

HARCOURT BUTLER TECHNICAL UNIVERSITY



Department of Computer Science Engineering
Session: **2022-2023**

ARTIFICIAL INTELLIGENCE **LAB WORK**

Submitted To:

Dr.Amit Gupta
Ms. Sonia Gupta
. Pooja Agnihotri

Submitted By:

AKSHITA GUPTA
190104010
Final B.Tech CSE

1. Write a LISP program to solve the water-jug problem using heuristic function.

CL-USER 1 > (start)

WATER JUG PROBLEM

FILL JUGNAME : (fill JUGNAME)

EMPTY JUGNAME : (empty JUGNAME)

PUT A TO B : (put A to B)

GIVE THE STARTING STATE OF EACH JUG IN
FROM (A B C) E.G (0 0 0)
MAX A=7, MAX B=4 AND MAX C=2 : (0 0 0)
NIL

CL-USER 2 > (fill B)

NEW STATE : (0 4 0)

NIL

CL-USER 3 > (put B to A)

NEW STATE : (0 4 0 0)

NIL

CL-USER 4 > (fill B)

NEW STATE : (4 4 0)

NIL

CL USER 5> (put B to A)

^{NEW STATE}
CL-USER : (7 1 0)

YOU WIN B = 1

NIL

CL-USER 6>

2. Create a compound object using Turbo Prolog.

domain

things = car (brand); flat; villa; networth (integer)

name, brand = symbol

predicates

owns (name, things)

clauses

owns (jeff, car (bmw)).

owns (jeff, villa).

owns (jeff, networth (43)).

owns (bill, car (audi)).

owns (bill, flat).

owns (bill, networth (40)).

3. Write a Prolog Program to show the advantage and disadvantage of green & red cuts.

Domains:

disease, indication = symbol
name-string

Predicates:

hypothesis(name, disease)
symptom(name, indication)
response(char)

go

goonce

clauses:

go:-

goonce

write("will you like to try again (y/n)?"),
response(Reply),
Reply = 'n'.

go.

goonce:-

write("what is the patient's name"), nl,
readln(Patient),

hypothesis(Patient, Disease), !,

write(Patient, "probably has", Disease), ?,

goonce:-

write ("sorry, i am not in a position to diagnose"),
 write ("the disease").

symptom (Patient, fever):-

write ("does", Patient, "has a fever (y/n)?"), nl,
 response (Reply),

Reply = 'y', nl.

symptom (Patient, rash):-

write ("does", Patient, "has a rash (y/n) ?"),
 nl, response (Reply),

Reply = 'y'

symptom (Patient, body-ache):-

write ("does", Patient, "has a body ache (y/n)?"),
 nl, response (Reply).

Reply = 'y', nl.

symptom (Patient, runny-nose):-

write ("does", Patient, "has a runny-nose (y/n)?"),
 response (Reply),

Reply = 'y'

hypothesis (Patient, flu):-

symptom (Patient, fever),

symptom (Patient, body-ache),

hypothesis (Patient, common-cold):-

symptom (Patient, body-ache),

symptom (Patient, runny-nose).

response (Reply):-

readchar (Reply),

write (Reply).

- Q. Write a prolog program to use BEST-FIRST SEARCH applied to the eight puzzle problem.

`test(Plan):-`

`write('Initial state:'), nl,`

`Init = [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4),
at(tile2,5), at(tile6,6), at(tile5,7), at(tile1,8),
at(tile7,9)],`

`write-go(Init),`

`Goal = [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4),
at(empty,5), at(tile5,6), at(tile6,7), at(tile7,8),
at(tile8,9)],`

`nl, write('Goal state:'), nl,`

`write(Goal), nl, nl,`

`solve(Init, Goal, Plan).`

`solve(SState, Goal, Plan):-`

`solve(SState, Goal, [], Plan).`

`is-movable(X1, Y1) :- (1 is X1-Y1); (-1 is X1-Y1);
(3 is X1-Y1); (-3 is X1-Y1).`

`solve(SState, Goal, Plan, Plan):-`

`is-subset(Goal, SState), nl,`

`write-go(Plan).`

solve(State, Goal, SoFar, Plan) :-

act(Action, Preconditions, Delete, Add),

is-subset(Preconditions, State),

\+ member(Action, SoFar),

delete-list(Delete, State, Remainder),

append(Add, Remainder, NewState),

solve(NewState, Goal, [Action| SoFar], Plan).

