Propositional logic

... continued

So far …

- Knowledge base
- Inference algorithms
- Resolution

- Resolution generates the Resolution closure
- Construction of closure makes resolution complete

- In many practical scenarios, no need for such power!

Forward and Backward Chaining

- Horn Form (restricted)
  - KB = conjunction of Horn clauses
  - Horn clause = definite clause or goal clause
    - Definite clause : $\bigwedge_j \alpha_j \Rightarrow \beta$
    - Goal clause : $\bigwedge_j \alpha_j \Rightarrow False$
  - e.g., KB: $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$
- Inference with Horn clauses: forward chaining or backward chaining algorithms. Easy to interpret, run in linear time
- Inference is Modus Ponens (for Horn Form): sound for Horn KB

$$\frac{\alpha_1, \dots, \alpha_k; \bigwedge_j \alpha_j \Rightarrow \beta}{\beta}$$

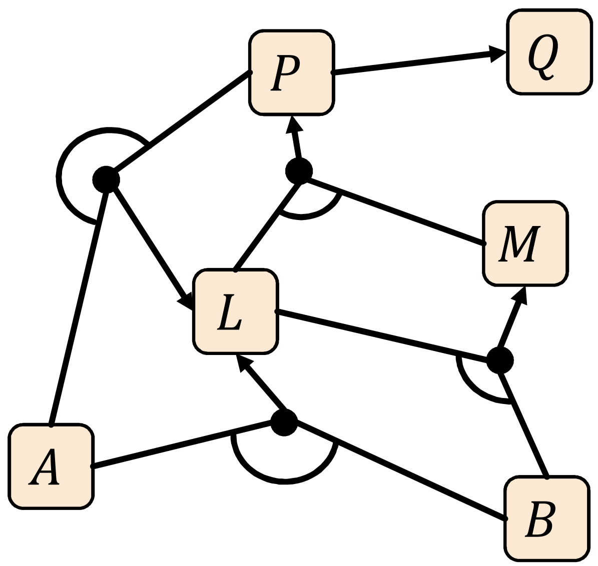
Forward Chaining (FC)

- Idea: Fire any rule whose premise is satisfied in the KB, add its conclusion to the KB, repeat until query is found

KB of horn clauses

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

AND-OR graph



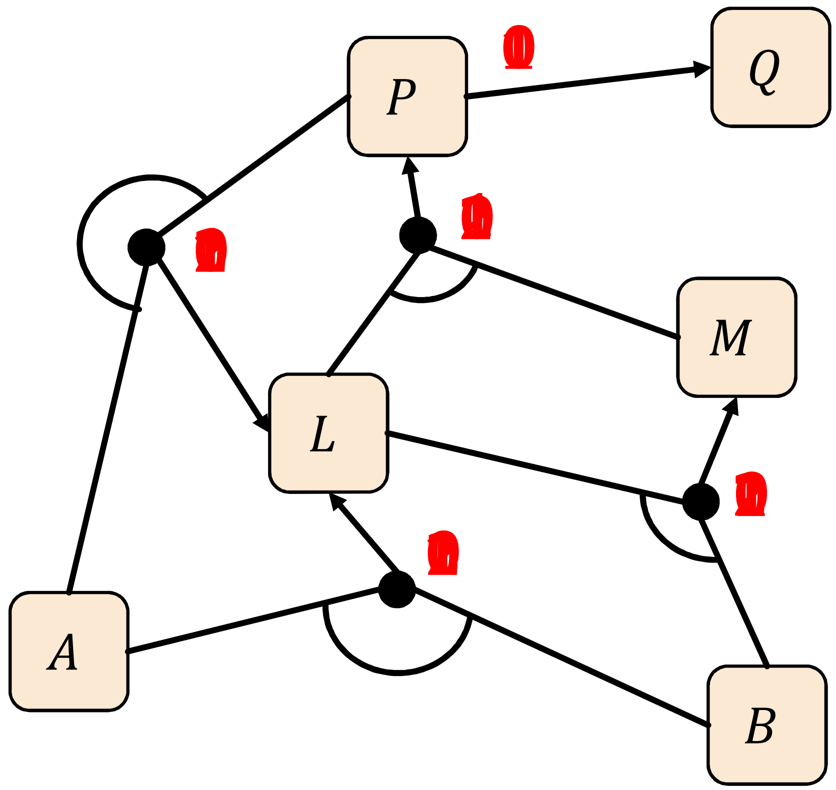
Forward Chaining (FC) Algorithm

- For every rule c, let $\mathrm{count}(c)$ be the number of symbols in c's premise.
- For every symbol s, let $\mathrm{inferred}(s)$ be initially $False$
- Let agenda be a queue of symbols (initially containing all symbols known to be true.
- While $\mathrm{agenda} \neq \emptyset$:
  - pop a symbol p from $agenda$; if it is q we're done
  - Set $inferred(p) = True$
  - For each clause $c \in KB$ such that p is in the premise of c, decrement $count(c)$. If $count(c) = 0$, add c's conclusion to $agenda$.

Forward chaining is sound and complete for Horn KB

Forward Chaining Example

Iteration 1: $[A, B]$
Iteration 2: $[B]$
Iteration 3: $[] \Rightarrow [L]$
Iteration 4: $[] \Rightarrow [M]$
Iteration 5: $[] \Rightarrow [P]$
Iteration 6: $[] \Rightarrow [L, Q]$
Iteration 7: $[Q]$
Iteration 8: $[]$



Proof of Completeness

FC derives every atomic sentence entailed by Horn KB

1. Suppose FC reaches a fixed point where no new atomic sentences are derived
2. Consider the final state as a model m that assigns true/false to symbols based on the inferred table
3. Every clause in the original KB is true in m

   $$\alpha_1 \land \cdots \land \alpha_k \Rightarrow \beta$$

4. Hence, m is a model of KB
5. If $KB \vDash q$, then q is true in every model of KB, including m.

Backward Chaining (BC)

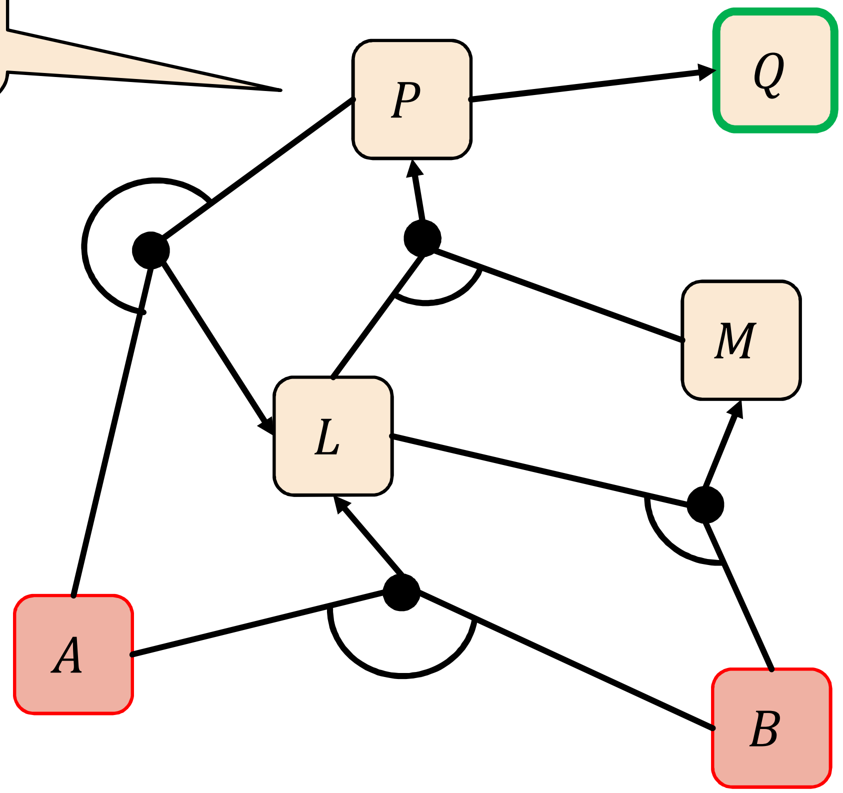
Backtracking depth-first search algorithm

Idea: work backwards from the query q

- To prove q by BC,
  - check if q is known already, or
  - prove by BC the premise of some rule concluding q
- Avoid loops: check if new subgoal is already on the goal stack
- Avoid repeated work: check if new subgoal
  - has already been proven true, or
  - has already failed

Backward Chaining Example

Hit a loop! Try something else



Forward vs. Backward Chaining

FC = data-driven reasoning

- e.g., object recognition, routine decisions
- May do a lot of work that is irrelevant to the goal

BC = goal-driven reasoning

- e.g., Where are my keys? How do I get into Google?
- Complexity of BC can be sublinear in $|KB|$.

