CSE643: Artificial Intelligence
Assignment 2: Knowledge Representation, Reasoning and Planning

Himanshu Raj (2022216)

November 8, 2024

Computational

1) Data loading and Knowledge Base creation

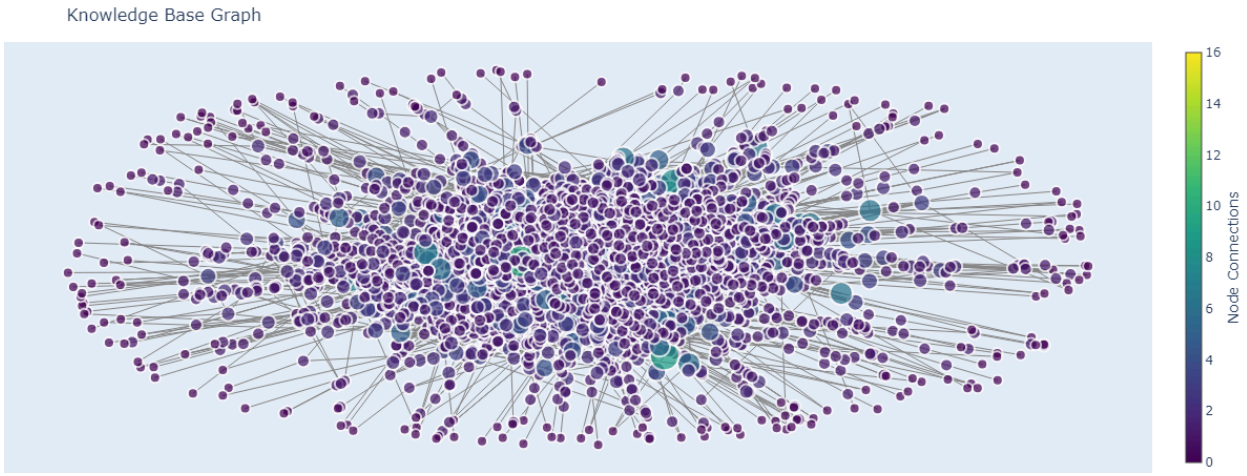
a) Top 5 busiest routes based on the number of trips.
Answer: [(5721, 318), (5722, 318), (674, 313), (593, 311), (5254, 272)]

b) Top 5 stops with the most frequent trips.
Answer: [(10225, 4115), (10221, 4049), (149, 3998), (488, 3996), (233, 3787)]

c) The top 5 busiest stops based on the number of routes passing through them.
Answer: [(488, 102), (10225, 101), (149, 99), (233, 95), (10221, 86)]

d) The top 5 pairs of stops (start and end) that are connected by exactly one direct route, sort them by the combined frequency of trips passing through both stops.
Answer: [((233, 148), 1433), ((11476, 10060), 5867), ((10225, 11946), 5436),
((11044, 10120), 5916), ((11045, 10120), 5610)]

Graph representation of route_to_stops from the knowledge base:
Stops are circles/nodes and they are connected by edges/routes.



2) Reasoning

a) Time Complexity and Memory Usage

I made 10,000 random test cases and tested query_direct_routes and direct_route_brute_force on these.

```
Total time for query_direct_routes: 109.120884 seconds
Total memory for query_direct_routes: 99.549805 MB
Total time for direct_route_brute_force: 69.847349 seconds
Total memory for direct_route_brute_force: 0.228882 MB
```

I observed that direct_route_brute_force is efficient by large margins than query_direct_routes in both time and memory. Query from pyDatalog takes way more time and space than iterating a dictionary stored in memory.

b) Intermediate steps in Reasoning process

In brute force approach, we are iterating a dictionary route_to_stops, which we created while creating knowledge base, and checking if both the stops exist in the list of stops mapped to a route_ID. If a route has both start and end stops, it is a valid route and is added in the answer list.

In query from pyDatalog approach, we check among all our facts if any route has both the stops, we define our predicate as:
DirectRoute(R, X, Y) <= RouteHasStop(R, X) & RouteHasStop(R, Y)
where RouteHasStop are facts which we added while initialising our datalog. This query return the valid routes that have both stops.

c) Comparing the overall number of steps involved

Time Complexity for brute force approach is O(m*n) where m is the number of route_IDs in dictionary and n is the maximum number of stops any route has.

Time Complexity for query in pyDatalog approach is O(RouteHasStop * RouteHasStop), because for every query we check in logs of RouteHasStop for routes that contain the required start and end stop. O(RouteHasStop) is O(m*n).
So final time complexity is O((m*n)^2) where m is the number of route_IDs in dictionary and n is the maximum number of stops any route has.

