Relatório Técnico

Projeto Adji-boto*

UC Inteligência Artificial 2024/2025

Realizado por: Raul Rohjans - 202100518

Introdução

O objetivo deste projeto foi desenvolver um algoritmo para resolver o puzzle Adji-boto* utilizando algoritmos de pesquisa. Os algoritmos implementados incluem:

- Pesquisa em Largura (BFS - Breadth-First Search)
- Pesquisa em Profundidade (DFS - Depth-First Search)
- Pesquisa A* com duas heurísticas diferentes (padrão e personalizada)

A implementação foi realizada em **Common Lisp**, e o projeto consiste em três ficheiros principais:

- `projeto.lisp`: Contém as funções principais e a interface de interação com o utilizador.
- `procura.lisp`: Implementa os algoritmos de pesquisa.
- `puzzle.lisp`: Define a mecânica do puzzle e as operações sobre os nós.

Funções Heurísticas

Heurística Padrão

A heurística padrão estima o número de movimentos restantes ao avaliar quantas peças ainda estão presentes no puzzle. É calculada da seguinte forma:

[h(n) = -]

Esta heurística fornece uma estimativa básica, mas não considera a distribuição das peças ou a complexidade dos movimentos.

Heurística Personalizada

A heurística personalizada adota uma abordagem mais refinada, medindo a dispersão das peças dentro do puzzle. A ideia é que uma distribuição mais concentrada das peças permite movimentos mais eficientes. Esta heurística é calculada da seguinte forma:

[h(n) = {dispersao das peças}]

Calcula o número de peças presentes em cada linha e coluna, priorizando movimentos que as concentrem em menos posições. Isto resulta numa pesquisa mais informada em comparação com a heurística padrão.

Detalhes da Implementação

Algoritmos de Pesquisa

Pesquisa em Largura (BFS)

O BFS explora a árvore de pesquisa nível por nível. Garante encontrar a solução mais curta, mas é intensivo em memória.

Pesquisa em Profundidade (DFS)

O DFS explora um ramo completamente antes de retroceder. Durante os testes, foi utilizado um limite de profundidade de 10 para evitar recursão excessiva. No entanto, soluções mais profundas podem não ser encontradas.

Pesquisa A*

O A* utiliza uma função heurística para guiar a pesquisa de forma mais eficiente em direção ao objetivo.

- **Heurística padrão**: Estima os movimentos restantes com base no número de peças restantes.
- **Heurística personalizada**: Avalia a dispersão das peças, favorecendo configurações onde as peças estão concentradas em menos colunas.

Funções Auxiliares

Manipulação do Tabuleiro

- `upper-entry-line(entry)`: Retorna a linha superior de um dado tabuleiro.
- `lower-entry-line(entry)`: Retorna a linha inferior de um dado tabuleiro.
- `entry-piece-count(entry)`: Conta o número de peças num tabuleiro.
- `replace-piece(line, cell, entry, piece)`: Substitui uma peça numa posição específica.

Operações sobre Nós

- `create-node(entry, parent-node, depth, removed-pieces, heuristic)`: Cria um nó para representação de estado.
- `expand-node(node)`: Expande um nó para gerar sucessores.
- `existsp(node, closed-nodes)`: Verifica se um nó já existe na lista fechada.

Limitações

1. **Problemas de Performance**:
    - BFS e DFS são impraticáveis para instâncias grandes do problema devido a restrições de memória.
    - A* depende fortemente da eficácia da função heurística.
2. **Casos Não Resolvidos**:
    - Para configurações de tabuleiro altamente complexas, nenhum dos algoritmos retorna uma solução devido a um stack overflow ou tempo de computação excessivo.

Análise Crítica

1. **Eficiência dos Algoritmos**:
    - **BFS**: Garante soluções mais curtas, mas consome muita memória.
    - **DFS**: Usa menos memória, mas pode não encontrar a solução ótima.
    - **A***: Fornece o melhor equilíbrio, mas exige heurísticas bem projetadas.
2. **Desempenho das Heurísticas**:
    - A **heurística padrão** é eficaz em casos simples, mas carece de insight estratégico.
    - A **heurística personalizada** melhora a eficiência ao priorizar configurações mais agrupadas de peças.
3. **Ineficiência do Algoritmo A***:
    - A implementação do A* por vezes resulta em erros ou execução ineficiente.
4. **Estatísticas**:
    - O programa exibe corretamente o caminho da execução, contudo, não apresenta métricas sobre a mesma, como nós gerados ou expandidos.

Conclusão

Este projeto implementou com sucesso algoritmos de pesquisa para resolver o puzzle Adji-boto*. O uso de heurísticas no A* demonstrou melhorias significativas sobre estratégias de pesquisa não informadas. No entanto, a implementação atual enfrenta dificuldades com instâncias grandes devido à profundidade de recursão e limitações de memória.

Possíveis Melhorias

- Otimizar DFS e BFS para evitar stack overflow.
- Adicionar mais funções heurísticas para o A* para melhor guiar a pesquisa.