Proof Methods

Applying inference rules (aka theorem proving)

- Generation of new sentences from old
- Proof = sequential application of inference rules
- Inference rules are actions in a search algorithm
- Proof can be more efficient: ignores irrelevant propositions
- Transformation of sentences into a normal form

Model checking

- Truth table enumeration (time complexity exponential in n)
- Improved backtracking, e.g. DPLL
- local search in model space (sound but incomplete), e.g. min-conflicts-like hill-climbing algorithm

Efficient Propositional Model Checking

Two families of efficient algorithms for propositional model checking:

- Complete backtracking search algorithms
  - DPLL algorithm (Davis, Putnam, Logemann, Loveland)
- Incomplete local search algorithms
  - WalkSAT algorithm

These algorithms test a sentence for satisfiability; used for inference.

Recall: Satisfiability is connected to entailment via

$KB \models \alpha$ if and only if $(KB \land \neg\alpha)$ is unsatisfiable

DPLL Algorithm

How are DPLL and CSP related?

Determine if a given CNF formula $\phi = C_1 \wedge \cdots \wedge C_m$ is satisfiable

Improvements over truth table enumeration:

1. Early termination

   (a) A clause is true iff any literal in it is true.

   (b) The formula ϕ is false if any clause is false.

2. Pure symbol heuristic

   Least constraining value

   Pure symbol: always appears with the same “sign” in all clauses.

   e.g., in $(A \vee \neg B) \wedge (\neg B \vee \neg C) \wedge (C \vee A)$, A and B are pure; C is impure.

   Make a pure symbol’s literal true: Doing this can never make a clause false.

   Ignore clauses that are already true in the model constructed so far.

3. Unit clause heuristic

   Most constrained variable

   Unit clause: only one literal in the clause.

   The only literal in a unit clause must be true.

DPLL Algorithm

function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of s
  symbols ← a list of the proposition symbols in s
  return DPLL(*clauses*, *symbols*, { })

---

function DPLL(*clauses*, *symbols*, *model*) **returns** *true* or *false*

  if every clause in *clauses* is true in *model* **then return** *true*
  if some clause in *clauses* is false in *model* **then return** *false*
  P, *value* ← FIND-PURE-SYMBOL(*symbols*, *clauses*, *model*)
  P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  value ← FIND-UNIT-CLAUSE(*clauses*, *model*)
  P is non-null **then return** DPLL(*clauses*, *symbols* – P, *model* ∪ {P=*value*})
  ← FIRST(*symbols*); *rest* ← REST(*symbols*)
  return DPLL(*clauses*, *rest*, *model* ∪ {P=*true*}) **or**
      DPLL(*clauses*, *rest*, *model* ∪ {P=*false*}))

Early Termination

Try to apply heuristics

If it doesn't work, brute force.

WalkSAT Algorithm

- Incomplete, local search algorithm
- Evaluation function: minimize the number of unsatisfied clauses
- Balance between greediness and randomness

WALKSAT Algorithm

CNF formula: $\phi = C_1 \wedge \cdots \wedge C_m$

1. Start with a random variable assignment $\ell_1 \dots \ell_n$, where $\ell_i \in \{True, False\}$
2. If $\vec{\ell}$ satisfies the formula return $\vec{\ell}$.
3. Choose a random unsatisfied clause $C_j \in \phi$
4. With probability p flip the truth value of a random symbol $x_i \in C_j$; else flip a symbol $x_i \in C_j$ that maximizes number of satisfied clauses in ϕ.
5. Repeat steps 2-4 $MaxFlips$ times.

Why is WalkSat incomplete?

How are WALKSAT and local search related?

Inference-Based Agents in the Wumpus World

A wumpus-world agent using propositional logic:

$\neg P_{1,1}$

$\neg W_{1,1}$

$B_{i,j} \Leftrightarrow (P_{i,j+1} \vee P_{i,j-1} \vee P_{i+1,j} \vee P_{i-1,j})$

$S_{i,j} \Leftrightarrow (W_{i,j+1} \vee W_{i,j-1} \vee W_{i+1,j} \vee W_{i-1,j})$

$W_{1,1} \vee W_{1,2} \vee \cdots \vee W_{4,4}$

$\neg W_{1,1} \vee \neg W_{1,2}$

$\neg W_{1,1} \vee \neg W_{1,3}$

...

Each i, j rule is its own proposition

There is exactly one Wumpus

64 distinct proposition symbols, 155 sentences

First-Order Logic (FOL)

AIMA Chapter 8

Outline

- Why FOL?
- Syntax and semantics of FOL
- Using FOL
- Wumpus world in FOL
- Knowledge engineering in FOL

Propositional Logic

Pros

- Declarative: tells agent what it needs to know to operate in its environment. No need to specify exact behavior
- Allows partial information via disjunction and negation (unlike many other data structures)
- Compositional: meaning of $A \wedge B$ derived from meanings of A and B.
- Context independent and unambiguous

Cons

- Limited expressive power: cannot concisely say “pits cause breezes in adjacent squares”.

First-Order Logic

- Propositional logic assumes that the world contains facts
- First-order logic (like natural language) assumes that the world contains
  - Objects: people, houses, numbers, colors, baseball games, wars, …
  - Relations: unary relations or properties such as red, round, prime, …, or more general n-ary relations such as brother of, bigger than, part of, comes between, …
  - Functions: father of, best friend, one more than, plus, …

Syntax of FOL: Basic Elements

Type	Examples
Constants	John, 2, NUS,…
Predicates (relations)	$Brother(x, y), x > y, \ldots$
Functions	$\sqrt{x}, LeftLeg(x), \ldots$
Variables	x, y, a, b
Connectives	$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall, \exists$

Atomic Sentences

Functions can be viewed as complex names for constants

Term: *constant* or *variable* or $function(x_1, \ldots, x_n)$

Atomic sentence: $predicate(x_1, \ldots, x_n)$ or $x_1 = x_2$

E.g.,

- $Brother(John, Richard)$
- $Length\big(LeftLeg(Richard)\big) = Length\big(LeftLeg(John)\big)$

Complex Sentences

Constructed from atomic sentences via connectives

$$\neg\alpha, \alpha_1 \land \alpha_2, \alpha_1 \lor \alpha_2, \alpha \Rightarrow \beta, \alpha \Leftrightarrow \beta$$

E.g.,

- $Sibling(John, Richard) \Rightarrow Sibling(Richard, John)$
- $(a \leq b) \lor (a > b)$
- $(1 > 2) \land \neg(1 > 2)$

Truth in First-Order Logic

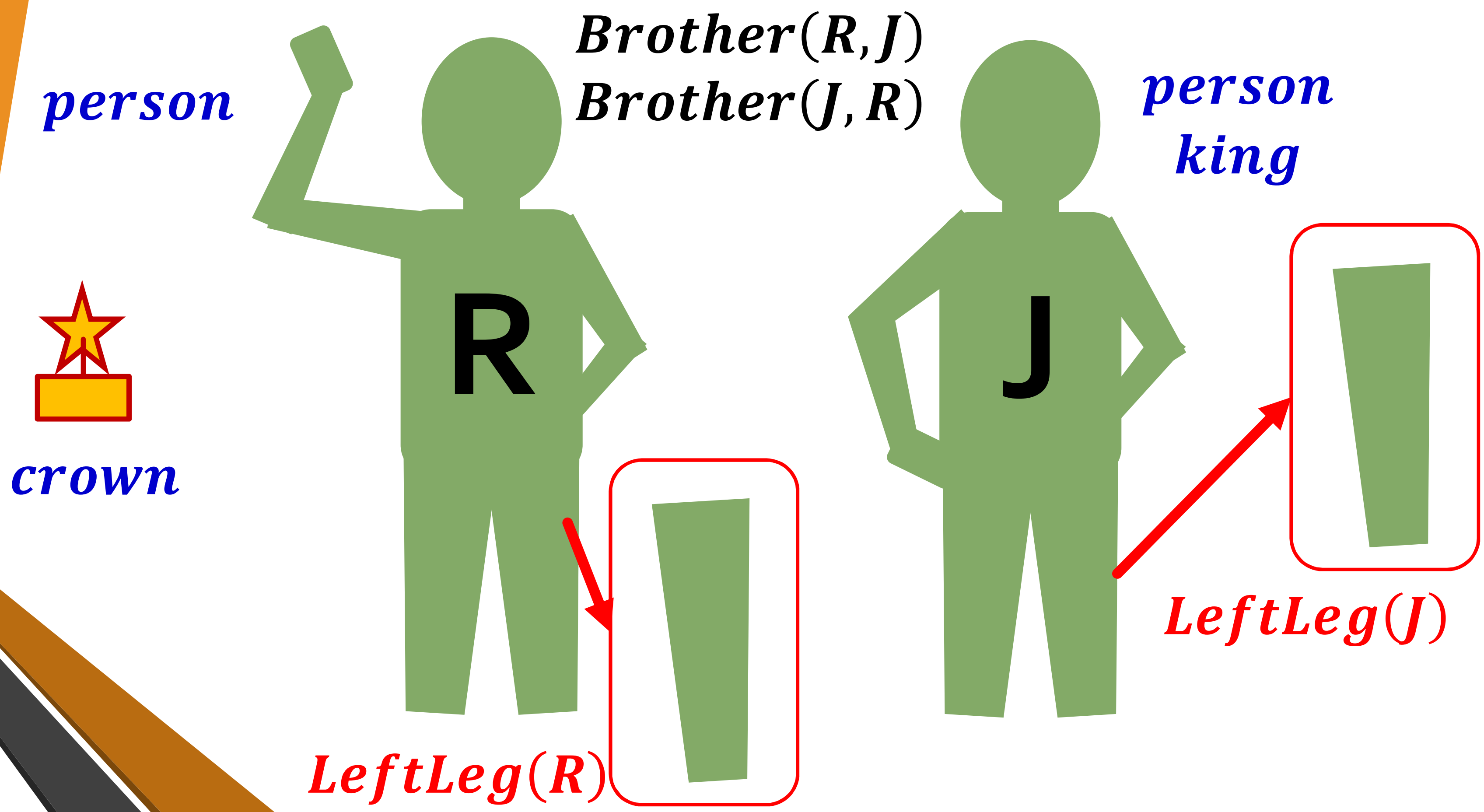
- Sentences are true in a model
- Model comprises a set of objects (domain elements) and an interpretation
- Interpretation specifies referents for

