MAC0425/5739 - Inteligência Artificial

Exercício-Programa 2 - Planejamento

Prazo limite de entrega no PACA: 21/05/2017 às 23:55

1 Introdução

Neste exercício-programa estudaremos a abordagem baseada em modelos para o problema de tomada de decisão sequencial em ambientes totalmente observáveis e determinísticos. Para isso desenvolveremos resolvedores automáticos de planejamento clássico baseados na linguagem STRIPS.

Os objetivos deste exercício-programa são:

(i) compreender a abordagem geral de problemas de planejamento clássico;

(ii) familiarizar-se com a linguagem de representação PDDL/STRIPS;

(iii) implementar um planejador automático baseado em busca heurística no espaço de estados;

(iv) implementar heurísticas independentes de domínio;

(v) comparar os resultados obtidos em problemas da competição de planejamento (IPC).

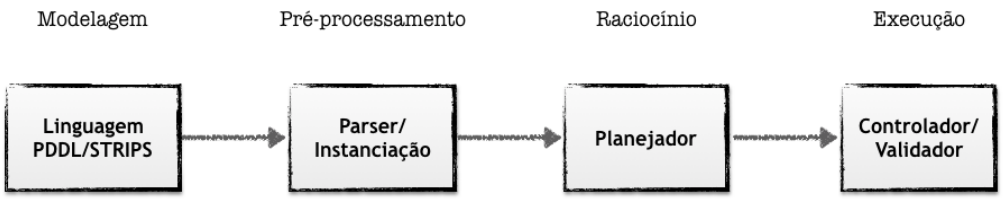

Figura 1: Esquema geral de um planejador clássico

1.1 Instalação

Para a realização deste EP será necessário ter instalado em sua máquina a **versão 3 do Python**.

Observação: você precisará ter instalado o **pacote ply** [1] que é necessário para automatizar a etapa de análise léxica-sintática (*parser*) dos arquivos de entrada PDDL/STRIPS.

Descompacte o arquivo **ep2.zip** e rode no diretório `pystrips/` o seguinte comando para testar a instalação:

```
$ python3 pystrips.py --help
$ python3 pystrips.py show pddl/robot/domain.pddl pddl/robot/boxes/problem02.pddl
```

[1] `http://www.dabeaz.com/ply/`

2 Planejamento clássico

Nesse exercício-programa você implementará um resolvedor automático para problemas de planejamento clássico. Os domínios e problemas de planejamento serão dados na linguagem PDDL/STRIPS conforme exemplos a seguir.

```
(define (domain robot)
    (:requirements :strips :equality :typing)

    (:types room box 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 (and (robot-at ?x) (not (= ?x ?y)))
        :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))))
```

```
(define (problem box02)
    (:domain robot)

    (:objects
        room1 room2 - room
        box1 box2 - box
        left right - arm)

    (:init
        (robot-at room1)
        (box-at box1 room1) (box-at box2 room1)
        (free left) (free right))

    (:goal
        (and (box-at box1 room2) (box-at box2 room2))))
```

3 Implementação

Arquivos que você precisará editar:

- **planner.py** onde os algoritmos de planejamento serão implementados;
- **heuristics.py** onde as heurísticas independentes de domínio serão implementadas;
- **validator.py** onde a rotina de validação de plano será implementada.

Arquivos que você precisará ler e entender:

- **domain.py** onde é definida a classe `Domain` que representa um domínio de planejamento;
- **problem.py** onde é definida a classe `Problem` que representa um problema de planejamento;
- **node.py** onde é definida a classe `Node` que representa um nó de problema de busca;
- **state.py** onde é definida a classe `State` que representa um estado do problema de planejamento;
- **util.py** onde estruturas de dados estão definidas para facilitar a implementação.

3.1 Parte I - Modelagem

A fim de se familiarizar com a modelagem de problemas de planejamento clássico na linguagem PDDL/STRIPS você deverá completar a especificação dos operadores STRIPS do domínio **gitplanner**[2] disponível no arquivo `pddl/gitplanner/domain.py`. Esse domínio é uma versão bastante simplificada (repositório com único *branch*, comandos simples sem opcões, *commit* por arquivo, ...) do *workflow* da ferramenta de controle de versão de código-aberto *git*.

Os detalhes do domínio gitplanner estão no próprio arquivo `domain.pddl` e nos arquivos de instâncias de problemas disponíveis no diretório `pddl/gitplanner/problems/`. Você deverá usar exclusivamente os predicados listados no arquivo `domain.pddl` na construção dos operadores.

```
(:predicates
  ;; file exists in workspace but has never been source-controlled
  (untracked ?f - file)

  ;; a file modification has been added to the staging area but not committed
  (staged ?f - file)

  ;; all changes were added to the staging area and then committed.
  (committed ?f - file)

  ;; already tracked file has no modifications in workspace
  (clean ?f - file)

  ;; file has been modified but this change has not been staged
  (modified-in-workspace ?f - file)

  ;; file has been deleted but this change has not been staged
  (deleted-in-workspace ?f - file))
```

[2]Esse exercício é baseado na proposta de problema disponível em `http://modelai.gettysburg.edu/2017/gitplanner/assignment.htm`. No entanto, note que para simplificar o problema diversas modificações foram feitas.

