UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

HENRICKY DE LIMA MONTEIRO

BUSCA HEURÍSTICA SIMBÓLICA PARA PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

QUIXADÁ

2025

HENRICKY DE LIMA MONTEIRO

BUSCA HEURÍSTICA SIMBÓLICA PARA PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciências da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciências da Computação.

Orientadora: Profa. Dra. Maria Viviane de Menezes.

QUIXADÁ

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M776b Monteiro, Henricky de Lima.
Busca Heurística Simbólica para Planejamento em Inteligência Artificial / Henricky de Lima Monteiro. – 2025.
73 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Ciência da Computação, Quixadá, 2025.
Orientação: Profa. Dra. Maria Viviane de Menezes.

1. Planejamento Automatizado. 2. Busca Heurística. 3. Busca Simbólica. 4. Inteligência Artificial. I. Título.

CDD 004

HENRICKY DE LIMA MONTEIRO

BUSCA HEURÍSTICA SIMBÓLICA PARA PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciências da Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciências da Computação.

Aprovada em: 29 de Julho de 2025

BANCA EXAMINADORA

Profa. Dra. Maria Viviane de Menezes (Orientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Atílio Gomes Luiz
Universidade Federal do Ceará (UFC)

Prof. Dr. Davi Romero de Vasconcelos
Universidade Federal do Ceará (UFC)

"Por nove noites seguidas, dez abutres negros circularam o Bastião. Então um deles expulsou a águia que vivia no topo da Torre de Papel."

(Glen Cook, *A Companhia Negra*)

AGRADECIMENTOS

Aos meus pais, Maria Iranice e Antônio Aguinaldo, e irmão, Pedro Anthony, pelo apoio e paciência.

A todos os professores que me inspiraram nesses anos de curso, em especial ao Prof. Dr. Atílio Gomes Luiz, Prof. Dra. Paulyne Matthews Jucá, Prof. Dr. Paulo de Tarso Guerra Oliveira, Prof. Anderson Lemos da Silva e à Prof. Dra. Maria Viviane de Menezes pela excelente orientação e paciência.

Aos professores participantes da banca examinadora Prof. Dr. Atílio Gomes Luiz e Prof. Dr. Davi Romero de Vasconcelos pelo tempo, pelas valiosas colaborações e sugestões.

Aos amigos que fiz durante essa jornada: Mário Filho, Hercules Florentino, Ryan Alves, Pedro Spinosa, João Almir, Andreina Maria, Luis Fernando, Carlos Ryan, Tiago Rodrigues, Abel Figueiredo, Leonardo Ferreira e Robert Bertoldo;

RESUMO

Planejamento automatizado é a subárea da Inteligência Artificial (IA) que estuda o desenvolvimento de algoritmos capazes de elaborar um plano de ações para que agentes inteligentes alcancem suas metas. A busca por um plano solução pode ser realizada para frente, gerando estados sucessores a partir de um dado estado inicial, ou para trás, gerando estados antecessores a partir de estados satisfazendo a meta. Esse é um problema P-SPACE e para lidar com tal complexidade, duas abordagens principais têm sido exploradas na literatura: a *busca heurística*, que utiliza estimativas para guiar a expansão de nós, e a *busca simbólica*, que emprega representações compactas dos estados por meio de Diagramas de Decisão Binária (BDDs). Em trabalhos anteriores, foi proposta a busca heurística simbólica para combinar as vantagens dessas duas abordagens Silva (2019). No entanto, a realização do cálculo heurístico em espaços de estados grandes é um passo que ainda merece atenção. Assim, Kissmann (2012) introduziu um limitante de tempo na geração de heurísticas para melhorar o desempenho da busca simbólica. O presente trabalho propõe incorporar o limitante de tempo no cálculo heurístico regressivo de (Silva, 2019) e avaliar o impacto dessa mudança na realização da busca simbólica A*. O algoritmo implementado será avaliado em *benchmarks* da Competição Internacional de Planejamento (IPC).

Palavras-chave: Planejamento Automatizado; Busca Heurística; Busca Simbólica; Diagramas de Decisão Binária.

ABSTRACT

Automated planning is a subfield of Artificial Intelligence (AI) that focuses on the development of algorithms capable of generating action plans that allow intelligent agents to achieve their goals. The search for a solution plan can be performed forward, generating successor states from an initial state, or backward, generating predecessor states from goal-satisfying states. This is a P-SPACE problem, and to deal with such complexity, two main approaches have been widely explored in the literature: heuristic search, which uses estimates to guide node expansion, and symbolic search, which employs compact representations of state sets through Binary Decision Diagrams (BDDs). Previous works proposed symbolic heuristic search as a way to combine the strengths of both approaches Silva (2019). However, computing heuristics in large state spaces remains a challenging task. To address this, Kissmann (2012) introduced a time limit during heuristic generation to improve the performance of symbolic search. This work proposes incorporating such a time limit into the symbolic heuristic computation of (Silva, 2019) and evaluating the impact of this modification on the performance of symbolic A* search. The proposed algorithm is evaluated on benchmarks from the International Planning Competition (IPC).

Keywords: Automated Planning; Heuristic Search; Symbolic Search; Binary Decision Diagrams.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

*A**	*A Star*
*BDDA**	*Binary Decision Diagram A**
BDD	*Binary Decision Diagram*
BFS	*Breadth-First Search*
BNF	*Backus Naur Form*
CWA	*Closed World Assumption*
DFS	*Depth-First Search*
GBFS	*Greedy Best-First Search*
ICAPS	*International Conference on Automated Planning and Scheduling*
IPC	*Internacional Planning Competition*
JVM	*Java Virtual Machine*
OBDD	*Ordered Binary Decision Diagram*
PDB	*Pattern Databases*
PDDL	*Planning Domain Definition Language*
QBF	*Quantified Boolean Formulas*
RBDD	*Reduced Binary Decision Diagram*
ROBDD	*Reduced and Ordered Binary Decision Diagram*
STRIPS	*STanford Research Institute Problem Solver*
UCS	*Uniform Cost Search*
IA	Inteligência Artificial

LISTA DE SÍMBOLOS

$\mathcal{D}$	Domínio de planejamento;
$\mathbb{P}$	Conjunto finito de átomos proposicionais;
$\mathbb{A}$	Conjunto finito de Ações;
$\mathbb{S}$	Conjunto finito de estados;
$\mathscr{L}$	Função de rotulação de estados ;
Σ	Sistema de transição de estados;
$\mathscr{T}$	Função de transição de estados;
$\mathcal{P}$	Problema de planejamento;
$\mathbb{G}$	Conjunto finito de estados meta;
δ	Função de custo de caminho;
φ	Fórmula proposiciona e estado meta;
ξ	Codificação;
$\mathbb{N}$	Conjunto dos números naturais;

SUMÁRIO

1 INTRODUÇÃO

Planejamento Automatizado é a subárea da Inteligência Artificial (IA) dedicada ao desenvolvimento de modelos e algoritmos autônomos que são capazes de planejar uma sequência de ações para que um agente inteligente alcance suas metas. O domínio de planejamento é uma descrição dos predicados, que representam as propriedades do agente e do ambiente, e das ações que representam as habilidades do agente em um dado ambiente (Russell; Norvig, 2013).

Um problema de planejamento é definido em termos do domínio de planejamento, do estado inicial e da meta de planejamento, e tem como solução um plano de ações, que define como um agente pode alcançar suas metas. Por sua vez, um planejador é um algoritmo que realiza uma *busca no espaço de estados* do domínio de planejamento, que pode ser representado explicitamente na forma de um grafo no qual cada estado é um nó e as transições de um estado para outro são representadas por arestas rotuladas por ações (Ghallab *et al.*, 2004).

O problema de decidir se existe ou não um plano solução para o problema de planejamento é PSPACE-completo (Russell; Norvig, 2013), o que significa que, durante a busca por uma solução, os algoritmos podem consumir uma quantidade exorbitante de memória, ocasionada por uma explosão de estados gerados. Usualmente, devido a este problema, um domínio de planejamento é descrito por meio de uma linguagem de descrição de ações. A *Planning Domain Definition Language* (*PDDL*) (Haslum *et al.*, 2019) tem sido adotada como a linguagem padrão para descrição de domínios e problemas de planejamento (Russell; Norvig, 2013). A Competição Internacional de Planejamento (do inglês *Internacional Planning Competition* (*IPC*)) é um evento que ocorre desde 1998, em conjunto com a Conferência Internacional de Planejamento (do inglês *International Conference on Automated Planning and Scheduling* (*ICAPS*)) (ICAPS, 2024), com o objetivo de impulsionar o aprimoramento da eficiência de planejadores.

Os planejadores realizam a busca por uma solução, gerando estados sucessores a partir do estado inicial (busca para frente) ou estados predecessores a partir da meta (busca para trás). Em ambos os casos, devido ao problema de explosão do espaço de estados, realizar a expansão de todos os nós necessários para se chegar a uma solução é uma tarefa inviável do ponto de vista computacional. Assim, na literatura de planejamento automatizado (Russell; Norvig, 2013) duas abordagens têm sido exploradas para contornar este problema:

- **Busca Heurística** (Hoffmann, 2001; Helmert, 2006; Richter; Westphal, 2008; Chen *et al.*, 2024);
- **Busca Simbólica** (Fourman, 2000; Edelkamp; Helmert, 2001; Menezes, 2014; Menezes *et*

al., 2014).

– Ou uma combinação de busca heurística e busca simbólica (Kissmann, 2012; Torralba *et al.*, 2014; Edelkamp *et al.*, 2015; Torralba *et al.*, 2017; Torralba *et al.*, 2018; Speck, 2022).

As buscas informadas, ou buscas heurísticas, utilizam o conhecimento específico de um problema para estimar, por meio de funções heurísticas, o quão próximo cada nó expandido está de um estado objetivo. Em IA, a busca $A*$ expande os nós com menor valor da função de avaliação $f(n) = g(n) + h(n)$, onde g é o custo para alcançar o nó e h é a estimativa do custo para se alcançar o objetivo (heurística) (Russell; Norvig, 2013). No entanto, em planejamento, as heurísticas devem ser independentes do domínio (Russell; Norvig, 2013), ou seja, devem ser capazes de guiar efetivamente a busca em uma grande variedade de problemas, sem depender de características específicas do domínio. Uma técnica bastante comum em planejamento para construir heurísticas independentes de domínios é utilizar um problema relaxado, isto é, um problema que possui menos restrições do que o problema original (Ghallab *et al.*, 2004).

A busca simbólica visa contornar o problema de explosão do espaço de estados por meio da representação de estados simbolicamente como um conjunto de estados, em vez de estados individuais. Nesta abordagem, estruturas de dados denominadas *Diagramas de Decisão Binária* (tradução do inglês, *Binary Decision Diagram* (*BDD*)) (McMillan, 1992), que são similares a uma árvore de decisão, são usadas para representar os estados e ações codificados como fórmulas da lógica proposicional.

O planejamento simbólico utilizando *BDD*s teve seu início na área de verificação de modelos para o desenvolvimento de planejadores não determinísticos. Recentemente, (Torralba *et al.*, 2017) propuseram unificar a capacidade de compactação da representação de estados da busca simbólica com algoritmos de busca heurística, tais como $A*$. Esta abordagem introduziu a técnica de busca denominada *Binary Decision Diagram* $A*$ (*BDD*$A*$), que superou as técnicas de busca estado da arte na maioria dos problemas *benchmark* da *IPC* em domínios determinísticos.

Dado o contexto, o trabalho de Silva (2019) propõe uma heurística para a busca simbólica em planejamento determinístico, calculada por meio de uma busca para trás no espaço de estados em um problema de planejamento relaxado. No entanto, essa abordagem enfrentou dificuldades em instâncias com grande espaço de estados, não conseguindo encontrar soluções devido ao consumo excessivo de memória e tempo na geração dos estados antecessores. Por outro lado, o trabalho de Kissmann (2012) propõe um limitante de tempo para a geração de heurísticas, o que torna possível a computação desses valores para problemas com muitos estados.

1.1 Objetivos

1.1.1 Objetivo Geral

Implementar e avaliar o cálculo heurístico proposto por Silva (2019), adicionando um limitante de tempo para a geração de estados predecessores, conforme proposto por Kissmann (2012), e implementar e avaliar a busca simbólica *A**.

1.1.2 Objetivos Específicos

- Implementar o cálculo heurístico regressivo proposto por Silva (2019), adicionando um limitante de tempo de computação para estados predecessores.
- Implementar uma busca heurística simbólica A* progressiva.
- Avaliar o desempenho dos algoritmos implementados utilizando domínios *benchmark* da *IPC*.

1.2 Organização do texto

Este trabalho está organizado conforme segue: O Capítulo 2 apresenta os conceitos teóricos relacionados ao planejamento clássico, incluindo como são realizadas a busca heurística e a busca simbólica, como é possível representar de maneira eficiente os domínios de planejamento e como são gerados os estados. O Capítulo 3 discute os trabalhos relacionados que serviram como base para o desenvolvimento desta pesquisa. Serão abordados tópicos como o trabalho de Torralba *et al.* (2017), heurísticas em planejamento automático, planejamento simbólico com heurísticas e um comparativo entre diferentes abordagens. O Capítulo 4 descreve os procedimentos metodológicos adotados, incluindo a estrutura da implementação, a organização dos algoritmos, o ambiente de desenvolvimento, as ferramentas utilizadas e o cronograma de execução.

O Capítulo 5 apresenta os resultados experimentais obtidos com a implementação dos algoritmos. São discutidos os domínios utilizados na avaliação, as métricas de desempenho consideradas, a análise comparativa entre os métodos testados e os impactos do uso do limitante de tempo na construção das heurísticas.

Por fim, o Capítulo 6 apresenta as conclusões deste trabalho, destacando as contribuições alcançadas, as limitações observadas e possíveis direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, apresentam-se os conceitos fundamentais necessários para compreender este trabalho. Na Seção 2.1, discute-se os conceitos de busca no espaço de estados, incluindo a utilização de Diagramas de Decisão Binária (BDDs) e a lógica proposicional para representar estados e transições, essenciais para o raciocínio simbólico. Em seguida, na Seção 2.2, aborda-se o planejamento automatizado, definindo formalmente um problema de planejamento e como representar ações utilizando linguagens de ações como *STanford Research Institute Problem Solver* (*STRIPS*). Serão exploradas as técnicas de busca para frente e para trás no espaço de estados, além do planejamento como busca heurística e simbólica, incluindo a codificação de ações em termos lógicos para aplicação em *BDD*s, o que permite a modelagem eficiente de problemas com estados e ações. E por fim a Seção 2.3 apresenta uma visão geral de *Planejamento como Busca Heurística Simbólica* e sobre a aplicação dessa abordagem no contexto deste trabalho.

2.1 Busca no Espaço de Estados

A busca por uma solução é um dos principais fundamentos da IA, uma vez que muitos problemas podem ser representados por meio da exploração do espaço de estados, onde o objetivo é encontrar o melhor caminho que leve o agente inteligente de uma situação inicial para suas metas (Russell; Norvig, 2013). Esses problemas de busca são frequentemente modelados como sistemas de transição de estados, com estados representando as possíveis configurações do mundo e transições representando a mudança entre estados provocadas por ações, conforme definido formalmente na Definição 2.1.1(Ghallab *et al.*, 2004).

Definição 2.1.1 (Sistema de Transição de Estados): Dados um conjunto de átomos proposicionais $\mathbb{P}$, que descrevem as características do agente e do ambiente, e um conjunto de ações $\mathbb{A}$ representando as habilidades do agente no ambiente, um sistema de transição de estados é definido por uma tupla $\langle \mathbb{S}, \mathscr{L}, \mathscr{T} \rangle$, em que:

- $\mathbb{S} = \{s_0, s_1, \cdots, s_n\}$ é um conjunto finito de estados possíveis do ambiente;
- $\mathscr{L} : \mathbb{S} \mapsto 2^{\mathbb{P}}$ é uma função de rotulação de estados; e
- $\mathscr{T} : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ é uma função de transição de estados.

Utilizaremos a suposição do mundo fechado (do inglês *Closed World Assumption* (*CWA*)) (Russell; Norvig, 2013) na qual consideramos na representação do estado apenas os átomos proposicionais que são verdadeiros, os átomos proposicionais que são falsos não são representados.

Deste modo, um **problema de busca** é definido formalmente pelos seguintes componentes (Russell; Norvig, 2013):

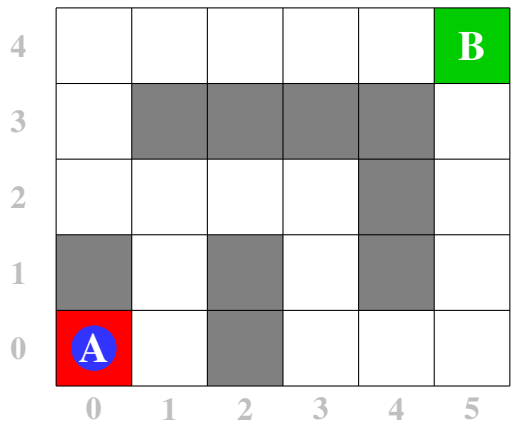
- **Estado inicial** s_0, em que $s_0 \in \mathbb{S}$;
- Uma descrição do **Sistema de transição de estados** $\langle \mathbb{S}, \mathscr{L}, \mathscr{T} \rangle$
- Uma **meta** que determina se um estado s pertence a um conjunto de estados objetivos ($\mathbb{G} \subset \mathbb{S}$);
- Uma **função de custo de caminho**, a soma dos custos, de cada passo, ou seja, um custo para sair do estado $s \in \mathbb{S}$ ao estado $s' \in \mathbb{S}$ por uma ação $a \in \mathbb{A}$:

$$\delta : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathbb{N}$$

, em que $\mathbb{N}$ é o conjunto dos números inteiros.

Assim, a solução para um problema de busca é encontrar **uma sequência de ações que leva o agente do estado inicial ao estado meta**.

Figura 1 – Um labirinto em que o agente deve sair do seu estado inicial (posição *A*) e alcançar a meta (posição B).



Fonte: Elaborado pelo autor baseado em University (2020).

O problema de encontrar um caminho em um labirinto é mais intuitivo para o entendimento de buscas. Na Figura 1 o agente, em azul, inicialmente na posição $(0,0)$ destacada na cor vermelho na figura, tem como objetivo alcançar a posição $(5,4)$, destacada na cor verde na figura, e as posições em cinza são paredes que impedem a passagem do agente. Assim, é

preciso em cada passo, expandir as posições possíveis, acumular os estados alcançados, gerando assim, uma fronteira de opções ainda não exploradas (também chamada de borda) e, a partir desses estados, decidir qual será expandido em uma próxima iteração.