Objects	$\rightarrow$	Constants
Relations	$\rightarrow$	Predicates
Functions	$\rightarrow$	Function Symbols

- An atomic sentence $predicate(x_1, \ldots, x_n)$ is true in a given model if the relation referred to by $predicate$ holds among the objects referred to by $x_1, \ldots, x_n$.

Models for FOL: Example 1

Model contains 5 objects, 2 binary relations (black), 3 unary relations (blue), 1 unary function (red)



Universal Quantification

- $\forall < variables >: < sentence >$
- e.g., everyone at NUS is smart: $\forall x: x \in NUS \Rightarrow Smart(x)$
- $\forall x: P(x)$ is true in a given model if P is true with x referring to each possible object in the model
- Roughly speaking, it is equivalent to the conjunction of instantiations of P

$$
\begin{aligned}
&\text{Alice} \in NUS \Rightarrow Smart(\text{Alice}) \\
&\wedge \text{Bob} \in NUS \Rightarrow Smart(\text{Bob}) \\
&\wedge \text{Claire} \in NUS \Rightarrow Smart(\text{Claire}) \\
&\ldots
\end{aligned}
$$

A Common Mistake to Avoid

- Typically, $\Rightarrow$ is the main connective with $\forall$
- Common mistake: using $\wedge$ as the main connective with $\forall$:

$$\forall x: x \in NUS \wedge Smart(x)$$

What does the above mean?

Existential Quantification

- $\exists < vars >: < sentence >$

  e.g., someone at NUS is smart: $\exists x: x \in NUS \wedge Smart(x)$

- $\exists x: P$ is true in a given model if P is true with x referring to at least one object in the model

- Roughly speaking, it is equivalent to the disjunction of instantiations of P

$$
\begin{aligned}
& Alice \in NUS \wedge Smart(Alice) \\
& \vee\, Bob \in NUS \wedge Smart(Bob) \\
& \vee\, Claire \in NUS \wedge Smart(Claire) \\
& \ldots
\end{aligned}
$$

Another Common Mistake to Avoid

- Typically, $\wedge$ is the main connective with $\exists$
- Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

$$\exists x: x \in NUS \Rightarrow Smart(x)$$

What does this mean?

Negation

- Negation of $\forall x: P(x)$ is $\exists x: \neg P(x)$
- Negation of $\exists x: P(x)$ is $\forall x: \neg P(x)$

$$\forall x: \left(\exists y: P(x, y)\right) \vee \left(\forall z: \exists y: \left(Q(x, y, z) \wedge P(y, z)\right)\right)$$

$$\Downarrow$$

$$\exists x: \left(\forall y: \neg P(x, y)\right) \wedge \left(\exists z: \forall y: \left(\neg Q(x, y, z) \vee \neg P(y, z)\right)\right)$$

Equality

- $x_1 = x_2$ is true under a given interpretation iff x_1 and x_2 refer to the same object
- With function: e.g., $Father(\text{John}) = \text{Henry}$
- With negation: e.g., definition of $Sibling$ in terms of $Parent$:

$$\forall x, y: Sibling(x, y)$$
$$\Leftrightarrow \Big(\neg(x = y)$$
$$\wedge \Big(\exists m, f: \neg(m = f) \wedge Parent(m, x)$$
$$\wedge Parent(f, x) \wedge Parent(m, y)$$
$$\wedge Parent(f, y)\Big)\Big)$$

Interacting with FOL KBs

- A Wumpus-world agent is using a FOL KB and perceives a smell, a breeze, and glitter at $t = 5$:

  TELL(KB, *Percept*([*Smell*, *Breeze*, *Glitter*, *None*, *None*], 5))

  ASK(KB, $\exists a$ *BestAction*(a, 5))

  - Quantified query: does the KB entail some best action at $t = 5$? Answer: Yes.

- ASKVARS(KB, S) returns the binding list or substitutions such that $KB \vdash S$

  - e.g., ASKVARS(KB, $\exists a$ *BestAction*(a, 5))
  - Answer: $\{a/Grab\}$ ← substitution (binding list)

KB for the Wumpus World

- Perception rule
  - Process agent’s inputs
  - “If observed a glitter at time t, set $\text{Glitter}(t) = True$”
- Reflex rule
  - Process agent’s outputs
  - $\forall t\text{: Glitter}(t) \Rightarrow \text{BestAction}(Grab, t)$
- Above rules yield $\text{BestAction}(\text{Grab}, 5)$

How would we write the above rule in propositional logic?

KB for the Wumpus World

Properties of squares:

- $\forall x, y, a, b\colon \mathrm{Adjacent}([x, y], [a, b]) \Leftrightarrow$
$$\big(x = a \wedge (y = b - 1 \vee y = b + 1)\big)$$
$$\vee \big(y = b \wedge (x = a - 1 \vee x = a + 1)\big)$$
- $\forall s, t\colon \mathrm{At}(\mathrm{Agent}, s, t) \wedge \mathrm{Breeze}(t) \Rightarrow \mathrm{Breezy}(s)$

Squares are breezy near a pit:

- $\forall s\colon \mathrm{Breezy}(s) \Leftrightarrow \exists r\colon \mathrm{Adjacent}(r, s) \wedge \mathrm{Pit}(r)$

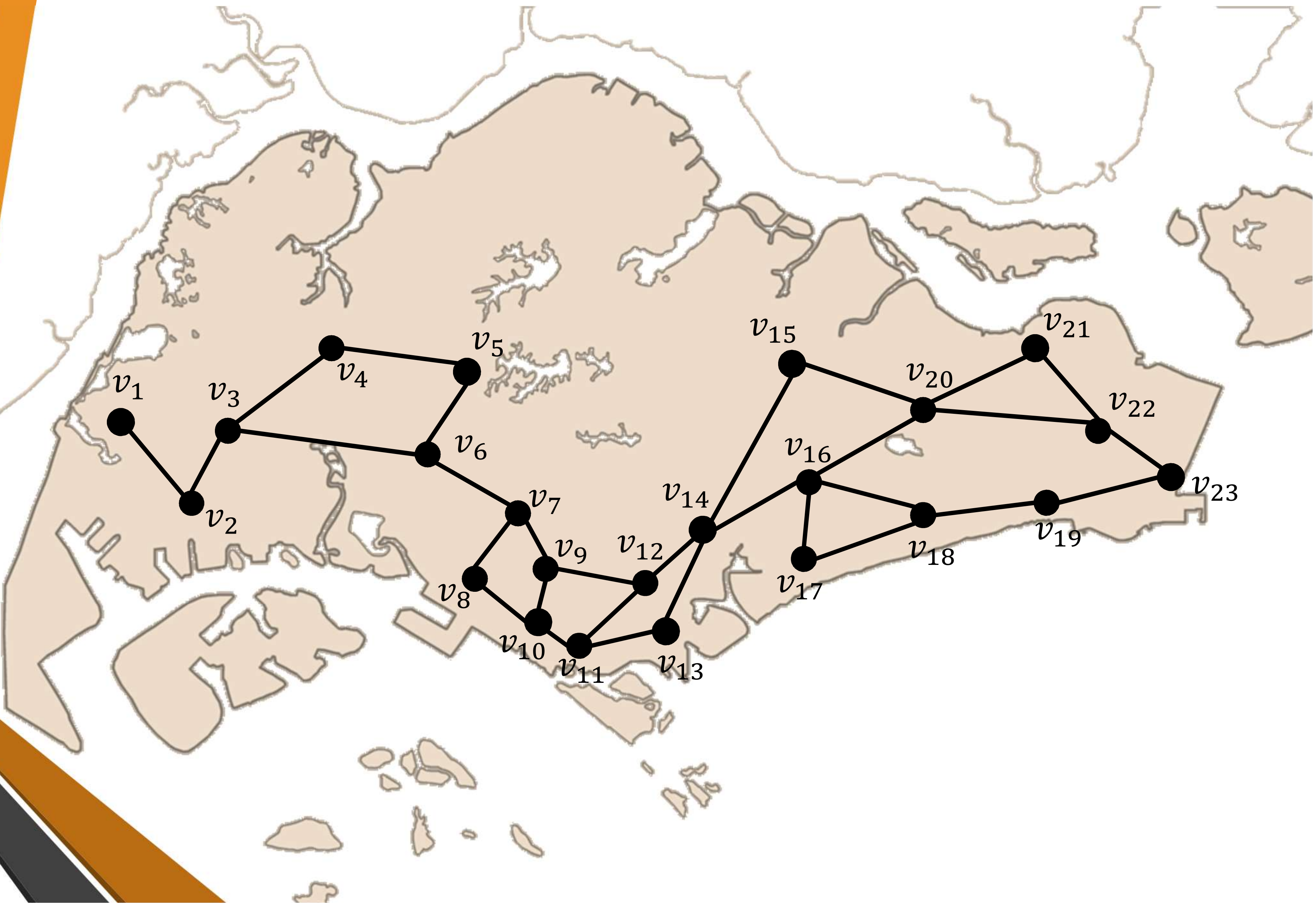
Knowledge Engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