act(move X, Y, Z),

[at(X, Y), at(empty, Z), is-movable(Y, Z)],

[at(X, Y), at(empty, Z)],

[at(X, Y), at(empty, Y)]).

is-subset([HIT], Set) :-

member(H, Set),

is-subset(T, Set).

is-subset([], -).

delete-list([HIT], CurState, NewState) :-

remove(H, CurState, Remainder),

delete-list(T, Remainder, NewState).

delete-list([], CurState, CurState).

remove(X, [X| T], T).

remove(X, [HIT], [HIR]); -

remove(X, T, R).

write_sol([]).

write_sol([H | T]):-

 write_sol(T),

 write(H), nl.

append([H | T], L1, [H | L2]):-

 append(T, L1, L2).

append([], L, L).

member(X, [X | _]).

member(X, [_ | T]):-

 member(X, T).

5. Implementation of the problem solving strategies:-
 Forward Chaining, Backward Chaining, Problem Reduction.

(ii) Forward chaining:-

```

  :- op(1100,xfx,if).
  :- op(1000,xfy,aud).
  :- op(900,xfy,or).
  :- dynamic rule/1.
  
```

forward(Facts):-

```
fixed_point(nil,[true],Facts).
```

```
fixed_point(Base,Base,Base):-!.
```

```
fixed_point(_,Base,Facts):-
```

```
  setof(Fact,derived(Fact,Base),NewFacts),
```

```
  ord-union(NewFacts,Base,NewBase),
```

```
  fixed_point(Base,NewBase,Facts).
```

derived(Fact,Base):-

rule(Fact if Condition),

satisfy(Base,Condition).

satisfy(Base,G1 and G2):-

!,

member(G1,Base),

satisfy(Base,G2).

satisfy(Base, G or G₂):-

!,
member(G, Base)
; satisfy(Base, G₂)).

satisfy(Base, Condition):-

member(Condition, Base).

(ii) Backward chaining :-

is_true(P, P):-

fact P.

is_true(C, C ∈ ProofTreeA):-

if A then C, is_true(A, ProofTreeA)

is_true(P₁ and P₂, ProofTree1 and ProofTree2):-

is_true(P₁, ProofTree1), is_true(P₂, ProofTree2).

is_true(P₁ or P₂, ProofTree1):- is_true(P₁, ProofTree1).

is_true(_ or P₂, ProofTree2):- is_true(P₂, ProofTree2)

6. Write a LISP Program to implement STEEPEST-ASCENT HILL CLIMBING.

```
(setf (get 's 'neighbours) '(a d)
      (get 'a 'neighbours) '(s b d)
      (get 'b 'neighbours) '(a c e)
      (get 'c 'neighbours) '(b)
      (get 'd 'neighbours) '(s a e)
      (get 'e 'neighbours) '(b d f)
      (get 'f 'neighbours) '(e))
```

```
(setf (get 's 'neighbours) '(0 3)
      (get 'a 'neighbours) '(4 6)
      (get 'b 'neighbours) '(7 6)
      (get 'c 'neighbours) '(11 6)
      (get 'd 'neighbours) '(3 0)
      (get 'e 'neighbours) '(6 0)
      (get 'f 'neighbours) '(11 3))
```

```
(defun straight-line-distance (node1 node2)
  "Compute the straight line distance between two nodes"
  (let ((coord1 (get node1 'coordinates))
        coord2 (get node2 'coordinates)))
    (sqrt (+ (expt (- (first coord1)) (first coord2)) 2)
          (expt (- (second coord1)) (second coord2)) 2
          )))))
```

(defun extend-path (path)

"Extend the path by consing on new neighbor nodes that don't result in loops"

(mapcar #'(lambda (new-node) (cons new-node path))

(remove-if

#'(lambda (neighbor) (member neighbor path)))

(get (first path) 'neighbors)))

(defun closerp (node1 node2 target)

"Returns true if node1 is closer to target than node2, else false"

(< (straight-line-distance node1 target)

(straight-line-distance node2 target)))

(defun hill-climb (start finish &optional (queue
(list (list start))))

"Perform hill climbing search for path from start to finish"

(cond ((endp queue) nil)

(if (eq finish (first (first queue)))

(reverse (first queue)))

(t

(print queue)

(hill-climb

start finish

(append (sort (extend-path (first queue))

#'(lambda (p1 p2).

(closerp (first p1) (first p2) finish)))
(rest queue))))))

2

12

7. Write a prolog program to implement COUNTER PROPAGATION NETWORK.

About counter propagation network:-

CPN is a multilayer network based on the combinations of I/p, O/p and clustering layers. The application of CPN are data compression, function approximation & pattern association.

