Graph Coloring Problem via SAT

February 23, 2022

1 Graph Coloring Problem via SAT

Antonin Novak (antonin.novak@cvut.cz), Combinatorial Algorithms, 2022

In this notebook, we demonstrate how to reduce the problem of Graph k-coloring into SAT. The problem is to find, whether the given undirected graph $G = (V, E)$ can be colored with k colors. The valid coloring is an assigments of numbers (colors) $\{1, \ldots, k\}$ to the verticies V, such that no two verticies connected by an edge have the same color.

1.1 Reduction

We will construct the instance of SAT problem as follows. We will introduce a propositional variablee $v_{i,k}$ with the meaning that the vertex $i \in V$ is colored with color k. We need to ensure the following: - each vertex has exactly one color assigned - for each edge $(i, j) \in E$, the vertices i and j cannot share the same color

We will construct clauses as follows: - $\forall i \in V : (v_{i,1} \vee v_{i,2} \vee \ldots \vee v_{i,k})$ = "each vertex has at least one color assigned" - $\forall i \in V, \forall 1 \leq r < q \leq k : (\neg v_{i,r} \vee \neg v_{i,q})$ = "each vertex has at most one color assigned" - $\forall k \in \{1, \ldots, k\}, \forall (i, j) \in E : (\neg v_{i,k} \vee \neg v_{j,k})$ = "same color cannot be assigned to connected verticies (i, j)"

1.2 SAT solver

Finally, we see that our clauses are already in CNF form, thus it can be fed into CNF-SAT solver, e.g., Kissat: https://github.com/arminbiere/kissat

[63]:
```
# parses graphs in DIMACS format
def parse_graph(filename):
    n = 0    # vertices
    m = 0    # edges
    E = []

    for line in open(filename, 'r').readlines():
        line = line.strip()
        if line.startswith('p'):
            line = line.split()
            n = int(line[2])
            m = int(line[3])
            continue
        if line.startswith('e'):
```

```
            line = line.split()
            E += [(int(line[1])-1, int(line[2])-1)]
    return n, m, E
```

[73]:
```
# reduction of k-coloring to SAT
def generate_kcoloring_sat_formula(n, m, E, k):
    clauses = []
    sat_vars = {}
    c = 1
    for i in range(n):
        for kdx in range(k):
            sat_vars[i, kdx] = c
            c += 1

    # at least one color for each vertex
    for i in range(n):
        clauses += [[sat_vars[i, kdx] for kdx in range(k)]]
    # at most one color for each vertex
    for i in range(n):
        for r in range(k):
            for q in range(r+1, k):
                clauses += [[-sat_vars[i, q], -sat_vars[i, r]]]

    # connected nodes cannot have the same color
    for e in E:
        for kdx in range(k):
            clauses += [[-sat_vars[e[0], kdx], -sat_vars[e[1], kdx]]]

    return len(sat_vars.keys()), len(clauses), clauses
```

[74]:
```
# translates clauses into DIMACS SAT format
def export_dimacs(n, m, clauses, filename='sat_input.txt'):
    f = open(filename, 'w+')
    print('p cnf {} {}'.format(n, m), file=f)
    for mdx in range(m):
        print(' '.join([str(var) for var in clauses[mdx] + [0]]), file=f)
    f.close()
```

[87]:
```
# uses instances from https://mat.tepper.cmu.edu/COLOR/instances.html

filename = 'myciel3.col'   # k=4
#filename = 'queen8_8.col'  # k=9
#filename = 'myciel6.col'   # k=7

n, m, E = parse_graph(filename)
k = 4
sat_n, sat_m, sat_clauses = generate_kcoloring_sat_formula(n, m, E, k=k)
```

```
export_dimacs(sat_n, sat_m, sat_clauses, filename='sat_input_k{}.txt'.format(k))
```

After compiling Kissat solver, you can run it with: `cat sat_input.txt | ./kissat`