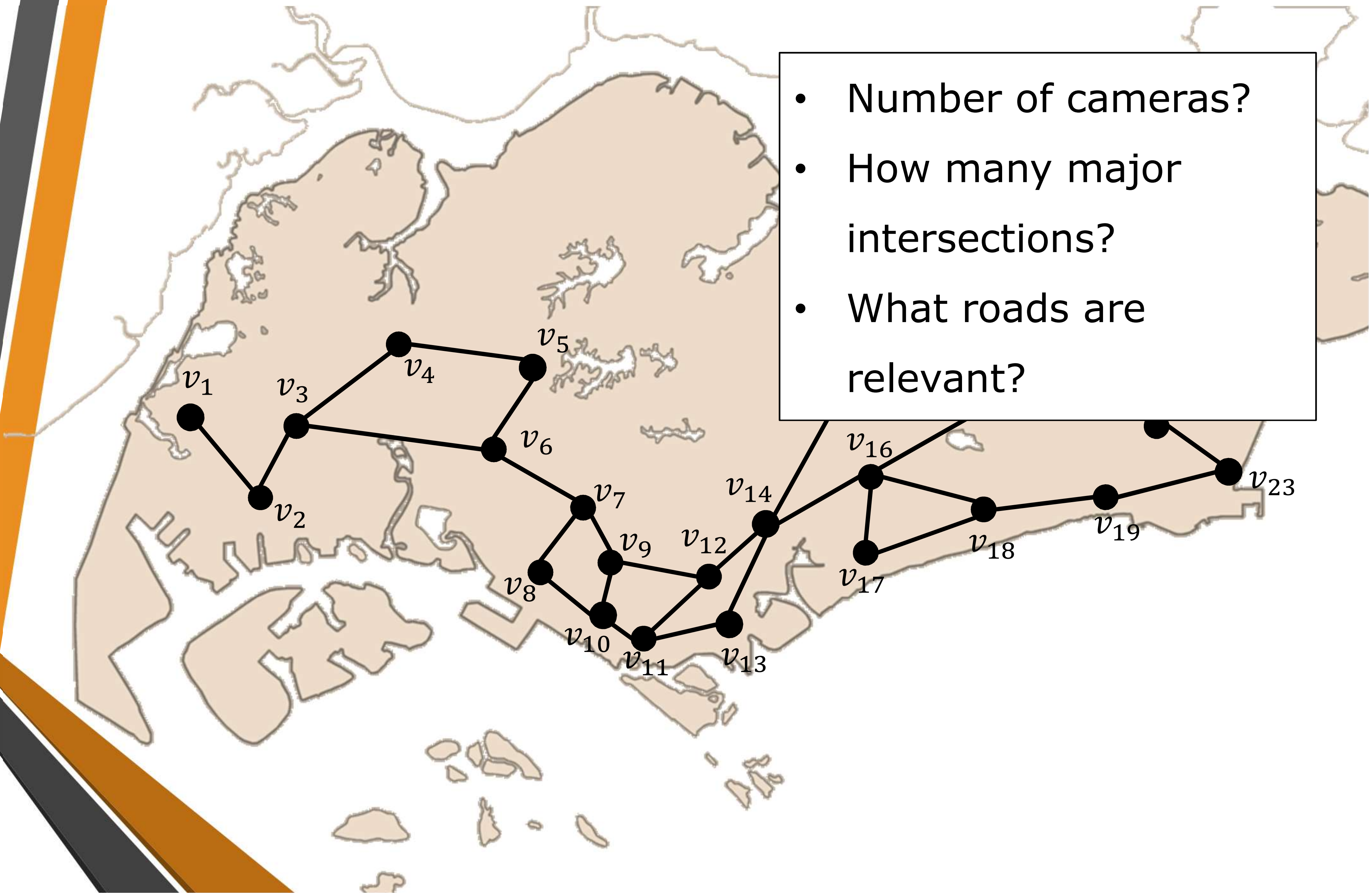
Optimal Traffic Management

- We are approached by the Singapore Police
- Want to optimally position traffic cameras in major intersections so as to cover all relevant roads.
- A camera in an intersection also covers adjacent ones.
- Please help!

Identify the Task



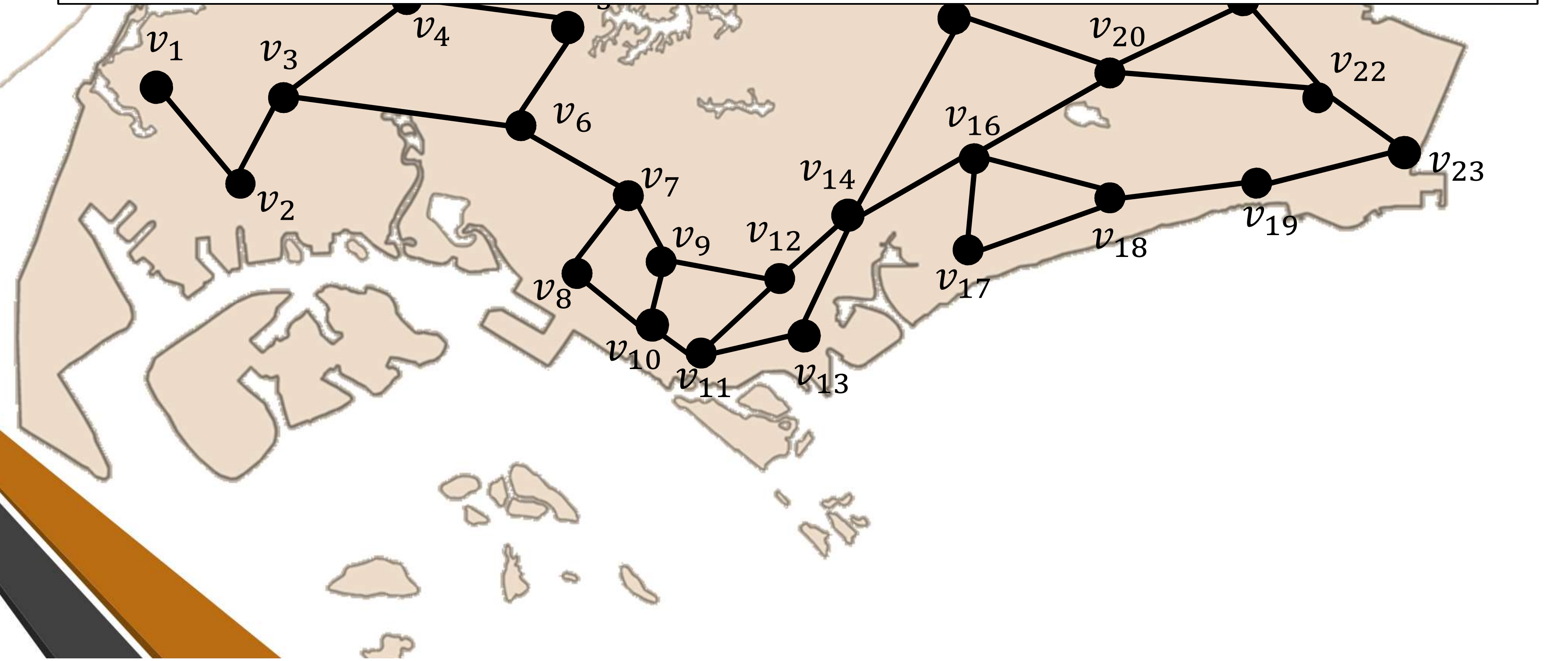
Assemble Relevant Knowledge



- Number of cameras?
- How many major intersections?
- What roads are relevant?