3) Planning

a) Time Complexity and Memory Usage
I made around ~8k random test cases and tested forward_chaining and backward_chaining on these.

```
Total time for forward_chaining: 1401.392396 seconds
Total memory for forward_chaining: 270.251334 MB
Total time for backward_chaining: 1152.113627 seconds
Total memory for backward_chaining: 263.657429 MB
```

I observed that backward_chaining is a bit more efficient than forward_chaining in terms of time and memory.

b) Intermediate steps in Reasoning process

We define a new predicate as:
OptimalRoute(R1, R2, X, Y, Z) <= DirectRoute(R1, X, Z) & DirectRoute(R2, Z, Y) & (R1!=R2)

In forward chaining, we first find a direct route from start stop to intermediate stop and then from intermediate stop to end stop and these routes should be different. We can either do this with a procedural method or just define the above predicate and query directly from datalog.

In backward chaining, we find a direct route from end stop to intermediate stop and then from intermediate stop to start stop and these routes should be different. We can either do this with a procedural method or just define the above predicate and query directly from datalog.

c) Comparing the overall number of steps involved

The overall number of steps involved is almost equal. Time complexity for both the chaining algorithms is O(DirectRoute * DirectRoute) and time complexity for DirectRoute is O(RouteHasStop * RouteHasStop).
So final complexity is O((m*n)^4) where m is the number of route_IDs in dictionary and n is the maximum number of stops any route has.

4) PDDL

a) Time Complexity and Memory Usage
I made around ~8k random test cases, the same used for forward and backward chaining

```
Total time for PDDL planning: 1005.648030 seconds
Total memory for PDDL planning: 150.940751 MB
```

PDDL planning turns out to be faster than both backward and forward chaining.

b) Intermediate steps in Reasoning process

We define new predicates as:
BoardRoute(R, X) <= RouteHasStop(R, X)
TransferRoute(R1, R2, Z) <= RouteHasStop(R1, Z) & RouteHasStop(R2, Z) & (R1!=R2)
PDDL(R1, R2, X, Y, Z) <= BoardRoute(R1, X) & TransferRoute(R1, R2, Z) & BoardRoute(R2, Y)

We board the start route that leads to intermediate stop, then at intermediate stop we transfer routes to routes that lead to end stop and we board this new route. We can either do this with a procedural method or just define the above PDDL predicate and query directly from datalog.

c) Comparing the overall number of steps involved

Time complexity is O(BoardRoute * TransferRoute * BoardRoute).O(BoardRoute) is O(RouteHasStop) and O(TransferRoute) is O(RouteHasStop * RouteHasStop).
So final complexity is O((m*n)^4) where m is the number of route_IDs in dictionary and n is the maximum number of stops any route has.

All algorithms (forward chaining, backward chaining, and PDDL) produce the same optimal route even after the constraints are applied due to their implementation at the core is more or less equivalent.

Himanshu Raj
2022216

Date : / /
Page No.

AI Assignment-2

Theoretical

Q1 Let G_t : ~~traj~~ traffic light is green at time t
R_t : light is red at time t
Y_t : light is yellow at time t

Rule 1
a) $(G_t \vee R_t \vee Y_t) \wedge \neg(R_t \vee Y_t) \wedge \neg(Y_t \vee G_t) \wedge \neg(G_t \vee R_t)$

Rule 2
b) $(G_t \rightarrow Y_{t+1}) \wedge (Y_t \rightarrow R_{t+1}) \wedge (R_t \rightarrow G_{t+1})$

Rule 3
c) $\neg(R_t \wedge R_{t+1} \wedge R_{t+2} \wedge R_{t+3}) \wedge \neg(Y_t \wedge Y_{t+1} \wedge Y_{t+2} \wedge Y_{t+3}) \wedge \neg(G_t \wedge G_{t+1} \wedge G_{t+2} \wedge G_{t+3})$

Q3 Propositional Logic - R (can read), L (is literate), I (is intelligent), D (is dolphin)

S1: $R \rightarrow L$
S2: $D \rightarrow \neg L$
S3, S4, S5: These statements can't be represented with propositional logic as we can't specify quantity in propositional logic.

Date : / /
Page No.

