Universidade do Algarve

Faculdade de Ciências e Tecnologias

Engenharia Informática

Inteligência Artificial

Containers++

Autor(s)

Márcio Felício a76999
Maria Anjos 79768
Miguel Rosa 76936

IA2425P03G03

Docente(s)

José Valente de Oliveira
Pedro Martins

Relatório apresentado em cumprimento dos requesitos da
Universidade do Algarve na cadeira de Inteligência Artificial para o grau de licenciatura em *Engenharia Informática*

24 de Outubro, 2024

Indíce

Capítulo 1

Introdução

Figure 1.1.

```
O6l3p5m1x2E9 M5u7 e6 o2t1s6I7 a2 A2
Isto e uM Exmpl0 a A
Estimated Heuristic Cost: 54.0

[A]
[E, x, m, p, l, O]
[I, s, t, o]
[a]
[e]
[u, M]

Results:
Total Cost (G): 54
Path Length (L): 12.0
Generated Nodes (G): 729
Expanded Nodes (E): 13
Penetrance (P = L / E): 0.9230769230769231
Total Time: 0.036655 seconds
```

Containers++ "Isto é um Exemplo a A"example.
Feito por: Márcio Felício, Maria Anjos, Miguel Rosa

Este problema envolve mover contentores de uma configuração inicial para uma configuração final com o menor custo acumulado possível. Cada contentor possui um custo específico para ser movido, e a solução deve seguir regras simples: apenas contentores que não possuem outros em cima podem ser movidos, seja para o chão ou para outra pilha. O desafio consiste em desenvolver um algoritmo de busca heurística no espaço de estados, capaz de encontrar o caminho de menor custo acumulado. Além disso, a solução precisa ser eficiente para lidar com instâncias maiores e mais complexas, conforme os casos estendidos propostos pelo problema.

Para resolver este problema, utilizámos o algoritmo de busca A*. A implementação, juntamente com a heurística desenvolvida, permitiu-nos lidar eficientemente com instâncias maiores e mais complexas em tempo hábil. A heurística desempenhou um papel fundamental para os bons resultados obtidos, já que sua elaboração cuidadosa foi essencial para o desempenho geral do algoritmo. Através dessa heurística, o A* foi capaz de encontrar soluções ótimas dentro de um tempo aceitável, assegurando o equilíbrio entre precisão e eficiência.