Decide on Vocabulary

- V – set of intersections
- $\mathrm{edge}(u, v) \in \{0,1\}$ – is there a road connecting u and v
- $c(v) \in \{0,1\}$ - there is a camera in location v.
- Maximal number of cameras - $k \in \mathbb{Z}_+$



Encode General Domain Knowledge

- Edges are bidirectional –

$$\forall u, v: \mathrm{edge}(u, v) \Leftrightarrow \mathrm{edge}(v, u)$$

- Coverage property –

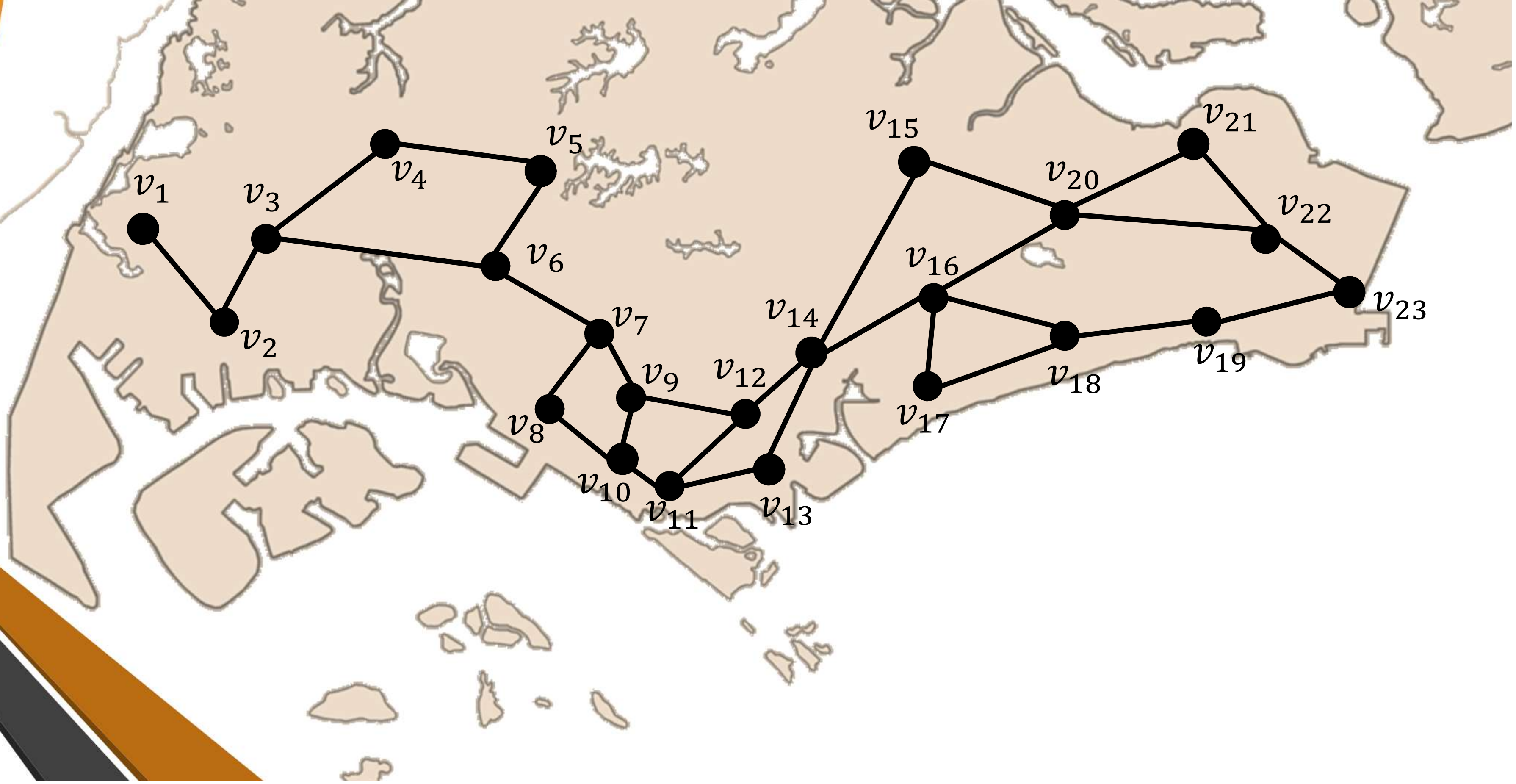
$$\mathrm{Covered}(u, v) \Leftrightarrow c(v) \vee c(u)$$

- Total coverage – $\mathrm{TotalCover}(V) \Leftrightarrow \forall e = \{u, v\} \in E: \mathrm{Covered}(e)$
- Is $U \subseteq V$ providing total coverage?

$$\mathrm{IsCovering}(U) \Leftrightarrow \left(\bigwedge_{u \in U} c(u)\right) \wedge \left(\bigwedge_{v \in V \setminus U} \neg c(v)\right) \wedge \mathrm{TotalCover}(V)$$

Encode the Specific Instance

- $V = \{v_1, \dots, v_{23}\}$
- $\text{edge}(v_1, v_2)$, $\text{edge}(v_2, v_3)$, $\text{edge}(v_3, v_4)$, $\text{edge}(v_3, v_6)$, …



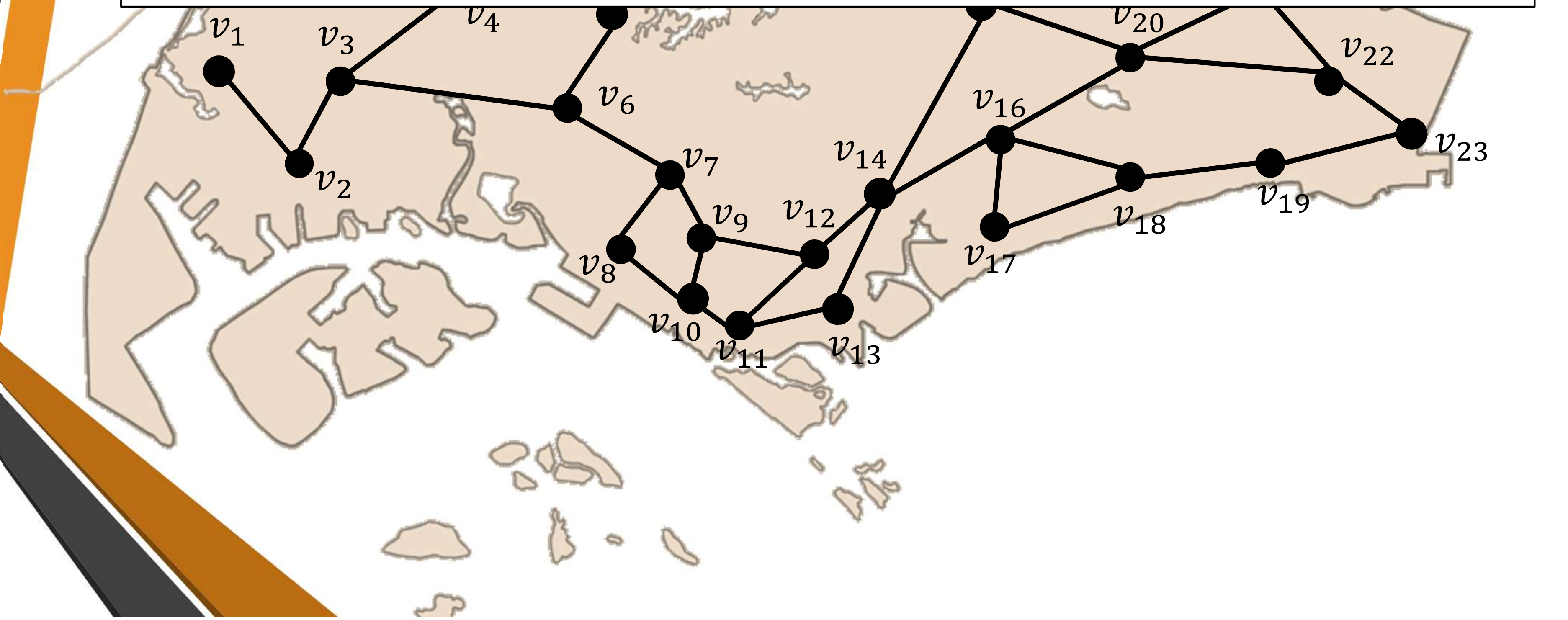
Pose Queries

- Is there a solution using k cameras?

