Primeiro Trabalho de Inteligência Artificial - 2017.1
Busca Custo Uniforme e A* (Opção 1)

Instruções de envio do trabalho e relatórios

- Todo o código fonte usado no trabalho deve ser enviado como tarefa no Classroom, até o dia 10/05, por apenas um dos componentes do grupo (que deve especificar o nome de todos os demais na tarefa).
- No mesmo email deve ser enviado o relatório do trabalho, indicando, entre outros, detalhes da implementação que não estejam claras no código.
- Pode ser usada qualquer linguagem de programação para a implementação do trabalho, mas a responsabilidade de execução correta é do grupo. Caso a execução necessite de algum pacote adicional, deve ser indicado no relatório.
- Deve ser enviado um relatório adicional, individual, contendo a sua participação no trabalho, com detalhes, e o quanto você acha que os demais componentes contribuíram para o trabalho. Envie como sua própria tarefa no Classroom.
- Valor : 0 - 10. A nota levará em consideração o código implementado, incluindo o quão eficiente e elegante está a sua implementação, bem como cada escolha que você fará no seu trabalho. Será dada atenção especial a implementação do trabalho do ponto de vista de modelagem em IA. Obviamente, a nota também levará em consideração a corretude e completude do trabalho.
- O grupo deverá ser formado de 2 a 5 alunos.
- Referências: esse trabalho foi adaptado do trabalho do curso CSC384 da Universidade de Toronto, professor Bacchus.

Especificação

Você deverá resolver um problema simplificado de entregas e empacotamento em um depósito, modelado a partir do problema bem mais complexo que os robôs da Amazon devem enfrentar todos os dias. Veja como funciona o problema real em https://www.youtube.com/watch?v=6KRjuuEVEZs
O depósito tem vários produtos, um conjunto de estações de empacotamento e um conjunto de robôs. Os produtos e as estações de empacotamento estão em localizações fixas espalhadas pelo depósito. Os robôs podem se mover pelo depósito. Um problema típico neste domínio é operar os robôs para que eles completem um conjunto de pedidos. Cada pedido requer que um produto em particular seja movido para um estação de empacotamento. Um robô deve ser associado a esse trabalho e ele se moverá para a localização do produto, pegará o produto e então se moverá para a estação de empacotamento para entregar o produto. Depois da entrega, o robô permanecerá na estação de empacotamento, até que ele seja associado a um outro trabalho. O problema é resolvido

quando todos os pedidos forem associados aos robôs e todos os robôs tenham completado todos os trabalhos para os quais eles foram associados. A qualidade da solução é medida de acordo com o tempo que a entrega final é realizada. Ou seja, uma solução melhor entrega os pedidos para as estações de empacotamento mais rápido. O problema é simplificado em diversas maneiras, comparado com a aplicação real. Algumas dessas simplificações incluem: o robô só pode carregar um produto de cada vez; cada pedido consiste de apenas um produto; não nos preocupamos com as ações envolvidas em pegar o produto e colocá-lo na estação de empacotamento—só pedimos para o robô pegar o produto em uma posição (x_1, y_1) e entregá-lo na posição (x_2, y_2), sem se preocupar com os passos intermediários.

Posições e movimentos do robô

Posições são especificadas usando um sistema de coordenadas simples *x-y*, em que *x e y* são inteiros, maiores ou iguais a zero. Os robôs podem ser mover entre quaisquer duas posições. Se ele está correntemente na posição (x_1, y_1) ele pode ir para a posição (x_2, y_2). Ele só se move na horizontal ou vertical e gasta uma unidade de tempo para se mover em qualquer direção, uma posição para a frente ou para trás, para a direita ou para a esquerda. Por exemplo, para se mover da posição (1,2) para (4,5), ele precisará de abs(1-4) + abs(2-5) = 6 unidades de tempo. Ou seja, ele precisa se mover +3 na direção x e +3 na direção y.

Concorrência

Quando temos vários robôs, temos que gerenciá-los executando entregas concorrentemente. Para conseguir fazer isso, os estados do **espaço de estados** incluirão (a) tempo corrente e (b) alguns eventos pendentes que acontecerão no futuro (mas apenas o futuro desse estado em particular - cada estado tem seus próprios eventos futuros). Isso permitirá que tenhamos um espaço de estados em que múltiplas atividades concorrentes estão acontecendo em qualquer estado. Todas essas atividades tem um fim em algum tempo no futuro. Os eventos futuros serão todos entregas completas de diferentes robôs, tal que temos um estado onde um grupo de robôs estão trabalhando concorrentemente em diferentes entregas. Cada entrega estará completa em algum tempo (diferente) no futuro. Para gerenciar tais estados, temos dois tipos diferentes de **ações:** (1) ações que iniciam atividades e atualizam o estado para incluir a informação sobre o tempo futuro quando estas atividades estarão completas. Esse tipo de ação gera um novo estado que terá o mesmo tempo que o estado corrente. Ou seja, esse tipo de ação não move o tempo para a frente e custa zero (uma vez que estamos tentando encontrar uma solução para completá-la no tempo mais cedo possível); (2) uma ação simples que move o tempo para a frente. Esta ação opera em um estado movendo o tempo para a frente, para o instante de tempo em que o evento futuro mais próximo (temporalmente) estará terminado. Estes eventos envolvem robôs terminando suas entregas. Essas ações custam uma quantidade igual a quantidade de tempo que foi movido.