Para definir formalmente o problema do labirinto, temos inicialmente que pensar no conjunto de átomos proposicionais $\mathbb{P}$ cuja combinação de valores-verdade podem descrever unicamente um estado do labirinto. Essa descrição pode ser pensada com predicados e variáveis que podem, posteriomente, ser instanciadas com valores constantes (resultando em átomos proposicionais), conforme a seguir:

- **at**$(\mathbf{x})$: predicado que descreve que o agente está na posição x do labirinto;
- **livre**$(\mathbf{x})$: predicado que descreve que a posição x do labirinto está livre, i.e, pode ser utilizada para passagem do agente e;
- **vizinho**$(\mathbf{x},\mathbf{y})$: predicado que descreve que a posição x do labirinto é vizinha da posição y.

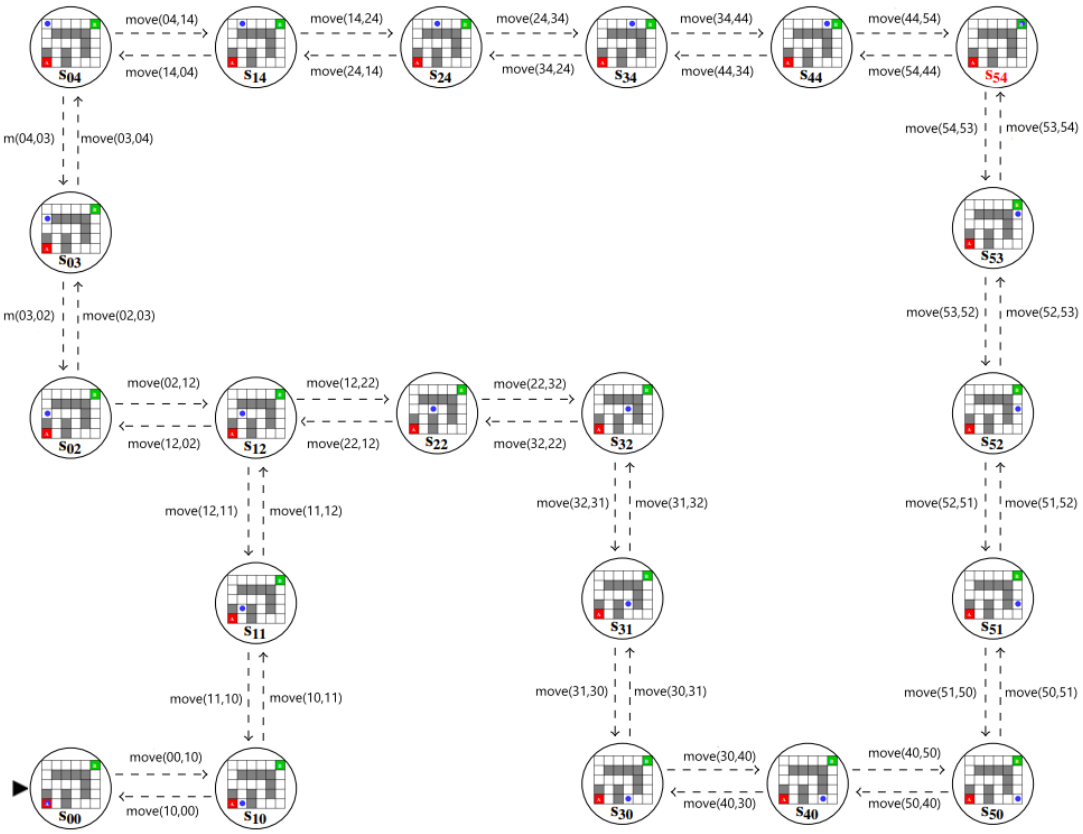
Para representar todas as possíveis configurações de atuação do agente no labirinto da Figura 1, instanciaremos os predicados acima com os valores de x correspondentes as posições do labirinto, isto é, $x = \{00, 10, ..., 54\}$:

Por exemplo, no estado s_{00} temos que: o agente está na posição 00 (cartesiana $(x:0, y:0)$) do labirinto (o átomo proposicional *at*(00) é verdade), as posições 00, 10, 11, 12, 02, 03, 04, 14, 24, 34, 44, 22, 32, 31, 30, 40, 50, 51, 52, 53 e 54 são livres (os átomos proposicionais livre(00), livre(10), ..., livre(54) são verdadeiros) e os vizinhos de s_{00} são as posições 10 e 01 (vizinho(00,10) e vizinho (00,01) são verdadeiros). As transições entre estados pode ser rotulada com uma ação do tipo $move(x,y)$ que causa a movimentação do agente que estava na posição x do labirinto para a posição y.

O problema do labirinto pode ser modelado da seguinte forma:

- **Estados**: Cada estado é $\{s_{00}, s_{01}, s_{02}, \cdots, s_{54}\} = \mathbb{S}$ é uma configuração possível de localização do agente no labirinto, sendo cada um rotulado ($\mathscr{L}$) por um conjunto de átomos proposicionais $\mathbb{P}$;
- **Estado inicial**: $s_{00} \in \mathbb{S}$ no qual o agente está na posição $(0,0)$ do labirinto;
- **Ações**: $\mathbb{A}$ = { mover(x, y) } em que x e y são posições do labirinto;
- **Sistema de transição de Estados**: conforme mostrado na Figura 2.
- **Meta**: estado s_{54} que define a posição (5,4) com a qual o agente deve chegar;
- **Custo de caminho:** Cada ação tem custo 1.

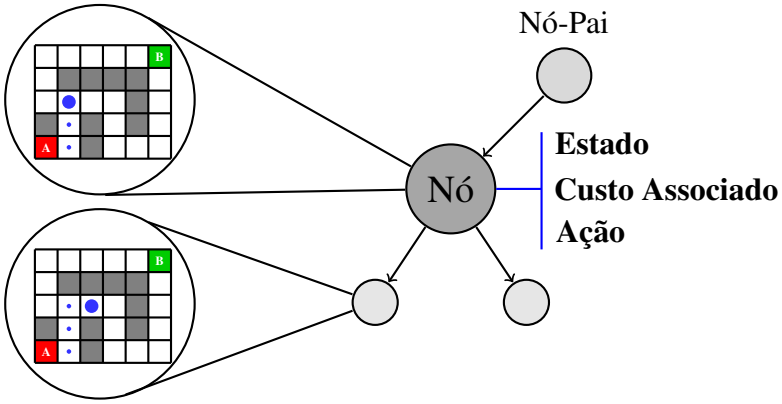
Figura 2 – Exemplo do modelo de transições para o problema do labirinto. Onde o estado inicial é definido pela seta preta (s_{00}) e o estado final com rotulo vermelho (s_{54}).



Fonte: Elaborado pelo autor.

Para resolver um problema formulado, **buscamos** uma solução que consiste em uma sequência de ações que partem do estado inicial s_0 e formam uma árvore de busca, conforme ilustrado na Figura 3. Nesta árvore, o nó raiz representa o estado inicial s_0, com custo acumulado zero e sem ação associada. Os nós subsequentes, derivados das ações possíveis, representam os estados alcançáveis a partir do estado inicial. Durante o processo de busca, a árvore é gerada expandindo-se à medida que testamos diferentes ações. Cada nó da árvore possui um custo associado e uma ação que leva ao seu estado. É importante notar que dois nós podem representar o mesmo estado se houver múltiplos caminhos para o mesmo objetivo, refletindo a diversidade de possíveis soluções para o problema. Utilizamos uma estratégia de busca para decidir qual nó da fronteira será expandido (Russell; Norvig, 2013).

Figura 3 – Representação de um fragmento da árvore de busca e como cada nó armazena um estado do problema.



Fonte: Elaborado pelo autor, baseado em (Russell; Norvig, 2013).

O algoritmo de busca em largura, *Breadth-First Search* (*BFS*), escolhe para expandir o nó que está a mais tempo na borda. Para isso, armazena os estados da borda em uma fila, estrutura de dados em que o primeiro elemento a entrar é o primeiro a sair (Szwarcfiter; Markenzon, 2010). Este algoritmo expande todos os caminhos igualmente, independente do custo de caminho, até encontrar a primeira solução (Cormen *et al.*, 2002). O algoritmo de busca em profundidade, *Depth-First Search* (*DFS*), escolhe o nó que está a menos tempo na borda. Armazenando os estados em uma pilha, estrutura de dados em que o primeiro a entrar é o primeiro a sair (Szwarcfiter; Markenzon, 2010). Já na busca de custo uniforme, *Uniform Cost Search* (*UCS*), cada nó n possui uma função de custo $g(n)$. Essa busca utiliza uma fila de prioridade como borda. Assim, em cada iteração, é escolhido o nó com menor valor $g(n)$ para expansão, priorizando os caminhos de menor custo (Russell; Norvig, 2013).

Esses algoritmos de busca são denominados busca sem informação, ou busca cega, uma vez que não há informações (guia) sobre o quão distante um nós expandido está de um estado meta. Por outro lado, a busca informada utiliza heurísticas para guiar a exploração ao estado meta.

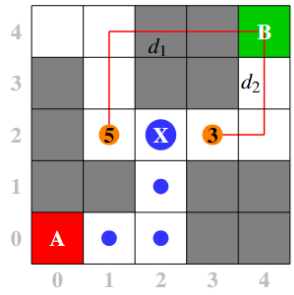
2.1.1 Busca Heurística no Espaço de Estados

Uma heurística é uma função que estima o quão promissor está um nó n em relação a um estado meta. Assim em algoritmos de busca heurística, adicionalmente à entrada do problema, faz-se necessário definir, para cada nó n, o seu valor heurístico $h(n)$. Exemplos de algoritmos de busca heurística incluem a busca gulosa de melhor escolha (do inglês *GBFS*), e o algoritmo $A*$ (Ghallab *et al.*, 2004). Ambos algoritmos utilizam filas de prioridades para armazenar os nós da

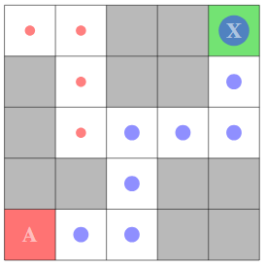
borda. No entanto, na busca *GBFS* os nós são ordenados segundo o valor $f(n) = h(n)$. Já na busca A^*, os nós são ordenados segundo o valor $f(n) = g(n) + h(n)$.

Em muitos casos a heurística é pensada em problemas relaxados, ou seja, removendo certos fatores ou impeditivo, muitas vezes considerações otimistas do problema (Russell; Norvig, 2013). No caso do problema do labirinto, podemos utilizar como heurística a distância *Manhattan* (Russell; Norvig, 2013) de cada nó para a posição meta, ignorando as paredes. Esta distância, também conhecida como distância L1 ou distância de táxi, é a soma das distâncias horizontais e verticais entre dois pontos em uma grade. A Figura 4 ilustra que no momento de uma bifurcação, Figura 4(a), onde o agente tem que escolher um dos lados, a distância *Manhattan* de cada opção para a meta, $d_1 = h(s_{12}) = 5$ e $d_2 = h(s_{32}) = 3$. Além disto, escolhendo arbitrariamente o agente pode expandir nós desnecessariamente ou seguir por um caminho sem saída, Figura 4(b), enquanto que guiado por uma heurística o agente pode chegar no estado meta sem muitos problemas (Figura 4(c)).

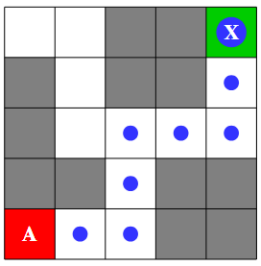
Figura 4 – Exemplos de busca com e sem heurística na escolha dos nós.



(a) Momento da Escolha.



(b) Exemplo da expanção desnecessária de nós por não usar heurística.



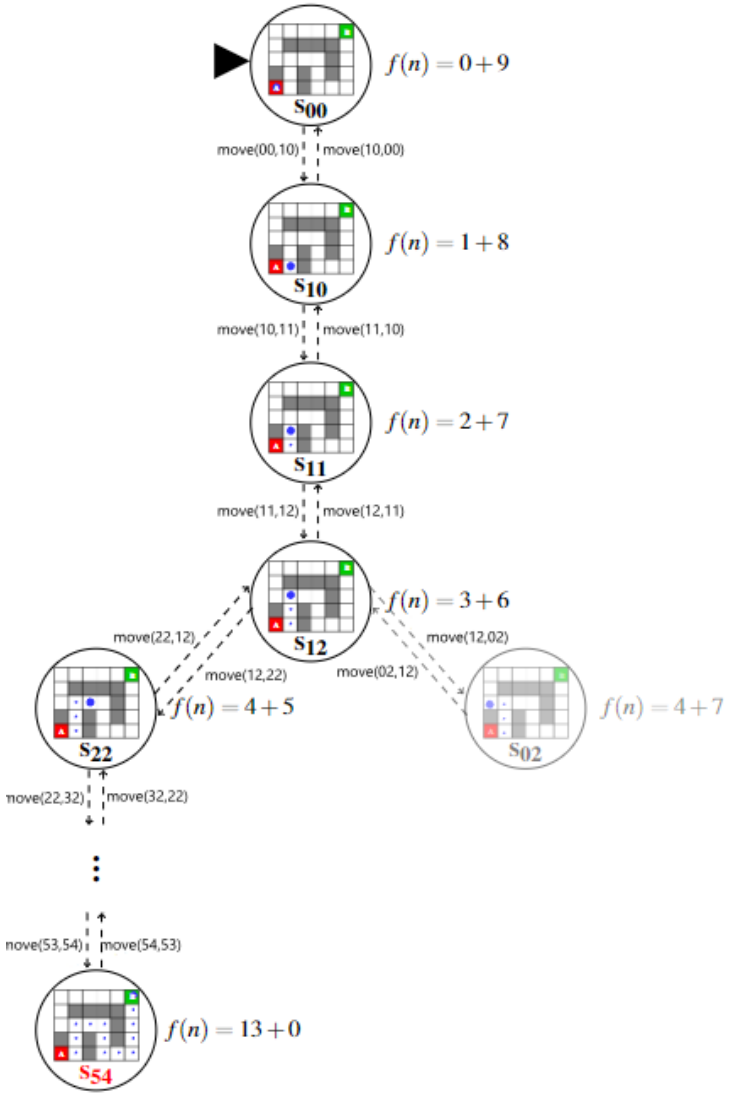
(c) Exemplo da expanção de nós quando se utiliza heurística.

Fonte: Elaborado pelo autor baseado em University (2020).

A execução do algoritmo *GBFS* pode ser representado na forma de árvore de busca (Figura 5), onde cada nó apresenta um estado s e contém um valor associado, função heurística $f(n) = h(n)$. Pode-se perceber que no momento em que o agente se encontra no estado s_{12} ele deve escolher um dentre dois possíveis estados sucessores: s_{02} ou s_{22}. Então, o algoritmo *GBFS*,

deve escolhe o estado com menor valor heurístico $h(n)$, menor distância de *Manhattan*, sendo eles h(s_{02}) = 7 e h(s_{22}) = 5. O algoritmo escolhe para expandir o estado s_{22}. Porém, pode-se perceber que o algoritmo acaba por expandir, neste caso, o maior caminho no lugar do menor. O problema está no fato do caminho escolhido, mais tarde, alcançar valores heurísticos maiores, ou seja, ele não tem a capacidade de perceber que o caminho escolhido era maior[1].

Figura 5 – Árvore de busca para o algoritmo *GBFS* no problema do labirinto. Onde a raiz da árvore é apontada por um triângulo preto. E cada nó tem sua função heurística $f(n) = g(n) + h(n)$, mesmo que o algoritmo olhe apenas para o valor heurístico $h(n)$.



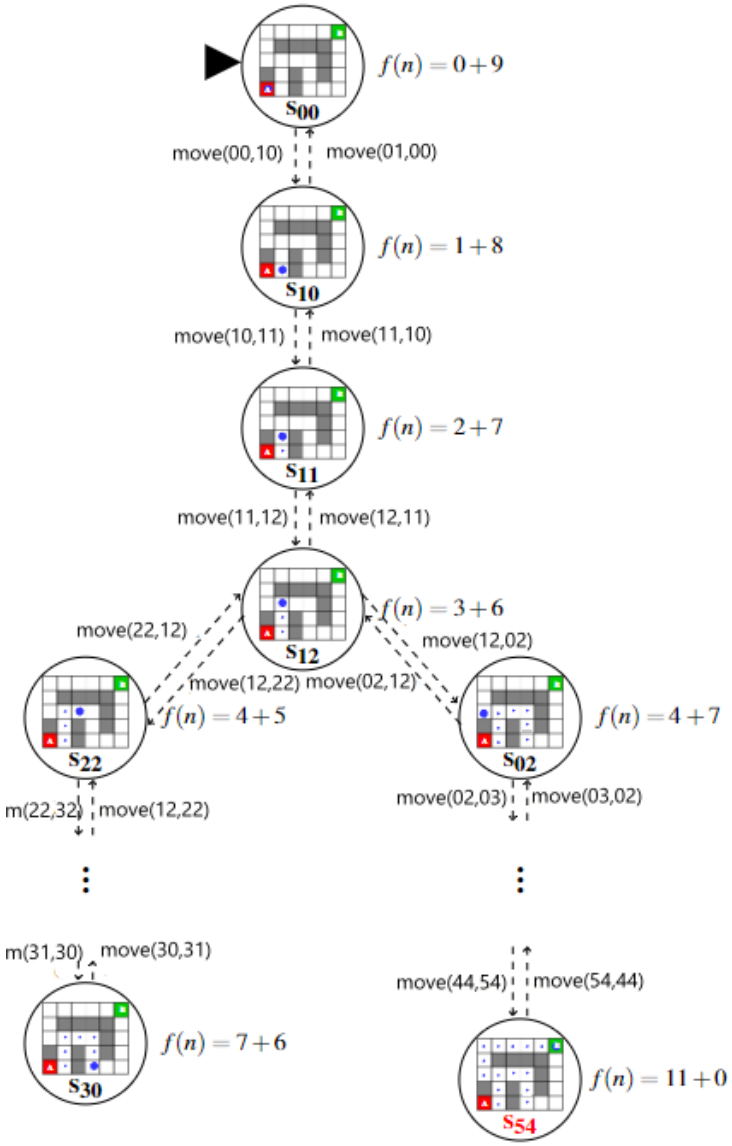
Fonte: Elaborado pelo autor.

