	1,032
TBAA*	1,064	1,018	1,009	1,006

Tabela A.12. Qualidade da solução para o mapa **hrt201n** com visibilidade total (visibilidade = ∞).

A.2 Tempo total de busca

Nessa seção mostramos as tabelas que indicam o tempo total de busca dos algoritmos nos testes realizados. Cada tabela mostra os resultados para um mapa em certa visibilidade, com os quatro tempos máximos de episódio de planejamento utilizados. Cada resultado mostrado é a média do tempo gasto para encontrar e primeira solução em todos os problemas do mapa em questão. Os resultados são mostrados em segundos.

	Tempo $\leq$ 0,025s	Tempo $\leq$ 0,050s	Tempo $\leq$ 0,075s	Tempo $\leq$ 0,100s
D* Lite	3,378s	3,378s	3,378s	3,378s
LRTA*	0,048s	0,048s	0,048s	0,048s
LRTA*(k)	0,054s	0,054s	0,054s	0,054s
LSSLRTA*	0,215s	0,318s	0,426s	0,526s
P-LRTA*	1,636s	2,827s	3,985s	5,634s
PRTA*	0,028s	0,028s	0,028s	0,028s
RTAA*	0,137s	0,214s	0,320s	0,384s
TBAA*	1,275s	1,381s	1,496s	1,483s

Tabela A.13. Tempo total de busca para o mapa **combat** sem visibilidade (visibilidade = 1).

	Tempo $\leq$ 0,025s	Tempo $\leq$ 0,050s	Tempo $\leq$ 0,075s	Tempo $\leq$ 0,100s
D* Lite	3,470s	3,470s	3,470s	3,470s
LRTA*	0,047s	0,047s	0,047s	0,047s
LRTA*(k)	0,073s	0,073s	0,073s	0,073s
LSSLRTA*	0,181s	0,250s	0,344s	0,382s
P-LRTA*	1,994s	3,053s	4,851s	6,541s
PRTA*	0,028s	0,028s	0,028s	0,028s
RTAA*	0,117s	0,172s	0,232s	0,272s
TBAA*	1,144s	1,273s	1,290s	1,359s

Tabela A.14. Tempo total de busca para o mapa **combat** com visibilidade parcial (visibilidade = 10).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	3,011s	3,011s	3,011s	3,011s
LRTA*	0,048s	0,048s	0,048s	0,048s
LRTA*(k)	0,076s	0,076s	0,076s	0,076s
LSSLRTA*	0,181s	0,247s	0,328s	0,375s
P-LRTA*	2,179s	3,509s	5,109s	6,885s
PRTA*	0,028s	0,028s	0,028s	0,028s
RTAA*	0,116s	0,168s	0,216s	0,240s
TBA*	0,371s	0,369s	0,368s	0,368s
TBAA*	0,920s	0,913s	0,907s	0,909s

Tabela A.15. Tempo total de busca para o mapa **combat** com visibilidade total (visibilidade = ∞).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	18,259s	18,259s	18,259s	18,259s
LRTA*	3,667s	3,667s	3,667s	3,667s
LRTA*(k)	3,707s	3,707s	3,707s	3,707s
LSSLRTA*	2,579s	3,447s	4,827s	6,112s
P-LRTA*	35,501s	43,991s	61,594s	74,027s
PRTA*	0,286s	0,286s	0,286s	0,286s
RTAA*	1,391s	2,479s	4,148s	5,288s
TBAA*	11,397s	35,844s	18,698s	20,741s

Tabela A.16. Tempo total de busca para o mapa **duskwood** sem visibilidade (visibilidade = 1).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	19,852s	19,852s	19,852s	19,852s
LRTA*	3,676s	3,676s	3,676s	3,676s
LRTA*(k)	3,927s	3,927s	3,927s	3,927s
LSSLRTA*	1,947s	1,713s	1,989s	2,186s
P-LRTA*	33,589s	47,914s	65,328s	78,594s
PRTA*	0,294s	0,294s	0,294s	0,294s
RTAA*	0,748s	0,805s	1,114s	1,371s
TBAA*	15,815s	15,005s	13,775s	14,969s

Tabela A.17. Tempo total de busca para o mapa **duskwood** com visibilidade parcial (visibilidade = 10).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	9,101s	9,101s	9,101s	9,101s
LRTA*	3,667s	3,667s	3,667s	3,667s
LRTA*(k)	3,860s	3,860s	3,860s	3,860s
LSSLRTA*	1,985s	1,867s	2,067s	2,160s
P-LRTA*	27,210s	36,478s	44,755s	52,731s
PRTA*	0,286s	0,286s	0,286s	0,286s
RTAA*	0,837s	0,766s	0,801s	0,859s
TBA*	1,600s	1,590s	1,589s	1,589s
TBAA*	3,044s	3,007s	2,995s	2,996s

Tabela A.18. Tempo total de busca para o mapa **duskwood** com visibilidade total (visibilidade = ∞).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	13,247s	13,247s	13,247s	13,247s
LRTA*	31,503s	31,503s	31,503s	31,503s
LRTA*(k)	24,326s	24,326s	24,326s	24,326s
LSSLRTA*	12,469s	13,706s	16,921s	19,563s
P-LRTA*	133,474s	248,232s	277,644s	482,523s
PRTA*	1,298s	1,298s	1,298s	1,298s
RTAA*	5,458s	7,744s	10,686s	13,981s
TBAA*	19,741s	24,213s	29,769s	33,636s

