Universidade Federal de Juiz de Fora

Departamento de Ciência da Computação

Inteligência Artificial

Régua-Puzzle: Uma Análise de Diferentes Algoritmos de Busca

Grupo 11

Ana Beatriz Kapps - 201835006

André Luiz dos Reis - 201965004AC

Felipe Israel de Oliveira Vidal - 201835041

Relatório de trabalho prático apresentado ao Professor Saulo Moraes Villela como requisito de avaliação da Disciplina DCC014 - Inteligência Artificial

Juiz de Fora

Setembro de 2021

Relatório Técnico

Ana Beatriz Kapps [1]
André Luiz dos Reis [1]
Felipe Israel de Oliveira Vidal [1]

1 Introdução

O presente trabalho tem como temática central descrever o uso de diferentes algoritmos de busca para encontrar um solução vencedora para o jogo Régua-puzzle. Para tal, foram desenvolvidos sete algoritmos:Backtracking, busca em Largura, Busca em Profundidade(Limitada), Busca Ordenada, Busca Gulosa, Busca A* e Busca IDA*. Esses foram avaliados ante um conjunto de instâncias. Os resultados obtidos, assim como a heurística utilizada, são alvo de discussão nesse texto.

A realização desse experimento se deu como forma de consolidar e aplicar os conceitos teóricos vistos durante a disciplina DCC014 - Inteligência Artificial durante o primeiro semestre de 2021 no modelo de Ensino Remoto Emergencial (ERE). Além de expor os discentes a situações-problemas que requerem abstração e a busca por soluções otimizadas e eficientes, tais fatos são altamente presentes no mercado de trabalho e em diversas linhas de pesquisas existentes.

Diante disso, na seção 2 descrevendo formalmente o problema e a estrutura e regras do jogo; na seção 3 mostrando a abordagem propostas e as heurpisticas utilizadas, enquanto a seção 4 apresenta os experimentos realizados e uma análise comparativa dos resultados obtidos. Por último, a seção 5 apresenta as conclusões do trabalho e propostas de trabalhos futuros.

[1]Departamento de Ciência da Computação – Instituto de Ciências Exatas
Universidade Federal de Juiz de Fora (UFJF)
Rua José Kelmer, s/n, Campus Universitário, Bairro São Pedro
CEP: 36036-900 – Juiz de Fora/MG – Brasil
{anabeatriz.kapps, andre.reis, felipe.vidal}@estudante.ufjf.br

2 Descrição do problema

Existem N blocos brancos(W), N blocos pretos(B) e uma posição vazia (para representação da posição vazia foi usado um traço) em uma régua alinhados com $2N + 1$ posições de blocos. Essas posições na régua variam$[1, \cdots, 2N + 1]$. O objetivo do jogo é colocar todos os blocos brancos do lado esquerdo dos blocos pretos.

A solução ótima para esse jogo é onde possui o menor custo, ou seja o menor número de movimentos possíveis.

Blocos podem pular para a posição vazia quando a posição vazia estiver distante de no máximo N casas da posição do bloco. Desta maneira, existem no máximo 2N movimentos legais (no caso do vazio estar exatamente no meio da régua). O custo de um pulo é igual à distância entre a posição do bloco e a posição vazia. Definimos a distância entre duas posições i e j, como $j - i$, sendo $0 < i < j \leq 2N + 1$.

Um exemplo de estados meta possíveis para solução do seguinte caso com N=2:

W	B	-	B	W

São:

W	W	B	-	B

W	W	-	B	B

W	-	W	B	B

-	W	W	B	B

W	W	B	B	-

3 Abordagens para o problema

Frente ao problema intratável outrora apresentado, e considerando a constante procura por custo-eficiência do atual modelo econômico (podendo ser melhor entendida como qualidade-tempo no contexto de programação), buscou-se por algoritmos que atendessem as expectativas projetadas.

