Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DCC642 - Introdução à Inteligência Artificial 2021/2
Professor: Luiz Chaimowicz
Monitor: Marcelo Lemos

Trabalho Prático 1: Algoritmos de Busca no Pac-Man

Valor: 20 pontos
Data de entrega: 17 de Dezembro de 2021

Detalhes do problema

O objetivo deste trabalho é praticar os conceitos e algoritmos de busca em espaço de estados e desenvolver a habilidade de análise comparativa de algoritmos. Para tal, disponibilizamos no Moodle da disciplina o código-fonte de uma plataforma de experimentação de algoritmos de busca no jogo Pac-Man. Para este trabalho, será necessário conhecimento básico de Python 3[1] e uso de terminal ou prompt de comando.

Em um arquivo específico deste código-fonte, você deverá implementar algoritmos de busca sem informação (**em profundidade, em largura e de custo uniforme**) e com informação (**gulosa e A***) em seus respectivos métodos, que estarão inicialmente vazios. Você então deverá escolher um conjunto de fases do Pac-Man dentre as fases fornecidas (ou criar novas) para comparar os algoritmos implementados, observando o número de nós expandidos por eles, seu tempo de execução, otimalidade da solução e quaisquer outras métricas relevantes. Você também deverá propor e implementar uma heurística admissível e consistente para o A* que minimize o número de nós expandidos em uma fase específica. Por último, você deverá escrever um relatório descrevendo os algoritmos implementados, decisões tomadas e suas justificativas, assim como a análise comparativa dos algoritmos e a heurística proposta.

A seguir, cada um destes passos será explicado em mais detalhes. **Leia com atenção. :)**

Passo 1: Familiarizar-se com o código-fonte do Pac-Man

Vá ao Moodle da disciplina, e faça o download do arquivo .zip "Código-fonte Inicial", na seção "Trabalho Prático 1: Busca em Espaço de Estados". Em seguida, extraia seu conteúdo em uma pasta de sua preferência. Dentre os arquivos extraídos, haverão diversos arquivos Python (.py) e duas pastas. Não é necessário entender o conteúdo de todos os arquivos (mas se quiser pode). Os arquivos e pastas que importam são:

- O arquivo `search.py`, onde você fará a implementação dos algoritmos de busca e da heurística para o A*. **Este é o único arquivo que você deve alterar.**
- O arquivo `util.py`, que contém as estruturas de dados Stack (pilha), Queue (fila) e PriorityQueue (fila de prioridades ou *heap*). **Você deve usar estas estruturas de dados na implementação dos algoritmos de busca**.
- A pasta `layouts`, que contém 37 fases do Pac-Man dentre as quais você pode escolher para comparar os algoritmos de busca.

[1] Um guia básico da linguagem está disponível em http://df.python.org.br/pycubator/index.html.

- O script `pacman.py`, usado para executar partidas do Pac-Man. Use `python pacman.py` para jogar uma partida de Pac-Man usando as setas do teclado. Para trocar a fase, use `python pacman.py --layout fase`, onde `fase` é o nome de um arquivo na pasta `layouts`. Para usar um agente de busca para jogar a fase, use `python pacman.py --layout fase --pacman SearchAgent --agentArgs fn=agente`, onde `agente` $\in$ {dfs, bfs, ucs, gs, astar}. Para especificar uma heurística (caso use a busca gulosa ou A*), use `python pacman.py --layout fase --pacman SearchAgent --agentArgs fn=agente,heuristic=heuristica`, onde `heuristica` é o nome de uma função no arquivo `search.py`, como `nullHeuristic`. Por fim, para ver todos os parâmetros disponíveis no script, use `python pacman.py --help`.
- O script `autograder.py`, usado para verificar a corretude da implementação dos algoritmos de busca, assim como sua pontuação nos passos 2 a 6 deste trabalho. Use `python autograder.py` para verificar todos estes passos ao mesmo tempo ou `python autograder --question passoX` para verificar o passo `X`, onde `X` $\in \{2, 3, 4, 5, 6\}$.

Para completar o **Passo 1**, você deverá entender como executar partidas do Pac-Man em fases diferentes usando algoritmos diferentes e autoavaliar sua implementação. Todos os comandos exemplificados acima devem executar sem erros, apresentando o resultado esperado. Apesar de não valer pontos, este passo é essencial para que você consiga completar os passos a seguir. Em caso de dúvidas, não hesite em usar o fórum "Dúvidas / Discussão" do Moodle da disciplina ou contatar o monitor.

Passo 2: (2 pontos) Implementar a busca em profundidade (DFS)

Para completar o **Passo 2**, você deverá implemetar o método `depthFirstSearch` no arquivo `search.py`. O método recebe como entrada um parâmetro chamado `problem`, da classe `SearchProblem`, que contém informações sobre o problema que você vai precisar para implementar o algoritmo (os métodos disponíveis nesta classe podem ser vistos na linha 22 até a 62). O método deve retornar uma lista de ações (use as ações fornecidas pela classe `SearchProblem`). **Importante: sua implementação deste algoritmo deve usar a versão busca em grafo**, que evita a expansão de estados já visitados.

