Relatório do Software Sistema de Busca em Espaço de Estados

Raphael Victor (RA: 189362)

Henrique Gusman (RA: 191234)

Vinicius Haas (RA: 189815)

Resumo

Este trabalho apresenta o desenvolvimento de um sistema computacional interativo voltado para a resolução de problemas clássicos de Inteligência Artificial por meio de algoritmos de busca em espaço de estados. Utilizando a linguagem Python e bibliotecas como Tkinter para interface gráfica, o sistema permite ao usuário resolver dois problemas clássicos: o 8-Puzzle e o problema dos Missionários e Canibais, aplicando os algoritmos de Busca em Largura (BFS) e Busca A* (A-Star). O software oferece funcionalidades avançadas como configuração de estados iniciais personalizados, monitoramento em tempo real do progresso da busca, cancelamento de execução, comparação de desempenho entre algoritmos e visualização detalhada dos caminhos de solução. As métricas coletadas incluem tempo de execução, memória utilizada, número de nós expandidos e comprimento da solução. Os testes realizados demonstraram a robustez, eficiência e usabilidade da ferramenta, que se mostrou eficaz na promoção do entendimento prático dos algoritmos de busca em Inteligência Artificial. A proposta contribui para o processo de ensino-aprendizagem ao oferecer uma interface intuitiva e recursos de análise que facilitam a compreensão dos conceitos envolvidos.

Introdução

A busca em espaço de estados é um dos pilares fundamentais da Inteligência Artificial, fornecendo uma abordagem sistemática para resolver problemas que podem ser modelados como um conjunto de estados e transições entre eles. Formalmente, um problema de busca em espaço de estados pode ser definido por uma quádrupla (S, s_o, T, G), onde S é o conjunto de todos os estados possíveis, s_o é o estado inicial, T é a função de transição que define os estados sucessores, e G é o conjunto de estados objetivo.

A importância dos algoritmos de busca se destaca pela sua aplicabilidade em diversas áreas, incluindo planejamento automático, jogos, robótica, sistemas de navegação, resolução de quebra-cabeças e otimização. Esses algoritmos variam em estratégia, desde buscas não informadas (cegas) que exploram o espaço sem conhecimento adicional, até buscas informadas (heurísticas) que utilizam estimativas para guiar a exploração de forma mais eficiente.

No contexto educacional, o estudo de algoritmos de busca promove o desenvolvimento do raciocínio algorítmico, da capacidade de análise de complexidade

e da compreensão de trade-offs entre tempo e espaço. É fundamental para disciplinas como Inteligência Artificial, Análise de Algoritmos e Estruturas de Dados. No entanto, muitos estudantes enfrentam dificuldades em compreender o comportamento dinâmico desses algoritmos apenas por meio de descrições teóricas.

Este trabalho implementa dois problemas clássicos da literatura: o 8-Puzzle, um quebra-cabeça deslizante que requer planejamento e heurísticas eficientes para ser resolvido; e o problema dos Missionários e Canibais, que explora restrições lógicas e busca de caminhos válidos. Ambos os problemas são resolvidos através de dois algoritmos fundamentais: a Busca em Largura (BFS), que garante encontrar a solução ótima em termos de número de passos, e o algoritmo A*, que utiliza heurísticas para encontrar soluções ótimas de forma mais eficiente.

Dessa forma, este trabalho visa desenvolver uma ferramenta interativa para aplicação e análise de algoritmos de busca, permitindo ao usuário configurar problemas, executar buscas com monitoramento em tempo real, comparar o desempenho de diferentes estratégias e visualizar detalhadamente os caminhos de solução encontrados. A proposta tem como objetivo principal facilitar a compreensão prática dos conceitos de busca em Inteligência Artificial por meio de uma interface gráfica intuitiva, promovendo uma aprendizagem mais efetiva e experimental dos fundamentos teóricos.