Conforme relatado em [1], é preciso achar uma solução ótimo e algoritmos de busca podem ser interessantes. Deste modo, implementou-se sete algoritmos de busca capazes de fornecer propostas satisfatórias. Os detalhes e especificações dessas abordagens são alvo das discussões seguintes.

Nas seguintes seções, são destacadas as metodologias e escolhas feitas durante a execução do trabalho, junto com uma breve apresentação das estruturas de dados utilizadas.

3.1 Método Não Informados de Busca

Os Métodos Não Informados de Busca são estruturalmente independentes do domínio que exploram uma grande quantidade de alternativas, por isso são conhecidos também como métodos fracos.

Em geral, esses métodos utilizam pouca memória, são exaustivos e não possuem função de avaliação.

Estes métodos atuam unicamente sobre o modelo em grafo de estados e implementam uma forma organizada e sistemática de se realizar uma busca nos mesmos.

Os principais métodos não informados são:

- Irrevogável
- Backtracking
- Busca em Largura
- Busca em Profundidade
- Busca Ordenada

Os métodos *Irrevogável* e *Backtracking* são os mais simples dessa família, gerando apenas um filho por vez.

3.2 Backtracking

O algoritmo Backtracking tem a possibilidade de voltar ao caminho (estado) anterior (pai, em geral), sempre que o processo de busca se encontrar em situação de impasse, e tenta-se outro caminho.

Dessa forma, o algoritmo pode terminar com fracasso, se não houver solução (caminho da raiz ao vértice solução), já que todas as possibilidades de caminho são examinadas. Ou o algoritmo pode terminar com sucesso, se houver solução.

A estratégia utilizada para implementação desse algoritmo foi alternando a troca do bloco com a posição vazia, de forma uniforme para os dois lado, começando pela esquerda.

Toda a lógica utilizada na implementação pode ser visualizada através do pseudocódigo ilustrado na Figura 1 foi usado como base.

Figura 1 – Pseudocódigo que foi usado como base para a implementação

```
Início
   S := estado inicial;
   N := S; Fracasso := F; Sucesso := F;
   Enquanto não (Sucesso ou Fracasso) faça
      Se R(N) <> vazio então
         Selecione o operador r de R(N);
         N := r(N);
         Se N é solução então
            Sucesso := T;
         Fim-se;
      Senão
         Se N = S então
            Fracasso := T;
         Senão
            N := pai(N);
         Fim-se;
      Fim-se;
   Fim-enquanto;
Fim.
```

Fonte: Notas Aulas DCC014, 2021

3.3 Busca em Largura e Profundidade(Limitada)

Nos métodos de Busca em Largura e Profundidade, todos os filhos de cada vértice são gerados de uma só vez e colocados na árvore. Na árvore de busca, pode-se distinguir dois tipos de vértices:

Abertos: que são os vértices instalados na árvore que não foram explorados, ou seja,

cujos filhos ainda não foram obtidos.

Fechados: que são os vértices instalados na árvore já explorados, isto é, cujos filhos também estão instalados na árvore.

A busca em largura e em profundidade diferencia-se na seleção dos vértices, na busca em largura o vértice selecionado é o que está a mais tempo na lista de abertos, se comporta como fila. Já na busca em profundidade é selecionado o vértice instalado mais recentemente na lista, se comporta como pilha.

Toda a lógica utilizada na implementação pode ser visualizada através do pseudocódigo ilustrado na Figura 2 foi usado como base.

