Assignment 2
Theory

1. Propositions to represent state of traffic light at time t are defined as follows:

G_t : Light is green at time t

Y_t : Light is yellow at time t

R_t : Light is red at time t

Rule 1 : $(G_t \lor Y_t \lor R_t) \land (\neg G_t \lor \neg Y_t) \land (\neg G_t \lor \neg R_t) \land (\neg Y_t \lor \neg R_t)$

Rule 2 : $(G_t \to Y_{t+1}) \land (R_t \to G_{t+1}) \land (Y_t \to R_{t+1})$

Rule 3 : $((G_t \land G_{t+1} \land G_{t+2}) \to \neg G_{t+3}) \land ((R_t \land R_{t+1} \land R_{t+2}) \to \neg R_{t+3})$
$\land ((Y_t \land Y_{t+1} \land Y_{t+2}) \to \neg Y_{t+3})$

2. Predicates in FOL language

Node(x) : x is a node

Color(c) : c is a color in $C : \{c_1, \ldots, c_k\}$

HasColor(x, c) : Node x has color c

Edge(x, y) : Directed edge from node x to node y exists

Clique(x, c) : Node x is part of clique for color c

Axioms

Let $x, y, z, w, v \in$ domain of Node and $c \in$ domain of Color

1. $\forall x,y\, (Edge(x,y) \to \forall c\, (HasColor(x,c) \to \neg HasColor(y,c)))$

2. $\exists x,y\, (x \neq y \land HasColor(x, Yellow) \land HasColor(y, Yellow))$
$\land \forall z\, (HasColor(z, Yellow) \to (z = x \lor z = y))$

3. $\forall x\, (HasColor(x, Red) \to \exists y\, (HasColor(y, Green) \land Edge(x,y))$
$\lor \exists y,z\, (Edge(x,y) \land Edge(y,z) \land HasColor(z, Green))$
$\lor \exists y,z,w\, (Edge(x,y) \land Edge(y,z) \land Edge(z,w) \land HasColor(w, Green))$
$\lor \exists y,z,w,v\, (Edge(x,y) \land Edge(y,z) \land Edge(z,w) \land HasColor(v, Green)$
$\land Edge(w,v))$

4. $\forall c\,(\mathrm{Color}(c) \rightarrow \exists x\,(\mathrm{HasColor}(x,c) \land \mathrm{Node}(x)))$

5. $\forall c\,((\mathrm{Color}(c) \rightarrow \exists x(\mathrm{Node}(x) \land \mathrm{HasColor}(x,c) \land \forall y\,(\mathrm{Node}(y) \rightarrow (\mathrm{HasColor}(y,c) \leftrightarrow \mathrm{Clique}(x,c) \land \mathrm{Clique}(y,c) \land (x=y \lor (\mathrm{Edge}(x,y) \land \mathrm{Edge}(y,x))))))))$

3. PL representation

1. Cannot be expressed ~~to~~ due to inability to express universal statement.
2. Cannot be expressed due to inability to express universal statement.
3. Cannot be expressed due to inability to express existential statement.
4. Cannot be expressed due to inability to express existential statement.
5. Cannot be expressed due to inability to express existential and universal statement.

FOL Representation

Predicates

$\mathrm{Read}(x)$:	x can read
$\mathrm{Literate}(x)$:	x is literate
$\mathrm{Intelligent}(x)$:	x is intelligent
$\mathrm{Dolphin}(x)$:	x is a dolphin

Axioms

1. $\forall x\,(\mathrm{Read}(x) \rightarrow \mathrm{Literate}(x))$
2. $\forall x\,(\mathrm{Dolphin}(x) \rightarrow \neg \mathrm{Literate}(x))$
3. $\exists x\,(\mathrm{Dolphin}(x) \land \mathrm{Intelligent}(x))$
4. $\exists x\,(\mathrm{Intelligent}(x) \land \neg \mathrm{Read}(x))$
5. $\exists x\,(\mathrm{Dolphin}(x) \land \mathrm{Intelligent}(x) \land \mathrm{Read}(x)) \land \forall y\,((\mathrm{Dolphin}(y) \land \mathrm{Intelligent}(y) \land \mathrm{Read}(y)) \rightarrow \neg \mathrm{Literate}(y))$

Check satisfiability of fourth sentence

Let α be axiom 4.