Estrutura do Software

1. Escolha das Bibliotecas

Para viabilizar o desenvolvimento do sistema proposto, foram selecionadas bibliotecas que oferecem suporte eficiente à construção de interfaces gráficas, gerenciamento de estruturas de dados e monitoramento de recursos computacionais. A linguagem Python foi escolhida por sua clareza sintática, ampla adoção acadêmica e rico ecossistema de bibliotecas.

A biblioteca **Tkinter** foi utilizada para a construção da interface gráfica, fornecendo widgets nativos para entrada de dados, exibição de resultados e controles interativos. Sua integração nativa com Python facilita o desenvolvimento de aplicações desktop multiplataforma sem dependências externas complexas.

O módulo **tracemalloc** foi empregado para o monitoramento preciso do consumo de memória durante a execução dos algoritmos, permitindo a coleta de métricas detalhadas sobre o uso de recursos. O módulo **time** foi utilizado para medir o tempo de execução de cada algoritmo.

Para o gerenciamento de estruturas de dados, foram utilizadas as implementações nativas do Python: **Queue** para a Busca em Largura e **heapq** para a implementação da fila de prioridade do algoritmo A*. O módulo **threading** foi empregado para executar as

buscas em threads separadas, mantendo a interface responsiva durante operações longas.

Adicionalmente, foram utilizadas classes de dados (**dataclass**) para estruturação das métricas de desempenho e **copy.deepcopy** para garantir a independência de estados durante comparações de algoritmos.

2. Modelagem dos Problemas

2.1. 8-Puzzle

O 8-Puzzle é um quebra-cabeça clássico composto por uma grade 3×3 contendo 8 peças numeradas de 1 a 8 e um espaço vazio. O objetivo é reorganizar as peças a partir de uma configuração inicial até alcançar o estado objetivo, onde os números estão ordenados sequencialmente e o espaço vazio ocupa a posição inferior direita.

A classe **PuzzleState** encapsula toda a lógica necessária para representar e manipular estados do puzzle:

- **Representação do Estado:** Uma matriz 3×3 armazena a configuração atual do tabuleiro, onde o valor 0 representa o espaço vazio.
- **Heurística de Manhattan:** Implementa o cálculo da distância de Manhattan como função heurística, somando as distâncias que cada peça precisa percorrer para alcançar sua posição objetivo.
- **Geração de Sucessores:** O método get_neighbors() gera todos os estados válidos alcançáveis movendo o espaço vazio em uma das quatro direções possíveis (cima, baixo, esquerda, direita).
- **Custos:** Mantém os valores g (custo do caminho), h (heurística) e f (g + h) para uso no algoritmo A*.

2.2. Missionários e Canibais

O problema dos Missionários e Canibais é um quebra-cabeça lógico clássico onde três missionários e três canibais precisam atravessar um rio usando um barco que comporta no máximo duas pessoas. A restrição crítica é que, em nenhuma margem, os canibais podem superar numericamente os missionários, caso contrário os missionários serão devorados.

A classe **MissionaryState** modela este problema:

- **Representação do Estado:** Armazena o número de missionários e canibais na margem esquerda, além da posição do barco.

- **Validação de Estados:** O método is_valid() verifica se um estado satisfaz todas as restrições do problema, garantindo que os canibais nunca superem os missionários em nenhuma margem.
- **Movimentos Possíveis:** Implementa cinco tipos de movimentos: 1M, 2M, 1C, 2C e 1M1C, gerando estados sucessores válidos.
- **Heurística Simples:** Utiliza o número total de pessoas na margem esquerda como estimativa heurística.

3. Implementação dos Algoritmos de Busca

3.1. Busca em Largura (BFS)

A Busca em Largura é um algoritmo de busca não informada que explora o espaço de estados em níveis, garantindo encontrar a solução com menor número de passos (solução ótima para grafos não ponderados). A implementação utiliza uma fila FIFO (First In, First Out) para manter a ordem de exploração dos estados.

