Third Practical Project

Inteligência Artificial - LEIC-2122SV

Group 12, Erasmus+

July 7th, 2022

1 First exercise

The program can be divided into three parts:

1.1 Puzzle-specific code:

```
% Example 1

% # # # # # o - Sokoban
% #       # x - Box
% #   x   # * - Goal
% # o   * #
% # # # # #

sokoban(a).
% ...

box(1).
% ...

state0([sokoban(a, 1, 1), box(1, 2, 2),
empty(2, 1), empty(3, 1),
empty(1, 2), empty(3, 2),
empty(1, 3), empty(2, 3), empty(3, 3)]).

solve1 :- state0(S), plan(S, [box(1, 3, 1)], P), show_pop(P).
```

1.2 POP planner:

```
%
% Figure 18.5 Partial order planning program.
%

% Partial Order Planner, using CLP(FD) and iterative deepening search
%
```

```
% Partially ordered plan = pop( Actions, OpenConditions, TrueConditions, FinishingTime)
%
% Actions = [ Action1:Time1, Action2:Time2, ...] Actions and their execution times
% OpenConditions = [ Cond1:Time1, Cond2:Time2, ...]
% TrueConds = [ Cond1:Time11/Time12, Cond2:Time21/Time22, ... ]
% Note: Ordering constraints are implemented as CLP(FD) constraints

:- use_module(library(clpfd)). % Load library for CLP(FD)

:- op(100, fx, ~). % Notation for negative effects of an action

% plan(StartState, Goals, Plan, Finish):
% Plan is partially ordered plan that achieves Goals from StartState at time Finish
%
plan(StartState, Goals, Plan) :-
    add_intervals(0, StartState, TrueConds, Finish),    % StartState true at time 0
    add_times(Finish, Goals, OpenConds),                % Goals should be true at time Finish
    EmptyPlan = pop([], OpenConds, TrueConds, Finish),  % No actions in initial plan
    MaxActions in 0..100,                               % Maximally 100 actions in plan
    indomain(MaxActions),                               % Enforce iterative deepening search
    Finish in 1..MaxActions,                            % Domain for finishing time of Plan
    depth_first(EmptyPlan, SolutionPath, MaxActions),   % Search in space of POP's
    once(indomain(Finish)),                             % Minimize finishing time
    append(_, [Plan], SolutionPath).                    % Working plan is last element of solution


% s(POP, NewPOP): successor relation between partially ordered plans
%     NewPOP is POP with the first open condition in POP achieved
%
s( pop(Acts, [Cond:Time | OpenPs], TrueConds, Fin),
  pop(Acts, OpenPs, TrueConds, Fin) ) :-
    member(Cond:Time1/Time2, TrueConds), % Cond already true between Time1 and Time2
    Time1 #< Time, Time #=< Time2.       % Constrain Time to interval Time1/Time2

s( pop(Acts, [Cond:Time | OpenPs0], TrueConds0, Fin),
     pop([Action1:Time1 | Acts], OpenPs, TrueConds, Fin) ) :-
    effects(Action1, Effects),              % Look for action that may achieve Cond
    del(Cond, Effects, RestEffects),        % Cond in Effects, that is Action1 achieves Cond
    can(Action1, PreConds1),                % Preconditions for Action1
    0 #< Time1, Time1 #< Time,              % Action1 must occur after 0 and before Time
    add_times(Time1, PreConds1, NewOpenPs),                   % Add Time1 to all preconditions
    add_intervals(Time1, RestEffects, RestEffectsTimes, Fin), % Add time intervals to all effects