$\neg\alpha$: $\neg(\exists x\,(\text{Intelligent}(x) \wedge \neg\text{Read}(x))) \equiv \forall x(\text{Intelligent}(x) \rightarrow \text{Read}(x))$

Using axiom 1-3 and $\neg\alpha$ in our KB, we derive their CNF form

S_1 : $\neg\text{Read}(x) \vee \text{Literate}(x)$

S_2 : $\neg\text{Dolphin}(x) \vee \neg\text{Literate}(x)$

S_3 : $\{\text{Dolphin}(a), \text{Intelligent}(a)\}$

S_4 : $\neg\text{Intelligent}(x) \vee \text{Read}(x)$

Using S_3 and S_4, we get $\{\text{Read}(a)\}$ ($\neg\text{Intelligent}(a) \wedge \text{Intelligent}(a)$ leads to contradiction) which is added to S_3.

S_3 : $\{\text{Dolphin}(a), \text{Intelligent}(a), \text{Read}(a)\}$

Using S_1 and S_3, we get $\{\text{Literate}(a)\}$ which is added to S_3.

S_3 : $\{\text{Dolphin}(a), \text{Intelligent}(a), \text{Read}(a), \text{Literate}(a)\}$

Using S_2 and S_3, we get $\{\neg\text{Literate}(a)\}$ which is added to S_3.

But Literate(a) and $\neg$Literate(a) leads to empty clause which is a contradiction.

Therefore, α entails KB and $\underset{\text{axiom 4}}{\underset{\downarrow}{\alpha}}$ is satisfiable.

Check satisfiability of fifth sentence

Let β be axiom 5.

$\neg\beta$: $\neg(\exists x\,(\text{Dolphin}(x) \wedge \text{Intelligent}(x) \wedge \text{Read}(x)) \wedge \forall y((\text{Dolphin}(y) \wedge \text{Intelligent}(y) \wedge \text{Read}(y)) \rightarrow \neg\text{Literate}(y)))$

: $\forall x\,(\neg\text{Dolphin}(x) \vee \neg\text{Intelligent}(x) \vee \neg\text{Read}(x)) \vee \exists y\,(\text{Dolphin}(y) \wedge \text{Intelligent}(y) \wedge \text{Read}(y) \wedge \text{Literate}(y))$

Using axiom 1-4 and $\neg\beta$ in our KB, we derive their CNF form

C_1 : $\neg\text{Read}(x) \vee \text{Literate}(x)$

C_2 : $\neg\text{Dolphin}(x) \vee \neg\text{Literate}(x)$

C_3 : $\{\text{Dolphin}(a), \text{Intelligent}(a)\}$

C_4 : $\{\text{Intelligent}(b), \neg\text{Read}(b)\}$

C_5 : $\underbrace{(\neg\text{Dolphin}(x) \vee \neg\text{Intelligent}(x) \vee \neg\text{Read}(x))}_{\rightarrow C_{51}} \vee \underbrace{\{\text{Dolphin}(c), \text{Intelligent}(c), \text{Read}(c), \text{Literate}(c)\}}_{\rightarrow C_{52}}$

Using C_2 and C_3, we get {¬Literate(a)} which is added to C_3

C_3: {Dolphin(a), Intelligent(a), ¬Literate(a)}

$\boxed{\text{Using } C_{51}}$

Using C_3 and C_{51}, we get {¬Read(a)} which is added to C_3

C_3^{51}: {Dolphin(a), Intelligent(a), ¬Literate(a), ¬Read(a)}

Using C_3 and C_1, we get {Literate(a)} which is added to C_3. But this leads to contradiction due to Literate(a) and ¬Literate(a) producing empty clause.