Figura 2 – Pseudocódigo que foi usado como base para a implementação

```
Início
   Defina(abertos); S := raiz; Fracasso := F; Sucesso := F;
   Insere(S, abertos); Defina(fechados);
   Enquanto não (Sucesso ou Fracasso) faça
      Se abertos = vazio então Fracasso := T;
      Senão
         N := Primeiro(abertos); {Pilha(topo), Fila(primeiro)}
         Se N = solução então Sucesso := T;
         Senão
            Enquanto R(N) <> vazio faça
               Escolha r de R(N); New(u); u := r(N);
               Insere(u, abertos);
               Atualiza R(N);
            Fim-enquanto;
            Insere(N, fechados);
         Fim-se;
      Fim-se;
   Fim-enquanto;
Fim.
```

Profundidade

Largura

Fonte: Notas Aulas DCC014, 2021

Cabe destacar que nos algoritmos de busca em largura e profundidade implementos, o algoritmo sempre verifica se o estado gerado é solução.

Na busca em profundidade, após diversos testes realizados a seguinte estratégia de limitação foi usada: $10 \times qtd\_blocos$

3.4 Busca Ordenada

O algoritmo de busca ordenada seleciona o vértice N em Abertos tal que o custo do caminho da raiz até N seja mínimo. Uma lista de abertos foi implementada, onde a sua ordenação é realizada no final do algoritmo.

Toda a lógica utilizada na implementação pode ser visualizada através do pseudocódigo ilustrado pela Figura 3 que foi usado como motivação.

Figura 3 – Pseudocódigo que foi usado como base para a implementação

```
Início
   Defina(abertos); S := raiz; Calcule(f(S)); Fracasso := F;
   Insere(S, abertos); Defina(fechados); Sucesso := F;
   Enquanto não (Sucesso ou Fracasso) faça
      Se abertos = vazio então Fracasso := T;
      Senão
         N := Primeiro(abertos); /*nó com o menor f*/
         Se N = solução então Sucesso := T;
         Senão
            Enquanto R(N) <> vazio faça
               Escolha r de R(N); New(u); u := r(N);
               Calcule(f(u)); Insere(u, abertos);
               Atualiza R(N);
            Fim-enquanto;
            Insere(N, fechados);
         Fim-se;
      Fim-se;                          Busca Ordenada: f(n) = g(n);
   Fim-enquanto;                         Busca Gulosa: f(n) = h(n);
Fim.                                        Busca A*: f(n) = g(n) + h(n);
```

Fonte: Notas Aulas DCC014, 2021

3.5 Método Informados de Busca

Os algoritmos deste métodos utilizam a estratégia de busca pela melhor escolha. Em linhas gerais, um nó n é selecionado para ser explorado baseado em uma função que avalia a probabilidade deste pertencer ao caminho ótimo. Usualmente denotamos a função de avaliação por $f(n)$, qual mede a distância de n até o nó objetivo.

Entretanto, nem sempre somos capazes (ou é eficiente) de calcular $f(n)$ propriamente dita. Assim, fundamentamos uma função heurística $h(n)$ que informa o custo estimado do menor caminho de n até o vértice/estado objetivo. Os próximos algoritmos são considerados como membros deste método.

3.6 Busca Gulosa

Em suma, a busca gulosa visa a expansão do nó mais próximo à meta, assumindo apenas a função heurístico. Matematicamente falando, temos que $f(n) = h(n)$.

Por sua semelhança com a busca em profundidade, a busca gulosa apresente os mesmo defeitos: nem sempre a solução ótima é encontrada, caso exista mais de uma solução; e pode entrar em um caminho de *loop* circular, nunca explorando novos caminhos.

3.7 Busca A*

A busca A* avalia nó com menor valor de avaliação entre todos os abertos, com o intuito de encontrar a solução de menor custo. A busca encontra a solução ótima caso a heurística $h(n)$ for admissível.

Diferente da busca Gulosa, a busca A* considera o custo até o nó n, além do custo do nó atual até o objetivo. Isto é, $f(n) = g(n) + h(n)$, sendo $g(n)$ o custo do caminho inicial até o nó n, e o restante já definido anteriormente.