$$\exists u_1, \ldots, u_k \colon \mathrm{IsCovering}(\{u_1, \ldots, u_k\})$$

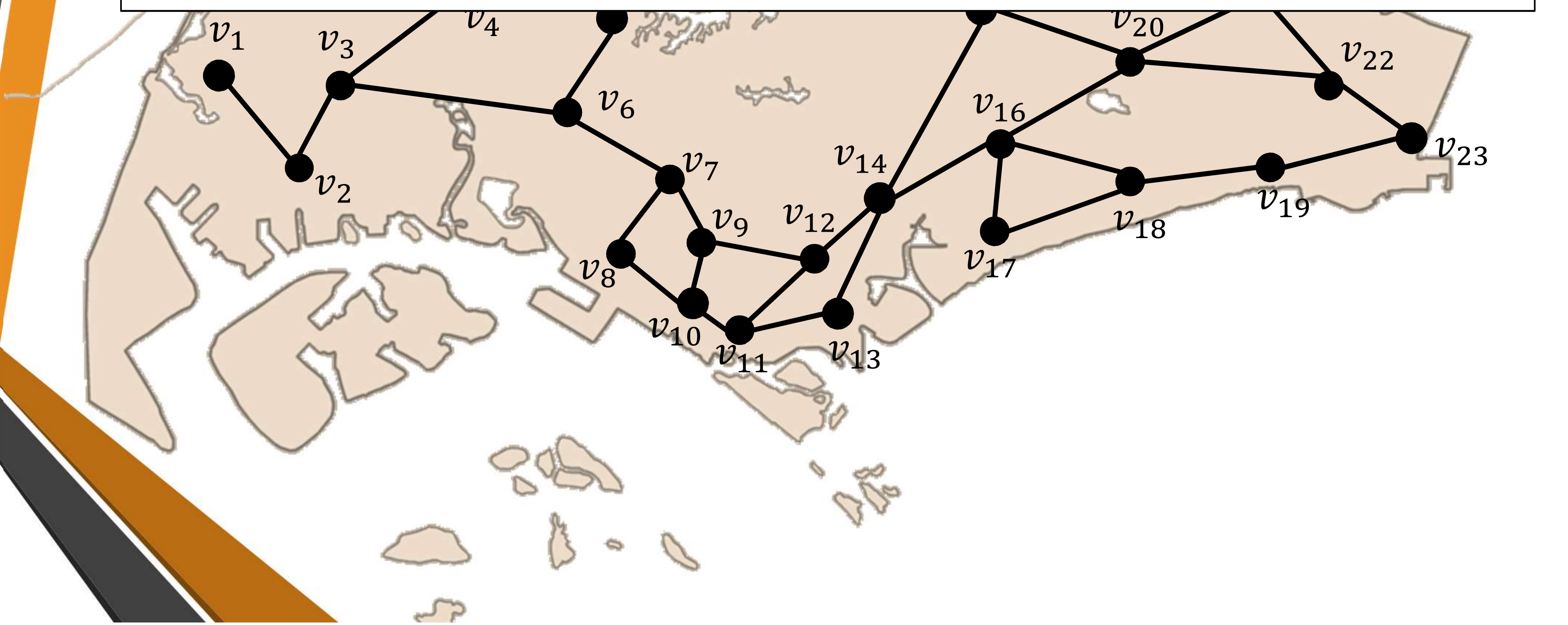
- Will a specific solution work?

$$\mathrm{IsCovering}(\{v_2, v_4, v_6, v_{10}, v_{12}, v_{16}\})$$



Debug Database

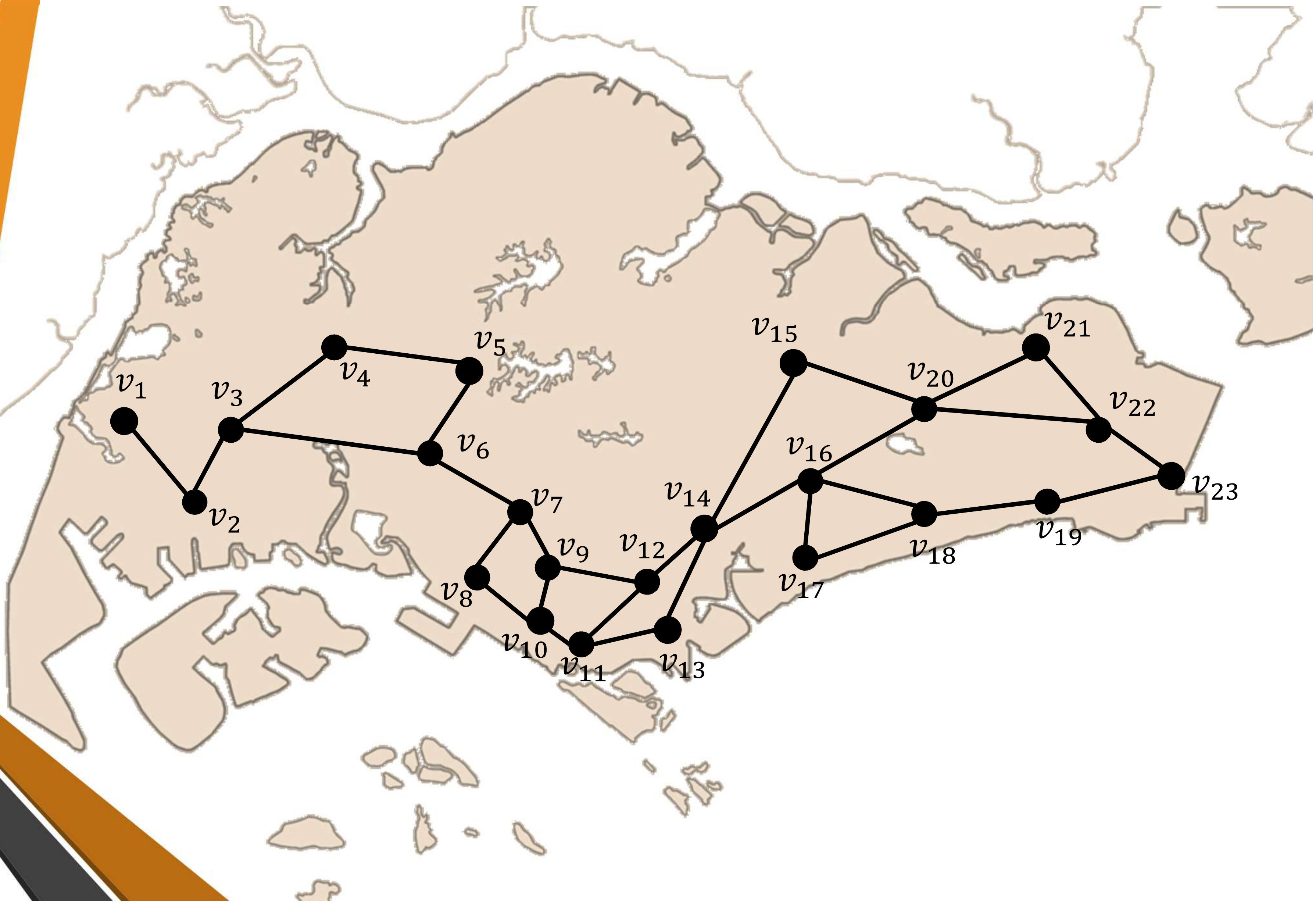
- $\forall u, v\colon \mathrm{edge}(u, v) \Rightarrow u \in V \land v \in V$
- $\forall u, v\colon \mathrm{edge}(u, v) \Rightarrow u \neq v$
- $\forall v\colon c(v) \Rightarrow v \in V$
- …