We now use C_{52}

$\boxed{\text{Using } C_{52}}$

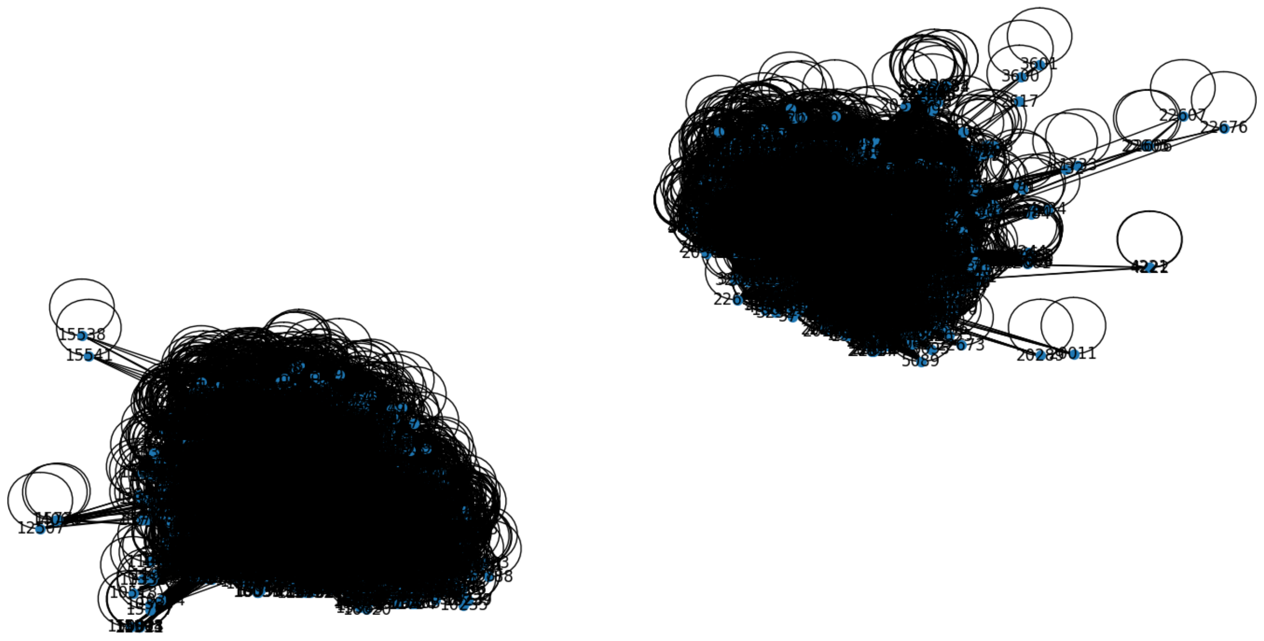
Using C_2 and C_{52}, we get {¬Dolphin(c)} or {¬Literate(c)}, either of which added to C_{52}, will lead to a contradiction by producing empty clause with their respective negated form.

Since using C_{51} or C_{52} gives empty clause which leads to a contradiction, β entails KB and $\underset{\substack{\downarrow \\ \text{axiom 5}}}{\beta}$ is satisfiable.

AI ASSIGNMENT — Knowledge Representation, Reasoning and Planning

Computational

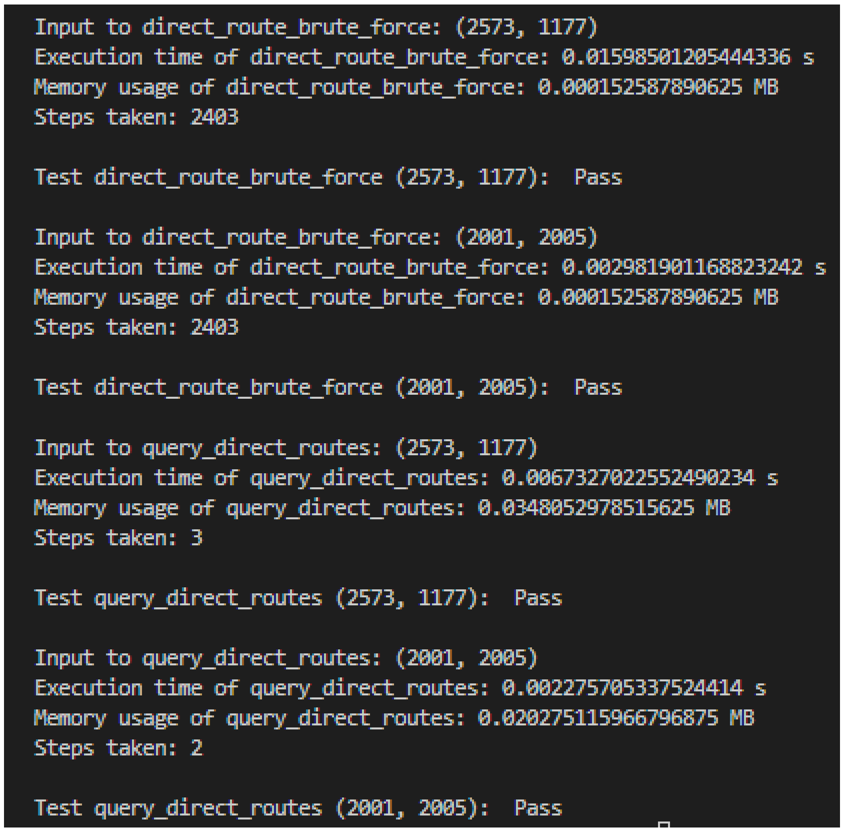
1.



route_to_stops graph

2.