This model is 3 layer Neural Network that performs input-output data mapping, producing an output vector(y) in response to i/p vector(x) on the basis of competitive learning.

Full counterpropagation network:-

Full CPN represents a large number of vector pairs x_i, y_j by adaptively constructing a look-up table. The full CPN works best if the inverse function exists & is continuous.

Training Algorithm for Full CPN:-

Step 0:- Set the initial wts. & initial learning rate.

Step 1:- Perform steps 2-7 if stopping condition is false for phase-I training.

Step 2:- For each of the training input vector pair x_i, y_i presented, perform Step 3-5.

Step 3:- Make the X-input layer activations to vector X .
Make the Y-input layer activations to vector Y .

Step 4:- Find the winning cluster unit, If dot product method is used, find the cluster unit Z_j with target net input : for $j=1$ to p

$$Z_{inj} = \sum_{i=1}^n x_i v_{ij} + \sum_{k=1}^m y_k w_{kj}$$

If euclidean distance method is used, find the cluster unit Z_j whose squared distance from i/p vector is the smallest

$$D(j) = \sum_{i=1}^n (x_i - v_{ij})^2 + \sum_{k=1}^m (y_k - w_{kj})^2$$

Step 5:- Update the weights over the calculated winner unit Z_j

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \alpha [x_i - v_{ij}(\text{old})] \quad i=1 \text{ to } n$$

$$w_{kj}(\text{new}) = w_{kj}(\text{old}) + \beta [y_k - w_{kj}(\text{old})] \quad k=1 \text{ to } m$$

Step 6:- Reduce the learning rates α & β

$$\alpha(t+1) = 0.5 \alpha t$$

$$\beta(t+1) = 0.5 \beta t$$

Step 7:- Test stopping condition for phase-I training

Step 8:- Perform Step 9-15 when stopping condition is false for phase-II training.

Step 9: Perform 10-13 for each training input pair x_i, y_i .
Hence α & β are small constant values.

Step 10: Make the X-input layer activations to vector x .
Make the Y-input layer activations to vector y .

Step 11: find the winning cluster unit (use formulae for Step 4). Take the winner unit index as j

Step 12: Update the weights entering into unit z_j
 $v_{ij}(\text{new}) = v_{ij}(\text{old}) + \alpha [x_i - v_{ij}(\text{old})] \quad i=1-n$
 $w_{kj}(\text{new}) = w_{kj}(\text{old}) + \beta [y_k - w_{kj}(\text{old})] \quad k=1-m$

Step 13:- Update the weights from unit z_j to o/p layers.

$$t_{ji}(\text{new}) = t_{ji}(\text{old}) + b[x_i - t_{ji}(\text{old})] \quad i=1 \text{ to } n$$

$$u_{jk}(\text{new}) = u_{jk}(\text{old}) + a[y_k - u_{jk}(\text{old})] \quad k=1 \text{ to } m$$

Step 14:- Reduce the learning rates a & b .

$$a(t+1) = 0.5 a t$$

$$b(t+1) = 0.5 b t$$

Step 15:- Test stopping condition for phase-II training.

8. Development of a small Expert System using PROLOG.

Expert System on Diagnosis of Mental Disorders.

diagnose:-

write('This is an expert system for diagnosis of mental disorders.'), nl,

write('There are several questions you need to answer for diagnosis of mental disorders.'), nl, nl, disorder(X),

write('Condition was diagnosed as'),

write(X),

write('.').

diagnose:-

write('The diagnose was not found.').

question(Attribute, Value):-

retract(yes, Attribute, Value), !.

question(Attribute, Value):-

retract(_, Attribute, Value), !, fail.

question(Attribute, Value):-

write('Is the'),

write(Attribute),

write(' - '>,

write(Value),

write('?'),

read(Y),

asserta(retract(Y, Attribute, Value)).