Você deverá completar as listas de precondições e efeitos dos seguintes operadores (que implementam os comandos em comentários logo acima):

```
;; git add <new-file>
(:action git-add-new ... )

;; git add <old-file>
(:action git-add ... )

;; git rm <old-file>
(:action git-rm ... )

;; git checkout -- <old-file>
(:action git-checkout ... )

;; git reset -- <old-file>
(:action git-reset ... )

;; git reset -- <new-file>
(:action git-reset-new ... )

;; git commit <staged-file>
(:action git-commit ... )
```

Observação: Se você não tem nenhuma experiência com a ferramenta *git* sugerimos os tutoriais *on-line* disponíveis em `https://learnxinyminutes.com/docs/git/` e `https://try.github.io/`.

Para facilitar essa parte do EP, você poderá utilizar o editor *on-line* de arquivos PDDL que está disponível em `http://editor.planning.domains/`. Uma vez modelado o domínio do problema, você poderá resolver cada problema usando o *solver* disponível no próprio site. O objetivo desse exercício é verificar se você entendeu a sintaxe e semântica dos operadores STRIPS. Esse conhecimento será necessário nas próximas partes do EP.

3.2 Parte II - Planejamento

Nessa parte do EP vamos implementar um **planejador automático baseado em busca heurística progressiva. Em particular, vocês deverá implementar a busca Weighted A* (WA*)** que é similar a busca informada A* mas com a função de avaliação de nós dada por: $f(n) = g(n) + W \cdot h(n)$, onde W é um parâmetro do algoritmo. Dessa forma, será possível ponderar a importância da heurística na avaliação dos nós da fronteira, o que por sua vez acabará influenciando no comportamento da busca. Note que se $W = 1$ o algoritmo se reduz ao A* tradicional, e se $W = 0$ o algoritmo se reduz a uma busca de custo uniforme.

Você deverá completar a definição das seguintes classes e funções:

- classe `ProgressionPlanning` no arquivo `planner.py`;
- funções heurísticas `h_add` e `h_max` no arquivo `heuristics.py`;
- função heurística `h_ff` (obrigatório **APENAS** para alunos da pós-graduação).

Para testar sua implementação separamos 3 conjuntos de problemas:

- `test0.sh` problemas triviais para *debug* e verificação de erros;
- `test1.sh` problemas fáceis (podem ser resolvidos sem heurística);
- `test2.sh` problemas difíceis (necessitam de heurísticas);
- `test3.sh` problemas BÔNUS

Para testar seu planejador, rode o seguinte comando (com os parâmetros desejados):

```
# ./test[0,1,2,3].sh <heuristic> <weight>
# <heuristic>: naive, max, add, ff
# <weight> heuristic weight (WA*)

# (0) debug
$ ./test0.sh naive 0
$ ./test0.sh max 1

# (1) busca não-informada
$ ./test1.sh naive 0

# (2) busca WA* com h_add
$ ./test2.sh add 1
$ ./test2.sh add 3

# (3) busca WA* com h_ff
$ ./test3.sh ff 1
$ ./test3.sh ff 3
```

Observação: você deverá conseguir resolver (pelo menos) todos os problemas to `test0.sh` e `test1.sh` sem heurística e também todos os problemas do `test2.sh` com heurística `h_add`.

3.3 Parte III - Validação

Nessa parte do EP vamos implementar um **validador automático de planos**. Complete a implementação da função `validate` no arquivo `validation.py`.

4 Relatório

Observação: a entrega do relatório é obrigatória APENAS para alunos de pós-graduação.

Após o desenvolvimento da parte prática, você deverá testar seus algoritmos e redigir um relatório claro e sucinto (máximo de 4 páginas). Assim, você deverá:

(a) compilar em tabelas os resultados obtidos para as diversas heurísticas implementadas em termos de **tempo de execução, tamanho da solução, numéro de nós explorados e visitados, e fator de ramificação médio**;

(b) discutir os méritos e desvantagens de cada heurística, no contexto dos dados obtidos;

(c) sugerir possíveis melhorias e relatar dificuldades.

5 Entrega

Você deve entregar um arquivo `EP2-NUSP-NOME-COMPLETO.zip` contendo todos os arquivos do projeto em Python3 juntamente com o relatório em PDF (para os alunos de pós-graduação).

Não esqueça de identificar cada arquivo com seu nome e número USP! No código coloque um cabeçalho em forma de comentário.

6 Critério de avaliação

6.1 Parte prática

O critério de avaliação será da seguinte forma:

- Parte I (Modelagem): 4 pontos
- Parte II (Planejamento): 20 pontos (graduação); 28 pontos (pós-graduação)
  - `./test0.sh naive 0`: 2 pontos
  - `./test1.sh naive 0`: 4 pontos
  - `./test1.sh add 3`: 6 pontos
  - `./test2.sh add 3`: 8 pontos
  - `./test2.sh ff 3`: 8 pontos (obrigatório **APENAS** para pós-graduação)
  - `./test3.sh`: 8 pontos (**BÔNUS** para todos os alunos)
- Parte III (Validação): 2 pontos

Observação: para os itens da Parte II você deverá ser capaz de resolver todos os problemas com as heurísticas pedidas para receber crédito total.

6.2 Parte teórica

Relatório (20 pontos)

- Compilação de Resultados: 5 pontos
- Comparação e Discussão: 15 pontos

Observação: O relatório será avaliado principalmente pela sua forma de interpretar comparativamente os desempenhos de cada heurística. Não é necessário discutir detalhes de implementação, mas deve ficar claro que você compreendeu os resultados obtidos conforme as características teóricas de cada heurística.