Tabela A.19. Tempo total de busca para o mapa **lak503d** sem visibilidade (visibilidade = 1).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	13,058s	13,058s	13,058s	13,058s
LRTA*	31,514s	31,514s	31,514s	31,514s
LRTA*(k)	24,859s	24,859s	24,859s	24,859s
LSSLRTA*	9,891s	7,988s	7,308s	6,374s
P-LRTA*	98,889s	119,236s	152,536s	154,532s
PRTA*	1,304s	1,304s	1,304s	1,304s
RTAA*	3,330s	2,130s	2,327s	2,550s
TBAA*	81,293s	27,938s	22,235s	22,005s

Tabela A.20. Tempo total de busca para o mapa **lak503d** com visibilidade parcial (visibilidade = 10).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	3,997s	3,997s	3,997s	3,997s
LRTA*	31,503s	31,503s	31,503s	31,503s
LRTA*(k)	24,224s	24,224s	24,224s	24,224s
LSSLRTA*	10,088s	8,271s	7,172s	6,790s
P-LRTA*	83,341s	103,786s	105,108s	120,033s
PRTA*	1,298s	1,298s	1,298s	1,298s
RTAA*	3,267s	2,134s	1,877s	1,643s
TBA*	0,566s	0,566s	0,566s	0,566s
TBAA*	0,962s	0,958s	0,957s	0,957s

Tabela A.21. Tempo total de busca para o mapa **lak503d** com visibilidade total (visibilidade = ∞).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	18,384s	18,384s	18,384s	18,384s
LRTA*	14,000s	14,000s	14,000s	14,000s
LRTA*(k)	10,047s	10,047s	10,047s	10,047s
LSSLRTA*	8,117s	10,532s	13,344s	16,876s
P-LRTA*	145,918s	200,416s	188,337s	197,211s
PRTA*	1,029s	1,029s	1,029s	1,029s
RTAA*	4,364s	7,832s	11,207s	14,097s
TBAA*	33,547s	40,029s	42,804s	40,199s

Tabela A.22. Tempo total de busca para o mapa **hrt201n** sem visibilidade (visibilidade = 1).

	Tempo ≤ 0,025s	Tempo ≤ 0,050s	Tempo ≤ 0,075s	Tempo ≤ 0,100s
D* Lite	18,292s	18,292s	18,292s	18,292s
LRTA*	13,874s	13,874s	13,874s	13,874s
LRTA*(k)	10,319s	10,319s	10,319s	10,319s
LSSLRTA*	4,962s	3,987s	3,801s	4,030s
P-LRTA*	81,570s	100,199s	125,139s	138,443s
PRTA*	1,027s	1,027s	1,027s	1,027s
RTAA*	1,707s	1,617s	2,086s	2,490s
TBAA*	83,692s	58,876s	46,329s	48,043s

Tabela A.23. Tempo total de busca para o mapa **hrt201n** com visibilidade parcial (visibilidade = 10).

	Tempo $\leq$ 0,025s	Tempo $\leq$ 0,050s	Tempo $\leq$ 0,075s	Tempo $\leq$ 0,100s
D* Lite	3,483s	3,483s	3,483s	3,483s
LRTA*	14,000s	14,000s	14,000s	14,000s
LRTA*(k)	10,167s	10,167s	10,167s	10,167s
LSSLRTA*	5,185s	4,308s	4,068s	3,819s
P-LRTA*	59,131s	72,872s	78,470s	81,485s
PRTA*	1,029s	1,029s	1,029s	1,029s
RTAA*	1,840s	1,370s	1,168s	1,030s
TBA*	0,560s	0,561s	0,562s	0,561s
TBAA*	0,979s	0,979s	0,980s	0,980s

Tabela A.24. Tempo total de busca para o mapa **hrt201n** com visibilidade total (visibilidade = ∞).

A.3 Execuções até convergência

Nessa seção mostramos as tabelas que indicam o número total de execuções gastas pelos algoritmos para convergir até a solução ótima nos mapas testados. Cada tabela mostra os resultados para um mapa, com os quatro tempos máximos de episódio de planejamento utilizados. Cada resultado mostrado é a média do número de execuções gastas em todos os problemas do mapa em questão.

	Tempo $\leq$ 0,025s	Tempo $\leq$ 0,050s	Tempo $\leq$ 0,075s	Tempo $\leq$ 0,100s
LRTA*	149,025	149,025	149,025	149,025
LRTA*(k)	128,625	128,625	128,625	128,625
LSSLRTA*	18,650	12,950	11,375	9,475
P-LRTA*	7,950	6,225	5,275	4,500
RTAA*	13,350	7,475	7,325	7,425
TBAA*	2,650	2,750	2,900	2,775

Tabela A.25. Número de execuções até a convergência para o mapa **arena** com visibilidade parcial (visibilidade = 10).

	Tempo $\leq$ 0,025s	Tempo $\leq$ 0,050s	Tempo $\leq$ 0,075s	Tempo $\leq$ 0,100s
LRTA*	118,600	118,600	118,600	118,600
LRTA*(k)	52,350	52,350	52,350	52,350
LSSLRTA*	12,900	10,525	8,850	5,925
P-LRTA*	5,400	4,525	4,050	3,950
RTAA*	26,025	21,175	18,150	15,700
TBAA*	6,000	5,050	5,025	4,825

Tabela A.26. Número de execuções até a convergência para o mapa **den101d** com visibilidade parcial (visibilidade = 10).