A busca $A*$, por sua vez, busca balancear o valor heurístico $h(n)$ com a quantidade de passos $g(n)$, através da função heurística $f(n) = g(n) + h(n)$. Na prática, ele prefere um caminho heuristicamente pior, se puder chegar na meta mais rápido. A Figura 6 ilustra que no

[1] Pode-se ver essa diferença de caminho na Figura 20 no Apêndice A.

momento da escolha, s_{12}, o algoritmo $A*$ vai continuar escolhendo o estado s_{22} por ter o valor da função heurística $f(s_{22}) = 4+5 = 9$ menor que $f(s_{02}) = 4+7 = 11$. No entanto, no momento que expandir o nó s_{30} o algoritmo "percebe", por meio da fila de prioridade, que $f(s_{30}) > f(s_{02})$, logo vai começar a expandir um novo caminho promissor, removendo, neste caso, da fila de prioridade o nó com estado s_{02}, até chegar no estado meta.

Figura 6 – Árvore de busca para o algoritmo $A*$ no problema do labirinto.



Fonte: Elaborado pelo autor baseado em University (2020).

Podemos perceber que embora o algorítimo $A*$ seja de fácil implementação, algoritmo da Figura 7, a escolha da heurística pode ser um desafio, pois, quanto melhor for a heurística melhor se resolve o problema com menos estados explorados. No entanto, heurísticas nem sempre são ótimas, uma vez que podem superestimar ou subestimar o custo real da solução.

Quando isso acontece, o algoritmo pode explorar caminhos subótimos, resultando em soluções que não são as mais eficientes. Dessa forma, a heurística, apesar de poderosa, não garante a obtenção da solução ideal, mas sim uma solução suficientemente boa para uso prático (Russell; Norvig, 2013).

Figura 7 – Peseudocódigo da busca informada.

Algorithm 1 Peseudocódigo do algoritmo *A**.

Require: *initialState*: **State**, *goalState*: **State**, *actions*: **Action[ ]**
Ensure: Caminho do *initialState* ao *goalState* se encontrado

```
1:  seja frontier uma PriorityQueue
2:  seja explored uma empty set
3:  seja fCost = 0 + 0
4:  frontier.enfileira(Node(initialState,fCost))
5:
6:  while frontier não é vazia do
7:      seja node = frontier.desenfileira()
8:      seja state = node.state
9:      if state == goalState then
10:         retorna Solução(node)
11:     end if
12:
13:     for each action em actions do
14:         seja childState = Progredir(state, action)
15:         fCost = action.cost + HeuristicCost(state, goalState)
16:         seja childNode = Node(childState, fCost, action, node)
17:         if childNode não está em explored or não está em frontier then
18:             frontier.enfileira(childNode)
19:         else if childNode em frontier com custo maior then
20:             substitua o nó em frontier por childNode
21:         end if
22:     end for
23: end while
24: return Falha()
```

Fonte: Baseado no algoritmo proposto por Russell e Norvig (2013).

2.1.2 Busca Simbólica no Espaço de Estados

A busca simbólica contorna o problema da explosão do espaço de estados, utilizando a representação de conjuntos de estados no lugar de representar estados individuais, geralmente um grafo. Esse tipo de representação é denominada representação simbólica do espaço de estados (Huth; Ryan, 2004). Além disso, este tipo de busca é capaz de raciocinar sobre a representação

simbólica, o que chamamos de raciocínio simbólico.

Na busca simbólica, estruturas de dados chamadas Diagramas de Decisão Binária (BDDs, do inglês *Binary Decision Diagrams*) são utilizadas para representar o conjunto de estados e transições e raciocinar sobre essas representações.

2.1.2.1 Lógica Booleana Quantificada

A Lógica Booleana Quantificada (do inglês *Quantified Boolean Formulas* (*QBF*)) (Buning *et al.*, 1995) é uma extensão da lógica proposicional com quantificadores existencial e universal sobre os valores-verdade dos átomos proposicionais. Formalmente, a sintaxe da lógica QBF pode ser definida por uma gramática na forma de *Backus Naur* (do inglês *Backus Naur Form* (*BNF*)) como mostrado na Definição 2.1.2.

Definição 2.1.2 (Sintaxe da Lógica QBF): (Buning *et al.*, 1995)

$$\varphi ::= p \mid (\neg\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid \exists p.\varphi \mid \forall p.\varphi,$$

em que p representa qualquer átomo proposicional e cada ocorrência φ a direita de '::=' representa qualquer fórmula já construída.

A quantificação existencial dos valores de um átomo proposicional p em uma fórmula φ é definida como a seguir (Buning *et al.*, 1995):

$$\exists p.\,\varphi \equiv \varphi[1/p] \vee \varphi[0/p] \tag{2.1}$$

Em que $\varphi[1/p]$ é a fórmula resultante de se substituir cada ocorrência de p na fórmula φ por 1 (valor verdade) e $\varphi[0/p]$ é a fórmula resultante de se substituir cada ocorrência de p na fórmula φ por 0 (valor falso).

A quantificação universal dos valores de um átomo proposicional p em uma fórmula φ é definida como a seguir (Buning *et al.*, 1995):

$$\forall p.\,\varphi \equiv \varphi[1/p] \wedge \varphi[0/p] \tag{2.2}$$

Exemplo Seja $\varphi = (p \wedge q) \vee r$ uma fórmula proposicional sobre o conjunto de proposições $\mathbb{P} = \{p, q, r\}$, a quantificação existencial da fórmula φ sobre os valores da proposição p é dada por:

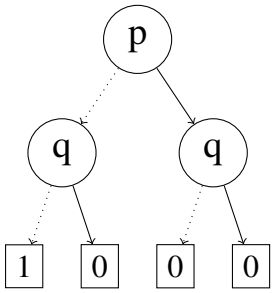
$$
\begin{array}{lcccc}
\exists p.\varphi & \equiv & \varphi[1/p] & \vee & \varphi[0/p] \\
 & \equiv & ((1 \wedge q) \vee r) & \vee & ((0 \wedge q) \vee r)) \\
 & \equiv & (q \vee r) & \vee & (r) \\
 & \equiv & (q \vee r) & \vee & (r) \\
 & \equiv & q & \vee & r
\end{array}
$$

Percebe-se que a aplicação do quantificação existencial de p sobre a fórmula φ tem o efeito de filtrar a ocorrência de p na fórmula φ.

2.1.2.2 Diagramas de Decisão Binária

Diagramas de Decisão Binária (Huth; Ryan, 2004) são estruturas de dados, semelhantes à árvores de decisão binária, que permitem uma representação compacta de funções proposicionais as quais só possuem representações exponenciais em outros sistemas tais como tabelas-verdades e formas normais conjuntivas (*QBF*). *BDD*s são grafos acíclicos, finitos e direcionados com um único nó inicial, em que os nós internos são rotulados com átomos proposicionais e os nós terminais são rotulados com os valores-verdade 0 (falso) e 1 (verdadeiro). Cada nó interno *n* possui duas arestas saindo: uma aresta pontilhada, representando que o átomo proposicional correspondente ao nó possui valor-verdade 0 e; uma aresta sólida, representando que o átomo proposicional correspondente ao nó possui valor-verdade 1.

Figura 8 – Exemplo de uma Árvore de Decisão Binária para a fórmula $\neg(p \vee q)$.



Fonte: Baseado em (Huth; Ryan, 2004).

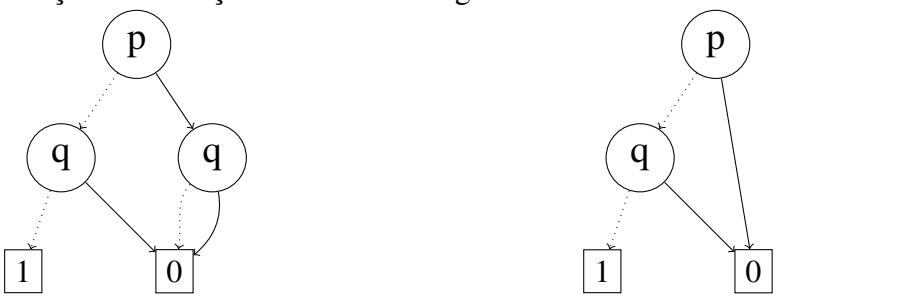
A Figura 8 ilustra a árvore de decisão binária que representa as possíveis valorações para a fórmula $\neg(p \vee q)$. Nesta fórmula, por exemplo, quando p possui valor-verdade 0 (aresta pontilhada saindo do nó p) e q possui valor-verdade 0 (aresta pontilhada saindo do nó q), a fórmula possui valor verdade 1. No entanto, note que há redundâncias nesta representação e que possíveis simplificações poderiam ser realizadas de modo que a estrutura continue representando os possíveis valores-verdade das fórmulas, mas que façam isso em uma estrutura mais compacta.

As possíveis reduções que podem ser feitas são:

- **R1**: Eliminação de nós terminais redundantes;
- **R2**: Eliminação de testes redundantes;
- **R3**: Eliminação de nós internos redundantes.

Ao realizar todas as possíveis reduções R1, R2 e R3, dizemos que a árvore está na forma reduzida, chamada *Reduced Binary Decision Diagram* (*RBDD*). Outra importante característica é a ordenação nos átomos proposicionais dispostos ao longo de qualquer caminho do *BDD*. Assim, se uma ordem dos átomos for mantida para todos os caminhos do *BDD*, dizemos que o *BDD* é ordenado (do inglês *Ordered Binary Decision Diagram* (*OBDD*)). Os *BDD*s ordenados e reduzidos são denominados (do inglês *Reduced and Ordered Binary Decision Diagram* (*ROBDD*)). Neste trabalho, chamaremos os *RBDD*s de apenas *BDD*s.

Figura 9 – Aplicação das reduções ao BDD da Figura 8.



(a) Eliminação de nós terminais redundantes. (b) Eliminação de nós e testes redundantes.

Fonte: Baseado em (Huth; Ryan, 2004).

A Figura 9(a) ilustra a aplicação da redução **R1** (remoção de folhas redundantes) no BDD da Figura 8 e; a Figura 9(b) ilustra a aplicação da redução R2 e R3 na Figura 8. Como nenhuma outra redução pode ser aplicada e a ordem em todos caminhos do *BDD* é *p* depois *q*, podemos dizer que a Figura 9(b) é um *ROBDD*.

2.1.2.3 Representação de Estados e Conjunto de Estados como Fórmulas Proposicionais

Nesta seção, apresentaremos como estados e transições de um sistema de transição de estados (aplicado ao problema do labirinto, Figura 1) podem ser representados como fórmulas da lógica proposicional[2]. Essa representação é fundamental para realização do raciocínio simbólico desta estrutura.

[2] O problema do labirinto é uma ótimo exemplo para analogias, porém deixa as fórmulas extensas e exaustivas de representar.

A modelagem de um domínio precisa ser bem definida. Neste trabalho, seguimos com a modelagem do problema do labirinto (Figura 10). Nele, como a única ação do agente, *move*, altera apenas sua posição no ambiente, podemos, para fins de simplificação das fórmulas, considerar fixos os elementos estruturais do ambiente, como posições livres e relações de vizinhança.

- at(xy): o agente pode estar em qualquer posição do espaço cartesiano 6×5, ou seja, todas as posições (x,y) com $x \in [0,5]$ e $y \in [0,4]$. Denotado por: at$(00), \ldots,$ at(54)
- livre(xy): representa que as posições (x,y) que não são paredes. Todas as posições são consideradas livres, exceto: 01, 20, 21, 13, 23, 33, 53, 42, 41. Denotaremos por livre$(00), \cdots,$ livre(54)
- vizinho(xy,zw): indica que (zw) é uma posição adjacente (vizinhos em uma das quatro direções) à posição (xy), respeitando os limites do labirinto. Denotaremos por : vizinho$(00,10), \ldots,$ vizinho$(54,44)$

Assim, definimos o conjunto de todas as proposições possíveis $\mathbb{P}$ por:

$$\mathbb{P} = \{\text{at}(00), \ldots, \text{at}(54), \text{livre}(00), \cdots, \text{livre}(54), \text{vizinho}(00,10), \ldots, \text{vizinho}(54,44)\}$$

Codificação de um estado: Um estado $s \in \mathbb{S}$ pode ser representado como uma conjunção de todos os átomos proposicionais $p \in \mathbb{P}$ que estão definidos pela função de rotulação de s, $\mathscr{L}(s)$, e com a conjunção da negação de todos os átomos proposicionais $q \in \mathbb{P}$ não definidos em $\mathscr{L}(s)$. A codificação proposicional ξ de um estado $s \in \mathbb{S}$ é denotada pela fórmula (Huth; Ryan, 2004):

$$\xi(s) = \bigwedge_{p \in \mathscr{L}(s)} p \wedge \bigwedge_{q \in \mathbb{P} \setminus \mathscr{L}(s)} \neg q \tag{2.3}$$

Por exemplo, sendo a rotulação do estado inicial s_{00}, no qual o agente está na posição $(0,0)$ do labirinto (Figura 10),

$$\mathscr{L}(s_{00}) = \{at(00), livre(00), \cdots, livre(54), vizinho(00,10), \cdots, vizinho(54,44)\},$$

a codificação proposicional deste estado é:

$$\xi(s_{00}) = [at(00) \wedge livre(00) \wedge \cdots \wedge livre(54) \wedge vizinho(00,10) \wedge \cdots \wedge vizinho(54,44)]$$
$$\wedge \quad [\neg at(10) \wedge \cdots \wedge \neg at(54) \wedge \neg \text{livre}(01) \wedge \cdots \wedge \neg \text{livre}(41)]$$

Por sua vez, a codificação proposicional do estado s_{10}, no qual o agente está na posição (1,0) do labirinto e tem como vizinhos s_{00} e s_{11} (Figura 10),

$$\mathscr{L}(s_{10}) = \{at(10), livre(00), ..., livre(54), vizinho(10,00), vizinho(10,11), vizinho(10,20)\},$$

a codificação proposicional deste estado é:

$$\xi(s_{10}) = [at(10) \wedge livre(00) \wedge \cdots \wedge livre(54) \wedge vizinho(00,10) \wedge \cdots \wedge vizinho(54,44)]$$
$$\wedge \quad [\neg at(00) \wedge \cdots \wedge \neg at(54) \wedge \neg \text{livre}(01) \wedge \cdots \wedge \neg \text{livre}(41)]$$

Codificação de conjunto de estados: Um conjunto de estados $X \subseteq \mathbb{S}$ é representado simbolicamente como uma disjunção das fórmulas proposicionais representando os estados pertencentes a X, dada pela seguinte fórmula (Huth; Ryan, 2004):

$$\xi(X) = \bigvee_{s \in X} \xi(s) \tag{2.4}$$

Apenas pra exemplificar, dado um conjunto de estados $X = \{s_{00}, s_{11}\}$, temos

$$\xi(X) = \xi(s_{00}) \vee \xi(s_{10})$$

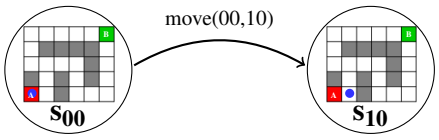
2.2 Planejamento Automatizado

Planejamento Automatizado é a subárea da IA que estuda o raciocínio de escolha de ações para que um agente inteligente alcance suas metas (Russell; Norvig, 2013). O ambiente em que o agente atua é denominado *domínio de planejamento* $\mathscr{D}$ que pode ser definido em termos de um sistema de transição de estados Σ no qual os estados $a \in \mathbb{S}$ são as possíveis configurações do ambiente e as transições são ocasionadas pelas ações $a \in \mathbb{A}$(Ghallab *et al.*, 2004), como mostrado na Definição 2.1.1. No entanto, a descrição completa do sistema de transição de estados não é apropriada para a maioria dos domínios reais devido ao problema da explosão do espaço de

estados (Russell; Norvig, 2013). No labirinto, por exemplo, conforme aumenta o tamanho do labirinto, há um aumento exponencial de proposições para representar as possíveis localizações do agente e as possíveis configurações de cada posição do labirinto. Assim, a comunidade de planejamento automatizado desenvolveu uma forma de representar os domínios por meio de linguagens de ações tais como STRIPS (*Stanford Research Institute Problem Solver*) (Fikes; Nilsson, 1971), PDDL (*Planning Domain Description Language*) (Aeronautiques *et al.*, 1998) e RDDL (*Relational Dynamic Influence Diagram Language*)(Sanner *et al.*, 2010).

Em linguagens como *STRIPS* e *PDDL* as ações do agente são representadas em termos de suas pré-condições e efeitos. As pré-condições são os átomos proposicionais que devem ser verdadeiros em um estado para que a ação seja executada. Os efeitos são os literais que tem seus valores modificados com a execução da ação (i.e, eram verdadeiros no estado anterior e passam a ser falso no próximo estado ou eram falsos no estado anterior e passam a ser verdadeiros). Os efeitos são divididos em dois conjuntos: os efeitos positivos, que são aqueles que passam a ser verdadeiros no estado sucessor, e os efeitos negativos, que são aqueles que tornam-se falsos no estado sucessor (Ghallab *et al.*, 2004).

Figura 10 – Dois estados s_{00} e s_{10} e uma transição rotulada por ação no domínio do labirinto.



Fonte: Elaborado pelo autor baseado em Silva (2019).

Assim, considere, por exemplo, no domínio do labirinto a ação *move*(00,10), Figura 10, que é responsável por mover o agente que está na posição 00 para a posição 10. Esta ação pode ser descrita em uma linguagem de ações, conforme mostrado na Figura 11. As pré-condições desta ação são: at(00) (o agente deve estar na localização 00), livre(10) (deve ser possível para o agente transitar para a localização 10) e vizinho(00,10) (as localização 00 e 10 devem ser vizinhas). Os efeitos positivos são (lista de adição): é $\{at(10)\}$. Os efeitos negativos (lista de deleção) são: $\{at(00)\}$.

Figura 11 – A ação STRIPS move(00,10) que move o agente da posição 00 para a posição 10 do labirinto.

move(00**,** 10)
precond$(move(00,10))$: { at(00), vizinho(00,10), livre(10) }
$add(a) : \{at(10)\}$
$del(a) : \{at(00)\}$

Fonte: Elaborado pelo autor.

Por sua vez, um *problema de planejamento* $\mathcal{P}$ (Definição 2.2.3), é dado em termos do *domínio do planejamento*, do *estado inicial* do agente e um conjunto de estados de aceitação, também denominados estados *meta*, a ser alcançado. A solução é um *plano*, uma sequência de ações que leva o agente do estado inicial a um estado meta (Ghallab *et al.*, 2004).

Definição 2.2.3 (Problema de planejamento): Um problema de planejamento é definido por uma tupla $\mathcal{P} = (\mathcal{D}, s_0, \mathbb{G})$, onde:

- $\mathcal{D}$ é o domínio de planejamento;
- $s_0 \in \mathbb{S}$ é o estado inicial do ambiente; e
- $\mathbb{G} = \{s_1, s_2, ..., s_n\} \subseteq \mathbb{S}$ é o conjunto de estados meta.