Características da implementação:

- **Estrutura de Dados:** Utiliza Queue do módulo queue para gerenciar a fronteira de exploração.
- **Controle de Visitados:** Mantém um conjunto (set) de estados já visitados para evitar ciclos e reexpansão.
- **Garantia de Otimalidade:** Por explorar em níveis, garante encontrar o caminho com menor número de ações.
- **Limite de Nós:** Implementa um limite configurável para evitar consumo excessivo de recursos em problemas difíceis.

3.2. Busca A* (A-Star)

O algoritmo A* é uma busca informada que utiliza heurísticas para guiar a exploração, priorizando estados mais promissores. Combina o custo real do caminho (g) com uma estimativa heurística até o objetivo (h) através da função de avaliação f = g + h.

Características da implementação:

- **Fila de Prioridade:** Utiliza heapq para implementar uma fila de prioridade mínima, ordenando estados por f(n).
- **Heurísticas Admissíveis:** Implementa a distância de Manhattan para o 8-Puzzle, garantindo admissibilidade e otimalidade.
- **Eficiência:** Explora significativamente menos nós que BFS em problemas com boas heurísticas.

- **Desempate:** Utiliza id(estado) como segundo critério de ordenação para garantir comparabilidade consistente.

4. Funcionalidades Avançadas

4.1. Cancelamento de Busca

O sistema implementa um mecanismo robusto de cancelamento que permite ao usuário interromper buscas longas sem encerrar a aplicação. Utiliza threading.Event como flag de cancelamento, verificado a cada iteração do algoritmo. Quando acionado, a busca é interrompida graciosamente, retornando métricas parciais da execução.

4.2. Monitoramento de Progresso

Uma barra de progresso visual informa ao usuário o andamento da busca em tempo real. A cada 100 nós expandidos, a interface é atualizada mostrando a porcentagem de progresso em relação ao limite configurado. Isso proporciona feedback contínuo e permite ao usuário avaliar se deve aguardar ou cancelar a operação.

4.3. Coleta de Métricas

O sistema coleta métricas detalhadas de cada execução através da classe SearchMetrics:

- **Tempo de Execução:** Medido com precisão usando time.time().
- **Memória Utilizada:** Monitorada através de tracemalloc, reportada em megabytes.
- **Nós Expandidos:** Conta o número total de estados explorados.
- **Comprimento da Solução:** Número de passos no caminho encontrado.
- **Status da Busca:** Indica se a solução foi encontrada, não foi encontrada ou foi cancelada.

4.4. Comparação de Algoritmos

A funcionalidade de comparação executa ambos os algoritmos no mesmo problema e apresenta uma tabela comparativa lado a lado. Isso permite ao usuário compreender empiricamente as diferenças de desempenho entre estratégias de busca informada e não informada.

5. Interface Gráfica e Usabilidade

A interface gráfica foi projetada seguindo princípios de design centrado no usuário, organizando as funcionalidades em seções lógicas e distintas:

5.1. Seção de Configuração

Permite selecionar o problema (8-Puzzle ou Missionários e Canibais), o algoritmo (A* ou BFS) e configurar o limite máximo de nós a serem expandidos. A interface adapta-se dinamicamente ao problema selecionado, habilitando ou desabilitando campos conforme necessário.

5.2. Entrada de Estado Inicial

Para o 8-Puzzle, oferece uma grade 3×3 de campos de entrada onde o usuário pode configurar manualmente o estado inicial. O sistema valida a entrada, garantindo que todos os números de 0 a 8 sejam usados exatamente uma vez. Para Missionários e Canibais, o estado inicial é fixo (3M, 3C na margem esquerda).

5.3. Controles de Execução

Botões claramente identificados permitem executar a busca, cancelar durante a execução, comparar algoritmos e limpar os resultados. O estado dos botões é gerenciado dinamicamente para prevenir ações inválidas.

5.4. Visualização de Resultados

Uma área de texto scrollável e somente leitura exibe os resultados de forma organizada, incluindo métricas de desempenho e o caminho completo da solução com representação visual de cada estado. A formatação clara facilita a análise dos resultados.

