CAAM 476 & INDE 546 — Spring 22

Final Project

Due Date: May 3, 2022 (Tuesday)

This project involves implementation of two solution methods for a well-studied combinatorial optimization problem, The Maximum Clique Problem, and comparing and contrasting their performances through solving real-life instances. The problem description as well as the solution methods are presented below.

The Maximum Clique Problem. Given a simple, undirected graph $G = (V, E)$, a set of pairwise adjacent vertices $C \subseteq V$ is called a *clique*, i.e., $\{i, j\} \in E$, for every pair of vertices $i, j \in C$. A cliques is called *maximal* if it is not a proper subset of another clique (it is not contained in a larger clique) and *maximum* if it is of maximum cardinality (there is no larger clique in the graph). The Maximum Clique Problem (Max Clique) is to find a maximum clique in an input graph G; the cardinality (size) of a maximum clique in G is called the *clique number* of the graph and is denoted by $\omega(G)$. Max Clique is one of the classical NP-hard problems of combinatorial optimization, which finds a wide range of applications and has been studied extensively in the literature [1].

1. Integer Programming

A straightforward integer-programming formulation of Max Clique is as follows:

$$\begin{aligned} \omega(G) = \max \quad & \textstyle\sum_{i \in V} x_i \\ \text{s.t.} \quad & x_i + x_j \leq 1, \ \forall \{i, j\} \notin E, \\ & x_i \in \{0, 1\}, \ \forall i \in V, \end{aligned} \tag{1}$$

where each vertex $i \in V$ is represented by a binary variable $x_i \in \{0, 1\}$. The constraints guarantee each feasible solution of (1) is the characteristic vector of a clique in G; hence, the objective function represents the cardinality of a clique. An optimal solution to this formulation characterizes a maximum clique in the graph, i.e., an optimal solution to Max Clique is given by the vertices whose variables have the value 1 in an optimal solution of (1).

2. A Combinatorial Algorithm

A well-known solution method for Max Clique is the *combinatorial branch-and-bound* algorithm proposed by Tomita and Kameda [2]. The algorithm relies on the adjacency of the vertices to form subproblems in the branching phase. To help forming small subproblems, the algorithm involves an initialization step (before starting the branch-and-bound procedure), which sorts the vertices in an ascending order of their degrees. To draw an upper bound on the clique number of subproblems, the algorithm employs a simple heuristic procedure for Graph Coloring. A *proper* coloring of a graph is an assignment of colors (natural numbers) to the graph vertices such that every two adjacent vertices have different colors. Finding a minimum number of colors needed to color a graph properly is known as The Graph Coloring Problem (Graph Coloring) and is NP-hard. Heuristic algorithms for Graph Coloring aim to find reasonably-good solutions for this problem, i.e., a proper coloring with a few number of colors, fast. Note that if a graph has a clique of cardinality k, at least k colors are needed to color the graph properly; hence, any proper coloring of a graph provides an upper bound on its clique number.

Assignments:

For the following assignments, five input instances (taken from The Second DIMACS Implementation Challenge) have been provided. In each instance (`.txt`) file, every line identifies an edge in the graph; for example "3 2" implies $\{3, 2\} \in E$, where 3 and 2 are the labels of the end-vertices. You should be able to construct the adjacency matrix of the graph easily from the input.[1] Submit your code as `.ipynb`. file or `.py` files. You will receive credit only if your code runs with no error, generates the requested output, and produces correct solutions.

1. Write a script that inputs an instance of Max Clique, i.e., an undirected graph, and calls the GUROBI solver to solve the problem using formulation (1). Your code should output a maximum clique in the graph and its cardinality, i.e., the clique number of the graph, as well as the time (in CPU seconds) it takes for the solver to solve the problem. Limit the solution time to *one hour*; if the code terminates due to the time limit without completing the search, it should output the best solution found. Your submission for this part should be named `IP_lastName_firstNameInitial`, e.g., `IP_Hosseinian_M.ipynb`.

2. Implement the combinatorial branch-and-bound algorithm for Max Clique proposed by Tomita and Kameda [2]. Your code should input an instance of Max Clique and output a maximum clique in the graph and its cardinality, i.e., the clique number of the graph, as well as the time (in CPU seconds) it takes for the algorithm to solve the problem. Limit the solution time to *one hour*; if the code terminates due to the time limit without completing the search, it should output the best solution found. Your submission for this part should be named `CBB_lastName_firstNameInitial`, e.g., `CBB_Hosseinian_M.ipynb`.

3. Submit your answers to the following questions as a single PDF file:

   (a) Print the output of your codes on the five instances provided.

   (b) For each instance, which method performs faster and by what factor? Is there a correlation between the performance (solution time) and graph density?

   (c) Consider the (vertex) weighted counterpart of Max Clique: given a vertex-weighted undirected graph (each vertex is assigned a weight), find a clique of maximum weight (sum of the weights on its vertices) in the graph. Obviously, a maximum (cardinality) clique is not necessarily a maximum weight clique in the graph. Can the above methods be easily modified to address this problem? Explain.

   (d) Based on your answers, which method would you prefer and why?

References

[1] Immanuel M Bomze, Marco Budinich, Panos M Pardalos, and Marcello Pelillo. The maximum clique problem. In *Handbook of combinatorial optimization*, pages 1–74. Springer, 1999.

[2] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global optimization*, 37(1):95–111, 2007.

[1] In DIMACS instances, the vertices have been labeled starting from 1. The total number of vertices in the graph is the maximum end-vertex label in the list of edges, which is also included in the instance names. You can use `list = numpy.loadtxt("instance.txt" , dtype=str)` to input the list of edges and then read them one by one (and convert `str` to `int`) to populate the adjacency matrix.