Exemplo

Suponha que tenhamos os produtos abaixo em suas respectivas posições:

```
[['prod1', (0,5)], ['prod2', (1,5)], ['prod3', (2,5)]]
```

e as estações de empacotamento nas suas posições:

```
[['pack1', (0,0)], ['pack2', (1,0)]]]
```

e a seguinte lista de pedidos para ser concluída:

```
[['order2', 'prod3', 'pack1'], ['order3', 'prod2', 'pack2'],['order4',
'prod3', 'pack1']]
```

Como queremos entregar dois items de prod3 para a estação de empacotamento 1, no nosso domínio simplificado, isso deve ser feito com dois pedidos distintos.
Finalmente, temos dois robôs. Para especificar os robôs e capturar o fato que o robô poderia estar trabalhando correntemente em uma entrega, precisamos fornecer informações sobre o status de cada robô.

```
[['r1', 'idle', (0,0)], ['r2', 'on_delivery', (1,0), 8]]
```

Aqui, o robô r1 está correntemente ocioso e parado na posição (0,0). O robô r2 está em uma entrega. Não sabemos sua localização exata (e nem precisamos saber), mas sabemos que depois da sua entrega ele estará posicionado na coordenada (1,0) e que gastará 8 unidades de tempo para terminar a entrega. Finalmente, o tempo corrente no estado é 0.

Ações
Nesse exemplo, temos 4 possíveis ações: pegar o robô ocioso e colocá-lo para trabalhar no pedido 2 (*order2*); no pedido 3; ou no pedido 4; ou podemos mover o tempo para completar a entrega pendente mais próxima.

Ação 1: Associar *r1* ao pedido *order2.* Para entregar esse pedido, r1 terá que se mover de sua posição corrente (0,0) para pegar o produto *prod3* que está em (2,5). Então ele terá que se mover de (2,5) para a estação de empacotamento *pack1,* localizada em (0,0). Isso significa que a entrega será executada em 14 unidades de tempo a partir do instante corrente, que é 0, e ao final r1 estará na posição (0,0). O novo estado gerado por essa ação terá a lista de entregas atualizada, onde *order2* foi removida:

```
[['order3', 'prod2', 'pack2'], ['order4', 'prod3', 'pack1']]
```

O novo estado também terá uma atualização no status do robô r1, tal que sua informação se tornará:
[['r1', 'on_delivery, (0,0), 14], ['r2', 'on_delivery', (1,0), 8]]
Observe que r2 não é afetado por essa ação.
2. Podemos associar *r1* ao pedido *order3*. Essa ação atualiza a lista de pedidos e o status de r1.

3. Podemos associar *r1* ao pedido *order4*. Essa ação atualiza a lista de pedidos e o status de r1.

4. Podemos mover o tempo para a frente, para completar a entrega mais próxima (temporalmente). Nesse exemplo, temos apenas uma entrega em progresso, que será concluída no instante de tempo 8. Então essa ação não muda a lista de pedidos, mas muda o status do robô:
[[’r1’, ’idle’, (0,0)], [’r2’, ’idle’, (1,0)]]

Ou seja, r2 se torna ocioso na posição em que a entrega foi feita. Além disso, o tempo nesse novo estado moveu para 8. Uma vez que começamos no instante de tempo 0, o custo dessa ação é 8 (as outras ações tiveram custo 0, conforme especificado anteriormente).

Uma ação de mover no tempo é executada para mover o tempo para a frente para o instante mais próximo possível. Pode acontecer que mais de uma entrega seja concluída em tal instante de tempo. Nesse caso, o status de todos os robôs que tiveram ações concluídas devem ser atualizados. Por exemplo, suponha que tenhamos o seguinte status para os robôs:

```
[[’r1’, ’on_delivery’, (0,0), 14], [’r2’, ’on_delivery’, (1,0), 8], [’r3’, ’on_delivery’, (0,5), 8]]
```

Uma ação de mover no tempo (e observe que essa é a única possível aqui, visto que não temos robôs ociosos) irá para o instante de tempo 8 e atualizará o status dos robôs r2 e r3. O novo status será:

```
[[’r1’, ’on_delivery’, (0,0), 14], [’r2’, ’idle’, (1,0)], [’r3’, ’idle’, (0,5)]]
```

O que fazer:

- Implementar uma representação de estado para esse problema, considerando todos os elementos necessários para construir a árvore de busca. Isso inclui o instante de tempo, a lista de pedidos não atendidos ainda, e o status de cada robô.
- Implementar o modelo de transição e as ações. Lembre que temos dois tipos de ações: deliver(<robot-name>, <product-name>,<packing_station-name>), que deve remover o pedido e atualizar o status dos robôs e custam zero; e move_forward(newTime), que cria um estado sucessor onde o tempo é atualizado para newTime e todos os robôs com entregas agendadas para newTime são atualizados (se tornam ociosos). O custo dessa ação é o incremento de tempo (newTime - tempo corrente).
- Implementar uma função de inicialização de estado.
- Implementar uma forma de acessar os dados do estado e os demais objetos no depósito.
- Implementar uma função heurística para esse domínio.
- Implementar a busca de custo uniforme e a busca A* para resolver esse problema. Ao final, deve ser exibida a solução para o problema (o caminho na árvore de busca do estado inicial para o objetivo). O objetivo é que a lista de pedidos esteja vazia e os robôs ociosos, ou seja todos os produtos foram entregues para a estação de empacotamento.

Teste Possível:

[['prod1', (0,0)], ['prod2', (0,10)], ['prod3', (0, 20)]],

[['pack1', (20,0)], ['pack2', (20,10)], ['pack3', (20,20)]],

0,

[['prod1', 'pack1'], ['prod2', 'pack2'], ['prod3', 'pack3']],

[['r1', 'on_delivery', (20,10), 14],

['r2', 'on_delivery', (20,20), 8],

['r3', 'on_delivery', (20,0), 8]]