Capítulo 2

Diagrama de Classes UML Containers++

2.1 Apresentação do Diagrama

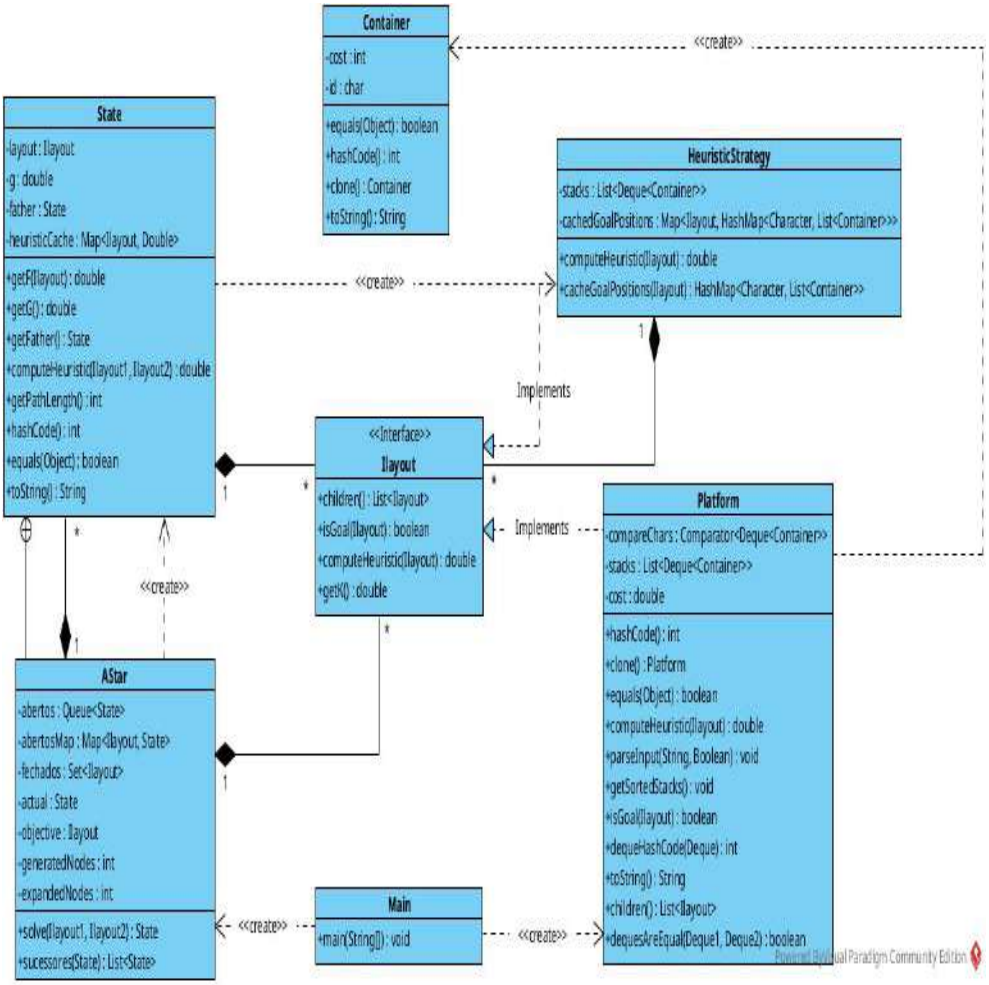

Figure 2.1: Diagrama UML do projeto baseado no padrão de projeto Strategy (Para melhor visualização da imagem, a mesma encontra-se na pasta UML)

2.2 Breve Descrição e Responsabilidade de cada Classe

Classe Cliente (Main)

- A classe Cliente é responsável por fornecer um ponto de entrada para execução e testes das funcionalidades principais do projeto Containers++. Ela permite que o usuário insira configurações iniciais e finais de contentores e realiza a execução do algoritmo A* com uma heurística específica, exibindo os resultados, como o custo minimo acumulado e a configuração goal.

Classe AStar

- Classe que trata da implementação do algoritmo A* para resolver o problema dos contentores, utilizando a heurística definida na classe `Platform`.
- Invariantes:
  - A heurística (h(n)) deve ser admissível, ou seja, nunca deve superestimar o custo real para alcançar o objetivo.
  - O custo acumulado (g(n)) de cada nó não deve ser negativo.
  - A fila de prioridade (openSet) deve estar ordenada de acordo com o valor mínimo de `f(n) = g(n) + h(n)`.

Classe State

- A classe State representa um estado específico no espaço de busca do algoritmo A*, contendo informações sobre a configuração atual de contentores e o custo acumulado para chegar a essa configuração. Essa classe é essencial para armazenar o progresso da busca e calcular o custo total do caminho.
- Invariantes:
  - O custo acumulado (g(n)) deve refletir com precisão o custo real necessário para atingir o estado atual a partir do estado inicial, assegurando uma avaliação correta dos caminhos durante a busca.
  - A comparação entre objetos State deve ser consistente para possibilitar uma correta priorização na fila de prioridades do algoritmo A*, permitindo que o algoritmo expanda sempre o estado com menor custo (h(n) + g(n)) estimado.

Classe HeuristicStrategy

- A classe HeuristicStrategy é responsável por calcular o custo heurístico estimado (h(n)) para uma configuração específica de contentores, com o objetivo de auxiliar o algoritmo A* a encontrar o caminho de custo mínimo para a configuração objetivo.
- Invariantes:
  - O cache cacheGoalPositions deve armazenar corretamente as posições de cada contentor para as configurações objetivo já processadas, de modo a evitar cálculos repetitivos.
  - O valor retornado pelo método computeHeuristic nunca deve superestimar o custo real para transformar a configuração atual na configuração objetivo, garantindo assim a admissibilidade da heurística no contexto do algoritmo A*.