6. Tratamento de Erros e Robustez

O sistema implementa múltiplas camadas de validação e tratamento de erros:

- **Validação de Entrada:** Verifica se os valores inseridos são numéricos e estão no intervalo válido.
- **Verificação de Completude:** Garante que todos os números necessários estão presentes no 8-Puzzle.
- **Prevenção de Execução Concorrente:** Impede múltiplas buscas simultâneas que poderiam comprometer os resultados.
- **Mensagens Informativas:** Fornece feedback claro sobre erros, orientando o usuário na correção.
- **Limite de Recursos:** Protege contra consumo excessivo de memória e tempo em problemas intratáveis.

Resultados

Durante a fase de testes, o software demonstrou robustez e eficiência na resolução dos problemas propostos. Foram realizados testes extensivos com diferentes configurações

de estados iniciais para ambos os problemas, variando o nível de dificuldade e a distância até a solução.

Testes do 8-Puzzle

Para o 8-Puzzle, foram testadas configurações desde estados triviais (já resolvidos ou próximos da solução) até estados que requerem mais de 20 movimentos para serem solucionados. Os resultados mostraram que:

- O algoritmo A* consistentemente expandiu significativamente menos nós que BFS, com reduções frequentemente superiores a 80% em problemas complexos.
- A heurística de Manhattan provou-se eficaz, guiando a busca diretamente em direção à solução.
- Em estados iniciais com 15+ movimentos de distância, BFS frequentemente atingiu o limite de nós antes de encontrar a solução, enquanto A* conseguiu resolver o problema.
- O tempo de execução do A* foi consistentemente inferior, mesmo considerando o overhead do cálculo heurístico.
- O consumo de memória também foi menor no A*, devido ao menor número de estados mantidos na fronteira.

Testes do Problema dos Missionários e Canibais

Para este problema, que possui uma solução conhecida de 11 passos, os testes revelaram:

- Ambos os algoritmos encontraram a solução ótima de forma consistente.
- BFS expandiu aproximadamente 25-30 nós até encontrar a solução.
- A* expandiu cerca de 15-20 nós, mostrando eficiência mesmo com uma heurística simples.
- O tempo de execução foi mínimo para ambos (< 0.01 segundos), dado o espaço de estados relativamente pequeno.
- A diferença de desempenho foi menos pronunciada que no 8-Puzzle, refletindo a simplicidade do espaço de busca.

Funcionalidades da Interface

Os testes de usabilidade confirmaram que:

- A validação de entrada funcionou corretamente, rejeitando configurações inválidas com mensagens claras.

- O cancelamento de busca operou de forma responsiva, interrompendo execuções longas em menos de 1 segundo.
- A barra de progresso atualizou-se suavemente, fornecendo feedback visual adequado.
- A comparação de algoritmos executou ambas as buscas sequencialmente sem problemas, apresentando resultados lado a lado de forma clara.
- A interface permaneceu responsiva durante todas as operações, graças ao uso de threading.
- A área de resultados apresentou os caminhos de solução de forma legível e organizada.

Análise de Desempenho

As métricas coletadas permitiram análises quantitativas precisas:

- **Eficiência Temporal:** A* mostrou-se 3 a 10 vezes mais rápido que BFS em problemas complexos do 8-Puzzle.
- **Eficiência Espacial:** A redução no número de nós expandidos traduziu-se diretamente em menor consumo de memória.
- **Otimalidade:** Ambos os algoritmos garantiram soluções ótimas quando aplicáveis.
- **Escalabilidade:** O sistema lidou bem com diferentes níveis de complexidade, graciosamente reportando quando limites foram atingidos.

Discussão

Os resultados obtidos estão em conformidade com a teoria de algoritmos de busca em Inteligência Artificial, validando empiricamente os conceitos estudados na literatura. A superioridade consistente do A* sobre BFS em problemas com boas heurísticas confirma a importância do uso de informação específica do domínio para guiar a busca.