3.8 Busca IDA*

A busca IDA* veio para adaptar a busca A* com o intuito de reduzir o uso de memória por meio de um aprofundamento iterativo. Utiliza a mesma função $f(n)$ da A*, mas realiza cortes por um patamar, expandidos somente os vértices cujo $f(n) \leq$ patamar. Os vértices com $f(n) \geq$ patamar são descartados na iteração, e o menor valor δ de $f(n)$ dos vértices descartados é guardado.

O valor do patamar inicial é o valor de f da raiz. Caso o algoritmo não encontre solução, uma nova iteração com patamar $= \delta$ é iniciada, sendo o processo repetido até que não haja mais δ para atualização inicial.

A busca IDA* controí uma árvore de busca similar ao algoritmo *backtracking*, e tende, assintoticamente, ao tempo de processamento do A*, caso a heurística $h(n)$ seja consistente.

3.9 Principais Estruturas e Classes

A classe principal utilizada foi a State, ela é utilizada por todas as outras classes para o gerenciamento de todos os estados.

Outra classe utilizada foi a GameState, onde ele controla o status do jogo: Sucess, Fail ou Playing.

Foi criada também, uma classe para cada algoritmo, onde sua função é executar o mesmo.

3.10 Principais Métodos e Funções

As principais funções são: generate_initial_state que gera o estado iniciai, generate_final_states onde gera uma lista de estados finais. Foi implementada uma função também que verifica se o estado é a solução do problema ou não.

3.11 Metodologia e Organização

Frente ao singular tempo disponibilizado para a execução do trabalho, e um desconhecimento prévio de algumas estruturas utilizadas e heurísticas para o jogo, optou-se por uma metodologia de *per-programming*.

A fim de potencializar nosso tempo, utilizou-se o *Google Calendar* para organização das reuniões iniciais e, como a ferramenta permite, o link para as sala de reunião do *Google Meet*, bem como a disponibilização da gravação de todas as reuniões para potenciais dúvidas. Junto a isso, utilizou-se a ferramente *Trello* para a separação das tarefas e sub-tarefas, baseando-se em modelo de metodologias ágeis, além de possibilitar uma visão em tempo real para toda a equipe do andamento das atividades.

Ante o exposto, definiu-se a linguagem Python para ser utilizada no trabalho, realizou-se reuniões iniciais para o entendimento do problema e as relações entre as diferentes classes do projeto. Após isso, as tarefas foram realizadas de maneira conjunta a fim de maximizar o tempo e entediamento de todos os envolvimentos.

Ao final, foram realizados verificações e testes cruzados por todos os integrantes do grupo, além da confecção do presente relatório.

3.12 Dificuldades enfrentadas

Notou-se notária dificuldade em abordagens sobre o problema, foi notória a falta de metodologias presentes na literatura que pudessem servir como base para esse trabalho.

Junto a isso, a criação das Heurísticas foi outra dificuldade encontrada, visto que não foram encontradas muitas abordagens, então foi preciso realizar muitos testes para encontrar uma heurística ideal para os algoritmos.

Por fim, devido às dificuldades relatadas acimas, não foi possível finalizar a execução deste trabalho, faltando a implementação e análise do algoritmo IDA*.

4 Experimentos computacionais

Frente a necessidade de validar a abordagem desenvolvida, os algoritmos foram submetidos a diversos testes. Detalhes dos experimentos e ponderações sobre os resultados são apresentados a seguir.

4.1 Descrição das instâncias

Como base para os testes, utilizamos 6 (seis) instâncias. Suas descrições podem ser vistas na tabela 1, a qual apresenta o tamanho n, além de destacar o estado inicial.

Tabela 1 – Descrição das instâncias utilizadas

Instância	tamanho(N)	estado inicial
I3	3	BBB-WWW
I4	4	BBBB-WWWW
I5	5	BBBBB-WWWWW
I6	6	BBBBBB-WWWWWW
I7	7	BBBBBBB-WWWWWWW

Fonte: Os Autores, 2021

4.2 Ambiente computacional do experimento e conjunto de parâmetros

Buscando minimizar as interferências externas nos experimentos, todos os experimentos foram realizados utilizando a linguagem Python, através do interpretador Python 3.9.2 64-bit numa máquina com o sistema operacional macOS Big Sur, com processador Processardor 2,3 GHz Intel Core i5 Dual-Core e 8GB de memória RAM 2133 MHz LPDDR3.