Classe Platform

- Esta classe contém construtores que criam objetos Platform e o seu principal objetivo é criar sucessores de uma certa configuração.
- Invariantes:
  - Cada contentor na configuração inicial deve ter um custo de movimento associado, caso contrário, uma `PlatformException` será lançada.
  - As pilhas de contentores devem estar organizadas e comparadas com base na ordem lexicográfica do identificador do contentor na base.

Classe Container

- Esta classe contém construtores que criam objetos Container e o seu principal objetivo é armazenar os valores primitivos associados aos contentores ((id)character e custo de movimento)
- Invariantes:
  - O custo de movimento do contentor deve ser maior ou igual a zero, caso contrário, uma `ContainersException` será lançada.

Interface Ilayout

- Interface que define os métodos fundamentais para representar uma configuração de contentores, incluindo geração de sucessores, teste de objetivo e cálculo de custo heurístico.
- Invariantes:
  - Cada classe que implementa `Ilayout` deve implementar métodos para gerar sucessores, estimar o custo heurístico e verificar igualdade com uma configuração objetivo.

Exception ContainersException

- Exceção personalizada lançada quando uma operação com um contentor viola as restrições impostas pela classe `Container`, como custo de movimento negativo.

Exception PlatformException

- Exceção personalizada lançada quando uma configuração de `Platform` viola os requisitos da configuração inicial, como contentores sem custo associado.

Capítulo 3

Heuristica Implementada

3.1 Explicação:

A heurística calculada pelo método `computeHeuristic` estima o custo restante para atingir a configuração objetivo, comparando a disposição atual dos contentores com a disposição de destino. Utilizando uma estrutura de cache (`cacheGoalPositions`), a heurística armazena as posições-alvo dos contentores para evitar cálculos repetidos e melhorar a eficiência.

Para cada pilha:

- **Identificação da Posição Base**: A heurística identifica o identificador do contentor na base da pilha atual e usa-o para encontrar a pilha correspondente na configuração objetivo.
- **Comparação das Pilhas**: Se a pilha correspondente existe na configuração objetivo, o código compara os contentores em ambas as pilhas até a altura mínima entre as duas. Ao encontrar uma discrepância (ou seja, um contentor fora da posição correta), marca o índice do primeiro contentor fora do lugar. A partir desse ponto, soma o custo de todos os contentores restantes na pilha atual (pois todos estão fora de lugar).
- **Pilhas com Tamanhos Diferentes**: Se a pilha atual é mais alta que a correspondente no objetivo, a heurística adiciona o custo dos contentores extras (ou seja, aqueles que não têm correspondência na pilha de destino).
- **Pilha Inexistente no Objetivo**: Se a pilha atual não existe na configuração objetivo (ou seja, não tem base correspondente), o custo de movimento de todos os contentores da pilha é somado, uma vez que nenhum deles faz parte da configuração de destino.

Capítulo 4

Opções de Projeto Tomadas

4.1 Padrão de Projeto Strategy:

No contexto do algoritmo A*, que atua como o "contexto"no padrão Strategy, adotamos uma interface comum para o cálculo da heurística, permitindo que diferentes estratégias de estimativa de custo sejam trocadas e testadas conforme a necessidade. Embora, neste projeto, tenha sido implementada apenas uma estratégia concreta de heurística, o uso do padrão Strategy foi planejado para oferecer flexibilidade. Dessa forma, novas heurísticas podem ser facilmente adicionadas no futuro, bastando implementar a mesma interface, sem modificar o código central do algoritmo A*.

