Graph Colouring via Evolutionary Optimisation

Optimisation Midterm Examination

Kritchanat Thanapiphatsiri

Department of Computer Engineering
Kasetsart University

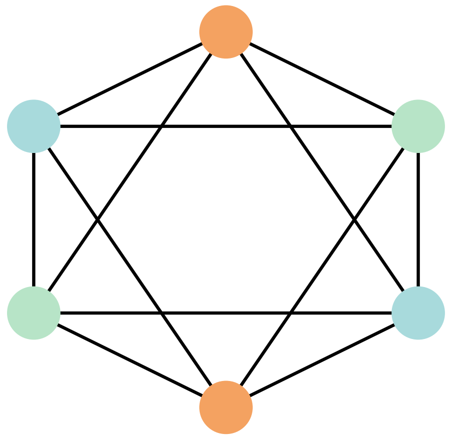
February 13, 2026

Problem

Graph Colouring (NP-hard)

Given an undirected graph $G = (V, E)$, assign a colour to each vertex so that adjacent vertices have different colours. The goal is to minimise the number of colours used (the chromatic number – $\chi(G)$).

- Decision version: can G be coloured with k colours?
- Optimisation version: minimise the number of colours



Why it matters

- Frequency assignment in wireless networks
- Exam/meeting timetabling
- Register allocation in compilers
- Map colouring and visualisation

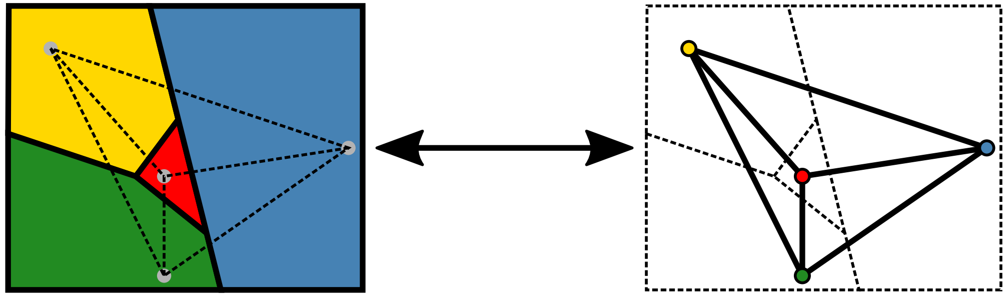


Image: Wikimedia Commons

Related problems

- k-colouring (feasibility)
- Maximum clique / maximum independent set (bounds)
- Graph colouring as a special case of constraint satisfaction

Formulation

Variables and constraints

Let $x_v \in \{1, ..., K\}$ be the colour of vertex v.

Constraints:

$$\forall (u, v) \in E : x_u \neq x_v$$

Optimisation target: minimise the number of used colours.

Optimisation-friendly objective

We use an unconstrained fitness that penalises conflicts:

$$\text{fitness}(x) = M \cdot \sum_{(u,v)\in E} 1[x_u = x_v] + |\{x_v : v \in V\}|$$

- M is a large penalty (e.g., $|E| + 1$)
- Works with heuristic search (GA, SA, PSO)

Approach

Representation

- Individual: integer vector of length $|V|$
- Gene: colour index in $\{1, ..., K\}$

Evolutionary algorithm outline

1. Initialise random population
2. Evaluate fitness
3. Select parents (tournament)
4. Crossover + mutation
5. Elitism, repeat until stop