Eficácia das Heurísticas

A heurística de Manhattan para o 8-Puzzle demonstrou ser altamente eficaz, sendo admissível (nunca superestima o custo real) e consistente (satisfaz a desigualdade triangular). Isso garantiu tanto a otimalidade quanto a eficiência do A*. A diferença dramática no número de nós expandidos ilustra como uma boa heurística pode transformar um problema intratável em um problema facilmente resolvível.

Para o problema dos Missionários e Canibais, a heurística simples (número de pessoas na margem esquerda) mostrou-se menos informativa, resultando em ganhos mais

modestos. Isso sugere que o desenvolvimento de heurísticas melhores poderia trazer benefícios adicionais, embora o problema já seja facilmente resolvível.

Trade-offs entre Algoritmos

A comparação direta entre BFS e A* revelou trade-offs importantes:

- **BFS:** Simples de implementar, não requer conhecimento do domínio, garante otimalidade, mas pode ser ineficiente em espaços de busca grandes.
- **A*:** Mais eficiente quando boas heurísticas estão disponíveis, requer desenvolvimento de funções heurísticas específicas do domínio, tem overhead adicional de cálculo heurístico.

Aspectos de Implementação

A decisão de implementar cancelamento via threading e flags demonstrou-se acertada, proporcionando uma experiência de usuário superior. A alternativa de bloquear a interface durante buscas longas seria inaceitável do ponto de vista de usabilidade.

O uso de tracemalloc para monitoramento de memória forneceu métricas precisas que seriam difíceis de obter de outra forma. Isso permitiu análises quantitativas rigorosas do consumo de recursos.

Valor Educacional

Do ponto de vista pedagógico, o sistema cumpre seu objetivo de tornar os conceitos abstratos de busca em Inteligência Artificial tangíveis e experimentáveis. A capacidade de configurar problemas, executar buscas e visualizar resultados passo a passo facilita a compreensão de como os algoritmos realmente funcionam.

A funcionalidade de comparação direta entre algoritmos é particularmente valiosa, permitindo que estudantes vejam empiricamente as diferenças de desempenho e compreendam quando cada abordagem é mais apropriada.

Limitações e Considerações

Algumas limitações devem ser reconhecidas:

- O sistema é limitado a dois problemas específicos, embora a arquitetura seja extensível.
- Para o 8-Puzzle, nem todas as configurações iniciais são solucionáveis (apenas metade das permutações possíveis), mas o sistema não detecta isso automaticamente.
- O limite de nós, embora configurável, é uma proteção necessária mas pode impedir a resolução de alguns casos muito complexos.

- A visualização é textual; uma representação gráfica animada poderia ser mais intuitiva.

Conclusão

O desenvolvimento do Sistema de Busca em Espaço de Estados alcançou plenamente os objetivos propostos, oferecendo uma ferramenta interativa, robusta e educacionalmente valiosa para o estudo prático de algoritmos de busca em Inteligência Artificial. A implementação dos algoritmos BFS e A* para resolver os problemas clássicos do 8-Puzzle e Missionários e Canibais demonstrou tanto correção teórica quanto eficiência prática.

As funcionalidades avançadas implementadas - incluindo cancelamento de busca, monitoramento de progresso, coleta detalhada de métricas e comparação direta de algoritmos - elevam o sistema além de uma simples implementação acadêmica, tornando-o uma ferramenta genuinamente útil para experimentação e aprendizado.

A validação empírica dos conceitos teóricos através de testes extensivos confirma que o sistema não apenas funciona corretamente, mas também fornece insights valiosos sobre o comportamento e desempenho relativo dos diferentes algoritmos. A superioridade demonstrada do A* em problemas com boas heurísticas ilustra claramente a importância da busca informada.