First order Logic :

$R(x)$ - x can red
$L(x)$ - x is litterate
$I(x)$ - x is intteligent
$D(x)$ - x is a dolphin

S1: $\forall x \quad R(x) \rightarrow L(x)$
S2: $\forall x \quad D(x) \rightarrow \neg L(x)$
S3: $\exists x \quad D(x) \wedge I(x)$
S4: $\exists x \quad I(x) \wedge \neg R(x)$
S5: $\exists x (D(x) \wedge I(x) \wedge R(x)) \wedge \forall x ((D(x) \wedge I(x) \wedge R(x)) \rightarrow \neg L(x))$

Resolution Refutation - for S4

KB- $\forall x \ (\neg R(x) \vee L(x))$,
$\forall x \ (\neg D(x) \vee \neg L(x))$ and
$\exists x \ (D(x) \wedge I(x))$ ~~$\forall x \neg (D(x) \vee I(x))$~~

(S4) query - $\exists (x) \ (I(x) \wedge \neg R(x))$
negation of query - $\forall x (\neg I(x) \vee R(x))$

KB $\wedge \neg$query, also we break down S3 as
$(D(x)) \wedge (I(x))$
so for some c, S3 gives $D(c)$ and $I(c)$.

$\neg R(x) \vee L(x)$
$\neg D(x) \vee \neg L(x)$
$D(c)$
$I(c)$ } from S_3
$\neg I(x) \vee R(x)$

Date : / /
Page No.

from D(c) and $\neg D(x) \vee \neg L(x)$

↳ $\neg L(c)$

from $\neg L(c)$ and $\neg R(x) \vee L(x)$

↳ $\neg R(c)$

from I(c) and $\neg I(x) \vee R(x)$

↳ R(c)

from R(c) and $\neg R(c)$

↳ {} - empty clause

We get an empty clause, this means that ¬query should be false, our assumption is wrong. Thus by contradiction query (S4) is valid. or satisfiable.

Resolution Refutation for S_5

KB - $\forall x\ (\neg R(x) \vee L(x))$,
$\forall x\ (\neg D(x) \vee \neg L(x))$,
$\exists x\ (D(x) \wedge I(x))$ — gives D(c) and I(c)
$\exists x\ (I(x) \wedge \neg R(x))$ — gives I(c) and $\neg R(c)$

query - S_5 - $\exists x (D(x) \wedge I(x) \wedge R(x)) \wedge$

this gives $(D(c) \wedge I(c) \wedge R(x))$

$\forall x\ ((D(x) \wedge I(x) \wedge R(x)) \rightarrow \neg L(x))$

↳ $\forall x\ (\neg(D(x) \wedge I(x) \wedge R(x)) \vee \neg L(x))$

simplifying S_5, we get $\neg L(c)$

negation of query - L(c)

Date : / /
Page No.

from D(c) and $\neg D(x) \vee \neg L(x)$
$\hookrightarrow \neg L(c)$

from $\neg L(c)$ and $\neg R(x) \vee L(x)$
$\hookrightarrow \neg R(c)$

we are left with $\neg R(c)$, $I(c)$, $I(c)$, $\neg R(c)$ and $L(c)$

we can not resolve any further and we do not get an empty clause. This means ¬query is valid, so (ss)query is invalid or unsatisfiable.

Q2 Let Node (x) : x is a node of graph $\in N$.
Edge (x,y) : directed edge from x to y $\in R$.
Color (x,c) : color of node x is c, $c \in C$.

R1: $\forall x \forall y \left(\text{Edge}(x,y) \wedge \text{Node}(x) \wedge \text{Node}(y)\right) \rightarrow \forall c \left(\text{color}(x,c) \wedge \neg \text{color}(y,c)\right)$

R2: ~~$\exists x \exists y \forall z \forall t \; \text{Node}(x) \wedge \text{Node}(y) \wedge \text{color}(x, \text{yellow}) \wedge \text{color}(y, \text{yellow}) \wedge \forall z \left(\text{color}(z, \text{yellow}) \rightarrow (z = x) \vee (z = y)\right)$~~

we define Reach (x,y,4) means we can reach y from x in no more than 4 steps.

R_3: $\forall x \left(\text{Node}(x) \wedge \text{color}(x, \text{Red})\right) \rightarrow \exists y \left(\text{Node}(y) \wedge \text{color}(y, \text{Green}) \wedge \text{Reach}(x,y,4)\right)$

Date : / /
Page No.

R2: $\exists x \exists y \, (Node(x) \wedge Node(y)) \wedge (color(x, yellow) \wedge color(y, yellow) \wedge \forall z \, (color(z, yellow) \rightarrow ((z = x) \vee (z = y))))$

R4: $\forall c \in C \; \exists x \; (Node(x) \wedge color(x, c))$

~~R5~~

R5: $\forall c \in C \; \exists x \exists y \, (Clique(x, c) \wedge Clique(y, c) \wedge \forall z \, (Clique(z, c)) \rightarrow color(z, c)))$
$\forall c \forall d \, (c \neq d) \rightarrow \forall x \, (Clique(x, c) \rightarrow \neg Clique(x, d))$

define Clique (x, c): x is a subset of nodes forming a clique of color c, $c \in C$ and $x \subset N$.