Foram realizados testes com variações do tamanho n do tabuleiros, sendo $n \in \{3\,, \cdots\,, 6\}$.

4.3 Resultados quanto à qualidade e tempo

Após a execução das heurísticas de construção, obtivemos os resultados para análise.

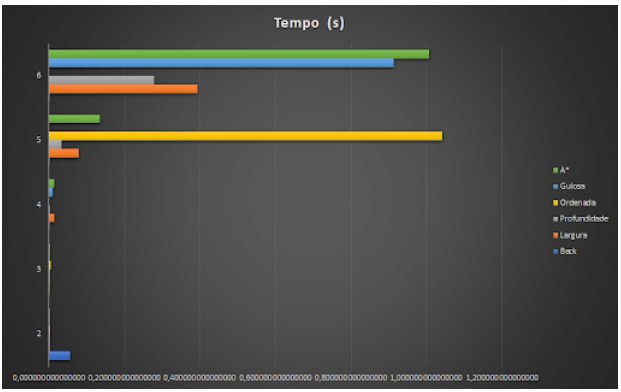
Tabela 2 – Tempo de Execução dos Algoritmos

N	Back	Largura	Profundidade	Ordenada	Gulosa	A*
2	0,055748956874000	0,000362873077393	0,000254869461060	0,000544071197510	0,000377893447876	0,000329017639160
3	0,000159978866577	0,002395153045654	0,000890254974365	0,004603862762451	0,001885175704956	0,002121925354004
4	0,000212907791138	0,013889074325562	0,001880168914795	0,06612396240234375	0,010345935821533	0,014183044433594
5	0,000155210494995	0,079065084457397	0,032135963439941	1,041545867919920	0,088775873184204 1	0,134177923202514
6	0,000246763229370	0,394245147705078	0,278784036636352	*	0,913479804992675	1,007909154891960

Fonte: Os Autores, 2021

A Tabela 2 exibe os melhores resultados (coluna 2) encontrados pelos algoritmos de busca enquanto que as demais colunas (3 a 8) apresentam os respectivos desvios percentuais em relação a estes resultados. A Figura 4 mostra uma comparação gráfica dos resultados, é possível notar um ótimo tempo para tabuleiros menores, e a discrepância de tempo para a Busca Ordenada.

Figura 4 – Comparação Gráfica do Tempo de Execução dos Algoritmos



Fonte: Os Autores, 2020

Os resultados inerentes à quantidade de nós visitados por cada algoritmos são ilustrados pela tabela 3 e pela figura 5.

Tabela 3 – Desvio Percentual Médio do Tempo

N	Back	Largura	Profundidade	Ordenada	Gulosa	A*
2	4	22	12	30	13	6
3	6	123	36	256	60	61
4	8	599	81	1688	277	373
5	10	2716	2123	9760	1259	1967
6	12	11912	10691	51954	5544	9845

Fonte: Os Autores, 2020

É possível notar uma relação diretamente proporcional entre a quantidade de nós visitados e o tamanho n do tabuleiro.