No domínio do labirinto (Figura 1) podemos definir um problema de planejamento em que s_{00} é o estado inicial e a meta é fazer o agente alcançar o estado s_{54} (i.e., φ = *at*(54)). No problema definido anteriormente um possível plano solução é a sequência de ações $move(00,10)$, $move(10,11)$, $move(11,12)$, $move(12,02)$, $move(02,03)$, $move(03,04)$, $move(04,14)$, $move(14,24)$, $move(24,34)$, $move(34,44)$ e $move(44,54)$.

Para obter um plano solução, um algoritmo de planejamento precisa explorar o espaço de estados de forma sistemática, expandindo o espaço de estados para frente (a partir do estado inicial) ou pra trás (a partir do conjunto de estados meta) (Russell; Norvig, 2013).

2.2.1 Busca para Frente no Espaço de Estados

A busca para frente no espaço de estados inicia-se a partir do estado inicial s_0, verificando em cada passo quais ações $a \in \mathbb{A}$ são *aplicáveis* para geração de estados sucessores. Cada estado sucessor é, por sua vez, expandido e o processo encerra quando um estado s satisfaz a meta de planejamento (i.e., $s \models \varphi \in \mathbb{S}$) é gerado. É importante ressaltar que o sistema de

transição de estados, como o da Figura 2, é gerado sob demanda por meio das ações STRIPS (Figura 11) a partir de um dado estado inicial.

Uma ação $a \in \mathbb{A}$ é aplicável em um dado estado $s \in \mathbb{S}$ se $precond(a) \subseteq \mathscr{L}(s)$, isto é, se o estado satisfaz as pre-condições da ação. O estado sucessor de um estado s por meio de uma ação a é computado da seguinte forma: para cada ação aplicável em s, remove-se os efeitos negativos da ação no estado e adiciona-se os efeitos positivos desta mesma ação no estado s (Ghallab *et al.*, 2004).

A busca para frente a partir de um estado s por uma ação a, computa um estado sucessor $s' \in \mathbb{S}$ como a seguir:

$$\mathscr{L}(s') = prog^a(s) = (\mathscr{L}(s) \setminus del(a)) \cup add(a) \quad [\text{se } precond(a) \subseteq \mathscr{L}(s)] \tag{2.5}$$

2.2.2 Busca para trás no Espaço de Estados

A busca para trás no espaço de estados inicia a partir do conjunto de estados metas $s_g \in \mathbb{G}$. Esta busca gera, em cada passo, um conjunto de estados predecessores até que o estado inicial seja alcançado. Essa busca também é denominada busca no espaço de *estados relevantes*, uma vez que consideramos apenas ações que são relevantes para meta (Ghallab *et al.*, 2004).

Uma ação a é relevante a um estado s se ela contribui com as propriedades satisfeitas em s, i.e., no mínimo um dos efeitos da ação a deve pertencer ao estado s e também a ação a não deve ter um efeito que negue um propriedade do estado s, i.e., a é relevante para s: $add(a) \cap \mathscr{L}(s) \neq \oslash$ e $del(a) \cap \mathscr{L}(s) = \oslash$. Assim, dado a descrição de um estado[3] s', a descrição de um estado predecessor (parcial) s é dada por:

$$\mathscr{L}(s') = regr^a(s) = (\mathscr{L}(s) \setminus add(a)) \cup precond(a). \tag{2.6}$$

2.2.3 Planejamento como Busca Heurística

Devido ao problema da explosão do espaço de estados, é necessário ter uma espécie de guia (heurística) para auxiliar na escolha do melhor estado a ser expandido em cada passo da busca. As estratégias de busca heurística são capazes de identificar se um estado a ser expandido é "mais promissor" que outro (Russell; Norvig, 2013) . A busca heurística tem sido a principal

[3] A descrição dos estados na busca regressiva é parcial., i.e., um estado pode representar um conjunto de estados.

abordagem utilizada por planejadores vencedores nas edições da Competição Internacional de Planejamento *IPC* (ICAPS, 2024).

Em planejamento, uma heurística deve ser **independente do domínio**, ou seja, pensada para ser aplicável em diferentes domínios descritos em uma linguagem de ações (Russell; Norvig, 2013). Uma das formas de se construir uma heurística independente de domínio é relaxar o problema original ignorando aspectos que o tornam difícil de resolver como, por exemplo, ignorando os efeitos negativos das ações (Ghallab *et al.*, 2004).

2.2.4 Planejamento como Busca Simbólica

Uma outra estratégia para contornar o problema da explosão do espaço de estados é utilizar a busca simbólica que expande conjuntos de estados no lugar de expandir estados individuais. A busca simbólica é tópico de pesquisa em Planejamento Automatizado desde o trabalho de (Fourman, 2000), o qual propôs o planejador PROPPLAN, e dos trabalhos que relacionam a área de planejamento com verificação de modelos (*planning as model checking*) (Cimatti *et al.*, 2003). Mais recentemente, planejadores baseados em busca simbólica foram os vencedores de competições internacionais de planejamento (Taitler *et al.*, 2024; Bongartz, 2018; Vallati *et al.*, 2015; Edelkamp *et al.*, 2015) segundo ICAPS (2024).

Para realizar a busca simbólica é necessário representar estados e ações como fórmulas da lógica proposicional. Estas fórmulas são codificadas como diagramas de decisão binária e o raciocínio simbólico (expansão de estados) é realizado por meio de operações nesses diagramas. A codificação de estados é tal qual mostrado na Seção 2.1.2 e a codificação de ações é realizada como mostrada a seguir.

Codificação de ações: Uma ação $a = \langle \text{precond}(a), efeitos(a) \rangle$ é representada simbolicamente como sendo um par de fórmulas $a = \langle \xi(\text{precond}(a); \xi(\text{efeitos}(a)) \rangle$ onde:

– $\xi(\text{precond}(a))$ é um literal ou uma conjunção de literais que representam as precondições da ação a:

$$\xi(\text{precond}(a)) = \bigwedge_{p \in \text{precond}(a)} p \tag{2.7}$$

– $\xi(\text{efeitos}(a))$ é um literal ou uma conjunção de literais que representam os efeitos positivos de a e os efeitos negativos de a:

$$\xi(\text{efeitos}(a)) = \bigwedge_{q \in \text{add}(a)} q \quad \wedge \bigwedge_{r \in \text{del}(a)} \neg r \tag{2.8}$$

Como exemplo, mostraremos a codificação da ação move$(00,10)$ representada na Figura 2, sendo representada pela fórmula:

$$\xi(\text{move}(00,10)) = \langle \xi(\text{precond}(\text{move}(00,10))), \xi(efeitos(\text{move}(00,10))) \rangle$$

em que,

$$\xi(\text{precond}(a)) = \text{at}(00) \ \wedge \ \text{vizinho}(00,10) \ \wedge \ \text{livre}(10)$$

$$\xi(efeitos(a)) = \text{at}(10) \ \wedge \ \neg\text{at}(00)$$

Adicionalmente é definido para cada ação $a \in \mathbb{A}$ o conjunto modifica(a) contendo todos os átomos proposicionais presentes nos efeitos da ação a. Por exemplo, para a ação move$(00,10)$ temos:

$$\text{modifica}(move(00,10)) = \{ \ at(00), at(10) \ \}$$

Este conjunto especial será utilizado para filtrar as proposições do estado simbólico s, por meio da aplicação do quantificador existencial (equação 2.1). Dessa forma, removemos do estado os efeitos associados à ação — tanto positivos quanto negativos[4] — evitando contradições lógicas na fórmula. Após essa remoção, os efeitos positivos da ação podem ser adicionados de forma segura ao estado resultante.

2.2.4.1 Busca Simbólica para Frente no Espaço de Estados

Dado um conjunto de estados $X \subseteq \mathbb{S}$ e um conjunto de ações $a \in \mathbb{A}$, representadas por fórmulas proposicionais, Fourman (2000) propôs a computação de estados sucessores a partir dessa representação, uma operação denominada progressão simbólica.

A progressão simbólica do conjunto X por uma ação a devolve um conjunto de estados $Y \in \mathbb{S}$ que pode ser alcançado ao executarmos a ação a em estados de X. A progressão de X pela ação a pode ser expressa por Fourman (2000):

$$\xi(progr(X,a)) = \exists \, modifica(a) \cdot (\xi(X) \wedge \xi(\text{precond}(a))) \wedge \xi(efeitos(a)) \qquad (2.9)$$

[4] Ou apenas os positivos, no caso do problema relaxado.

A Equação 2.9 é a versão simbólica da Equação 2.5 para cálculo de estados sucessores. A fórmula $\xi(\text{progr}(X,a))$ representa o conjunto Y de estados alcançáveis a partir da aplicação de uma ação a sobre estados do conjunto X. Para isso, é necessário: (i) verificar a aplicabilidade da ação a no conjunto de estados X por meio da conjunção de $\xi(X)$ com $\xi(\text{precond}(a))$; (ii) eliminar as variáveis presentes nos efeitos da ação a por meio da aplicação da quantificação existencial das variáveis em modifica(a) e; (iii) realizar a conjunção desta fórmula com os efeitos da ação a.

Utilizaremos o exemplo da Figura 10, onde $a = \text{move}(00,10)$ e o conjunto $\xi(X)$ pode ser descrito pela Fórmula 2.4, onde $X_0 = \{s_{00}\}$, o primeiro passo é calcular $\xi(X_0) \vee \xi(\text{precond}(a))$, de tal forma:

$$\begin{aligned}\xi(X_0) \vee \xi(\text{precond}(a)) = {} & [(at(00) \wedge livre(00) \wedge \cdots \wedge livre(54) \wedge vizinho(00,10) \wedge \cdots \wedge vizinho(54,44)) \wedge \\ & (\neg at(10) \wedge \neg \text{at}(11) \wedge \cdots \wedge \neg at(54) \wedge \neg \text{livre}(01) \wedge \cdots \wedge \neg \text{livre}(41))] \\ & \vee [\text{at}(00) \ \wedge \ \text{vizinho}(00,10) \ \wedge \ \text{livre}(10)] \\ = {} & \xi(X)\end{aligned}$$

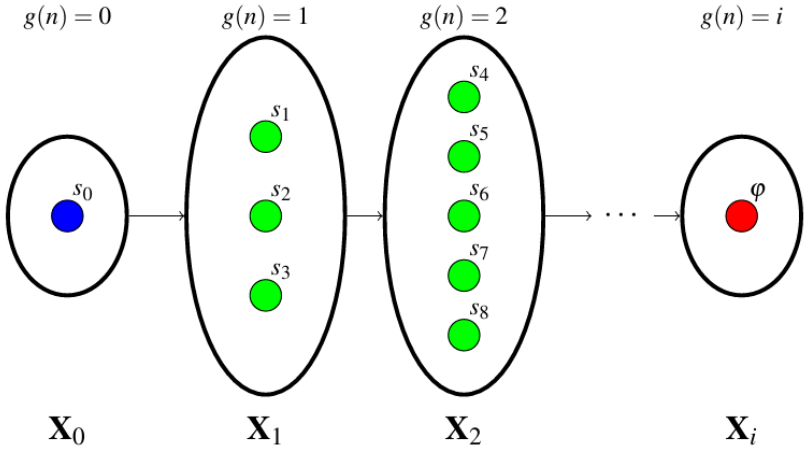
Percebe-se que para uma ação a válida a fórmula se mantem, porém para uma ação inválida teremos uma contradição na fórmula[5]. Assim, temos,

$$\begin{aligned}\xi(\text{prog}(X_0, move(00,10))) &= \exists\, modifica(a) \cdot (\xi(X_0) \wedge \xi(\text{precond}(a))) \wedge \xi(efeitos(a)) \\ &= \exists\{\text{at}(10), \text{at}(00)\} \cdot [\xi(X_0) \wedge (\text{at}(00) \ \wedge \ \text{vizinho}(00,10) \wedge \\ &\quad \text{livre}(10))] \wedge (\text{at}(10) \ \wedge \ \neg\text{at}(00)) \\ &= \exists \text{at}(10) \cdot \exists \text{at}(00) \cdot [\xi(X_0)] \wedge (\text{at}(10) \ \wedge \ \neg\text{at}(00)) \\ &= [livre(00) \wedge \cdots \wedge livre(54) \wedge vizinho(00,10) \wedge \cdots \wedge \\ &\quad vizinho(54,44)) \ \wedge (\neg\text{at}(11) \wedge \cdots \wedge \neg at(54) \wedge \neg\text{livre}(01) \wedge \\ &\quad \cdots \wedge \neg\text{livre}(41))] \wedge (\text{at}(10) \ \wedge \ \neg\text{at}(00)) \\ &= [\text{at}(10) \wedge \text{livre}(10) \wedge \cdots \wedge \text{livre}(54) \wedge \text{vizinho}(00,10) \wedge \cdots \wedge \\ &\quad \text{vizinho}(54,44)) \ \wedge (\neg\text{at}(00) \wedge \neg\text{at}(11) \wedge \cdots \wedge \neg at(54) \wedge \\ &\quad \neg\text{livre}(01) \wedge \cdots \wedge \neg\text{livre}(41))] \\ &= \xi(s_{10})\end{aligned}$$

[5] Como $\text{at}_{00} \wedge \cdots \wedge \neg\text{at}_{00}$.

Logo, a progressão do conjunto de estados $X_0 = \{s_{00}\}$ sobre a ação $a = \text{move}(00,10)$ resulta $X' = \{s_{10}\}$. Assim, para progredir de uma camada X_k para a camada X_{k+1} temos que agrupar, pela fórmula 2.4, todos os estados alcançáveis a partir de X_i por todos as ações válidas (Figura 12).

Figura 12 – Busca Progressiva por camadas.



Fonte: Elaborado pelo autor baseado no trabalho de Silva (2019).

2.2.4.2 Busca simbólica para trás no Espaço de Estados

Dado um conjunto de estados $X \subseteq \mathbb{S}$ e um conjunto de ações $\mathbb{A}$, representadas por fórmulas proposicionais, Fourman (2000) propôs a computação de estados predecessores $Y \subseteq \mathbb{S}$ a partir da especificação dessa reepresentação, uma operação denominada regressão simbólica.

A regressão simbólica computa o conjunto de estados predecessores Y que leva a estados pertencentes ao conjunto X. A regressão de X pela ação $a \in \mathbb{A}$ pode ser expressa por (Fourman, 2000):

$$\xi(\text{regr}(X,a)) = \xi(\text{precond}(a)) \wedge \exists \text{modifica}(a) \cdot (\xi(\text{efeitos}(a)) \wedge \xi(X)) \tag{2.10}$$

A Equação 2.10 é a versão simbólica da *computação regressiva de um estado s por uma ação a* (Equação 2.6) para cálculo de estados predecessores. A fórmula $\xi(\text{regr}(X,a))$ representa o conjunto Y de estados predecessores a partir da aplicação da ação a em estados do conjunto X. Para isso é necessário: (i) eleger o subconjunto X de estados alcançados pelos efeitos de a por meio da conjunção dos efeitos da ação com $\xi(X)$. Se algum efeito de a é relevante para algum estado X, então $\xi(\text{efeitos}(a)) \wedge \xi(X) \neq 0(\perp)$; (ii) eliminar as variáveis presentes nos efeitos da ação a por meio da aplicação da quantificação existencial das variáveis em modifica(a); (iii) realizar a conjunção desta fórmula com $\xi(\text{precond}(a))$.

Ao contrário da progressão, que trabalha com estados totalmente definidos (assumindo a hipótese do mundo fechado), a regressão considera que um estado é parcialmente definido, representando um conjunto de estados. Em outras palavras, *s* é um estado abstrato.

2.3 Planejamento como Busca Heurística Simbólica

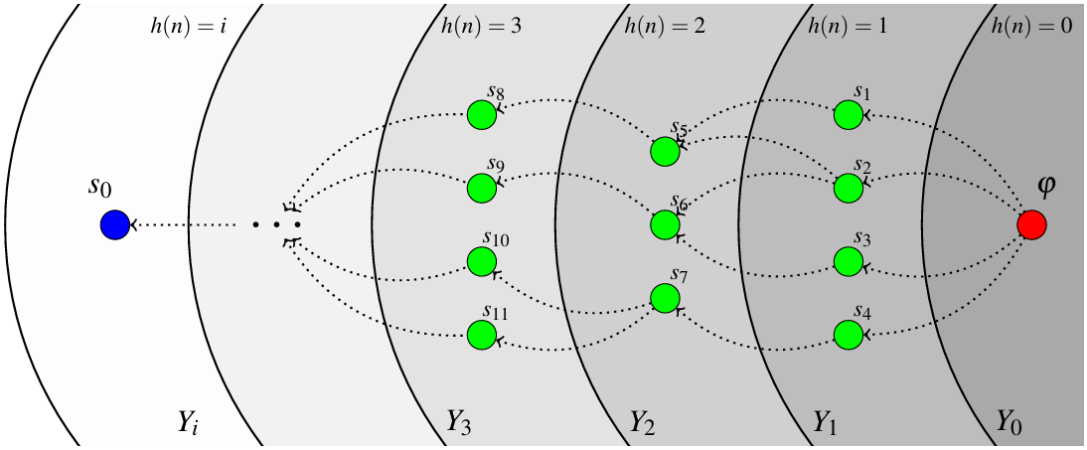
O planejamento como busca heurística simbólica combina as duas principais estratégias para contornar o problema da explosão do espaço de estados: (i) a representação compacta do espaço de estados da busca simbólica e; (ii) o guia (heurística) para estimar os nós mais promissores a serem expandidos. Proposta inicialmente por Edelkamp e Reffel (1998) e utilizada nos planejadores *MIPS* de Edelkamp e Helmert (2001) e *GAMER* de Edelkamp e Kissmann (2009), essa abordagem só recentemente mostrou resultados promissores em problemas de planejamento com grandes espaços de estados. Isso se deve ao uso de operações simbólicas entre conjuntos de *estados e ações*, em vez de estados e transições (Menezes, 2014; Santos *et al.*, 2019; Santos *et al.*, 2022), como discutido anteriormente.

A busca heurística simbólica tem atraído a atenção dos pesquisadores porque os planejadores que a utilizam começaram a superar os planejadores baseados em busca heurística tradicionais. Torralba *et al.* (2017) apresentaram a busca simbólica *A** (*BDDA**) para problemas de planejamento determinístico no planejador *cGamer*.

Assim como na busca *A**, a *BDDA** expande primeiro os estados com menor função de avaliação $f(n) = g(n) + h(n)$. No entanto, a *BDDA** agrupa estados com os mesmos valores de $g(n)$ e $h(n)$ e os expande simultaneamente. A função heurística é pré-computada e representada como uma lista de *BDD*s.