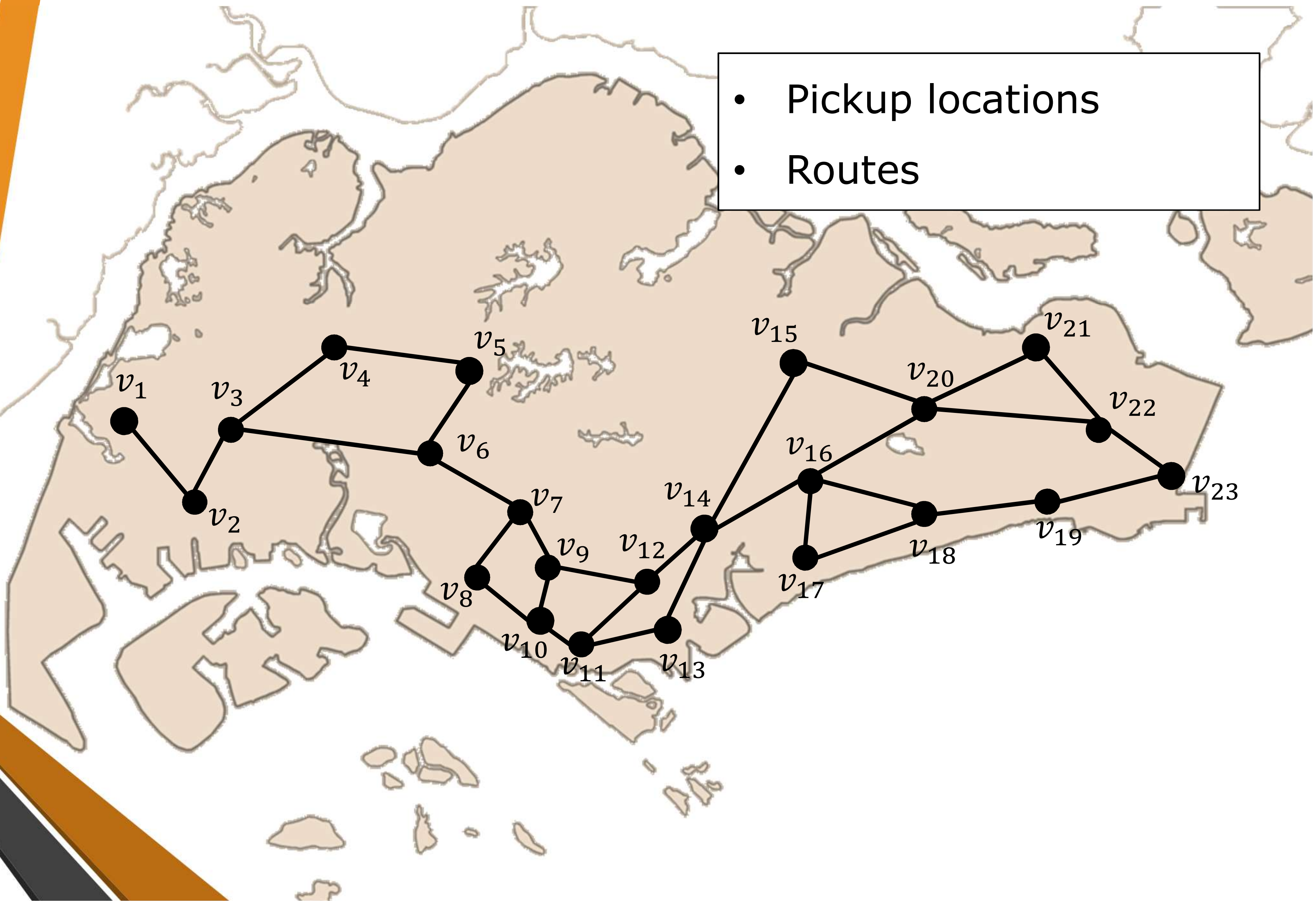
Waste Disposal

- We are approached by a Waste Disposal Service
- Want to optimally collect garbage from various locations.
- Don’t want to visit same location twice

Identify the Task

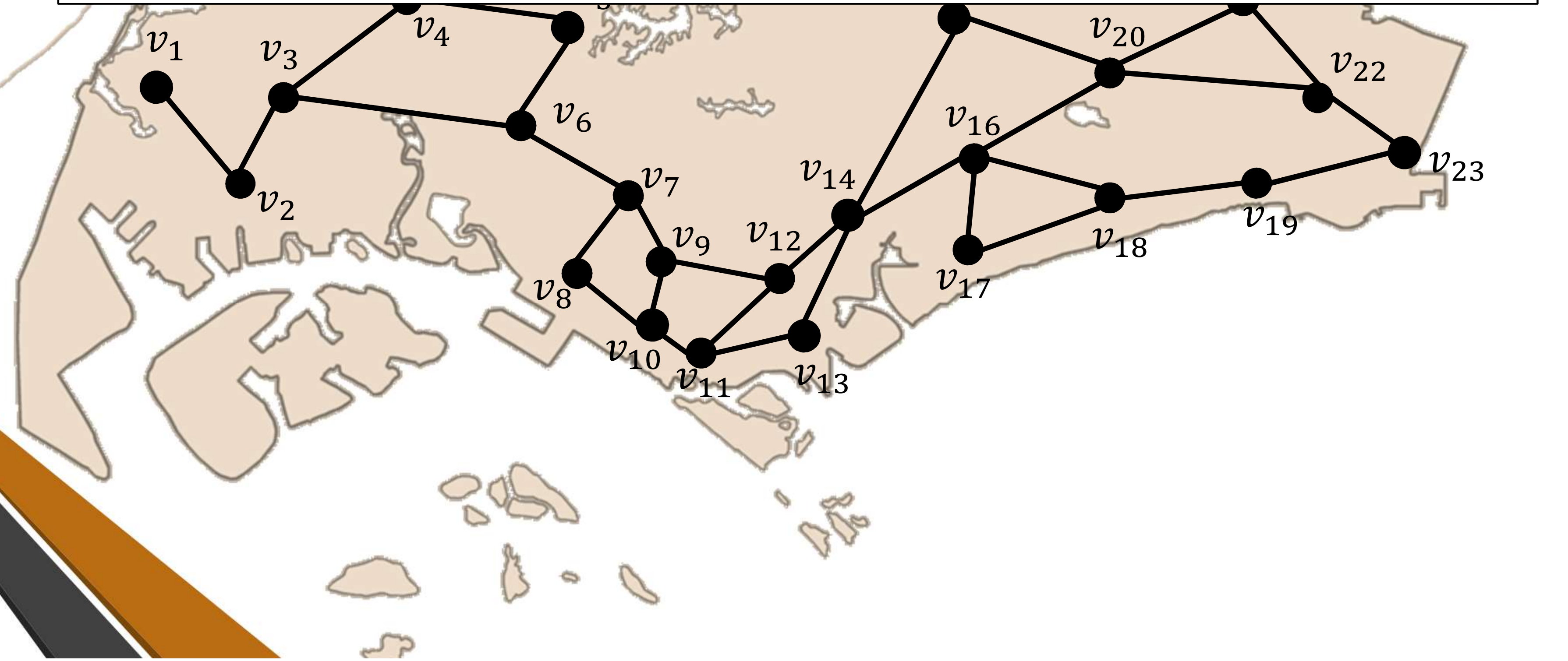


Assemble Relevant Knowledge



Decide on Vocabulary

- V – set of locations
- $\mathrm{edge}(u, v) \in \{0,1\}$ – is there a road connecting u and v
- $\mathrm{next}(u, v) \in \{0,1\}$: we move from u to v.
- Start location: $\mathrm{start}(v)$



Encode General Domain Knowledge

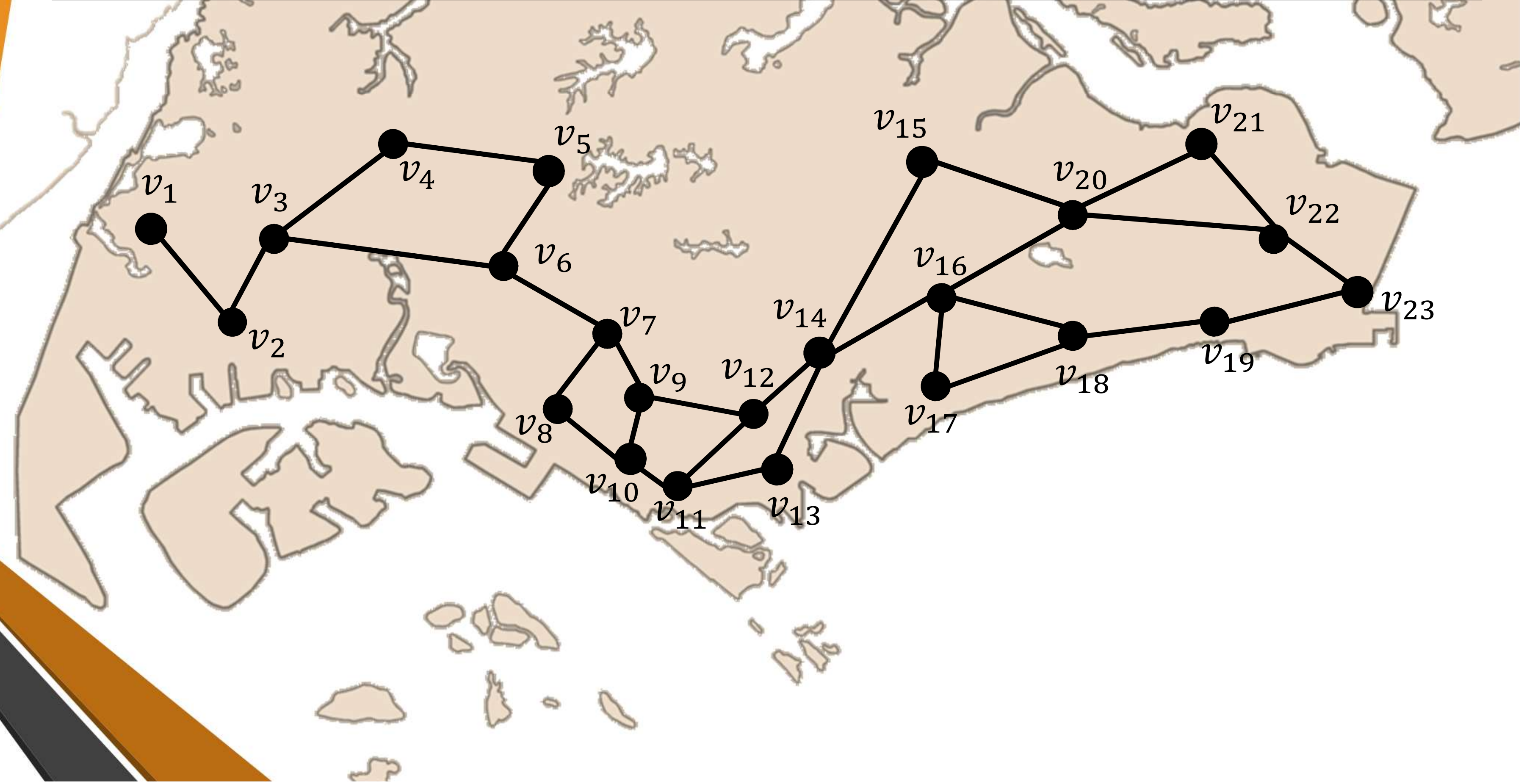
- $\text{edge}(u, v) \in \{0,1\}$: there is an edge between u and v.
- Start location is unique:

$$\exists v_0 : \left(v_0 \in V \wedge \text{start}(v_0)\right) \wedge \left(\forall v : \text{start}(v) \Rightarrow (v = v_0)\right)$$