    Time #=< Time2,                         % Achieved condition must be true until Time
    add_conds([Cond:Time1/Time2 | RestEffectsTimes], TrueConds0, TrueConds), % Add effects to True
    append(NewOpenPs, OpenPs0, OpenPs).       % Add preconditions of Action to goals
```

```
% add_conds(Conds, TrueConds, NewTrueConds):
%    Add conditions Conds to list TrueConds, and set corresponding precedence constraints
%
add_conds([], TrueConds, TrueConds).

add_conds([CondTime | Conds], TrueConds0, TrueConds) :-
    no_conflict(CondTime, TrueConds0), % No conflict between CondTime and TrueConds0
    add_conds(Conds, [CondTime | TrueConds0], TrueConds).


% no_conflict(CondTime, TrueConds0):
%   Set constraints to ensure no conflict between CondTime and TrueConds0
%
no_conflict(_, []).

no_conflict(CondTime, [Cond1Time1 | TrueConds]) :-
    no_conflict1(CondTime, Cond1Time1),
    no_conflict(CondTime, TrueConds).

no_conflict1(CondA:Ta1/Ta2, CondB:Tb1/Tb2) :-
    inconsistent(CondA, CondB), !, % CondA inconsistent with CondB
    (Ta2 #=< Tb1; Tb2 #=< Ta1)     % Ensure no time overlap between CondA and CondB
    ;
    true. % Case when CondA consistent with CondB - no constraint needed

% add_times(Time, Conds, TimedConds)
%
add_times(_, [], []).

add_times(Time, [Cond | Conds], [Cond:Time | TimedConds]) :-
    add_times(Time, Conds, TimedConds).

% add_intervals(Time, Conds, TimedConds, Finish):
%   every condition in Conds true from Time till some later time
%
add_intervals(_, [], [], _).

add_intervals(Time, [Cond | Conds], [Cond:Time/Time2 | TimedConds], Finish) :-
    Time #< Time2, Time2 #=< Finish, % Cond true from Time until Time2 =< Finish
    add_intervals(Time, Conds, TimedConds, Finish).


% depth_first(POP, SolutionPath, MaxActionsInPOP):
```

```
%   Depth-first search, with respect to number of actions, among partially ordered plans
%
depth_first(POP, [POP], _) :-
    POP = pop(_, [], _, _). % No open preconditions - this is a working plan

depth_first(First, [First | Rest], MaxActions) :-
    First = pop(Acts, _, _, _),
    length(Acts, NActs),
    (NActs < MaxActions, !          % # actions in plan is below MaxActions
     ;
     Second = pop(Acts, _, _, _)), % # actions in plan at maximum, no action may be added
    s(First, Second),
    depth_first(Second, Rest, MaxActions).


% Display all possible execution schedules of a partial order plan
%
show_pop(pop(Actions, _, _, _)) :-
    instantiate_times(Actions),              % Instantiate times of actions for readability
    setof(T:A, member(A:T, Actions), Sorted), % Sort actions according to times
    nl, write('Actions = '), write(Sorted),   % Write schedule
    fail                                     % Backtrack to produce other schedules
    ;
    nl, nl.                                  % All schedues produced


% instantiate_times( Actions): instantiate times of actions respecting ordering constraints
%
instantiate_times([]).

instantiate_times([_:T | Acts]) :-
    indomain(T),                    % A value in domain of T
    instantiate_times(Acts).


del( X, [X | Tail], Tail).

del( X, [Y | Tail], [Y | Tail1]) :-
    del( X, Tail, Tail1).
```