Alguns exemplos de testes para seu DFS são:

```
python pacman.py --layout tinyMaze --pacman SearchAgent --agentArgs fn=dfs

python pacman.py --layout mediumMaze --pacman SearchAgent --agentArgs fn=dfs

python pacman.py --layout bigMaze --zoom 0.5 --pacman SearchAgent --agentArgs
                                    fn=dfs
```

Ao executar um destes comandos, o Pac-Man aparecerá jogando a fase de acordo com a solução dada por seu DFS. A fase mostrará os estados que foram explorados pelo algoritmo e sua ordem de exploração (estados em vermelho mais intenso foram explorados primeiro). No terminal, o comando exibirá o custo total da solução, o tempo usado pelo algoritmo, a quantidade de nós expandidos e a pontuação do Pac-Man. Se seu DFS demorar vários segundos para encontrar uma solução para estas fases, desconfie da sua implementação. De qualquer forma, para testar se sua implementação está correta, utilize o *autograder*:

```
python autograder.py --question passo2
```

Ele fará alguns testes e, caso todos resultem em `PASS`, sua implementação está correta e você completou o **Passo 2**. Se algum resultar em `FAIL`, há algo errado e ele mostrará o que é. Ele também mostrará quantos pontos você ganhou neste passo (2/2 se estiver tudo certo, ou 0/2 caso contrário). Os passos 3 até 5 serão bastante similares a este, e apenas as diferenças serão explicadas.

Passo 3: (2 pontos) Implementar a busca em largura (BFS)

Para completar o **Passo 3**, você deverá implemetar o método `breadthFirstSearch` no arquivo `search.py`. O método recebe como entrada um parâmetro chamado `problem`, da classe `SearchProblem`, e deve retornar uma lista de ações. **Importante: termine a busca ao *expandir* um estado objetivo, e não ao *gerar* um estado objetivo (ou seja, não use o *early goal test)***. Um exemplo de teste para seu BFS é:

```
python pacman.py --layout mediumMaze --pacman SearchAgent --agentArgs fn=bfs
```

Seu BFS deve encontrar a solução ótima para esta fase. Teste sua corretude com `python autograder.py --question passo3`.

Passo 4: (2 pontos) Implementar a busca de custo uniforme (UCS)

Para completar o **Passo 4**, você deverá implementar o método `uniformCostSearch` no arquivo `search.py`. O método recebe como entrada um parâmetro chamado `problem`, da classe `SearchProblem`, e deve retornar uma lista de ações. Um exemplo de teste para seu UCS é:

```
python pacman.py --layout mediumMaze --pacman SearchAgent --agentArgs fn=ucs
```

Seu UCS também deve encontrar a solução ótima para esta fase. Teste sua corretude com `python autograder.py --question passo4`.

Passo 5: (2 pontos) Implementar a busca gulosa (best first search)

Para completar o **Passo 5**, você deverá implementar o método `greedySearch` no arquivo `search.py`. O método recebe como entrada um parâmetro chamado `problem`, da classe `SearchProblem`, e um chamado `heuristic`, que é uma função que recebe um estado e o problema, e retorna a heurística para aquele estado. O método deve retornar uma lista de ações. Um exemplo de teste para sua busca gulosa é:

```
python pacman.py --layout mediumMaze --pacman SearchAgent
    --agentArgs fn=gs,heuristic=manhattanHeuristic
```

Dependendo da forma que sua busca gulosa for implementada, ela pode não encontrar a solução ótima para esta fase. Teste sua corretude com `python autograder.py --question passo5`.

Passo 6: (2 pontos) Implementar a busca A*

Para completar o **Passo 6**, você deverá implementar o método `aStarSearch` no arquivo `search.py`. O método recebe como entrada um parâmetro chamado `problem`, da classe `SearchProblem`, e um chamado `heuristic`, que é uma função que recebe um estado e o problema, e retorna a heurística para aquele estado. O método deve retornar uma lista de ações. Um exemplo de teste para seu A* é:

```
python pacman.py --layout mediumMaze --pacman SearchAgent
    --agentArgs fn=astar,heuristic=manhattanHeuristic
```

Seu A* também deve encontrar a solução ótima para esta fase. Teste sua corretude com `python autograder.py --question passo6`.

Passo 7: (2 pontos) Implementar uma nova heurística para o A*

Nos passos anteriores, o exemplo de teste fornecido sempre resulta em execuções rápidas dos algoritmos. No entanto, em algumas fases (ou problemas) nem mesmo o A* consegue terminar a busca de forma rápida. Nesses casos, o uso de uma heurística melhor é uma forma possível para diminuir o tempo de execução. Um exemplo é a fase `trickySearch` com o problema `FoodSearchProblem`, que requer que o Pac-Man obtenha não apenas um ponto de comida, mas vários (e é, portanto, uma fase difícil).

Para completar o **Passo 7**, você deverá propor uma heurística para o A* e a implementar no método `foodHeuristic` no arquivo `search.py`. O método recebe como entrada um parâmetro chamado `state`, que é estado para o qual a heurística será calculada e um parâmetro chamado `problem`, da classe `SearchProblem`.