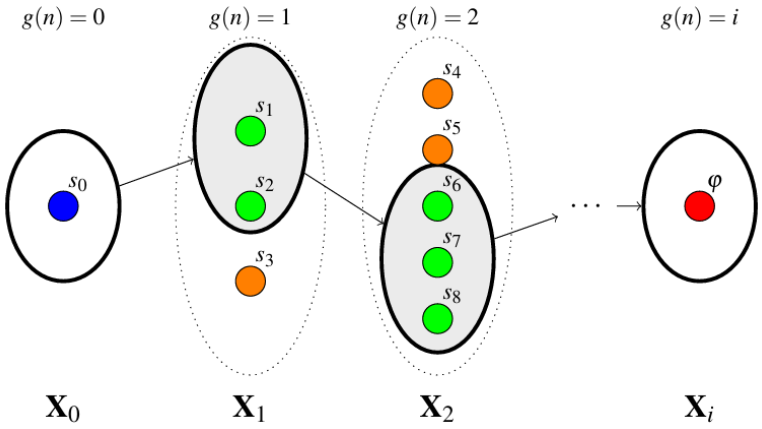
O trabalho de Silva (2019) propõe uma abordagem, baseado em Fourman (2000), para computar os valores heurísticos em planejamento simbólico usando busca regressiva seguida de uma busca progressiva utilizando a heurística computada. A busca regressiva (*backward*) sobre um problema relaxado constrói camadas Y de *BDD*s, representando os estados $s \in \mathbb{S}$ alcançáveis a partir da meta $\varphi \in \mathbb{S}$ em um determinado número de passos (Figura 13). Essas camadas permitem estimar o valor heurístico $h(n)$, o quão longe qualquer estado *s* está da meta φ. A busca progressiva (*forward*), por sua vez, utiliza essa heurística para guiar a expansão dos estados mais promissores (Figura 14). No entanto, esse trabalho apresentou dificuldades em problemas maiores, onde a busca regressiva não conseguia encontrar um caminho no espaço de estados da meta ao estado inicial, impedindo o início da busca progressiva.

Figura 13 – Busca regressiva no problema relaxado para estimar o valor heurístico



Fonte: Elaborado pelo autor baseado no trabalho de Silva (2019).

Figura 14 – Busca progressiva no problema real filtrado pela heurística.



Fonte: Elaborado pelo autor baseado no trabalho de Silva (2019).

No entanto o trabalho de Kissmann (2012) mostrou como abordaram esse problema com o uso de limitantes de tempo durante a geração de heurísticas em bancos de dados de padrões (do inglês *Pattern Databases* (*PDB*)), uma técnica para armazenar heurísticas pré-computadas (Culberson; Schaeffer, 1998). Kissmann (2012) propôs que, ao limitar o tempo de cálculo, é possível gerar heurísticas parciais, mas úteis, que aceleram a busca progressiva ao auxiliar na exploração de estados baseados em camadas parcialmente processadas.

Com base nessas ideias, este trabalho propõe uma melhoria na abordagem de Silva (2019), incorporando o limitante de tempo sugerido por Kissmann (2012) na computação regressiva da heurística. Podemos, assim, assegurar que a busca regressiva sempre resulte em uma heurística viável, mesmo que incompleta, permitindo que a busca progressiva seja executada, equilibrando os processos de cálculo de heurística e exploração de estados.

3 TRABALHOS RELACIONADOS

Esta seção apresenta os principais trabalhos relacionados, com foco em buscas heurísticas simbólicas baseadas em *BDD*s. São destacados os métodos de regressão simbólica propostos por Silva (2019), o uso de limite de tempo por Kissmann (2012), e melhorias recentes como em Torralba *et al.* (2017). O objetivo é evidenciar avanços e limitações que motivam esta pesquisa.

3.1 Busca Heurística Simbólica para Planejamento Determinístico

Em Silva (2019), a autora desenvolveu um algoritmo para realizar busca heurística simbólica $A*$ na representação de estados e ações, proposta por Menezes *et al.* (2014), de um dompinio de planejamento. Sua principal contribuição foi quanto à proposta de realização do cálculo heurístico por meio de uma busca simbólica regressiva em um problema de planejamento relaxado. No entanto, em problemas com um espaço de estados grande (como, por exemplo, o problema `Logistics-8` da *IPC* com 99 proposições e 171 ações), já não foi possível realizar o cálculo heurístico proposto.

Este trabalho propõe incorporar um limitante de tempo para o cálculo heurístico, conforme proposto em Kissmann (2012), na busca regressiva no problema relaxado proposta por Silva (2019) e avaliar o impacto desse limitante na resolução de problemas com um grande espaço de estados.

3.2 *Symbolic Search in Planning and General Game Playing*

Em Kissmann (2012), o autor propôs um algoritmo que utiliza *BDD*s para representar e raciocinar sobre estados e *transições* do domínio de planejamento. Ele introduziu o uso de *Pattern Databases* (*PDB*s), uma técnica que armazena heurísticas pré-computadas, e um limitante de tempo para computação dos valores heurísticos.

A relevância do trabalho de Kissmann (2012) para esta pesquisa está na introdução do conceito de limitante de tempo na geração de heurísticas. A proposta é incorporar esse limitante ao modelo de Silva (2019). Vale ressaltar também que o algoritmo de Silva (2019) realiza *raciocínio simbólico sobre ações* enquanto que Kissmann (2012) realiza *raciocínio simbólico sobre transições*.

3.3 cGamer - Efficient Symbolic Search for Cost-Optimal Planning

Em Torralba *et al.* (2017), os autores desenvolveram um algoritmo que utiliza *BDD*s para representar e *raciocinar sobre estados e ações* do domínio de planejamento. O planejador implementa a busca bidirecional de custo uniforme e a busca $A*$ com cálculo heurístico *PDB*s (do inglês *Pattern Database* (Culberson; Schaeffer, 1998). Os autores concluem que a busca simbólica supera em muitos *benchmarks* a busca heurística explícita utilizando as melhores heurísticas da literatura tais como `LM-Cut` (Helmert; Domshlak, 2009).

Em relação a este trabalho, embora ambos utilizem *BDD*s para manipulação simbólica de estados e ações, a principal diferença reside na forma de cálculo heurístico. Enquanto os autores em (Torralba *et al.*, 2014) realizam o cálculo heurístico com *PDB*s, este trabalho realiza o cálculo heurístico por meio da regressão simbólica em um problema de planejamento relaxado (como em Silva (2019)), com a introdução de um limitante de tempo, inspirado na abordagem de Kissmann (2012).

3.4 Comparativo entre as Abordagens

O Quadro 1 apresenta uma comparação entre os trabalhos relacionados e a pesquisa desenvolvida neste trabalho.

Quadro 1 – Quadro Comparativo dos Trabalhos Relacionados

Trabalho	**Tipo de Busca**
(Kissmann, 2012)	Simbólica Bidirecional e A* com heurística PDB limitada por tempo, representação e raciocínio sobre estados e transições
(Torralba *et al.*, 2017)	Simbólica Bidirecional e A* com heurística PDB limitada por tempo, com representação e raciocínio sobre estados e ações
(Silva, 2019)	Simbólica A* e heurística obtida a partir da busca regressiva no problema relaxado com representação e raciocínio sobre estados e ações
Este trabalho	Simbólica A* e heurística limitada por tempo obtida a partir da busca regressiva no problema relaxado com representação e raciocínio sobre estados e ações

Fonte: Elaborado pelo autor.

4 BUSCA HEURÍSTICA SIMBÓLICA COM REGRESSÃO LIMITADA E PROGRESSÃO GUIADA

4.1 Visão Geral da Abordagem

A busca simbólica tem se consolidado como uma estratégia promissora para enfrentar o problema da explosão do espaço de estados em planejamento automatizado, especialmente por meio do uso de Diagramas de Decisão Binária (*BDD*) para representar conjuntos de estados e ações de forma compacta (Edelkamp; Helmert, 2001). Dentre as abordagens existentes, o trabalho de Silva (2019) propôs uma heurística simbólica obtida por regressão relaxada, capaz de guiar uma busca progressiva de maneira informada.

Nessa proposta, a heurística é construída a partir de uma busca regressiva no problema relaxado — uma versão do domínio em que os efeitos negativos das ações são ignorados. A cada iteração, são identificados conjuntos de estados que se encontram a determinadas distâncias da meta (Figura 13). Essas *camadas simbólicas* são utilizadas posteriormente para estimar a proximidade de qualquer estado à meta, por meio de operações lógicas sobre *BDD*s.

Durante a busca progressiva, a heurística serve como base para guiar a expansão dos estados. A estimativa de distância é calculada por meio da interseção entre o estado atual e as camadas regressivas. Essa abordagem permite priorizar estados potencialmente mais promissores, reduzindo significativamente o esforço de busca — especialmente em domínios com grandes espaços de estados (Figura 14).

Apesar de seus avanços, a abordagem de Silva (2019) apresenta duas limitações principais. Primeiro, observou-se que, em problemas mais complexos, a regressão pode se tornar um gargalo computacional, impedindo a conclusão do processo e, consequentemente, **inviabilizando a geração dos valores da função heurística**. Essa limitação compromete a aplicabilidade do método, especialmente em domínios com grandes espaços de estados. Segundo, a progressão desenvolvida não apresenta as propriedades típicas de algoritmos clássicos como o $A*$, conforme proposto por Russell e Norvig (2013).

Os algoritmos de busca informada $A*$ e *GBFS*, conforme descritos em Russell e Norvig (2013), utilizam uma fila de prioridades para gerenciar os estados a serem explorados, guiando a busca por meio de uma função heurística. No caso do $A*$, a função de avaliação é dada por $f(n) = g(n) + h(n)$, onde g representa o custo acumulado até o estado atual e h é a estimativa heurística do custo até o objetivo. Já o *GBFS* utiliza apenas o valor heurístico, isto é, $f(n) = h(n)$,

para priorizar a exploração dos estados. Esse comportamento difere da implementação proposta por Silva (2019), onde a progressão não adota esse modelo clássico de avaliação.

Diante dessas limitações, este trabalho propõe uma metodologia alternativa que visa contornar esses problemas sem abrir mão da estrutura heurística simbólica. Inspirado por Kissmann (2012), introduzimos um limitante de tempo na fase de regressão, de modo que a construção da heurística seja interrompida após um tempo máximo configurável, produzindo assim uma heurística parcial. Para mitigar possíveis perdas de informação, a última camada gerada é complementada com os estados ainda não alcançados, criando uma complemento($\neg$*reached*) que reforça o direcionamento da busca progressiva mesmo diante de informações incompletas (Figura 15). A ideia é permitir que durante a progressão todo estado, mesmo os não alcançados na regressão, possam ter um valor heurístico.

Além disso, propomos o uso dessa heurística em uma busca progressiva baseada nos algoritmos *A** e *GBFS*, onde a expansão dos estados é guiada por funções de avaliação heurística. A Figura 18 ilustra esse processo de busca simbólica progressiva orientada por heurísticas, conforme implementado nesta metodologia.

Dessa forma, a metodologia adotada neste trabalho articula dois objetivos centrais:

- Tornar a construção da heurística simbólica viável, mesmo em instâncias complexas, por meio da limitação de tempo na regressão;
- Aplicar essa heurística como guia para uma busca progressiva simbólica, adotando uma estratégia de expansão informada.

A seguir, detalhamos cada etapa desse processo, desde a regressão com tempo limitado até a aplicação da heurística na progressão, passando pela estrutura lógica do sistema e pelas decisões metodológicas que orientaram sua implementação.

4.2 Inclusão de um Limitante de Tempo na Regressão

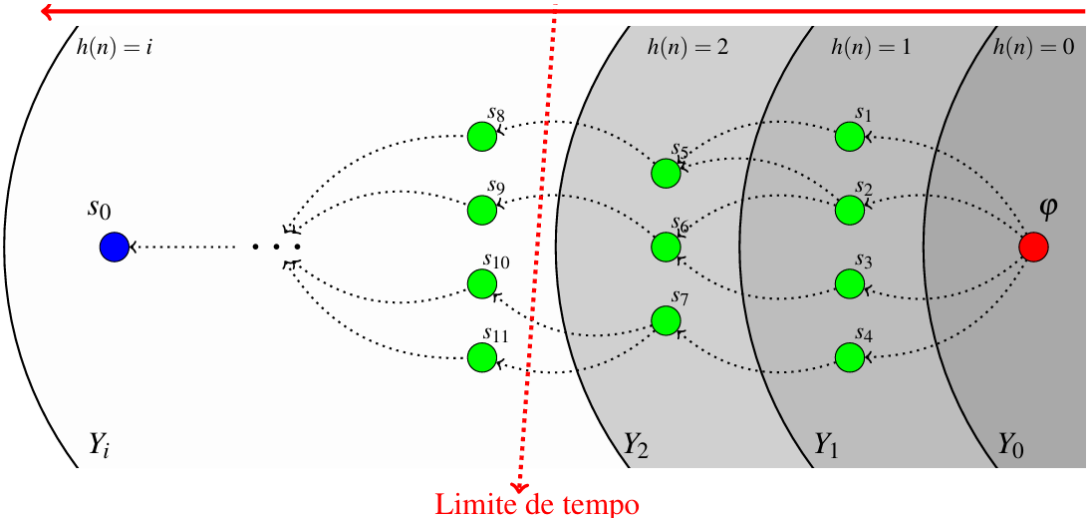
Como discutido anteriormente, a construção da heurística simbólica proposta por Silva (2019) depende de uma regressão relaxada a partir da meta, aplicada simbolicamente sobre o domínio. A cada iteração, são obtidos novos conjuntos de estados que podem atingir a meta após uma certa sequência de ações, formando uma estimativa da distância de qualquer estado até o objetivo. No entanto, em domínios mais complexos essa regressão pode consumir tempo e memória de forma excessiva, comprometendo sua conclusão e inviabilizando o uso da heurística.

Para contornar esse gargalo, este trabalho propõe a inclusão de um limitante de

tempo durante a construção da heurística regressiva. Essa estratégia foi inspirada na abordagem de Kissmann (2012), que introduziu mecanismos de interrupção controlada em buscas simbólicas para garantir que a execução se mantenha dentro de limites computacionais aceitáveis.

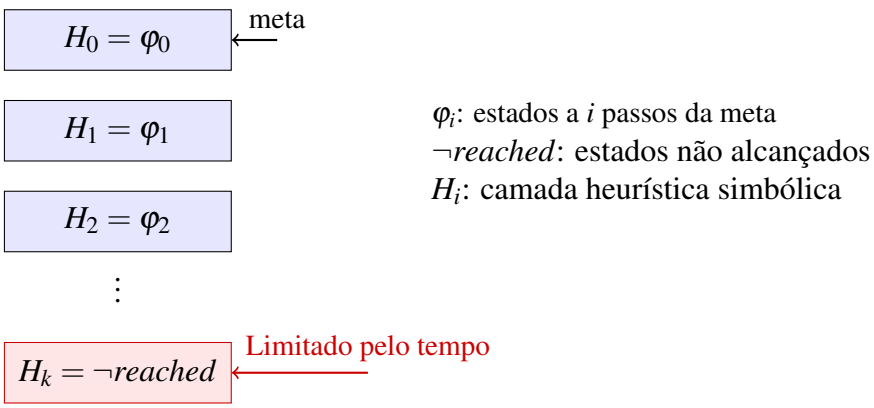
A ideia central consiste em permitir que a regressão relaxada seja executada apenas até um tempo máximo pré-configurado (Figura 15). Caso o tempo seja excedido antes da heurística estar completamente construída, o processo é interrompido e a informação disponível até o momento é utilizada de forma parcial. Para isso, a última camada gerada pela regressão é analisada, e os estados que não foram alcançados até aquele ponto são explicitamente marcados. Esses estados "não vistos"passam a compor uma camada negativa da heurística, assumindo um papel complementar ao conjunto de estados já explorados (Figura 16).

Figura 15 – Busca regressiva com interrupção por tempo.



Fonte: Elaborado pelo autor baseado no trabalho de Silva (2019).

Figura 16 – Construção da heurística com interrupção por tempo.



Fonte: Elaborado pelo autor.

Essa camada negativa permite preservar a propriedade informativa da heurística mesmo com a regressão incompleta. Durante a busca progressiva, quando um estado coincide com essa camada, infere-se que ele pertence a uma região do espaço que a regressão não conseguiu explorar no tempo disponível - indicando, portanto, uma distância estimada maior em relação à meta. Por outro lado, estados que coincidem com as camadas construídas previamente podem ser considerados mais promissores, pois foram comprovadamente encontrados na regressão, ainda que parcial. O algoritmo, Figura 17, apresenta o pseudocódigo da criação desta heurística.

Figura 17 – Peseudocódigo da regressão com limitante de tempo.

Algorithm 2 Construção da heurística simbólica com limite de tempo

Require: *goal*: conjunto de estados que satisfazem a meta
Ensure: Vetor heurístico $H = \{H_0, H_1, \ldots, H_k\}$, onde H_i é a camada i da regressão

seja *reached* $\leftarrow$ goal
seja $Z \leftarrow$ goal
seja $H \leftarrow$ vetor vazio
inicie o cronômetro
adicione $H_0 \leftarrow Z$
while Z **não é vazio do**
    if tempo limite excedido **then**
        adicione $H_{k+1} \leftarrow \neg reached$
        encerra
    end if
    Z $\leftarrow$ **regredir**(Z)
    Z $\leftarrow$ Z **menos** reached
    reached $\leftarrow$ reached $\cup$ Z
    adicione próxima camada $H_i \leftarrow Z$
end while
retorna H

Fonte: Elaborado pelo autor inspirado em Silva (2019).

A introdução desse limitante de tempo torna a construção da heurística mais robusta e adaptável, garantindo que a busca progressiva não seja comprometida, ou não executada, pela regressão cega não conseguir encontrar o estado inicial em tempo hábil. Com isso, amplia-se o conjunto de domínios e instâncias nos quais a abordagem simbólica pode ser aplicada de maneira prática.

Na próxima seção, detalharemos como essa heurística, mesmo quando incompleta, é utilizada para guiar uma busca progressiva informada.