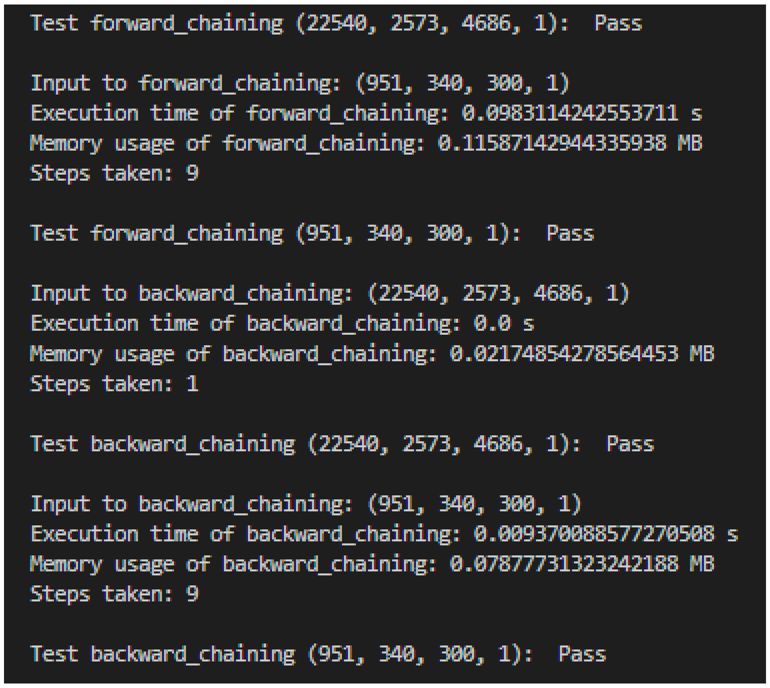
(a), (b), (c)



```
Input to direct_route_brute_force: (2573, 1177)
Execution time of direct_route_brute_force: 0.01598501205444336 s
Memory usage of direct_route_brute_force: 0.000152587890625 MB
Steps taken: 2403

Test direct_route_brute_force (2573, 1177):  Pass

Input to direct_route_brute_force: (2001, 2005)
Execution time of direct_route_brute_force: 0.002981901168823242 s
Memory usage of direct_route_brute_force: 0.000152587890625 MB
Steps taken: 2403

Test direct_route_brute_force (2001, 2005):  Pass

Input to query_direct_routes: (2573, 1177)
Execution time of query_direct_routes: 0.006732702255249023 4 s
Memory usage of query_direct_routes: 0.0348052978515625 MB
Steps taken: 3

Test query_direct_routes (2573, 1177):  Pass

Input to query_direct_routes: (2001, 2005)
Execution time of query_direct_routes: 0.002275705337524414 s
Memory usage of query_direct_routes: 0.020275115966796875 MB
Steps taken: 2

Test query_direct_routes (2001, 2005):  Pass
```

3.

(a), (b), (c)

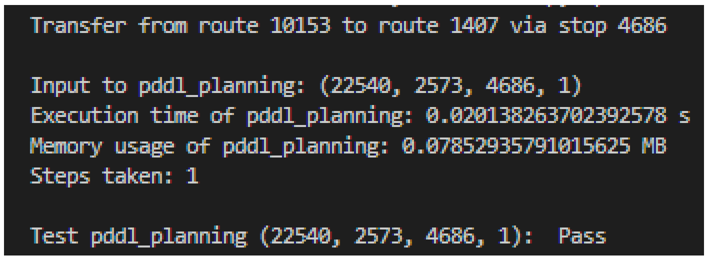


```
Test forward_chaining (22540, 2573, 4686, 1):  Pass

Input to forward_chaining: (951, 340, 300, 1)
Execution time of forward_chaining: 0.0983114242553711 s
Memory usage of forward_chaining: 0.11587142944335938 MB
Steps taken: 9

Test forward_chaining (951, 340, 300, 1):  Pass

Input to backward_chaining: (22540, 2573, 4686, 1)
Execution time of backward_chaining: 0.0 s
Memory usage of backward_chaining: 0.02174854278564453 MB
Steps taken: 1

Test backward_chaining (22540, 2573, 4686, 1):  Pass

Input to backward_chaining: (951, 340, 300, 1)
Execution time of backward_chaining: 0.009370088577270508 s
Memory usage of backward_chaining: 0.07877731323242188 MB
Steps taken: 9

Test backward_chaining (951, 340, 300, 1):  Pass
```