- **Modularidade e Reusabilidade**: Com a interface de heurística separada, o cálculo da estimativa de custo é isolado em uma classe específica. Isso permite que o código seja modular, facilitando a manutenção e a reutilização da lógica de heurística em outras partes do projeto ou em novos projetos que utilizem o A*.
- **Facilidade de Teste e Comparação de Heurísticas**: A estrutura do Strategy permite que heurísticas diferentes sejam testadas e comparadas facilmente. Caso seja necessário otimizar o algoritmo A* para diferentes configurações ou instâncias de contentores, basta implementar uma nova classe de heurística e passar essa implementação para o algoritmo. Isso facilita a experimentação e a otimização de desempenho.
- **Adaptabilidade a Novos Problemas**: O padrão Strategy torna o algoritmo A* mais adaptável a problemas de busca que necessitam de estimativas de custo específicas. Se, por exemplo, o problema dos contentores for expandido para incluir novos tipos de contentores ou diferentes regras de movimento, uma nova classe de heurística pode ser desenvolvida para refletir esses requisitos sem alterar o núcleo do algoritmo.
- **Separação de Responsabilidades**: O padrão Strategy segue o princípio da responsabilidade única, já que cada classe de heurística lida apenas com o cálculo da estimativa de custo. Isso mantém o código mais organizado e facilita a compreensão das diferentes partes do projeto, especialmente para quem precisar revisá-lo ou expandi-lo.
- **Uso de Cache para Otimização**: No contexto das heurísticas, foi implementado um mecanismo de cache para armazenar o layout de configuração objetivo, otimizando o desempenho ao evitar cálculos repetitivos para layouts similares. Esse cache é uma melhoria de performance que reforça a eficiência da heurística e maximiza o potencial do Strategy ao lidar com grandes espaços de busca.

Capítulo 5

Testes Unitários

Testes Unitários encontram-se no ficheiro src/UnitTests em anexo.

Capítulo 6

Resultados, Análises e Discussões

6.1 Resultados

Sample Input 1
Initial configuration - A1B2C3 E1D5
Goal configuration - CA B ED

Sample input 2
Initial configuration - O6l3p5m1x2E9 M5u7 e6 o2t1s6l7 a2 A2
Goal Configuration - Isto e uM ExmplO a A

Instância	Nós Expandidos (E)	Nós Gerados (G)	Comprimento da Solução (L)	Penetração (P = L/E)	Tempo (s)	Memória (Megabytes)
Sample Input 1	22	120	3	0.136	0.029	0.96 MB
Sample Input 2	OutOfMemoryError	OutOfMemoryError	OutOfMemoryError	OutOfMemoryError	OutOfMemoryError	OutOfMemoryError

Table 6.1: Resultados do Algoritmo A* com Busca de Custo Uniforme (UCS)

Instância	Nós Expandidos (E)	Nós Gerados (G)	Comprimento da Solução (L)	Penetração (P = L/E)	Tempo (s)	Memória (Megabytes)
Sample Input 1	4	29	3	0.75	0.008	0.96 MB
Sample Input 2	70	4223	12	0.171	0.089	2.83 MB

Table 6.2: Resultados do Algoritmo A* com Heurística

6.2 Análises

6.2.1 Instância Sample Input 1

- **Nós Expandidos (E)**: O A* com heurística expande apenas 4 nós, significativamente menos que os 22 nós expandidos pelo UCS, evidenciando uma busca mais direcionada.
- **Nós Gerados (G)**: O número de nós gerados também é consideravelmente menor, totalizando 29, o que representa uma redução expressiva em relação ao UCS.
- **Comprimento da Solução (L)**: O comprimento do caminho da solução é de 3 movimentos, mantendo-se o mesmo em ambas as abordagens e confirmando que a heurística é admissível.
- **Penetração (P)**: A penetração do A* com heurística é de 0.75, muito superior ao UCS (0.136), o que indica que o A* com heurística é significativamente mais eficiente ao evitar expansões desnecessárias.

- **Tempo (s)**: O tempo de execução com heurística é de apenas 0.008 segundos, uma melhora notável em comparação com os 0.029 segundos do UCS.
- **Memória (MB)**: Tanto o A* com heurística quanto o UCS utilizam aproximadamente 0.96 MB de memória para essa instância. Isso indica que, para o caso mais simples, o consumo de memória é similar entre as abordagens.