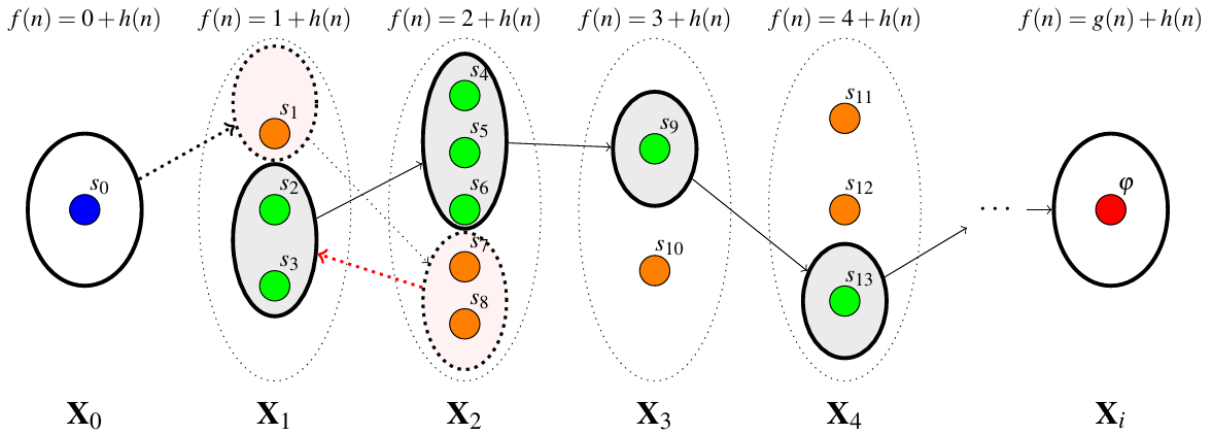
4.3 Busca Progressiva Guiada por Heurística Simbólica

Após a construção da heurística simbólica via regressão relaxada, conforme descrito no algoritmo da Figura 17, aplica-se essa estrutura para guiar a busca progressiva. A ideia central é empregar uma busca informada, utilizando os algoritmos $A*$ e *GBFS*, algoritmo da Figura 7, priorizando a expansão de estados com menor estimativa de distância até a meta, conforme determinado pela heurística.

A estimativa heurística $f(n)$ para um estado n é calculada com base em sua sobreposição com as camadas da heurística. Para isso, cada estado simbólico $s \in \mathbb{S}$ é decomposto em subconjuntos, utilizando uma função auxiliar que verifica, camada a camada, quais partes de s pertencem às fórmulas armazenadas na heurística. Cada fragmento de s que possui interseção não nula com uma camada H_i é considerado um candidato com estimativa $h(s) = i$.

Dentre os fragmentos identificados, seleciona-se o que possui menor valor de h, o qual é então progredido simbolicamente para gerar novos estados sucessores. Os demais fragmentos do estado atual também são considerados, podendo ser adicionados à fronteira conforme a função de avaliação os remove da fila de prioridades. A Figura 18 apresenta que a camada inicial X_0 gerou dois subconjuntos na camada X_1. Selecionou o subconjunto que contem o estado s_1, progrediu para a próxima camada, porém a fila de prioridades selecionou para remover o nó que tem o subconjunto da camada X_1 que contem os estados s_2 e s_3, o que levou para a solução.

Figura 18 – Exemplo de busca simbólica progressiva com fragmentação heurística, onde a fila prioriza o subconjunto mais promissor em X_1 após um caminho inicial pouco efetivo.



Fonte: Elaborado pelo autor baseado no trabalho de Silva (2019).

Nesta implementação, a busca progressiva simbólica foi desenvolvida de forma genérica, suportando tanto o algoritmo $A*$ quanto o *GBFS*, por meio da parametrização da função de avaliação $f(n)$. A estrutura nó define como calcular $f(n)$ com base nos valores de custo acumulado $g(n)$ e heurístico $h(n)$, herdados do estado pai. Assim, é possível configurar o comportamento da busca em tempo de execução, utilizando $f(n) = g(n) + h(n)$ para $A*$ ou $f(n) = h(n)$ para *GBFS*, sem alterar a lógica principal do algoritmo.

O algoritmo da Figura 19 apresenta um pseudocódigo resumido que ilustra o funcionamento da busca progressiva guiada pela heurística simbólica. A fronteira é inicializada com o estado inicial e sua heurística estimada. Iterativamente, remove-se da fila o estado com menor valor de f e verifica-se a condição de meta. Em caso negativo, aplica-se a progressão simbólica sobre os fragmentos com heurística mínima, e os sucessores são adicionados à fronteira, desde que ainda não tenham sido explorados.

Figura 19 – Pseudocódigo da Busca progressiva simbólica guiada.

Algorithm 3 Busca progressiva simbólica genérica pela heurística

Require: Estado inicial s_0, heurística simbólica H, função de custo $f(g,h)$
Ensure: Solução ou falha

inicialize a fronteira como fila de prioridade
inicialize o conjunto de estados explorados
calcule $h(s_0)$ a partir de H
enfileire s_0 com custo $g = 0$, prioridade $f(0, h(s_0))$
while fronteira não está vazia **do**
    $n \leftarrow$ remover da fronteira o nó com menor f
    if n satisfaz a condição de meta **then**
        retorne solução reconstruída a partir de n
    end if
    adicione n ao conjunto de explorados
    divida o estado de n em fragmentos s_i com heurísticas h_i
    selecione o fragmento s_b com menor h_b
    progrida simbolicamente $s_b \rightarrow s'_b$
    crie nó filho n' com $g = n.g + 1$, $h = h_b$
    enfileire n' com prioridade $f(n'.g, n'.h)$
    for cada fragmento restante s_i **do**
        crie nó com $g = n.g + 1$, $h = h_i$
        enfileire com prioridade $f(g,h)$
    end for
end while
retorne falha

Fonte: Elaborado pelo autor, com base na implementação em Java.

A partir da definição da heurística simbólica parcial e sua aplicação em uma busca progressiva informada, foram implementadas diferentes estratégias para avaliar o impacto dessa abordagem na resolução de problemas de planejamento. As variantes desenvolvidas incluem tanto métodos baseados em abordagens anteriores quanto a proposta deste trabalho, permitindo uma comparação sistemática entre o uso de regressão completa, regressão com tempo limitado e progressão guiada. A seguir, detalhamos os métodos de busca avaliados.

4.4 Métodos de Busca Avaliados

Para avaliar o impacto do uso de heurísticas simbólicas com regressão limitada, foram implementadas quatro variantes de busca simbólica:

- **OLD**: abordagem baseada na busca heurística simbólica proposta por Silva (2019), sem alterações;
- **OLD_TIME**: variação da abordagem anterior com limite de tempo na fase de regressão, como descrito na Seção 4.2;
- **ASTAR**: aplicação da busca regressiva limitada por tempo, seguida da busca progressiva simbólica, conforme descrito na Seção 4.3, com função de avaliação definida por $f = g + h$.
- **GBFS**: aplicação da busca regressiva limitada por tempo, seguida da busca progressiva simbólica, conforme descrito na Seção 4.3, com função de avaliação definida por $f = h$.

Cada um desses métodos foi implementado como uma classe separada (`SearchOldMethod`, `SearchOldWithTimerMethod` e `SearchNewMethod` recebendo a função avaliadora) derivada de uma estrutura comum (`BaseSearch`).

4.5 Metodologia de Testes

Para avaliar o desempenho das diferentes abordagens, cada domínio e instância foi testado com os três métodos descritos na Seção 4.4, utilizando tempos máximos distintos para a fase de regressão e tempos fixos para a progressão:

- **Tempo máximo de regressão:** 5, 10 e 15 minutos (300 000 ms, 600 000 ms, 900 000 ms)
- **Tempo máximo de progressão:** 30 minutos (1 800 000 ms)

Os testes foram organizados por partições, com execução automatizada em blocos de domínio e instância, armazenando os *logs* de saída e métricas em arquivos distintos.A seleção da variante a ser executada é feita via parâmetro em linha de comando, conforme descrito a seguir.

5 ANÁLISE EXPERIMENTAL

5.1 Ambiente de Desenvolvimento

Os algoritmos propostos neste trabalho foram implementados em Java, versão 21, estruturados em um projeto modular utilizando o sistema de *build* `Maven`. Para manipulação de diagramas de decisão binária (*BDD*s), foi utilizada a biblioteca `JavaBDD` (Whaley, 2007), amplamente empregada em aplicações simbólicas devido à sua interface simples e compatibilidade. Neste trabalho, utilizou-se a versão `9.0` da biblioteca, disponível no repositório oficial do Maven (GitHub Community, 2023).

Os testes foram conduzidos em um ambiente virtual de alto desempenho, com o uso do *Google Colab Pro* com as seguintes especificações:

- **Sistema Operacional:** Ubuntu (Colab Pro)
- **Processador (CPU):** Intel Xeon, 2 vCPUs, entre 2.2 e 2.3 GHz
- **Memória RAM:** aproximadamente 51 GB
- **Ambiente Python 3**: utilizado apenas como interface de controle via notebooks
- **Java:** OpenJDK 21

5.2 Configuração de Execução

Para cada teste, o projeto foi empacotado como um arquivo `.jar` gerado via Maven, com execução automatizada por meio de *scripts* `.sh` configuráveis. A chamada padrão utilizada nos testes foi:

```
java -Xmx48g -Xms20g -jar GUI.jar <domínio> heuristic <instância> <tempo_back> <tempo_forward> <método>
```

Os parâmetros `-Xmx` e `-Xms` foram ajustados explicitamente para definir, respectivamente, a memória máxima e mínima disponível para a máquina virtual Java (*Java Virtual Machine* (*JVM*)). Essa configuração foi necessária para garantir que o *garbage collector* não limitasse o crescimento do *heap* de forma prematura durante buscas progressivas de longa duração, especialmente em domínios com alta expansão de estados. O valor de `-Xmx48g` permitiu que a *JVM* aproveitasse a maior parte da RAM disponível no ambiente, enquanto `-Xms20g` garantiu uma alocação inicial robusta para estruturas como BDDs e filas de prioridade.

5.3 Métricas Coletadas

As métricas coletadas variaram conforme o método de busca empregado, refletindo os diferentes aspectos computacionais avaliados durante os experimentos.

No método `OLD`, foram mensurados, para a busca regressiva, o número de camadas geradas e o tempo total de execução. Para a busca progressiva, consideraram-se o número de camadas alcançadas, o tempo total e o uso de memória.

O método `OLD_TIME`, que se diferencia por impor um tempo limite na regressão, registrou o número de camadas geradas até a interrupção, o tempo de execução até esse ponto e o uso de memória. A progressão, nesse método, manteve as mesmas métricas do caso anterior.

Já para os métodos baseados em busca heurística simbólica (`ASTAR` e `GBFS`), as métricas analisadas incluíram:

- Número de camadas alcançadas
- Tempo total de execução
- Número total de nós gerados
- Valor da heurística inicial ($h(s_0)$)
- Uso de memória

A métrica $h(s_0)$, conhecida como *heurística de abertura*, indica a estimativa de distância do estado inicial até o objetivo, conforme a heurística simbólica construída com base em Silva (2019). Em execuções interrompidas por tempo, observou-se que $h(s_0)$ tende a assumir um valor uma unidade acima da última camada computada pela busca regressiva, refletindo sua natureza parcial. Por outro lado, nos casos em que a busca regressiva completava sua execução, o valor de $h(s_0)$ geralmente era inferior à profundidade máxima alcançada pela regressão.

As seções seguintes apresentam os resultados quantitativos extraídos com base nessas métricas, seguidos por uma análise comparativa entre os métodos avaliados.

5.4 Descrição dos Benchmarks

As instâncias utilizadas nos testes deste trabalho foram retiradas da coleção *All-IPC (STRIPS)*, disponível no repositório *Planning.Domains* (Muise, 2019). Essa coleção reúne problemas PDDL extraídos de diversas edições da Competição Internacional de Planejamento (*IPC*).

Os experimentos foram realizados utilizando dois domínios clássicos da *IPC*: `Logistics`

e `Rovers`. A escolha desses domínios permite comparação direta com trabalhos anteriores (Silva, 2019) e avaliação em diferentes cenários.

5.4.1 Domínio `Logistics`

No domínio `Logistics`, o objetivo é transportar pacotes de seus pontos de origem até os destinos definidos, dentro de um ambiente composto por várias cidades. As cidades são conectadas entre si por transporte aéreo, enquanto o transporte dentro de cada cidade ocorre por rodovias, utilizando caminhões (Muise, 2019).

Se o envio for dentro da mesma cidade, basta carregar o pacote em um caminhão no local de origem, levá-lo até o destino e descarregá-lo. Para entregas entre cidades, o pacote deve ser levado até o aeroporto da cidade de origem (possivelmente com o uso de um caminhão), embarcado em um avião e transportado até o aeroporto da cidade de destino. Caso o destino final seja o próprio aeroporto, a entrega é concluída. Caso contrário, o pacote ainda precisa ser transportado localmente por caminhão até o destino final. Não há restrições de capacidade para caminhões e aviões.

A Tabela 1 mostra as características das instâncias testadas, que variam em complexidade conforme o número de objetos e ações. Para cada instância testada, utilizamos como referência o valor de *Upper Bound*, geralmente correspondente ao custo da solução encontrada por um dos planejadores vencedores da *IPC*, como o *LAMA* (Richter; Westphal, 2008). Esse valor serve como comparação para avaliar a qualidade das soluções geradas pelos métodos propostos.

Tabela 1 – Características das instâncias do domínio `Logistics`

Problema	**Proposições**	**Ações**	***Upper Bound***
Logistics-4	48	78	19
Logistics-6	48	78	25
Logistics-8	99	171	31
Logistics-10	168	308	45
Logistics-12	168	308	68
Logistics-14	275	648	58

Fonte: Adaptado de (Silva, 2019).

5.4.2 Domínio `Rovers`

O domínio `Rovers` é inspirado nas missões do *Mars Science Laboratory* e representa tarefas de exploração planetária realizadas por robôs autônomos. Os robôs se deslocam entre

pontos estratégicos no planeta para coletar amostras de solo e rocha, além de registrar imagens de objetos com diferentes tipos de câmeras — que podem capturar imagens coloridas ou em diferentes resoluções(Muise, 2019).

As amostras analisadas e as imagens obtidas devem ser transmitidas a uma estação espacial de suporte. Cada robô possui capacidades específicas de locomoção, coleta e análise, o que exige coordenação para cumprir os objetivos da missão.

A Tabela 2 mostra as características das instâncias testadas, que variam em complexidade conforme o número de objetos, ações e o valor de *Upper Bound*. Na competição, o domínio conta com 40 instâncias, mas apenas 8 foram selecionadas para os experimentos no trabalho de Silva (2019).

Tabela 2 – Características das instâncias do domínio Rovers

Problema	**Proposições**	**Ações**	***Upper Bound***
Rovers-01	35	63	10
Rovers-02	35	63	8
Rovers-03	44	76	11
Rovers-04	50	86	8
Rovers-05	59	144	22
Rovers-06	63	178	36
Rovers-07	68	151	18
Rovers-08	110	328	26

Fonte: Adaptado de Silva (2019).

5.5 Resultados por Domínio

Os resultados completos, abrangendo todas as instâncias avaliadas, estão organizados em tabelas no Apêndice B. Para manter a clareza da análise, esta seção apresenta apenas um subconjunto representativo dos experimentos realizados.

5.5.1 Desempenho no Domínio `Logistics`

Os experimentos no domínio `Logistics` foram conduzidos com diferentes tempos-limite (5, 10 e 15 minutos) para a regressão, permitindo uma análise abrangente dos métodos `OLD`, `OLD_TIME`, `GBFS` e `ASTAR`. As Tabelas 3 a 4 apresentam os resultados mais relevantes.

Observa-se que o método `OLD` tradicional enfrenta dificuldades crescentes à medida que a complexidade das instâncias aumenta. Para `logistics-8`, por exemplo, ele alcança apenas 16 camadas regressivas em 300 segundos, sem conseguir completar a progressão (Tabela 11). Já

o método OLD_TIME, ao adotar limitação temporal, produz heurística informadas até a camada regressiva 17, permitindo a progresão encontrar uma solução, na camada 34 (Tabela 12) em menos de 163 segundos, com desempenho próximo ao ótimo (*Upper Bound* na camada 31).

A comparação entre GBFS e ASTAR mostra comportamentos bastante similares em termos de profundidade alcançada, tempo de execução e consumo de memória. Ambos métodos atingem 34 camadas progressivas, com desempenho quase idêntico tanto no limite de 5 quanto de 15 minutos. No teste com 15 minutos (logistics-8), o GBFS apresenta uma leve vantagem em tempo (80s contra 77s), enquanto o número de nós expandidos e o uso de memória permanecem equivalentes (72 nós e 2128 MB em ambos). Isso sugere que, nesse domínio, a escolha entre GBFS e ASTAR pode ser feita com base em critérios secundários, já que o desempenho geral se mantém equilibrado, especialmente quando uma boa heurística é disponibilizada.

Tabela 3 – Resultados para Logistics 8 com tempo limite de 5 minutos

Método	**Camadas (R)**	**Tempo (R)**	**Camadas (P)**	**Tempo (P)**	**Nós(P)**	**Memória (MB)**
OLD	16	300s	–	–	–	–
OLD_TIME	17	313s	34	163s	–	1376
ASTAR	17	309s	34	151s	65	1360
GBFS	17	309s	34	145s	65	1360

Fonte: Elaborado pelo autor.

Tabela 4 – Resultados para Logistics 8 com tempo limite de 15 minutos

Método	**Camadas (R)**	**Tempo (R)**	**Camadas (P)**	**Tempo (P)**	**Nós(P)**	**Memória (MB)**
OLD	20	1196s	–	–	–	–
OLD_TIME	20	912s	34	57s	–	2128
ASTAR	19	913s	34	77s	72	2128
GBFS	19	915s	34	80s	72	2128

Fonte: Elaborado pelo autor.

Adicionalmente, as Tabelas 5 e 6 evidenciam os desafios de escalabilidade enfrentados pelos métodos ASTAR e GBFS no domínio Logistics, mesmo sob um tempo limite estendido de 15 minutos. Observa-se uma degradação acentuada e não-linear no desempenho à medida que o tamanho das instâncias aumenta.

A instância logistics-4 é resolvida com facilidade por ambos os métodos, consumindo apenas alguns segundos e pouca memória, com sucesso completo na resolução. No entanto, ao escalar para logistics-8, o número de camadas se reduz na regressão, e o tempo total se aproxima do limite, indicando uma complexidade estrutural crescente. Ambos os métodos conseguem avançar parcialmente, mas não completam a solução.

Na instância mais complexa, `logistics-14`, o desempenho se torna drasticamente limitado. O número de camadas geradas cai significativamente (apenas 6 em regressão para `ASTAR` e `GBFS`), o tempo de planejamento atinge o limite de 30 minutos, e o número de nós expandidos permanece irrisório. Além disso, os consumos de memória aumentam drasticamente — alcançando 28GB para `ASTAR` e 12.8GB para `GBFS` —, refletindo o custo da tentativa de expansão do espaço de estados sem sucesso.