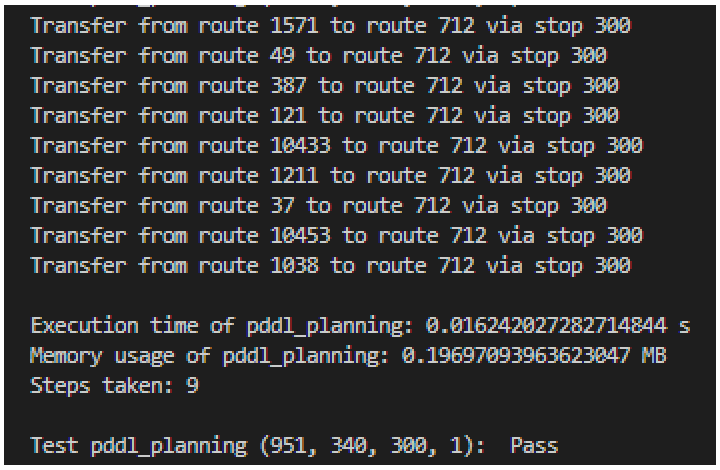
4.

4. 1.

(a), (b), (c)



```
Transfer from route 10153 to route 1407 via stop 4686

Input to pddl_planning: (22540, 2573, 4686, 1)
Execution time of pddl_planning: 0.020138263702392578 s
Memory usage of pddl_planning: 0.07852935791015625 MB
Steps taken: 1

Test pddl_planning (22540, 2573, 4686, 1):  Pass
```



```
Transfer from route 1571 to route 712 via stop 300
Transfer from route 49 to route 712 via stop 300
Transfer from route 387 to route 712 via stop 300
Transfer from route 121 to route 712 via stop 300
Transfer from route 10433 to route 712 via stop 300
Transfer from route 1211 to route 712 via stop 300
Transfer from route 37 to route 712 via stop 300
Transfer from route 10453 to route 712 via stop 300
Transfer from route 1038 to route 712 via stop 300

Execution time of pddl_planning: 0.016242027282714844 s
Memory usage of pddl_planning: 0.19697093963623047 MB
Steps taken: 9

Test pddl_planning (951, 340, 300, 1):  Pass
```

Analysis of routes from each algorithm

1. Forward Chaining: It is likely to produce the optimal route because it starts from the initial stop and systematically explores paths. However, it may not always prioritize the most efficient path early, especially when constraints such as transfers and via stops make the problem more complex.

2. Backward Chaining: This method can quickly eliminate non-feasible routes by starting at the goal, but it may return suboptimal paths if there are multiple ways to reach the start stop, especially when paths have multiple valid interchanges.

3. PDDL: If the PDDL problem is well-defined with clear constraints on via stops and maximum transfers, it is likely to find an optimal path. However, the flexibility of PDDL could lead to suboptimal results if constraints are not fine-tuned, or the problem space is not constrained well enough.

Effect of constraints on optimality

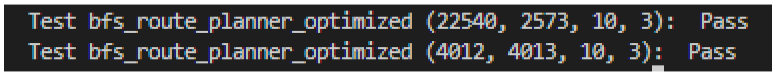
1. Via Stop Constraint: All algorithms produce the same results if this constraint is strictly enforced. However, if an algorithm like PDDL doesn't efficiently model this requirement, it could produce suboptimal routes.

2. Transfer Limits: This constraint can have a significant impact on suboptimality. Both forward and backward chaining can return suboptimal paths if they explore too many interchanges before finding an efficient path. PDDL could also return suboptimal paths if the transfer limit is too high, allowing more complex routes.

Comparison of algorithms

Algorithm	Optimality
Forward Chaining	Likely to find an optimal route, especially for simple, direct routes. Might explore inefficient paths first if there are multiple valid routes and interchanges.
Backward Chaining	Efficient at finding optimal paths when starting from a clear goal. Could return suboptimal paths when many valid paths exist with multiple interchanges.
PDDL Planning	Flexible and likely to find optimal routes with well-defined constraints. Suboptimal results possible if constraints aren't fine-tuned or if the model is too flexible.

4. 2.



```
Test bfs_route_planner_optimized (22540, 2573, 10, 3):  Pass
Test bfs_route_planner_optimized (4012, 4013, 10, 3):  Pass
```