Homework 6 – Theory of Computation

Encoding the k-Clique Problem into SAT

Alessandro Lorenzo Luca

Introduction. Graph-based social networks ask whether a group of k people are *all* mutually connected. Given a graph $G = (V, E)$ and an integer k, we translate the question "does G contain a k-clique?" into a propositional-logic (CNF) formula so that any SAT solver can answer it.

Encoding overview

For every vertex $v \in \{1, \ldots, n\}$ and every position $j \in \{1, \ldots, k\}$ we introduce the Boolean variable

$$x_{v,j} = \text{“vertex } v \text{ occupies position } j\text{”}.$$

Three clause families enforce:

A. Each position is filled by exactly one vertex.

B. No vertex appears in two different positions.

C. Vertices that are not adjacent in G cannot be chosen together.

Rule A: exactly one vertex per position

Idea. Fill every position once, never twice.

$$\boxed{\bigwedge_{j=1}^{k} \Big(\bigvee_{v=1}^{n} x_{v,j} \Big) \;\wedge\; \bigwedge_{j=1}^{k} \; \bigwedge_{1 \le u < v \le n} (\neg x_{u,j} \vee \neg x_{v,j})}$$

```python
# Rule A -- one vertex per position
for pos in closed_range(1, k):
    clauses.append([getVarNumber(vertex=v, slot=pos)
                    for v in closed_range(1, n)])
    for v1 in closed_range(1, n - 1):
        for v2 in closed_range(v1 + 1, n):
            clauses.append([
                -getVarNumber(vertex=v1, slot=pos),
                -getVarNumber(vertex=v2, slot=pos)
            ])
```

Explanation. The first clause for slot j guarantees that some vertex is written there; the binary clauses prevent two different vertices from sharing the same slot, so the occupant is unique.

Rule B: a vertex can appear only once

Idea. Re-using a vertex in two positions is forbidden.

$$\boxed{\bigwedge_{v=1}^{n} \bigwedge_{1 \le j < j' \le k} (\neg x_{v,j} \vee \neg x_{v,j'})}$$

```python
# Rule B -- a vertex cannot repeat
for v in closed_range(1, n):
    for p1 in closed_range(1, k - 1):
        for p2 in closed_range(p1 + 1, k):
            clauses.append([
                -getVarNumber(vertex=v, slot=p1),
                -getVarNumber(vertex=v, slot=p2)
            ])
```

Explanation. For every vertex we enumerate each pair of positions and add a clause that blocks the vertex from occupying both, ensuring uniqueness per vertex.

Rule C: forbid choosing non-adjacent pairs

Idea. Two vertices that are not connected by an edge must never both be selected.

$$\boxed{\bigwedge_{\{u,v\} \notin E} \bigwedge_{\substack{1 \le j,j' \le k \\ j \ne j'}} (\neg x_{u,j} \vee \neg x_{v,j'})}$$

```
# Rule C -- endpoints of a non-edge cannot both be chosen
for p1 in closed_range(1, k):
    for p2 in closed_range(1, k):
        if p1 == p2:
            continue
        for u in closed_range(1, n - 1):
            for v in closed_range(u + 1, n):
                if frozenset({u, v}) not in edges:
                    clauses.append([
                        -getVarNumber(vertex=u, slot=p1),
                        -getVarNumber(vertex=v, slot=p2)
                    ])
```

Explanation. Whenever $\{u, v\} \notin E$, these binary clauses forbid assignments that would place u and v in two different positions. Hence every pair among the chosen vertices is connected.

Why we chose these rules

Rules A and B together force the selection of exactly k distinct vertices, matching the size requirement of a k-clique. Rule C adds the mutual-adjacency requirement, turning that k-set into a clique.

How the rules deliver the answer

If the formula is *satisfiable*. Read the vertices that appear in positions $1, \ldots, k$ inside a satisfying assignment. A ensures every position is filled, B makes them all different, and C guarantees that every selected pair is an edge. So the k vertices form a k-clique in G.

If G already has a k-clique. Put those k vertices into the k positions and set every other variable to *false*. This fills every position (satisfying A), repeats no vertex (B), and violates no non-edge clause (C), hence the assignment satisfies the formula.

Thus the clause set is satisfiable exactly when G contains a k-clique.

What could we have done differently?