Esses resultados demonstram que, para domínios com crescimento estrutural denso como `Logistics`, os métodos baseados em *BDD* sofrem limitações críticas de memória e profundidade, reforçando a necessidade de estratégias adicionais ou maior refino do código.

Tabela 5 – Escalabilidade no domínio `Logistics` para o método `ASTAR` (15 minutos)

Instância	Camadas(R)	Tempo(R)	Camadas(P)	Tempo(R)	Nós(P)	Memória	Sucesso
logistics 4	22	3s	22	0.852s	61	16MB	Completo
logistics 8	19	913s	34	77s	72	2128MB	Parcial
logistics 14	6	949s	10	30min	11	28GB	Insuficiente

Fonte: Elaborado pelo autor.

Tabela 6 – Escalabilidade no domínio `Logistics` para o método `GBFS` (15 minutos)

Instância	Camadas(R)	Tempo(R)	Camadas(P)	Tempo(R)	Nós(P)	Memória	Sucesso
logistics 4	22	3s	22	0.846s	61	16MB	Completo
logistics 8	19	915s	34	80s	72	2128MB	Parcial
logistics 14	6	945s	10	30min	11	12.8GB	Insuficiente

Fonte: Elaborado pelo autor.

5.5.2 Desempenho no Domínio `Rovers`

Os experimentos conduzidos no domínio `Rovers` revelam um comportamento mais estável e previsível do que em `Logistics`, com instâncias resolvidas com sucesso mesmo nos tempos-limite mais restritos. As Tabelas 7 e 8 destacam os resultados da instância `rovers-6`, representativa por sua complexidade intermediária.

No tempo-limite de 300 segundos, o método `OLD` tradicional não consegue produzir heurísticas suficientes para suportar a fase progressiva. A regressão atinge apenas 24 camadas antes do tempo expirar. Já o método `OLD_TIME`, que emprega controle de tempo na regressão, alcança 25 camadas e consegue resolver a progressiva em 159 segundos, utilizando 4064MB de memória.

Os métodos `ASTAR` e `GBFS` apresentam desempenho superior. Ambos alcançam 26 camadas na regressiva, o que se mostra suficiente para permitir a resolução da fase progressiva

com 43 camadas. O ASTAR consome 145 segundos e 2896MB de memória, enquanto o GBFS se destaca pela maior eficiência, com apenas 51 segundos de tempo na progressiva e consumo reduzido de memória (1360MB). Isso demonstra que, mesmo com uma heurística limitada (26 camadas), os métodos baseados em busca conseguem explorar bem o espaço de estados no domínio Rovers.

Tabela 7 – Resultados para Rovers 6 com tempo limite de 5 minutos

Método	Camadas (R)	Tempo (R)	Camadas (P)	Tempo (P)	Nós (P)	Memória (MB)
OLD	24	300s	–	–	–	–
OLD_TIME	25	300s	43	159s	–	4064
ASTAR	26	301s	43	145s	172	2896
GBFS	26	302s	43	51s	122	1360

Fonte: Elaborado pelo autor.

Com o tempo estendido para 900 segundos, todos os métodos com heurísticas mais profundas - incluindo o OLD - demonstram um bom desempenho. Os métodos OLD, OLD_TIME, ASTAR e GBFS atingiram 41 camadas na regressão e resolvem a progreção com sucesso, alcançando 43 camadas.

Apesar do tempo elevado da fase regressiva (acima de 12 minutos no OLD), a progressão se mantém eficiente, com destaque para o OLD, que resolve a fase progressiva em apenas $8,9$ segundos. Já o GBFS também mantém bom desempenho (85s), enquanto o ASTAR apresenta um tempo mais elevado (255s), refletindo a diferença de seletividade da heurística.

Curiosamente, a memória registrada para os métodos ASTAR e GBFS nessa configuração é de apenas 16MB, o que contrasta com os valores altos observados anteriormente. Esse valor provavelmente representa apenas a memória da fase progressiva, ou um erro na coleta, mas não invalida a conclusão geral: as buscas são eficientes mesmo em instâncias intermediárias. O tempo da fase progressiva no GBFS (85s) é significativamente menor que no ASTAR (255s), refletindo a seletividade da busca gulosa em casos de heurísticas parciais.

Tabela 8 – Resultados para Rovers 6 com tempo limite de 15 minutos

Método	Camadas (R)	Tempo (R)	Camadas (P)	Tempo (P)	Nós (P)	Memória (MB)
OLD	41	886s	43	8,9s	–	16
OLD_TIME	41	729s	43	22s	–	1744
ASTAR	41	715s	43	255s	209	16
GBFS	41	729s	43	85s	121	16

Fonte: Elaborado pelo autor.

As Tabelas 9 e 10 abordam a escalabilidade dos métodos ASTAR e GBFS. Ambos são

eficazes em instâncias simples como `rovers-1` e intermediárias como `rovers-6`. No entanto, na instância mais complexa `rovers-8`, a regressão atinge apenas 14 camadas após 15 minutos de execução, e a fase progressiva se torna impraticável, excedendo o tempo-limite com apenas 11 camadas alcançadas. Isso evidencia que o gargalo está na fase de geração da heurística, e não na capacidade da busca em si.

Tabela 9 – Escalabilidade no domínio `Rovers` para o método `ASTAR` (15 minutos)

Instância	Camadas(R)	Tempo(R)	Camadas(P)	Tempo(P)	Nós(P)	Memória	Sucesso
rovers 1	11	0.72s	11	0.74s	27	16MB	Completo
rovers 6	41	715s	43	255s	209	16MB	Completo
rovers 8	14	925s	11	30min	12	13.2GB	Insuficiente

Fonte: Elaborado pelo autor.

Tabela 10 – Escalabilidade no domínio `Rovers` para o método `GBFS` (15 minutos)

Instância	Camadas(R)	Tempo(R)	Camadas(P)	Tempo(P)	Nós(P)	Memória	Sucesso
rovers 1	11	0.72s	11	0.74s	27	16MB	Completo
rovers 6	41	729s	43	85s	121	16MB	Completo
rovers 8	14	908s	11	30min	12	16GB	Insuficiente

Fonte: Elaborado pelo autor.

De forma geral, o domínio `Rovers` apresenta bons resultados com os métodos heurísticos quando a regressão é bem-sucedida. A limitação principal encontra-se na geração da heurística para instâncias grandes, onde o tempo e a memória necessários crescem exponencialmente. Apesar disso, o uso de busca guiada — especialmente com `GBFS` — mantém o número de nós e o tempo de busca em níveis controlados, sugerindo boa aplicabilidade prática dos métodos, desde que acompanhados de estratégias eficazes de controle na fase de regressão.

5.5.3 Acesso ao Código Fonte

Todo o código-fonte desenvolvido para este trabalho está disponível em repositório público no GitHub, com versão final preservada por meio do sistema de releases e tags, acessível no seguinte link:

- **Repositório Principal**: https://github.com/HenrickyL/AI-Planner-BDD
- **Versão Congelada (v1)**: https://github.com/HenrickyL/AI-Planner-BDD/tree/v1

O repositório contém a implementação dos algoritmos de busca, estruturas auxiliares, scripts de execução e análise dos dados, bem como instruções para reprodução dos experimentos.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho investigou o desempenho de algoritmos de busca simbólica no contexto do planejamento automatizado, com foco na construção de heurísticas por regressão e posterior aplicação de busca progressiva. Através da utilização de diagramas de decisão binária (*BDD*s), foi possível representar e manipular conjuntos simbólicos de estados de forma eficiente, explorando a viabilidade prática de diferentes estratégias de busca, incluindo métodos como o proposto por Silva (2019) (`OLD`), com adição do limitante de tempo durante a regressão proposta por Kissmann (2012) (`OLD_TIME`) e busca heurística simbólica informada proposta nesse trabalho (`ASTAR` e `GBFS`).

Os experimentos conduzidos nos domínios `Logistics` e `Rovers`, extraídos da competição internacional de planejamento (*IPC*), demonstraram a importância de incluir heurísticas para guiar a busca simbólica por uma solução.

As análises também evidenciaram que o controle de tempo na fase de regressão pode viabilizar soluções para instâncias onde métodos puramente exaustivos falham (Silva, 2019). Além disso, a abordagem `GBFS` destacou-se pelo melhor custo-benefício entre tempo de execução e consumo de memória, mantendo a qualidade das soluções geradas.

Todos os resultados obtidos, incluindo tabelas detalhadas por instância e método, estão apresentados no Apêndice B. O código-fonte do projeto está disponível em repositório público (Sessão 5.5.3), garantindo a reprodutibilidade e a transparência da pesquisa.

Uma possível linha de melhoria futura está relacionada ao controle da geração de *BDD*s. Essa técnica pode beneficiar especialmente métodos que *divide um estado de n em fragmentos* pra serem avaliados na fila de prioridades pela função heurística, que dependem intensamente da criação de novos *BDD*s. Além disso, investigações mais profundas sobre otimizações de desempenho, visando reduzir sobrecargas internas e tornar a execução mais eficiente, também se configuram como direções relevantes a serem exploradas.

A utilização de *BDD*s no planejamento simbólico se mostrou promissora, especialmente quando aliada a estratégias de busca informadas e controle de tempo. Embora desafios de escalabilidade persistam, os resultados indicam que a combinação entre representação simbólica e heurísticas progressivas pode representar uma alternativa viável e extensível dentro da área de planejamento automático.

REFERÊNCIAS

AERONAUTIQUES, C.; HOWE, A.; KNOBLOCK, C.; MCDERMOTT, I. D.; RAM, A.; VELOSO, M.; WELD, D.; SRI, D. W.; BARRETT, A.; CHRISTIANSON, D. *et al.* Pddl| the planning domain definition language. **Technical Report, Tech. Rep.**, 1998.

BONGARTZ, I. N. **Explaining unsolvable planning tasks**. Dissertação (Master's Thesis) – University of Freiburg, Freiburg, Germany, 2018.

BUNING, H. K.; KARPINSKI, M.; FLOGEL, A. Resolution for quantified boolean formulas. **Information and computation**, Elsevier, v. 117, n. 1, p. 12–18, 1995.

CHEN, D. Z.; THIÉBAUX, S.; TREVIZAN, F. Learning domain-independent heuristics for grounded and lifted planning. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [*S. l.*: *s. n.*], 2024. v. 38, n. 18, p. 20078–20086.

CIMATTI, A.; PISTORE, M.; ROVERI, M.; TRAVERSO, P. Weak, strong, and strong cyclic planning via symbolic model checking. **Artificial Intelligence**, Elsevier, v. 147, n. 1-2, p. 35–84, 2003.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos**: teoria e prática. 2. ed. [*S. l.*]: Campus, 2002.

CULBERSON, J. C.; SCHAEFFER, J. Pattern databases. **Computational Intelligence**, Wiley Online Library, v. 14, n. 3, p. 318–334, 1998.

EDELKAMP, S.; HELMERT, M. Mips: The model-checking integrated planning system. **AI magazine**, v. 22, n. 3, p. 67–67, 2001.

EDELKAMP, S.; KISSMANN, P. Optimal symbolic planning with action costs and preferences. In: CITESEER. **Twenty-First International Joint Conference on Artificial Intelligence**. [*S. l.*], 2009.

EDELKAMP, S.; KISSMANN, P.; TORRALBA, A. Bdds strike back (in ai planning). In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [*S. l.*: *s. n.*], 2015. v. 29, n. 1.

EDELKAMP, S.; REFFEL, F. Obdds in heuristic search. In: SPRINGER. **KI-98: Advances in Artificial Intelligence: 22nd Annual German Conference on Artificial Intelligence Bremen, Germany, September 15–17, 1998 Proceedings 22**. [*S. l.*], 1998. p. 81–92.

FIKES, R. E.; NILSSON, N. J. Strips: A new approach to the application of theorem proving to problem solving. **Artificial intelligence**, Elsevier, v. 2, n. 3-4, p. 189–208, 1971.

FOURMAN, M. P. Propositional planning. In: **Proceedings of AIPS-00 Workshop on Model-Theoretic Approaches to Planning**. [*S. l.*: *s. n.*], 2000. p. 10–17.

GHALLAB, M.; NAU, D.; TRAVERSO, P. **Automated Planning**: theory and practice. [*S. l.*]: Elsevier, 2004.

GitHub Community. **JavaBDD**: Java binary decision diagram library. 2023. Disponível em: https://mvnrepository.com/artifact/com.github.com-github-javabdd/com.github.javabdd. Acesso em: 10 mar. 2025.

HASLUM, P.; LIPOVETZKY, N.; MAGAZZENI, D.; MUISE, C.; BRACHMAN, R.; ROSSI, F.; STONE, P. **An introduction to the planning domain definition language**. [*S. l.*]: Springer, 2019. v. 13.

HELMERT, M. The fast downward planning system. **Journal of Artificial Intelligence Research**, v. 26, p. 191–246, 2006.

HELMERT, M.; DOMSHLAK, C. Landmarks, critical paths and abstractions: what's the difference anyway? In: **Proceedings of the International Conference on Automated Planning and Scheduling**. [*S. l.*: *s. n.*], 2009. v. 19, p. 162–169.

HOFFMANN, J. Ff: The fast-forward planning system. **AI magazine**, v. 22, n. 3, p. 57–57, 2001.

HUTH, M.; RYAN, M. **Logic in Computer Science**: Modelling and reasoning about systems. [*S. l.*]: Cambridge university press, 2004.

ICAPS. **International Conference on Automated Planning and Scheduling (ICAPS) - Competitions**. 2024. Disponível em: https://www.icaps-conference.org/competitions/. Acesso em: 06 abr. 2024.

KISSMANN, P. **Symbolic search in planning and general game playing**. Tese (Doutorado) – Universität Bremen, 2012.

MCMILLAN, K. **Symbolic model checking—An approach to the state explosion problem (Thesis)**. [*S. l.*], 1992.

MENEZES, M. V. de. **Mudanças em problemas de planejamento sem solução**. Tese (Doutorado) – Universidade de São Paulo, São Paulo, 2014.

MENEZES, M. V. de; BARROS, L. N. de; PEREIRA, S. do L. Symbolic regression for non-deterministic actions. 2014.

MUISE, C. **Planning.Domains - Online PDDL Editor and Benchmark Repository**. 2019. Disponível em: https://editor.planning.domains/. Acesso em: 15 jul. 2025.

RICHTER, S.; WESTPHAL, M. The lama planner using landmark counting in heuristic search. In: **Proceedings of IPC**. [*S. l.*: *s. n.*], 2008. v. 14.

RUSSELL, S.; NORVIG, P. **Inteligência Artificial**: Uma abordagem moderna. 3rd. ed. [*S. l.*]: ELSEVIER EDITORA, 2013. ISBN 0136042597.

SANNER, S. *et al.* Relational dynamic influence diagram language (rddl): Language description. **Unpublished ms. Australian National University**, v. 32, p. 27, 2010.

SANTOS, V. B. dos; BARROS, L. N. de; PEREIRA, S. do L.; MENEZES, M. V. de. Symbolic fond planning for temporally extended goals. In: INTERNATIONAL CONFERENCE ON AUTOMATED PLANNING AND SCHEDULING (ICAPS), 2022, Auckland, New Zealand. **Proceedings of the KEPS 2022 – Workshop on Knowledge Engineering for Planning and Scheduling, ICAPS 2022**. Auckland, New Zealand: AAAI Press, 2022. p. 1–9.

SANTOS, V. M. B. dos; BARROS, L. N. de; MENEZES, M. V. de. Symbolic planning for strong-cyclic policies. In: IEEE. **2019 8th Brazilian Conference on Intelligent Systems (BRACIS)**. [*S. l.*], 2019. p. 168–173.

SILVA, M. d. C. **Busca heurística simbólica para planejamento determinístico**. Trabalho de Conclusão de Curso (Graduação) – Universidade Federal do Ceará, Quixadá, 2019.

SPECK, D. Symbolic search for optimal planning with expressive extensions. **arXiv preprint arXiv:2204.00288**, 2022.

SZWARCFITER, J. L.; MARKENZON, L. **Estruturas de Dados e seus Algoritmos (3a. edição)**. [*S. l.*]: Editora LTC, 2010.

TAITLER, A.; ALFORD, R.; ESPASA, J.; BEHNKE, G.; FIŠER, D.; GIMELFARB, M.; POMMERENING, F.; SANNER, S.; SCALA, E.; SCHREIBER, D. *et al.* **The 2023 International Planning Competition**. [*S. l.*]: Wiley Online Library, 2024.

TORRALBA, A.; ALCÁZAR, V.; BORRAJO, D.; KISSMANN, P.; EDELKAMP, S. Symba*: A symbolic bidirectional a* planner. In: **International Planning Competition**. [*S. l.*: *s. n.*], 2014. p. 105–108.

TORRALBA, Á.; ALCÁZAR, V.; KISSMANN, P.; EDELKAMP, S. Efficient symbolic search for cost-optimal planning. **Artificial Intelligence**, Elsevier, v. 242, p. 52–79, 2017.

TORRALBA, Á.; LÓPEZ, C. L.; BORRAJO, D. Symbolic perimeter abstraction heuristics for cost-optimal planning. **Artificial Intelligence**, Elsevier, v. 259, p. 1–31, 2018.

UNIVERSITY, H. **CS50's Introduction to Artificial Intelligence with Python**. Cambridge, MA, 2020. Online course. Disponível em: https://cs50.harvard.edu/ai/2020/.