- Can only travel on edges: $\text{next}(u, v) \Rightarrow \text{edge}(u, v)$
- $\text{Visited}(v) \Leftrightarrow \exists u : \text{next}(u, v) \vee \text{start}(v)$
- $\text{Successor}(u, v) \Leftrightarrow \text{next}(u, v) \vee \exists w : \text{next}(u, w) \wedge \text{Successor}(w, v)$
- $\text{VisitedOnce}(v) \Leftrightarrow \text{Visited}(v) \wedge \neg\text{Successor}(v, v)$

Encode the Specific Instance

- $V = \{v_1, \ldots, v_{23}\}$
- $\text{edge}(v_1, v_2)$, $\text{edge}(v_2, v_3)$, $\text{edge}(v_3, v_4)$, $\text{edge}(v_3, v_6)$, …

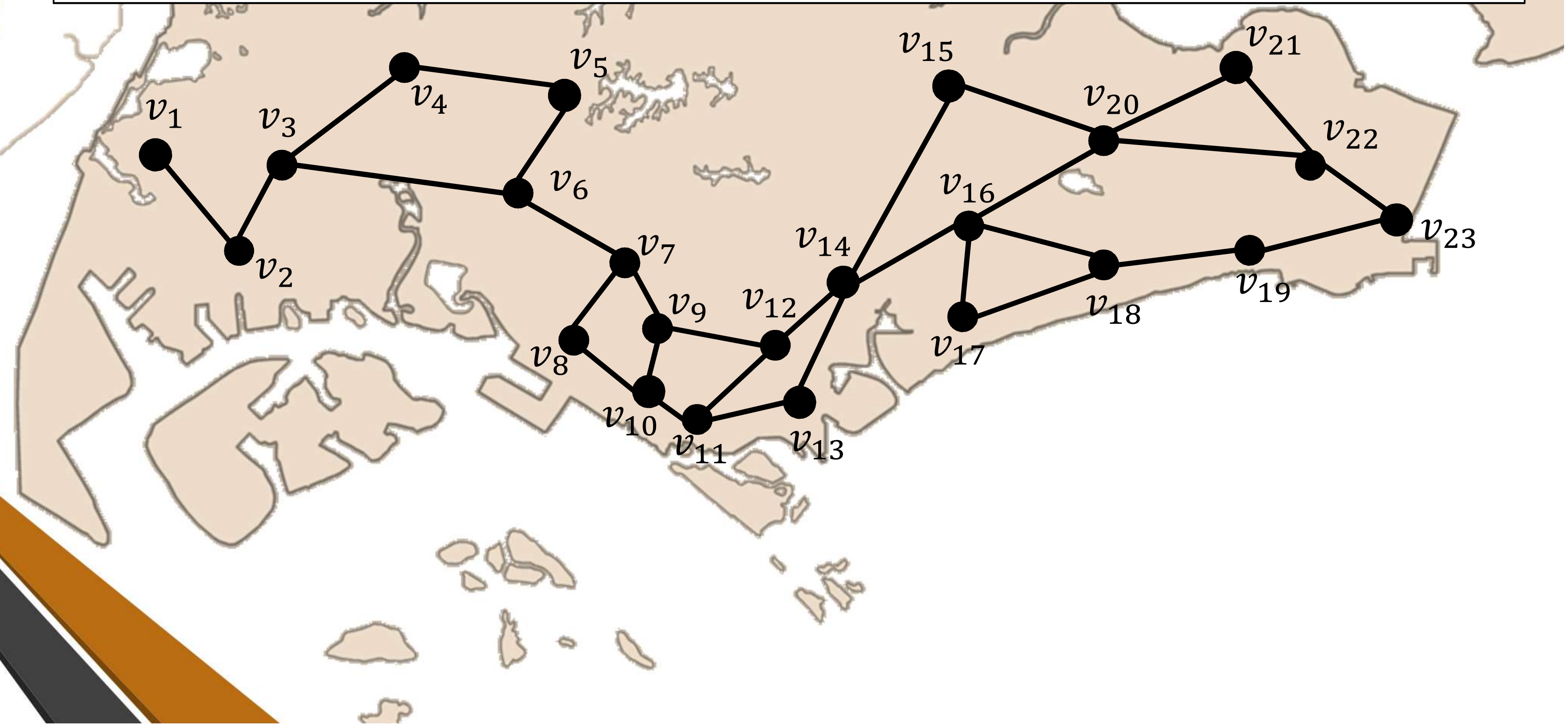


Pose Queries

- Is there a solution covering all vertices exactly once?

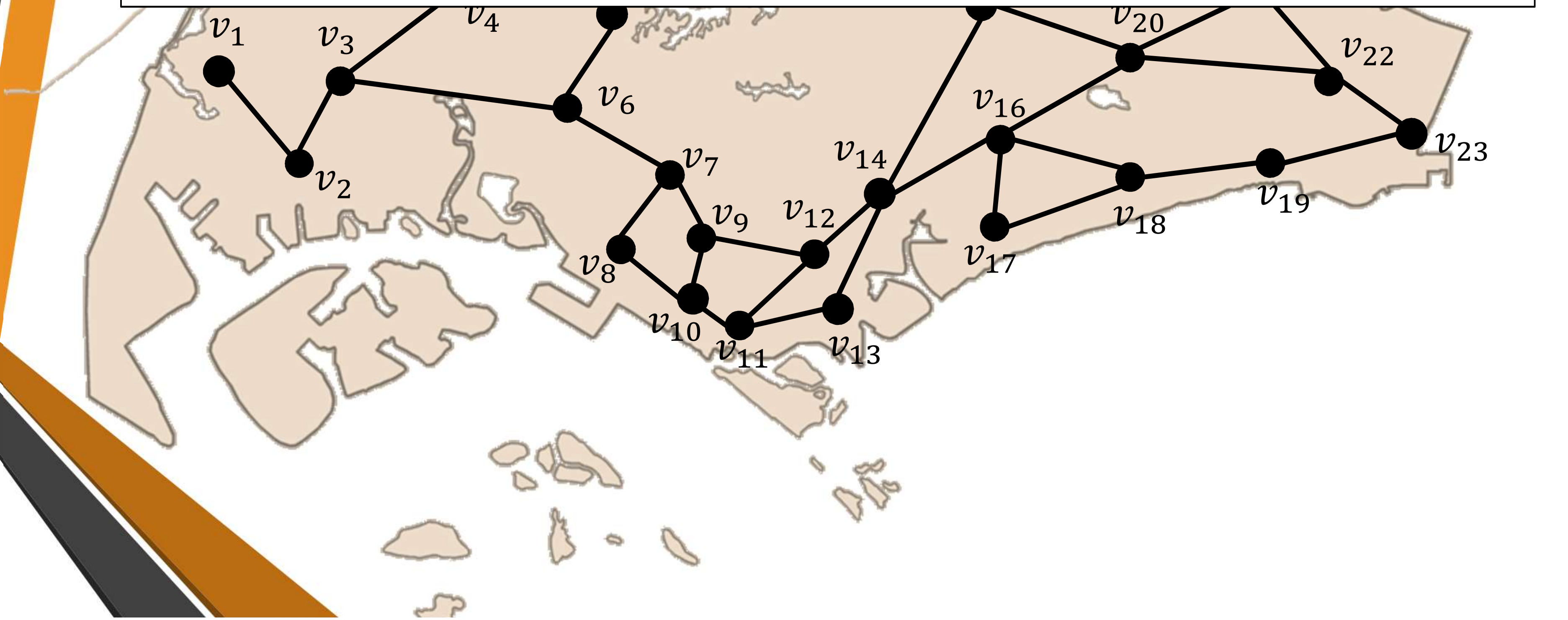
$$\forall v : (v \in V) \Rightarrow \text{VisitedOnce}(v)$$

- Will a specific solution work?



Debug Database

- $\forall u, v\colon \mathrm{edge}(u, v) \Rightarrow u \in V \wedge v \in V$
- $\forall u, v\colon \mathrm{edge}(u, v) \Rightarrow u \neq v$
- $\forall v\colon c(v) \Rightarrow v \in V$
- …



Summary

- First-order logic:
  - objects and relations are semantic primitives
  - syntax: constants, functions, predicates, equality, quantifiers
- Increased expressive power over propositional logic: sufficient to define many non-trivial problems