Tabela 4 – Iterações Médias

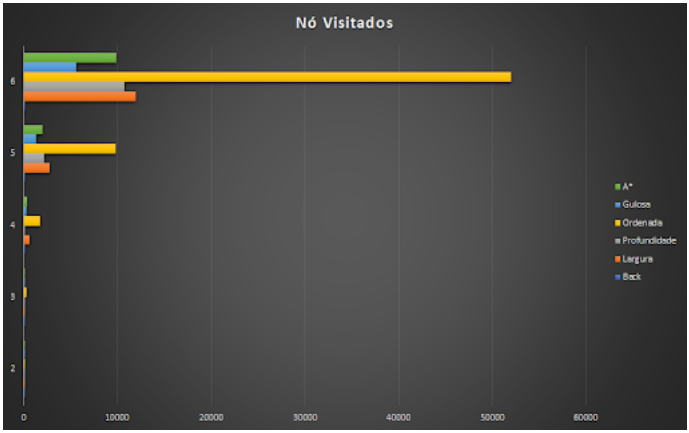
N	Back	Largura	Profundidade	Ordenada	Gulosa	A*
2	4	25	22	34	16	16
3	6	133	80	279	78	79
4	8	621	173	1759	348	477
5	10	2761	2260	9943	1510	2292
6	12	11999	10883	52383	6497	10843

Fonte: Os Autores, 2020

Por fim, tabela 4 e pela figura 6 ilustram a quantidade de nós gerados em cada iteração

por cada algoritmo.

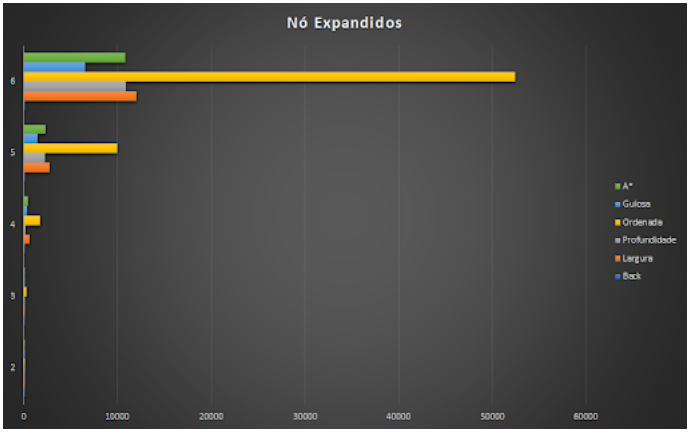
Figura 5 – Distribuição dos desvios percentuais do tempo da solução para cada instância analisada.



Fonte: Os Autores, 2020

É possível notar que a quantidade de nós gerados é próximo a quantidade de nós visitados. Esse fato comprova um bom controle de memória na geração dos estados a serem explorados.

Figura 6 – Distribuição dos desvios percentuais do tempo da solução para cada instância analisada.



Fonte: Os Autores, 2020

Por fim, analisou-se também o fator de ramificação média e a profundidade. Entretanto, tais métricas não serão avaliadas por esse relatório, mas podem ser analisadas no arquivo

output.txt no projeto.

5 Conclusão e Trabalhos Futuros

De modo geral, ao compararmos os resultados obtidos, podemos afirmar que o trabalho realizado alcançou resultados satisfatórios. Ademais, podemos concluir que o algoritmo *Backtracking* apresentou soluções de valor ótimo, por conta do estado inicial balanceado. Porém os algoritmos de ordenação e A* demonstraram valores insatisfatórios e com maior tempo de processamento, inviabilizando seu uso nesse contexto.

O uso de uma heurística que não garante a solução ótima geraria um tempo de processamento maior, o que inviabilizaria o uso de tal metodologia, haja vista a necessidade de desenvolvimento de tecnologias capazes de fornecer soluções em um curto espaço de tempo e com custos minimizados.

Findando, espera-se realizar o aperfeiçoamento das técnicas e heurísticas utilizadas, além de conhecer outras metodologias presentes na literatura. Com isso, será possível realizar trabalhos futuros com o fito de buscar propostas de soluções mais eficiências, tanto em custo operacional, bem como em tempo de processamento, para a problemática abordada.

Referências

[1] LELIANE. MAC 415 Exercício Programa 1 Busca - IME-USP. Disponível em: https://www.ime.usp.br/~leliane/IAcurso2006/slides/EP1-2006.pdf