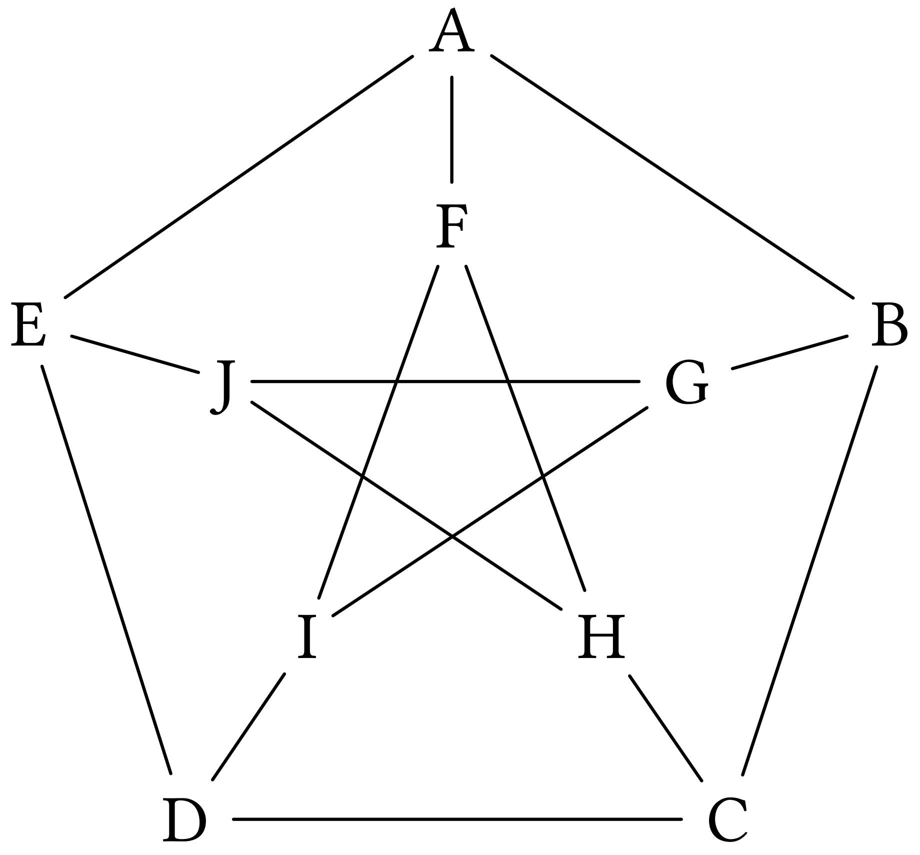
Why not solve directly

- Chromatic number is NP-hard
- Exact methods scale poorly
- Evolutionary search can reach optimal with enough exploration

Example

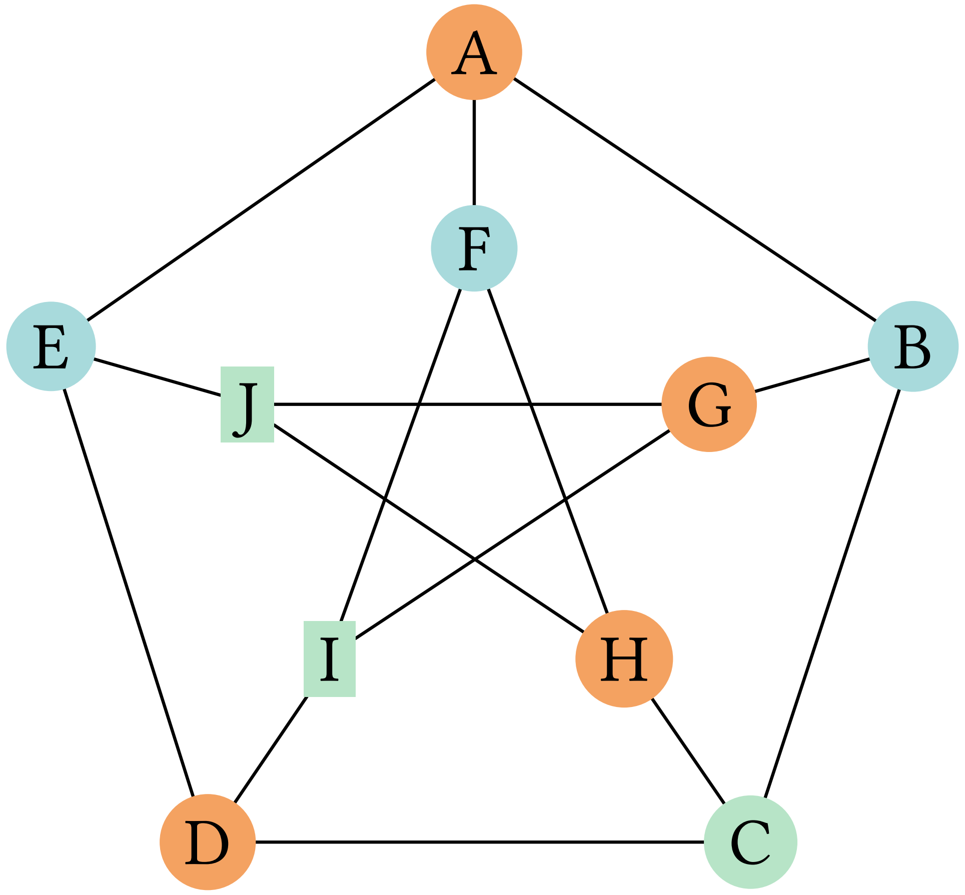
Example graph

Petersen graph (10 vertices, chromatic number 3):



Example colouring

One optimal 3-colouring (minimum possible):



Implementation

Python formulation highlights

- Graph as edge list
- Fitness = conflicts penalty + colour count
- GA with tournament selection and mutation

```py
@dataclass
class Graph:
    n: int
    edges: list[tuple[int, int]]
…                                                   …
    def fitness(self, colouring: Sequence[int], penalty: int) → int:
        conflicts = self.conflicts(colouring)
        colours = type(self).used_colours(colouring)
        return penalty * conflicts + colours
```

Output format

- Best colouring found
- Conflict count and colours used
- Ready for inspection or further refinement

```
$ python 6814001748_midterm.py

Best colouring: [2, 3, 1, 2, 3, 1, 2, 3, 3, 1]
Conflicts: 0
Colours used: 3
Fitness: 3
```

Discussion

Existing approaches

- Exact: branch-and-bound, ILP, SAT/CP
- Heuristic: DSATUR, greedy with reorder
- Metaheuristics: GA, SA, tabu search, ACO

Strengths and limitations

- Strength: flexible, easy to adapt
- Limitation: no optimality guarantee without enough search

Conclusion

Takeaways

- Graph colouring is NP-hard and practically important
- Evolutionary search offers a scalable path to good solutions
- The provided formulation supports future improvements

Thank you!

Any questions?