6.2.2 Instância Sample Input 2

- **Nós Expandidos (E)**: A* com heurística expande 70 nós, enquanto o UCS não consegue completar a busca devido a um `OutOfMemoryError`. Esse valor mais baixo demonstra a eficiência da heurística ao evitar expansões desnecessárias.
- **Nós Gerados (G)**: O A* com heurística gera um total de 4223 nós. Embora o número seja alto, ele permanece gerenciável e demonstra que a heurística está orientando a busca de forma eficiente.
- **Comprimento da Solução (L)**: O comprimento da solução é de 12 movimentos, o que é adequado para alcançar o objetivo, mantendo a admissibilidade da heurística.
- **Penetração (P)**: A penetração é de 0.171, consideravelmente melhor que o UCS, o que reafirma a eficiência da heurística em reduzir expansões.
- **Tempo (s)**: O tempo de execução é de 0.089 segundos, um valor aceitável que demonstra que o A* com heurística consegue resolver instâncias complexas sem problemas de memória.
- **Memória (MB)**: O consumo de memória do A* com heurística é de aproximadamente 2.83 MB. Em contraste, o UCS atinge um limite de memória (OutOfMemoryError), evidenciando que o A* com heurística é muito mais eficiente em termos de uso de memória para essa instância complexa.

6.3 Discussões

Os resultados indicam que o A* com heurística oferece uma vantagem clara em termos de eficiência e desempenho em ambas as instâncias analisadas:

- Para a Sample Input 1, a heurística reduz drasticamente o número de nós expandidos e gerados, além de diminuir o tempo de execução. O consumo de memória permanece em torno de 0.96 MB para ambas as abordagens, mas o A* com heurística demonstra uma busca mais eficiente e direcionada em comparação com o UCS.
- Para a Sample Input 2, a heurística permite que o algoritmo A* complete a busca com um número razoável de expansões e um tempo de execução aceitável, enquanto o UCS falha devido a limitações de memória. A penetração mais alta sugere que a heurística ajuda o algoritmo a manter o foco na solução, reduzindo a exploração desnecessária e otimizando o uso da memória. O A* com heurística utiliza cerca de 2.83 MB, o que é gerenciável, enquanto o UCS enfrenta problemas de alocação de memória.

Esses resultados demonstram que o uso de uma heurística admissível não só torna o A* mais escalável e eficiente, mas também otimiza o consumo de memória, especialmente em instâncias mais complexas. Isso confirma a importância de heurísticas bem projetadas para

melhorar o desempenho, reduzir o número de expansões e gerenciar a memória de forma eficiente em algoritmos de busca, tornando a resolução de problemas complexos mais viável e eficaz.

Capítulo 7

Conclusão

A conclusão do projeto Containers++ destaca a importância do uso de heurísticas e do padrão de projeto Strategy na resolução de problemas complexos de busca. Implementando o algoritmo A* com uma heurística admissível, conseguimos desenvolver uma solução eficiente e escalável, que otimiza o número de expansões e a utilização de memória em comparação com a busca de custo uniforme. As decisões de design, especialmente a implementação do Strategy, permitiram que diferentes heurísticas fossem testadas de forma modular, facilitando a adaptação e expansão para problemas similares no futuro.

Esse projeto não só reforçou habilidades em Inteligência Artificial e Java, como também evidenciou o valor de boas práticas de design para garantir a manutenibilidade e a eficácia do código. Em suma, o projeto Containers++ foi uma oportunidade enriquecedora para aplicar conceitos teóricos em um ambiente prático, proporcionando um aprendizado significativo para futuros desenvolvimentos em algoritmos de busca e design de software orientado a objetos.

Capítulo 8

Bibliografia

LaTeX
Disponível em: https://www.latex-project.org/.

Zotero
Disponível em: https://www.zotero.org/.

ia2024-25T5.pdf
Disponível em: https://tutoria.ualg.pt/2024/pluginfile.php/132103/mod_resource/content/2/ia2024-25T5.pdf.

Russel, Stuart; Norvig, Peter
Heuristic state-space search
Disponível em: https://tutoria.ualg.pt/2024/pluginfile.php/139936/mod_resource/content/1/ia2024-25T7.pdf.