UAlg FCT
UNIVERSIDADE DO ALGARVE
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Ano letivo: 2024/2025
UC: Inteligência Artificial

Relatório

Problema 2: Containers ++

PL1

Grupo 02

Andreia Qiu 79856

Eduarda Pereira 79749

Guilherme Carmo 79860

Descrição do Problema e Algoritmo Utilizado

Descrição do Problema

Este relatório descreve a implementação de um algoritmo de procura em Espaço de Estados (State Space Search), utilizando uma abordagem heurística conhecida por pesquisa Melhor- Primeiro com heurística A* (Best-First), aplicada ao problema de organização de containers em pilhas de acordo com uma configuração alvo. Este problema exige que os containers sejam movidos de uma pilha para outra até que o objetivo seja alcançado, minimizando o custo dos movimentos realizados.

O problema é caracterizado por um estado inicial, que representa a configuração atual das pilhas de containers, e, um estado objetivo, que define a configuração desejada. Cada movimento de containers envolve um custo específico, e a meta é encontrar uma sequência de movimentos que atinja o estado alvo no menor custo possível. O estado é representado pela classe Container, na qual cada instância armazena a configuração atual das pilhas e o custo associado a cada container, permitindo assim que o algoritmo calcule tanto o custo acumulado de um caminho $g(n)$ quanto o custo estimado para alcançar o objetivo $h(n)$, formando a função de avaliação $f(n) = g(n) + h(n)$.

Estrutura do código

O algoritmo utilizado é o A*, uma versão da procura Best-First que expande estados com base na função $f(n)$, com o intuito de minimizar o custo total estimado.

A função $g(n)$ acumula o custo dos movimentos desde o estado inicial até ao estado atual, enquanto a heurística, $h(n)$, fornece uma avaliação do custo restante até ao objetivo. Esta heurística foi projetada para ser admissível e consistente, ou seja, não sobrestima o custo real até ao objetivo e cumpre a condição $f(n) \leq f(n')$, garantindo que A* retorne uma solução ótima. No contexto deste problema, a heurística avalia o número de movimentos necessários para organizar os containers conforme o objetivo, somando os custos de containers mal posicionados na configuração atual.

A classe BestFirst implementa a lógica do algoritmo A*, através de uma PriorityQueue que gere estados abertos de forma eficiente, assegurando que o próximo estado a ser expandido seja aquele com menor valor de $f(n)$. De modo a evitar expansões redundantes, recorremos a uma estrutura HashSet para armazenar estados fechados, i.e., aqueles que já foram completamente explorados. Cada estado é representado pela classe State, que contém: layout atual (Container); uma referência ao estado pai, para reconstruir o caminho até ao objetivo; e o custo acumulado $g(n)$. A função $f(n)$ é então calculado como a soma entre $g(n)$ e $h(n)$.

Opções de Design

Diversas opções de design foram tomadas para garantir a eficiência e modularidade. Em primeiro lugar, a função heurística foi otimamente projetada para tornar o algoritmo A* completo, de maneira que encontre uma solução com o menor custo sempre que possível. Segue-se a PriorityQueue com a finalidade de ordenar estados abertos de acordo com o custo total estimado tendo sempre em conta a expansão dos estados mais promissores primeiro. Relativamente à parte modular, separamos as responsabilidades em classes - Container e BestFirst - de tal forma que simplifique a compreensão e reutilização do código.

Unit tests

Localizados na pasta src.

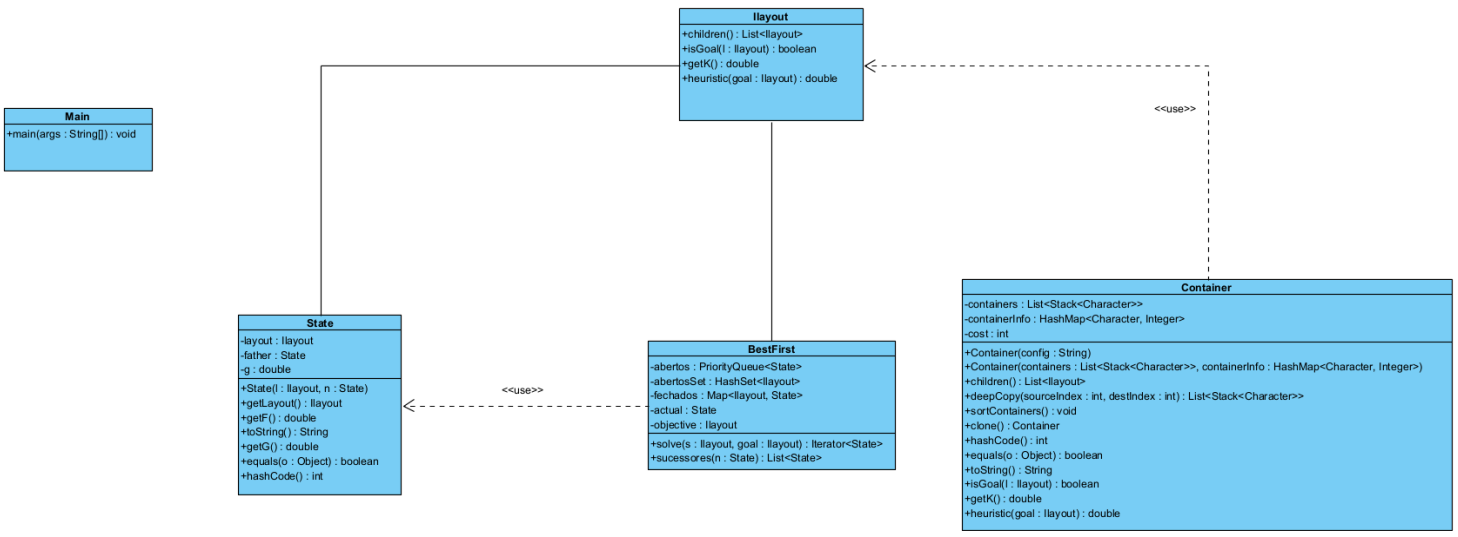
Os testes unitários desenvolvidos para este programa garantem a integridade das principais funcionalidades das classes Container e BestFirst, assegurando que o algoritmo de procura opere corretamente.

