Segundo Exercício Programa (EP2)

Busca Heurística para Planejamento

Data de entrega: 23/04/2015

Introdução

Considere o problema dos robôs dado em sala de aula.

Table 1Descrição PDDL para o domínio e problema do robô entregador de pacotes.

```
;; PDDL Domain Specification  File
 (define (domain robot)
  (:requirements :strips :equality :typing)
  (:types room box arm)
  (:constants left right - arm)
  (:predicates
    (robot-at ?x - room)
    (box-at ?x - box ?y - room)
    (free ?x - arm)
    (carry ?x - box ?y - arm)
  )
  (:action move
    :parameters (?x ?y - room)
    :precondition (robot-at ?x)
    :effect (and (robot-at ?y) (not (robot-at ?x)))
  )
  (:action pickup
    :parameters (?x - box ?y - arm ?w - room)
    :precondition (and (free ?y) (robot-at ?w) (box-at ?x ?w))
    :effect (and (carry ?x ?y) (not (box-at ?x ?w)) (not(free ?y)))
  )
  (:action putdown
    :parameters (?x - box ?y -arm ?w - room)
    :precondition (and (carry ?x ?y) (robot-at ?w))
    :effect (and (not(carry ?x ?y)) (box-at ?x ?w) (free ?y))
  )
)
```

```
;; PDDL Problem File
(define (problem robot1)
(:domain robot)
  (:objects
    room1 room2 - room
    box1 box2 box3 box4 - box
    left right - arm
  )
  (:init
    (robot-at room1)
    (box-at box1 room1)
    (box-at box2 room1)
    (box-at box3 room1)
    (box-at box4 room1)
    (free left)
    (free right)
  )
  (:goal
    (and
      (box-at box1 room2)
      (box-at box2 room2)
      (box-at box3 room2)
      (box-at box4 room2)
    )
  )
)
```

Planejamento como Busca no Espaço de Estados

A* Tree/Graph Search Algorithm

```
function aStarTreeSearch(problem, h)
fringe ← priorityQueue(new searchNode(problem.initialState))
allNodes ← hashTable(fringe)
loop
        if empty(fringe) then return failure
        node ← selectFrom(fringe)
        if problem.goalTest(node.state) then return pathTo(node)
        for successor in expand(problem, node)
           if not allNodes.contains(successor) then
              fringe ← fringe + successor   ;; add in priorityQueue  acording to evaluation function f
              allNodes.add(successor)
```

Figura 1. Algoritmo A* (Coursera).

Implementar o algoritmo de Busca Progressiva para problemas de planejamento. Você deve usar o algoritmo da busca A* em grafo do Coursera, conforme mostrado na Figura 2, porém modificando-o para que os métodos *problem.goalTest(node.state)* e *expand(problem, node*) raciocinem diretamente sobre a linguagem de ações. Para isso você deverá implementar os seguintes métodos (sendo A o conjunto dos operadores instanciados (ações)):

1. *goalTest(s)*): implemente o teste de meta verificando se o conjunto de literais da meta G esta contido no conjunto de literais do estado s;
2. *groundAllActions(Problem)*: que devolve o conjunto de todas as ações (operadores instanciados) do domínio, isto é, faz a proposicionalização de todas as ações (grounding);
3. *groundApplicableActions(Problem, s)*: que devolve um conjunto de ações (operadores instanciados) que podem ser aplicáveis para um estado s;
4. *matchApplicableActions(A, s):* dado o conjunto A de ações (operadores instanciados) esse método devolve o conjunto de ações aplicáveis em s;
5. *expand(a, s):* dada uma ação (proposicional) aplicável em s, gera os estados sucessores com a regra da progressão de ações STRIPS vista em sala de aula;

Além disso você deve implementar um parser para domínios e problemas especificados em PDDL considerando apenas os *requirements* do exemplo anterior.

Note que os dois métodos, *groundAllActions(Problem)* e *groundApplicableActions(s)*, são métodos alternativos para se fazer a instanciação dos operadores. Um dos objetivos desse EP é comparar o algoritmo de busca progressiva usando esses dois métodos, ou seja: (1) fazer a instanciação de todos os operadores antes da busca ou (2) fazer a instanciação para cada estado s visitado, isto é, instanciações sob demanda. Considere a heurística h=1 no A*, o que corresponde a uma estratégia de busca progressiva em largura (usaremos esse mesmo algoritmo com diferentes heurísticas no próximo EP).

A saída deve incluir:

- o plano,
- o tamanho do plano,
- o tempo em mili-segundos gasto pelo algoritmo,
- o número total de estados visitados (nós expandidas),
- o número total de estados gerados e
- o fator de ramificação médio da busca.

Para testar e fazer as comprações use o domínio e problema em PDDL do robô especificado na Tabela 1, para as 3 instâncias da Tabela 2 (especificar o estado inicial e meta para cada uma das instâncias de acordo com o exemplo da Tabela 1).

Table 2 Instâncias para teste.

(**:objects** room1 room2 - room box1 box2 - box left right - arm)	(**:objects** room1 room2 - room box1 box2 box3 box4 - box left right - arm)	(**:objects** room1 room2 - room box1 box2 box3 box4 box5 box6- box left right - arm)

O que você deve entregar

Entregar os arquivos fonte no Paca, bem como os arquivos de problemas em PDDL, e um relatório de no máximo 10 páginas**.** Cada equipe fará uma apresentação dos resultados.