1.3 Sokoban-like domain code:

```
:- op(100, fx, ~).

getResultant(X, Y, NX, NY, RX, RY) :-
```

```
  var(X), var(Y), var(NX), var(NY),
  (
  NX is RX, X is RX, NY is RY + 1, Y is RY + 2
  ;
  NX is RX, X is RX, NY is RY - 1, Y is RY - 2
  ;
  NX is RX + 1, X is RX + 2, NY is RY, Y is RY
  ;
  NX is RX - 1, X is RX - 2, NY is RY, Y is RY
  ).

getResultant(X, Y, NX, NY, RX, RY) :-
  var(X), var(Y), var(RX), var(RY),
  (
  RX is NX, X is NX, RY is NY + 1, Y is NY - 1
  ;
  RX is NX, X is NX, RY is NY - 1, Y is NY + 1
  ;
  RX is NX + 1, X is NX - 1, RY is NY, Y is NY
  ;
  RX is NX - 1, X is NX + 1, RY is NY, Y is NY
  ).

getResultant(X, Y, NX, NY, RX, RY) :-
  var(NX), var(NY), var(RX), var(RY),
  (
  NX is X, RX is X, NY is Y + 1, RY is Y + 2
  ;
  NX is X, RX is X, NY is Y - 1, RY is Y - 2
  ;
  NX is X + 1, RX is X + 2, NY is Y, RY is Y
  ;
  NX is X - 1, RX is X - 2, NY is Y, RY is Y
  ).

getAdjacent(X, Y, NX, NY) :-
  var(X), var(Y),
  (
  X is NX + 1, Y is NY
  ;
  X is NX - 1, Y is NY
  ;
  X is NX, Y is NY + 1
  ;
  X is NX, Y is NY - 1
```

```
  ).

getAdjacent(X, Y, NX, NY) :-
  var(NX), var(NY),
  (
  NX is X + 1, NY is Y
  ;
  NX is X - 1, NY is Y
  ;
  NX is X, NY is Y + 1
  ;
  NX is X, NY is Y - 1
  ).

can(move_sokoban(Sokoban, Position_X, Position_Y, NewPosition_X, NewPosition_Y),
  [sokoban(Sokoban, Position_X, Position_Y), empty(NewPosition_X, NewPosition_Y)]) :-
  sokoban(Sokoban),
  getAdjacent(Position_X, Position_Y, NewPosition_X, NewPosition_Y).

can(move_sokoban_and_box(Sokoban, Box, Position_X, Position_Y, NewPosition_X, NewPosition_Y,
  NewBoxPosition_X, NewBoxPosition_Y), [sokoban(Sokoban, Position_X, Position_Y),
  box(Box, NewPosition_X, NewPosition_Y), empty(NewBoxPosition_X, NewBoxPosition_Y)]) :-
  sokoban(Sokoban),
  getResultant(Position_X, Position_Y, NewPosition_X, NewPosition_Y,
  NewBoxPosition_X, NewBoxPosition_Y).

effects(move_sokoban(Sokoban, Position_X, Position_Y, NewPosition_X, NewPosition_Y),
  [sokoban(Sokoban, NewPosition_X, NewPosition_Y), empty(Position_X, Position_Y),
  ~sokoban(Sokoban, Position_X, Position_Y), ~empty(NewPosition_X, NewPosition_Y)]).

effects(move_sokoban_and_box(Sokoban, Box, Position_X, Position_Y, NewPosition_X, NewPosition_Y,
  NewBoxPosition_X, NewBoxPosition_Y), [sokoban(Sokoban, NewPosition_X, NewPosition_Y),
  box(Box, NewBoxPosition_X, NewBoxPosition_Y), empty(Position_X, Position_Y),
  ~sokoban(Sokoban, Position_X, Position_Y), ~box(Box, NewPosition_X, NewPosition_Y),
  ~empty(NewBoxPosition_X, NewBoxPosition_Y)]).

inconsistent(G, ~G).
inconsistent(~G, G).

inconsistent(sokoban(S1, P1X, P1Y), sokoban(S1, P2X, P2Y)) :- (P1X \== P2X; P1Y \== P2Y).
inconsistent(sokoban(S1, P1X, P1Y), sokoban(S2, P1X, P1Y)) :- S1 \== S2.

inconsistent(box(B1, P1X, P1Y), box(B1, P2X, P2Y)) :- (P1X \== P2X; P1Y \== P2Y).
inconsistent(box(B1, P1X, P1Y), box(B2, P1X, P1Y)) :- B1 \== B2.
```

```
inconsistent(sokoban(_, P1X, P1Y), box(_, P1X, P1Y)).
inconsistent(sokoban(_, P1X, P1Y), empty(P1X, P1Y)).
inconsistent(box(_, P1X, P1Y), empty(P1X, P1Y)).
```