Para a classe Container, os testes verificam a correta inicialização dos containers, a capacidade de gerar estados sucessores válidos, a precisão da função heurística, e a consistência dos métodos equals e hashCode. Estes testes garantem que cada configuração de container seja manipulada como esperado, especialmente durante a movimentação entre pilhas.

Na classe BestFirst, os testes focam na procura pesquisa por uma solução viável para uma configuração alvo, no cálculo correto dos custos acumulados e heurísticos, e na prevenção de ciclos e movimentos redundantes. Esses testes confirmam que o algoritmo prioritiza corretamente os estados mais promissores e evita revisitar estados já explorados, melhorando a eficiência do processo de procura.

Os testes proporcionam uma base sólida para validar a implementação, evitando erros e garantindo que o programa execute de forma eficiente e precisa.

Diagrama UML



Resultados, Análise e Discussão

De modo a avaliar o algoritmo implementado, preparamos alguns testes de eficiência e comparamos os 2 algoritmos: procura com Custo Uniforme e procura em Espaço de estados. Por essa razão, na classe State mantivemos a lógica para calcular g mas retiramos o método getF(), e no método Solve modificamos o comparador da fila de prioridade para ordenar apenas com base no valor G, uma vez que também não seria necessário para a procura com Custo Uniforme.

Utilizamos configurações iniciais e finais específicas para simular cenários realistas que representam diferentes níveis de complexidade no espaço de estados. A execução do programa fornece nos informações essenciais à análise do comportamento do algoritmo, como:

- Nós expandidos (E) : indica a quantidade de estados processados pelo algoritmo, ou seja, quantas vezes um estado foi retirado da fila de prioridades e expandido para gerar novos estados.
- Nós gerados (G) : representa o número total de estados criados pelo algoritmo, oferecendo uma visão sobre a quantidade de ramificações no espaço de procura.
- Comprimento da solução (L) : mede o comprimento do caminho encontrado entre o estado inicial e o estado final, refletindo a qualidade e a precisão da solução.
- Penetrância (P) : calculada como a razão entre o comprimento da solução e os nós expandidos, indica a profundidade média da pesquisa em relação aos nós expandidos.

Além disso, analisamos o tempo de execução em milissegundos e a memória utilizada em bytes, fundamentais para avaliar o custo computacional e aperfeiçoar o algoritmo.

Teste	Configuração Inicial	Configuração Final	Custo	E	G	L	P	Tempo Esperado (ms)	Memória utilizada (bytes)
1	A1B2C3 E1D5	CA B ED	6	18	88	4	0,222222	63	880880
2	N5K1F7B3 L9A4D3	LDA BF KN	30	10426	101125	9	0,000863	1589	55795280
3	M7B3A8 N9H3K1	HA NMB K	25	1300	848	7	0,005385	505	16725496
4	Q3A4 S4X6 E4D9	AXSQ DE	30	2308	128215	7	0,003033	395	11991720
5	G4 C5U2 I5 N7s4E3	CN sEG UI	28	26740	201606	8	0,000299	2562	45991616
6	J4S8L1 V2Z9 U5Y4 P1	LVYU PJSZ	41	316213	2545625	9	2,85E-05	24224	275627680
7	K6M2L1H7 T9W1D4	WKH TMLD	33	23220	182449	10	0,000431	2079	51051888
8	J2M3B5P1 N2X3 H5L1	LMPN XJBH	28	348399	2672594	11	3,16E-05	24435	434110464