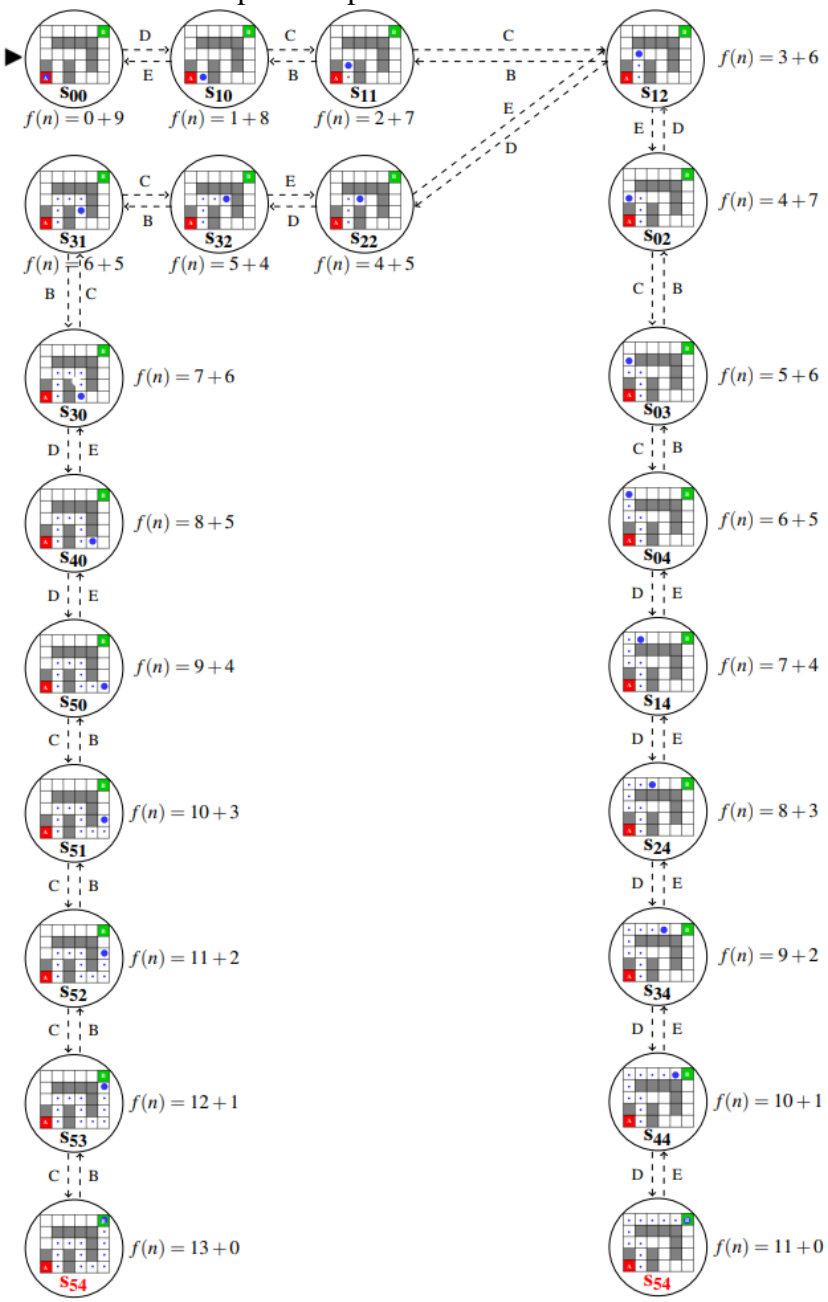
VALLATI, M.; CHRPA, L.; GRZEś, M.; MCCLUSKEY, T. L.; ROBERTS, M.; SANNER, S. The 2014 international planning competition: Progress and trends. **AI Magazine**, v. 36, n. 3, p. 90–98, 2015. Disponível em: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2571. Acesso em: 12 mai. 2024.

WHALEY, J. **JavaBDD**: Java binary decision diagram library. 2007. Disponível em: http://javabdd.sourceforge.net/. Acesso em: 10 mar. 2025.

APÊNDICE A – ÁRVORE DE BUSCA PARA O PROBLEMA DO LABIRINTO

A Figura 20 ilustra toda a árvore de busca para o problema do labirinto, Figura 1, onde cada nó tem um estado s, o valor da função heurística $f(n) = g(n) + h(n)$ e uma ação a[1] que leve para outro estado. Perceba que o custo, $g(n)$, é calculado pelo caminho, logo na bifurcação no estado s_{12} os valores de $g(n)$ para os estado s_{02} e s_{22} tem o mesmo valor.

Figura 20 – Árvore de busca completa no problema do labirinto.



Fonte: Elaborado pelo autor.

[1] C - Cima; B - Baixo; D - Direita; E - Esquerda.

APÊNDICE B – RESULTADOS COMPLETOS DOS EXPERIMENTOS

Este apêndice apresenta de forma detalhada todos os resultados obtidos nos experimentos conduzidos para os domínios `Logistics` e `Rovers`. Foram avaliados quatro métodos distintos de busca: `OLD`, `OLD_TIME`, `ASTAR` e `GBFS`, considerando três diferentes tempos limite de execução para cada método — 5, 10 e 15 minutos.

As tabelas a seguir organizam esses dados em sequência, facilitando a comparação e análise dos resultados completos que fundamentam as discussões apresentadas no corpo do trabalho.

B.1 Resultados pra o domínio `Logistics`

B.1.1 Tempo de 5 minutos

Tabela 11 – Resultados para o domínio `logistics` com método *OLD* e tempo limite de 5 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
logistics 4	22	2579	32	22	334	16
logistics 6	25	2432	32	*35_error	#	#
logistics 8	16	*300000	#	#	#	#
logistics 10	7	*300000	#	#	#	#
logistics 12	9	*300000	#	#	#	#
logistics 14	6	*300000	#	#	#	#

Fonte: Elaborado pelo autor.

Tabela 12 – Resultados para o domínio `logistics` com método *OLD_TIME* e tempo limite de 5 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
logistics 4	22	2817	32	22	383	16
logistics 6	25	2656	32	*35_error	#	#
logistics 8	17	*313250	32	34	163768	1376
logistics 10	7	*305501	5792	15	*1861991	9792
logistics 12	9	*305385	4064	15	*1887280	9792
logistics 14	6	*324420	4064	10	*1902968	9792

Fonte: Elaborado pelo autor.

Tabela 13 – Resultados para o domínio `logistics` com método *ASTAR* e tempo limite de 5 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo(ms)**	**Memória**	**Nº Camadas**	**Tempo(ms)**	**Nº Nós**	**Memória**
logistics 4	22	2695	16	22	536	61	16
logistics 6	25	2191	16	29	2830	1321	32
logistics 8	17	*309032	1168	34	151995	65	1360
logistics 10	8	*306201	5776	15	*1806580	16	9424
logistics 12	9	*306975	4048	15	*1802641	16	9040
logistics 14	6	*320699	4048	10	*1810297	11	9424

Fonte: Elaborado pelo autor.

Tabela 14 – Resultados para o domínio `logistics` com método *GBFS* e tempo limite de 5 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo(ms)**	**Memória**	**Nº Camadas**	**Tempo(ms)**	**Nº Nós**	**Memória**
logistics 4	22	2434	16	22	539	61	16
logistics 6	25	2328	16	624688	*1800005	1874065	348
logistics 8	17	*309409	1168	34	145645	65	1360
logistics 10	8	*303304	5776	15	*1813676	16	9424
logistics 12	9	*306503	4048	15	*1801914	16	9040
logistics 14	6	*307134	2512	10	*1801933	11	9040

Fonte: Elaborado pelo autor.

B.1.2 Tempo de 10 minutos

Tabela 15 – Resultados para o domínio `logistics` com método *OLD* e tempo limite de 10 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
logistics 4	22	2913	32	22	282	16
logistics 6	25	2689	32	*35_error	#	#
logistics 8	18	*600000	5599	#	#	#
logistics 10	8	*600000	10128	#	#	#
logistics 12	9	*600000	10144	#	#	#
logistics 14	6	*600000	`*memory`	#	#	#

Fonte: Elaborado pelo autor.

Tabela 16 – Resultados para o domínio `logistics` com método *OLD_TIME* e tempo limite de 10 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
logistics 4	22	3079	32	22	306	16
logistics 6	25	2787	32	*36_error	#	#
logistics 8	18	*615521	4064	34	88335	1760
logistics 10	8	*613793	4447	*14	*1869163	10528
logistics 12	9	*618440	4431	15	*1876416	10528
logistics 14	6	*629558	4431	10	*1855039	11296

Fonte: Elaborado pelo autor.

Tabela 17 – Resultados para o domínio `logistics` com método *ASTAR* e tempo limite de 10 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo(ms)**	**Memória**	**Nº Camadas**	**Tempo(ms)**	**Nº Nós**	**Memória**
logistics 4	22	2729	16	22	617	61	16
logistics 6	25	2605	16	28	3184	1321	32
logistics 8	18	*609956	4048	34	81318	70	1744
logistics 10	8	*609189	6927	15	*1814746	16	10512
logistics 12	8	*629028	6927	15	*1800983	16	10512
logistics 14	6	*643792	4415	10	*1810494	11	10512

Fonte: Elaborado pelo autor.

Tabela 18 – Resultados para o domínio `logistics` com método *GBFS* e tempo limite de 10 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo(ms)**	**Memória**	**Nº Camadas**	**Tempo(ms)**	**Nº Nós**	**Memória**
logistics 4	22	2781	16	22	629	61	16
logistics 6	25	2711	16	556678	*1800003	1670035	703
logistics 8	18	*624414	5776	34	57502	70	16
logistics 10	8	*637175	5199	15	*1807781	16	10512
logistics 12	9	*651551	6927	15	*1854300	16	10112
logistics 14	6	*638800	4431	10	*1801616	11	10512

Fonte: Elaborado pelo autor.

B.1.3 Tempo de 15 minutos

Tabela 19 – Resultados para o domínio `logistics` com método *OLD* e tempo limite de 15 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
logistics 4	22	2821	32	22	278	16
logistics 6	25	2588	32	*35_error	#	#
logistics 8	20	*1196486	5599	#	#	#
logistics 10	9	*1755317	*memory	#	#	#
logistics 12	10	*1848245	*memory	#	#	#
logistics 14	6	*1421548	*memory	#	#	#

Fonte: Elaborado pelo autor.

Tabela 20 – Resultados para o domínio `logistics` com método *OLD_TIME* e tempo limite de 15 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
logistics 4	22	2627	32	22	262	16
logistics 6	25	2602	32	*35_error	#	#
logistics 8	20	*912224	3679	34	57481	2128
logistics 10	9	*941929	6351	15	*1886913	31712
logistics 12	10	*949788	6751	15	*1869093	13216
logistics 14	6	*931185	5967	9	*1852101	33056

Fonte: Elaborado pelo autor.

Tabela 21 – Resultados para o domínio `logistics` com método *ASTAR* e tempo limite de 15 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo(ms)**	**Memória**	**Nº Camadas**	**Tempo(ms)**	**Nº Nós**	**Memória**
logistics 4	22	3168	16	22	852	61	16
logistics 6	25	2907	16	28	4269	1321	32
logistics 8	19	*913605	3663	34	77586	72	2128
logistics 10	9	*906241	5583	15	*1836631	16	10896
logistics 12	9	*942091	6735	15	*1842041	16	12816
logistics 14	6	*949164	7119	10	*1801599	11	28848

Fonte: Elaborado pelo autor.

Tabela 22 – Resultados para o domínio `logistics` com método *GBFS* e tempo limite de 15 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo(ms)**	**Memória**	**Nº Camadas**	**Tempo(ms)**	**Nº Nós**	**Memória**
logistics 4	22	3131	16	22	846	61	16
logistics 6	25	2877	16	543881	*1800004	1631643	496
logistics 8	34	*915837	3663	34	80377	72	2128
logistics 10	15	*912715	5583	15	*1802520	16	10512
logistics 12	15	*941117	941117	15	*1801561	16	12432
logistics 14	10	*945752	6735	10	*1831070	11	13200

Fonte: Elaborado pelo autor.

B.2 Resultados pra o domínio `Rovers`

B.2.1 Tempo de 5 minutos

Tabela 23 – Resultados para o domínio `Rovers` com método *OLD* e tempo limite de 5 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo(ms)**	**Memória**	**Nº Camadas**	**Tempo(ms)**	**Memória**
rovers 1	11	289	32	11	277	16
rovers 2	13	300	32	14	280	16
rovers 3	12	454	32	12	280	16
rovers 4	9	689	32	9	284	16
rovers 5	26	18785	32	27	3005	16
rovers 6	24	*300348	#	#	#	#
rovers 7	28	211062	32	28	11913	16
rovers 8	12	*300000	#	#	#	#

Fonte: Elaborado pelo autor.

Tabela 24 – Resultados para o domínio `Rovers` com método *OLD_TIME* e tempo limite de 5 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
rovers 1	11	297	32	11	274	16
rovers 2	13	326	32	14	281	16
rovers 3	12	468	32	12	291	16
rovers 4	9	716	32	9	303	16
rovers 5	26	18396	32	27	3073	16
rovers 6	25	*300911	32	43	159001	4064
rovers 7	28	196992	32	28	11228	16
rovers 8	12	*305563	2528	11	*1910187	12448

Fonte: Elaborado pelo autor.

Tabela 25 – Resultados para o domínio `Rovers` com método *ASTAR* e tempo limite de 5 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo(ms)**	**Memória**	**Nº Camadas**	**Tempo(ms)**	**Nº Nós**	**Memória**
rovers 1	11	450	16	11	461	27	16
rovers 2	13	459	16	14	479	40	16
rovers 3	12	577	16	12	464	31	16
rovers 4	9	843	16	9	481	22	16
rovers 5	26	15759	16	27	3890	84	16
rovers 6	26	*301214	1168	42	145443	172	2896
rovers 7	28	172998	16	28	9900	78	16
rovers 8	12	*304622	4048	12	*1864683	13	12800

Fonte: Elaborado pelo autor.

Tabela 26 – Resultados para o domínio `Rovers` com método *GBFS* e tempo limite de 5 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo(ms)**	**Memória**	**Nº Camadas**	**Tempo(ms)**	**Nº Nós**	**Memória**
rovers 1	10	447	16	11	476	27	16
rovers 2	13	457	16	14	476	36	16
rovers 3	12	590	16	12	454	31	16
rovers 4	9	790	16	9	466	22	16
rovers 5	26	16431	16	27	3008	75	16
rovers 6	26	*302204	1168	43	51385	122	1360
rovers 7	28	176410	16	28	10595	78	16
rovers 8	12	*307059	4048	11	*1807631	12	12432

Fonte: Elaborado pelo autor.

B.2.2 Tempo de 10 minutos

Tabela 27 – Resultados para o domínio `Rovers` com método *OLD* e tempo limite de 10 minutos.

		Regressivo			Progressivo		
Problema	**Expectativa**	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
rovers 1	10	11	207	32	11	194	16
rovers 2	8	13	206	32	14	199	16
rovers 3	11	12	294	32	12	197	16
rovers 4	8	9	495	32	9	209	16
rovers 5	22	26	12303	32	27	2018	16
rovers 6	36	41	594050	5792	43	5856	16
rovers 7	18	28	259392	32	28	13225	16
rovers 8	26	13	*600000	7519	#	#	#

Fonte: Elaborado pelo autor.

Tabela 28 – Resultados para o domínio `Rovers` com método *OLD_TIME* e tempo limite de 10 minutos.

		Regressivo			Progressivo		
Problema	**Expectativa**	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
rovers 1	10	11	228	32	11	195	16
rovers 2	8	13	220	32	14	195	16
rovers 3	11	12	317	32	12	198	16
rovers 4	8	9	468	32	9	201	16
rovers 5	22	26	12399	32	27	2024	16
rovers 6	36	41	520238	5792	43	5120	16
rovers 7	18	28	253806	32	28	13245	16
rovers 8	26	13	*610599	5792	11	*1875621	11680

Fonte: Elaborado pelo autor.

Tabela 29 – Resultados para o domínio `Rovers` com método *ASTAR* e tempo limite de 10 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Nº Nós**	**Memória**
rovers 1	11	529	16	11	527	27	16
rovers 2	13	521	16	14	563	40	16
rovers 3	12	644	16	12	538	31	16
rovers 4	9	903	16	9	551	22	16
rovers 5	26	18188	16	27	4546	84	16
rovers 6	33	*601906	4048	43	128382	962	1760
rovers 7	28	206030	16	28	13366	78	16
rovers 8	13	*617666	5775	11	*1827196	12	12800

Fonte: Elaborado pelo autor.

Tabela 30 – Resultados para o domínio `rovers` com método *GBFS* e tempo limite de 10 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Nº Nós**	**Memória**
rovers 1	11	530	16	11	542	27	16
rovers 2	13	530	16	14	542	36	16
rovers 3	12	650	16	12	525	31	16
rovers 4	9	893	16	9	537	22	16
rovers 5	26	18367	16	27	3682	75	16
rovers 6	33	*600566	4048	43	9224	141	16
rovers 7	28	207382	16	28	13927	78	16
rovers 8	13	*614367	5775	11	*1802596	12	12432

Fonte: Elaborado pelo autor.

B.2.3 Tempo de 15 minutos

Tabela 31 – Resultados para o domínio `Rovers` com método *OLD* e tempo limite de 15 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
rovers 1	11	259	32	11	232	16
rovers 2	13	278	32	14	253	16
rovers 3	12	571	32	12	250	16
rovers 4	9	1079	32	9	255	16
rovers 5	26	29051	32	27	4549	16
rovers 6	41	886590	5792	43	8949	16
rovers 7	28	232948	32	28	11805	16
rovers 8	14	*900000	6367	#	#	#

Fonte: Elaborado pelo autor.

Tabela 32 – Resultados para o domínio `Rovers` com método *OLD_TIME* e tempo limite de 15 minutos.

	Regressivo			Progressivo		
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Memória**
rovers 1	11	263	32	11	216	16
rovers 2	13	302	32	14	247	16
rovers 3	12	509	32	12	238	16
rovers 4	9	1086	32	9	260	16
rovers 5	26	29244	32	27	4524	16
rovers 6	36	*900871	4064	43	22480	1744
rovers 7	28	223876	32	28	11560	16
rovers 8	14	*910297	4447	11	*1914017	13216

Fonte: Elaborado pelo autor.

Tabela 33 – Resultados para o domínio `rovers` com método *ASTAR* e tempo limite de 15 minutos.

	Regressivo			Progressivo			
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Nº Nós**	**Memória**
rovers 1	11	721	16	11	747	27	16
rovers 2	13	742	16	14	774	40	16
rovers 3	12	864	16	12	749	31	16
rovers 4	9	1110	16	9	776	22	16
rovers 5	26	18864	16	27	5003	84	16
rovers 6	41	715199	5776	43	25556	209	16
rovers 7	28	207236	16	28	14106	78	16
rovers 8	14	*925582	5183	11	*1803727	12	13200

Fonte: Elaborado pelo autor.

Tabela 34 – Resultados para o domínio `rovers` com método *GBFS* e tempo limite de 15 minutos.

	Regressivo			**Progressivo**			
Problema	**Nº Camadas**	**Tempo (ms)**	**Memória**	**Nº Camadas**	**Tempo (ms)**	**Nº Nós**	**Memória**
rovers 1	11	729	16	11	746	27	16
rovers 2	13	748	16	14	752	36	16
rovers 3	12	925	16	12	758	31	16
rovers 4	9	1140	16	9	739	22	16
rovers 5	26	19887	16	27	3998	75	16
rovers 6	41	729278	5776	43	8550	121	16
rovers 7	28	208574	16	28	14035	78	16
rovers 8	14	*908668	6927	11	*1883481	12	16432

Fonte: Elaborado pelo autor.