Results:

`?-solve1.` results in the following output:

```
Actions =
[1:move_sokoban(a,1,1,1,2),
 2:move_sokoban_and_box(a,1,1,2,2,2,3,2),
 3:move_sokoban(a,2,2,2,3),
 4:move_sokoban(a,2,3,3,3),
 5:move_sokoban_and_box(a,1,3,3,3,2,3,1)]
```

2 Second exercise

Below is the implementation of a Neural Network for classification of hand-written digits:

```
import numpy
import scipy.special
import matplotlib.pyplot

import sys
import os

class neuralNetwork:

    # initialise the neural network
    def __init__(self, inputnodes, hiddennodes, outputnodes, learningrate):
        # set number of nodes in each input, hidden, output layer
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes

        # link weight matrices, wih and who
        # weights inside the arrays are w_i_j, where link is from node i to node j in the next lay
        # w11 w21
        # w12 w22 etc
        self.wih = numpy.random.normal(0.0, pow(self.inodes, -0.5), (self.hnodes, self.inodes))
        self.who = numpy.random.normal(0.0, pow(self.hnodes, -0.5), (self.onodes, self.hnodes))

        # learning rate
        self.lr = learningrate

        # activation function is the sigmoid function
```

```
        self.activation_function = lambda x: scipy.special.expit(x)

        pass

    # train the neural network
    def train(self, inputs_list, targets_list):
        # convert inputs list to 2d array
        inputs = numpy.array(inputs_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T

        # calculate signals into hidden layer
        hidden_inputs = numpy.dot(self.wih, inputs)
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)

        # calculate signals into final output layer
        final_inputs = numpy.dot(self.who, hidden_outputs)
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        # output layer error is the (target - actual)
        output_errors = targets - final_outputs
        # hidden layer error is the output_errors, split by weights, recombined at hidden nodes
        hidden_errors = numpy.dot(self.who.T, output_errors)

        # update the weights for the links between the hidden and output layers
        self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_outputs)), r

        # update the weights for the links between the input and hidden layers
        self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidden_outputs)),

        pass

    # query the neural network
    def query(self, inputs_list):
        # convert inputs list to 2d array
        inputs = numpy.array(inputs_list, ndmin=2).T

        # calculate signals into hidden layer
        hidden_inputs = numpy.dot(self.wih, inputs)
        # calculate the signals emerging from hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)

        # calculate signals into final output layer
        final_inputs = numpy.dot(self.who, hidden_outputs)
```

```
        # calculate the signals emerging from final output layer
        final_outputs = self.activation_function(final_inputs)

        return final_outputs

# number of input, hidden and output nodes
input_nodes = 784
hidden_nodes = 200
output_nodes = 10

# learning rate
learning_rate = 0.1

# create instance of neural network
n = neuralNetwork(input_nodes,hidden_nodes,output_nodes, learning_rate)

print ("neuralNetwork instantiated... ")

# load the mnist training data CSV file into a list
training_data_file = open("mnist_train.csv", 'r')
training_data_list = training_data_file.readlines()
training_data_file.close()

print ("mnist_train.csv loaded... ")

# train the neural network

# epochs is the number of times the training data set is used for training
epochs = 5

for e in range(epochs):
    # go through all records in the training data set
    for record in training_data_list:

        # print(record)

        # split the record by the ',' commas
        all_values = record.split(',')
        # scale and shift the inputs
        inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
        # create the target output values (all 0.01, except the desired label which is 0.99)
        targets = numpy.zeros(output_nodes) + 0.01
        # all_values[0] is the target label for this record
        targets[int(all_values[0])] = 0.99
        n.train(inputs, targets)
```

```
        pass
    pass

print ("neuralNetwork trained... ")

user_input = input ("Enter the path of a \'handwritten\' digit you\'d like the neural network to c
assert os.path.exists(user_input), "No file was found at \'"+str(user_input)+"\'..."

# evaluate
# load the digit CSV file into a list
digit_file = open(user_input, 'r')
digit_list = digit_file.readline()
digit_file.close()
all_values = digit_list.split(',')
inputs = (numpy.asfarray(all_values[0:]) / 255.0 * 0.99) + 0.01
outputs = n.query(inputs)
label = numpy.argmax(outputs)
print ("The neural network classified the digit as a \'"+str(label)+"\'")
# evaluate

user_input = input ("Would you like to try another digit? y/n: ")

while user_input == "y":

    user_input = input ("Enter the path of a \'handwritten\' digit you\'d like the neural network
    assert os.path.exists(user_input), "No file was found at \'"+str(user_input)+"\'..."

    # evaluate
    # load the digit CSV file into a list
    digit_file = open(user_input, 'r')
    digit_list = digit_file.readline()
    digit_file.close()
    all_values = digit_list.split(',')
    inputs = (numpy.asfarray(all_values[0:]) / 255.0 * 0.99) + 0.01
    outputs = n.query(inputs)
    label = numpy.argmax(outputs)
    print ("The neural network classified the digit as a \'"+str(label)+"\'")
    # evaluate

    user_input = input ("Would you like to try another digit? y/n: ")
```

Operation:

The program can be started from 'cmd' with `python second.py`.

3 Group

- Bartłomiej Jacak
- Kacper Muszyński
- Michał Sar