Gráfico 1 - Testes realizados ao algoritmo de pesquisa de custo uniforme (Uniform-cost search)

Teste	Configuração Inicial	Configuração Final	Custo	E	G	L	P	Tempo Esperado (ms)	Memória utilizada (bytes)
1	A1B2C3 E1D5	CA B ED	6	4	27	4	1	15	587256
2	N5K1F7B3 L9A4D3	LDA BF KN	30	67	720	9	0,134328	222	283184
3	M7B3A8 N9H3K1	HA NMB K	25	19	179	7	0,368421	58	1342208
4	Q3A4 S4X6 E4D9	AXSQ DE	30	14	167	7	0,5	31	1342320
5	G4 C5U2 I5 N7s4E3	CN sEG UI	28	20	380	8	0,4	75	2097152
6	J4S8L1 V2Z9 U5Y4 P1	LVYU PJSZ	41	135	1845	9	0,066667	125	2721448
7	K6M2L1H7 T9W1D4	WKH TMLD	33	115	1405	10	0,086957	186	3215856
8	J2M3B5P1 N2X3 H5L1	LMPN XJBH	28	255	3400	11	0,043137	295	1855320

Gráfico 2 - Testes realizados ao algoritmo de pesquisa em Espaço de Estados (State Space Search)

Custo Uniforme vs. Estado Espacial

O algoritmo de Custo Uniforme apresenta uma quantidade significativamente maior de nós gerados e expandidos em comparação ao algoritmo de procura em Espaço de Estados para todos os testes. Isso ocorre porque o Custo Uniforme expande os nós com base no custo cumulativo, sem considerar heurísticas para direcionar a pesquisa de maneira mais eficiente.

Como consequência, o Custo Uniforme exige muita mais memória e tempo de execução, especialmente para configurações mais complexas, como nos testes 5 a 8.

Nós Expandidos (E) e Gerados (G):

No algoritmo de Custo Uniforme, a quantidade de nós expandidos (E) e gerados (G) é muito maior, o que indica maior consumo de recursos computacionais. O número de nós gerados no Custo Uniforme chega a ser extremamente alto, como nos Testes 5 e 8 (316.213 e 348.399 nós expandidos, respectivamente).
Na procura em Espaço de Estados, os valores de E e G são mais baixos, indicando uma pesquisa mais orientada e eficiente. Por exemplo, no Teste 8, E é reduzido de 348.399 para 255, e G de 2.672.594 para 3.400, mostrando um uso mais económico dos recursos.

Comprimento da Solução (L) e Penetrância (P):

O comprimento da solução (L) é consistente em ambos os algoritmos, o que sugere que encontraram caminhos de qualidade semelhante. No entanto, a penetração (P), que reflete a profundidade média da pesquisa, é drasticamente menor no algoritmo de Custo Uniforme, especialmente para configurações complexas, com valores como 0,000299 no Teste 5 e $3,16 \times 10^{-5}$ no Teste 8. Isto indica que, apesar de expandir muitos nós, o algoritmo de Custo Uniforme explora grande parte do espaço de procura sem uma orientação efetiva.
Na pesquisa em Espaço de Estados, a penetrância (P) é mais alta, com valores consistentes acima de 0,05. Com tal evidência, percebemos que cumpre uma procura mais focada em estados que estão mais próximos da solução, especialmente com a aplicação de heurísticas.

Tempo e Memória Utilizada:
O tempo de execução e o uso de memória são drasticamente menores para a procura em Espaço de Estados. Por exemplo, no Teste 5, o Custo Uniforme leva 2.562 ms e utiliza 4.599.161.616 bytes, enquanto a procura em Espaço de Estados consome apenas 75 ms e utiliza 209.712 bytes.
Esta diferença destaca a eficiência da pesquisa em Espaço de Estados, tornando-a mais apropriada para resolver problemas com um grande espaço de procura onde o uso de heurísticas pode guiar a solução rapidamente e com menor consumo de recursos.

Conclusões

Este projeto demonstrou o potencial do algoritmo Space State Search para problemas de organização baseado em heurísticas. O uso de cache de hash e a otimização da comparação de objetos melhoraram a eficiência, mas o consumo de memória permanece um desafio para problemas de alta complexidade.

Para finalizar, constatamos que a pesquisa em Espaço de Estados é mais eficiente que o Custo Uniforme, tanto em termos de memória quanto de tempo de execução, especialmente para problemas mais complexos. A pesquisa em Espaço de Estados consegue obter soluções com o mesmo comprimento (L), mas com uma penetração muito mais alta (P), indicando uma procura mais vantajosa e mais em conta.

Referências Webgráficas

ia2024-25T7.pdf

https://blog.csdn.net/StreamlineWq/article/details/136537733

https://blog.csdn.net/bob595078694/article/details/123362253

https://www.geeksforgeeks.org/heuristic-function-in-ai/