$Y == \text{yes}$.

questionWithPossibilities(Attribute, Value, Possibilities) :-
 write('What is the patient's'), write(Attribute),
 write('?'), nl, write(Possibilities), nl,
 read(X),
 check_val(X, Attribute, Value, Possibilities),
 asserta(retract(yes, Attribute, X)),
 X == Value.

check_val(X, -, -, Possibilities) :- member(X, Possibilities).
 check_val(X, Attribute, Value, Possibilities) :-
 write(X), write(' is not a legal value, try again.'), nl,
 questionWithPossibilities(Attribute, Value, Possibilities).

i - dynamic (retract / 3).

food-amount(X) :- question(food-amount, X).
 symptom(X) :- question(symptom, X).
 mentality(X) :- question(mentality, X).
 cause(X) :- question(cause, X).
 indication(X) :- question(indication, X).
 social-skill(X) :- question(social-skill, X).
 condition(X) :- question(condition, X).
 consequence(X) :- question(consequence, X).
 speciality(X) :- question(speciality, X).
 face-features(X) :- question(face-features, X).

easy-features (X) :- question (easy-features, X).

brain-function (X) :- question (brain-function, X).

perceptions (X) :- question (perceptions, X).

behaviour (X) :- question with Possibilities (behaviour, X,
[repetitive-and-restricted, narcissistic, aggressive]).

disorder (anorexia-nervosa) :- type (eating-disorder),
consequence (low-weight),
food-amount (food-restriction).

disorder (bulimia-nervosa) :- type (eating-disorder),
consequence (purgings),
food-amount (binge-eating).

disorder (asperger-syndrome) :- type (neurodevelopmental-disorder),
specialty (psychiatry),
social-skill (low),
behavior (repetitive-and-restricted)

disorder (dyslexia) :- type (neurodevelopmental-disorder),
social-skill (normal),
perception (low),
symptom (trouble-reading).

disorder (autism) :- type (neurodevelopmental-disorder),
social-skill (low),
symptom (impaired-communication).

disorder (tourettes-syndrome) :- type (neurodevelopmental-disorder),
social-skills (normal),
specialty (neurology)
symptom (motor-tics).

disorder (bipolar-disorder) :- type (psychotic-disorder),
indication (elevated moods).

disorder (schizophrenia) :- type (psychotic-disorder),
indication (hallucinations).

disorder (down-syndrome) :- type (genetic-disorder),
symptom (delayed-physical-growth),
face-features (long-and-narrow),
ear-features (large),
brain-function (intellectual-disability).

disorder (fragile-X-syndrome) :- type (genetic-disorder),
face-features (small-chin-and-slanted-eyes),
brain-function (intellectual-disability).

type (eating-disorder) :- symptom (abnormal-eating-habits),
mentality (strong-desire-to-be-thin).

type (neurodevelopmental-disorder) :- condition (affected-nervous-system),
brain-function (abnormal),
cause (genetic-and-environmental).

type (psychotic-disorder) :- symptom (false-beliefs),
mentality (manic-depressive),
cause (genetic-and environmental).

type (genetic-disorder) :- cause (abnormalities-in-genome).

9.

Development a prolog program for monkey - banana problem.

do (state (middle, onbox, middle, hasn't) % grab banana
 grab,
 state (middle, onbox, middle, has)).

do (state (L, onfloor, L, Banana) % climb box
 climb,
 state (L, onbox, L, Banana)).

do (state (L1, onfloor, L1, Banana), % push box from L1→L2
 push (L1, L2),
 state (L2, onfloor, L2, Banana)).

do (state (L1, onfloor, Box, Banana), % walk from L1 to L2
 walk (L1, L2),
 state (L2, onfloor, Box, Banana)).

canget (state (-, -, -, has)).

canget (State1):-

do (State1, Action, State2),

canget (State2).

canget(state(-,-,-,has), []).

canget(State1, Plan) :-

do(State1, Action, State2),

canget(State2, PartialPlan),

add(Action, PartialPlan, Plan).

add(X, L, [X | L]).

10. Write a program for parsing a given sentence:-

parse a sentence into article, noun, verb, preposition & adjective.

$np([X]T, np(det(X), NP2), Rem) :-$
 $det(X),$
 $np2(T, NP2, Rem).$

$np(Sentence, Parse, Rem) :- np2(Sentence, Parse, Rem).$
 $np(Sentence, np(NP, PP), Rem) :-$
 $np(Sentence, NP, Rem1),$
 $pp(Rem1, PP, Rem).$

$np2([H]T, np2(noun(H), T) :- noun(H).$

$np2([H]T, np2(adj(H), Rest), Rem) :- adj(H), np2(T, Rest, Rem).$

$pp([H]T, pp(prep(H), Parse), Rem) :-$
 $prep(H),$
 $np(T, Parse, Rem).$

$vp([H][], vp(verb(H))) :-$
 $verb(H).$

$vp([H]T, vp(verb(H), Rem)) :-$
 $verb(H, Rem),$
 $pp(T, Rem, -).$

vp([HIT], vp(verb(H), Rem)):-
verb(H),
np(T, Rem, -).