Sua heurística deve reduzir ao máximo o número de nós expandidos especificamente na fase `trickySearch`. Além disso, sua heurística deve ser **admissível** e **consistente**. Para testar sua heurística, use:

```
python pacman.py --layout trickySearch --pacman SearchAgent --agentArgs
    fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic.
```

Ou, o comando equivalente:

```
python pacman.py --layout trickySearch --pacman AStarFoodSearchAgent.
```

Qualquer heurística válida (admissível e consistente) lhe dará 1 ponto neste passo. Você ganhará mais pontos de acordo com o número de nós que sua heurística expande para resolver a fase `trickySearch`:

Número de nós expandidos	**Pontuação no passo 7**
10001 ou mais	1/2 ponto
Entre 10000 e 6999	2/2 pontos
7000 ou menos	3/2 pontos (**ponto extra!**)

Para base de comparação, a heurística nula (o que deixa o A* equivalente ao UCS) expande mais de 16000 nós. Se sua heurística não for admissível e consistente, você receberá **zero** pontos. Dica: sua heurística resolve a fase `mediumSearch` de forma rápida? Se sim, ou sua heurística é (i) absurdamente boa; ou (ii) inconsistente.

Passo 8: (8 pontos) Escrever um relatório (em PDF)

Para completar o **Passo 8**, você deverá escrever um relatório no formato PDF. O relatório deverá seguir o modelo de trabalhos acadêmicos da SBC (que pode ser encontrado online[2,3]).

Para o relatório, você deverá escolher um conjunto de fases do Pac-Man dentre as fases fornecidas ou criar novas fases que destaquem as diferenças entre os algoritmos. Então, você deverá utilizar os algoritmos para resolvê-las,[4] observando o número de nós expandidos por eles, seu tempo de execução, otimalidade da solução e quaisquer outras métricas relevantes. Você deverá analisar os resultados obtidos e justificá-los. Por último, você também precisará descrever sua heurística proposta para o passo 7, como a pensou e implementou, seus resultados e por que ela é admissível e consistente. Em suma, o relatório deverá conter **todos** os seguintes tópicos:

- **Capa** - Especifique o título do trabalho, nome e número de matrícula da autora ou do autor.
- **Introdução** - Qual é o contexto do trabalho? Qual é o problema abordado e sua importância? Quais algoritmos são utilizados? Como foi, em linhas gerais, sua solução?
- **Metodologia e Análise Experimental** - Como planejou comparar os algoritmos e por quê? Quais métricas? Quais fases? Qual(is) computador(es) utilizou para os experimentos? Insira as comparações dos algoritmos. (Tabelas? Gráficos?) Qual é a interpretação dos resultados? Por que os resultados foram esses? Qual foi a heurística proposta para o Passo 7? Como chegou nela? Ela é admissível e consistente? Qual foi seu resultado?
- **Conclusão** - Resuma o conteúdo do seu trabalho. Quais suas conclusões gerais sobre os resultados? Encontrou dificuldades durante a realização do trabalho? Este trabalho lhe ajudou a praticar os conceitos e algoritmos de busca? Possui sugestões para futuros trabalhos?
- **Bibliografia** - Especifique as fontes consultadas para realização do seu trabalho (por exemplo: livros, páginas da Internet, slides da disciplina).

Entrega

Você deverá entregar um arquivo ZIP no Moodle da disciplina, no item "Entrega" da seção "Trabalho Prático 1: Busca em Espaço de Estados". O arquivo ZIP deve conter (i) Um arquivo Python `search.py` com a sua implementação dos algoritmos e da heurística, conforme pedido nos passos 2 a 7; e (ii) um arquivo PDF `relatorio.pdf` contendo seu relatório, conforme pedido no passo 8. Você deverá efetuar a entrega até o dia **17 de Dezembro de 2021** às **23:59**. Após o prazo, serão descontados $2^d - 1$ pontos, onde d é o número de dias de atraso arredondado para cima.

[2] https://www.overleaf.com/latex/templates/sbc-conferences-template/blbxwjwzdngr

[3] http://www.sbc.org.br/documentos-da-sbc/summary/169-templates-para-artigos-e-capitulos-de-livros/878-modelosparapublicaodeartigos

[4] Sinta-se livre para utilizar o problema de "Encontrar o único ponto de comida" (problema padrão), o problema de "encontrar todos os pontos de comida" (especificando `prob=FoodSearchProblem` como no passo 7) ou ambos.

Considerações finais

- Sinta-se livre para discutir o trabalho com seus colegas, mas o compartilhamento de código ou texto é plágio e será devidamente punido. A implementação dos algoritmos e heurística deve ser de sua autoria.
- Em caso de dúvidas, não hesite em perguntar na seção `Dúvidas / Discussão` do Moodle da disciplina (sua dúvida pode ser a dúvida de outras pessoas) ou procurar o monitor da disciplina via e-mail (marcelolemos@dcc.ufmg.br).
- Segundo dados do MECM (Minha Experiência Como Monitor), os alunos que começam a desenvolver o trabalho mais cedo obtém melhores resultados. Comece cedo, se possível!

Bom trabalho!