1 Inputs

The input data can be accessed from SQL database. As long as there is a way Python can to parse the input, it can be any type of file (JSON, CSV, etc.). Code is written to generate a random toy-world social connection of friends, so it's easy to control the size of vertices(represent as each person) and edges(represent as friendship between two persons) to conduct analysis of the running time, formula size, and to see the limit of feasible inputs that solved by SAT solver.

1.1 SAT $\leq_p$ K-clique

Reduction recipe of clique problem to SAT can be done by having a given SAT solver, and a one-to-one mapping function $f : [G]- > [B]$ that maps a graph $G = (V, E)$ to a Boolean formula, hence to reduce the clique problem to SAT problem.

For the clique problem we have a graph G=(V,E). we can imagine there are cliques with different sizes in this graph. Let k denote the size of sub-cliques in a graph. Hence, we have variables $v_{i,j}$, where $i \in V$ and $j \in range(k)$(this is the form in python denote numbers 0,1,. . . ,k-1). It not hard to see that there are overall $n \cdot k$ variables in this graph, where n is number of |V|.

A variable $v_(i, j) = 1$ means the j^{th} vertex i is in the clique of size k, otherwise, variable $v_(i, j) = 0$. To have a clique sub-graph, it should be a series of unique vertices to form a clique. Hence, we add each variable with disjunction to CNF clauses.

$$\bigvee_{\substack{i\in V\\ j\in k}} v_{i,j} \tag{1}$$

For the first constraint to form a clique in a graph, if size k is given, this means we have k different slots for the different vertex. If all of slots can be fully filled by some $v_(i, j)$, it means there is k-size clique satisfiable in this graph. We need to have every slot filled and at most only one variable for each slot; therefore, we add constraint (2) into conjunctions of clauses.

$$\bigwedge_{j=0}^{k-1} \sum_{i\in V} v_{i,j} = 1 \tag{2}$$

For the second constraint, every variable can or cannot be filled in into a slot(because each vertex has k variables, and some are not in the clique). There, we add constraint (3) into conjunctions of clauses.

$$\bigwedge_{i\in V} \sum_{j=0}^{k-1} v_{i,j} \leq 1 \tag{3}$$

For the last constraint, we must ensure the connectivity of this graph. To approach this constraint, we need to have a list of edges $\overline{E}$ ={(u,v) | (u,v)$\notin E$, $u \neq v$}, such that there is no such an edge between vertices u and v, into conjunctions of clauses.

$$\bigwedge_{\substack{(u,v)\in\overline{E}:u\neq v\\ q,w\in k:q\neq w}} \neg v_{u,q} \vee \neg v_{v,w} \tag{4}$$

If we put all constraints with conjunctions together, we can solve clique problem with a SAT solver, and the formula as follows,

$$(\bigvee_{\substack{i\in V\\ j\in k}} v_{i,j}) \wedge (\bigwedge_{j=0}^{k-1} \sum_{i\in V} v_{i,j} = 1) \wedge (\bigwedge_{i\in V} \sum_{j=0}^{k-1} v_{i,j} \leq 1) \wedge (\bigwedge_{\substack{(u,v)\in\overline{E}:u\neq v\\ q,w\in k:q\neq w}} \neg v_{u,q} \vee \neg v_{v,w}) \tag{5}$$

2 Method

2.1 Brute force algorithm

Brute-force algorithm as an oracle to tell us the maximum value of k, otherwise, the SAT solver will try to exhaust all possible ways to find a solution that doesn't even exist if given input k is larger than the value returned from brute-force algorithm. The pseudo code of brute-force algorithm is as follows,

```
1:  procedure BRUTE-FORCE-K-CLIQUE($G = V, E$)
2:      $C \leftarrow v \in V$                    ▷ Each vertex can represent as a clique
3:      for $clique \in C = c_1, c_2, ...$ do
4:          for $vertex \in V = v_1, v_2, ..., v_n$ do
5:              if $v \notin c$ : then
6:                  // is_clique() checking whether clique c is still a clique
7:                  // after adding a vertex v into c still a clique,
8:                  // and the running time is simply as $O(|V|^2)$
9:                  if $is\_clique(c, v)$: then $c.append(v)$
10:     // Only returns the maximum size cliques
11:     return $\{c \in C \mid len(c) == max(len(C))\}$
```

3 Conclusion

Brute-force is far faster and space efficient than the SMT solver method, because the latter algorithm used up most of time and space in constructing CNF clauses. The run time becomes even worse if there is no such an oracle that tells you the number maximum of k. Furthermore, the limit of SAT solver is the space usage. After a few couple of experiments, it always halts around input size(number of vertices) 60 to 70(it depends on how sparse the graph is in terms of amount of non-connected edges). The main issue is the limitation of physical memory usage in transforming input into the form that is accessible to the SAT problem. Based on the python memory monitor library, the program stops when the memory usage up to 1 GB.

In the experiment, plotting uses library Matplotlib in python based on input sizes, running time, and memory usage if memory usage is affordable. To see the result, we attached plotting graphs of running time, memory usage, number of clauses, and instructions to run our code in README in the RAR files.

4 Source

https://sites.google.com/site/haioushen/search-algorithm/cliquetosat