Do ponto de vista educacional, o sistema preenche uma lacuna importante entre teoria e prática, permitindo que estudantes experimentem com algoritmos de busca de forma interativa e obtenham feedback imediato sobre suas decisões de configuração. A interface intuitiva e a apresentação clara dos resultados facilitam a compreensão sem sacrificar o rigor técnico.

Trabalhos Futuros

O sistema estabelece uma base sólida para extensões futuras, incluindo:

- **Novos Problemas:** Implementação de outros problemas clássicos como Torre de Hanói, N-Rainhas, ou problemas de planejamento mais complexos.
- **Novos Algoritmos:** Adição de variantes como IDA*, RBFS, ou algoritmos de busca local como Hill Climbing e Simulated Annealing.
- **Visualização Gráfica:** Desenvolvimento de animações visuais que mostrem dinamicamente a exploração do espaço de estados.
- **Análise de Heurísticas:** Ferramentas para comparar diferentes funções heurísticas e visualizar seu impacto.
- **Detecção de Solubilidade:** Implementação de verificação prévia de se o 8-Puzzle é solucionável.

- **Exportação de Dados:** Capacidade de exportar métricas e resultados para análise posterior em ferramentas estatísticas.
- **Modo Tutorial:** Sistema de tutoriais interativos que guia o usuário através dos conceitos fundamentais.

Em suma, este projeto demonstra que é possível criar ferramentas educacionais que sejam simultaneamente rigorosas, eficientes e acessíveis, contribuindo significativamente para o processo de ensino-aprendizagem em Inteligência Artificial.

Tempo Gasto

Estima-se que o desenvolvimento do projeto foi realizado em cerca de 3 semanas de trabalho, divididas entre pesquisa, desenvolvimento, testes e documentação.

Distribuição do Tempo:

- **Pesquisa e Estudo Teórico:** ~8 horas
- **Implementação dos Algoritmos:** ~15 horas
- **Desenvolvimento da Interface Gráfica:** ~10 horas
- **Implementação de Threading e Cancelamento:** ~6 horas
- **Sistema de Métricas e Monitoramento:** ~5 horas
- **Testes e Debugging:** ~8 horas
- **Refinamento e Otimização:** ~4 horas
- **Documentação:** ~6 horas

Total Estimado: ~62 horas de trabalho

Referências

- RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 4th Edition. Pearson, 2020.
- HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. **IEEE Transactions on Systems Science and Cybernetics**, v. 4, n. 2, p. 100-107, 1968.
- KORF, R. E. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. **Artificial Intelligence**, v. 27, n. 1, p. 97-109, 1985.
- PEARL, J. **Heuristics: Intelligent Search Strategies for Computer Problem Solving**. Addison-Wesley, 1984.

- PYTHON SOFTWARE FOUNDATION. **Tkinter — Python interface to Tcl/Tk**. Disponível em: https://docs.python.org/3/library/tkinter.html. Acesso em: nov. 2025.
- PYTHON SOFTWARE FOUNDATION. **tracemalloc — Trace memory allocations**. Disponível em: https://docs.python.org/3/library/tracemalloc.html. Acesso em: nov. 2025.
- PYTHON SOFTWARE FOUNDATION. **heapq — Heap queue algorithm**. Disponível em: https://docs.python.org/3/library/heapq.html. Acesso em: nov. 2025.
- PYTHON SOFTWARE FOUNDATION. **threading — Thread-based parallelism**. Disponível em: https://docs.python.org/3/library/threading.html. Acesso em: nov. 2025.
- RATNER, D.; WARMUTH, M. K. The $(n^2 - 1)$-puzzle and related relocation problems. **Journal of Symbolic Computation**, v. 10, n. 2, p. 111-137, 1990.
- AMAREL, S. On representations of problems of reasoning about actions. **Machine Intelligence**, v. 3, p. 131-171, 1968.
- PYTHON SOFTWARE FOUNDATION. **Python 3 Documentation**. Disponível em: https://docs.python.org/3/. Acesso em: nov. 2025.