Approaching the Clique Problem With SAT Solvers

CIS 673: Computer Aided Verification

Matthew Howard

December 28, 2016

Abstract

Modern SAT solvers are able to solve real world instances of SAT within realistic time and space restrictions. General purpose solvers have been effective in solving artificial intelligence problems, circuit optimizations, logic puzzles, and verification problems. In this project, we evaluate the application of a SAT solver to the classical graph theoretic CLIQUE problem.

1 The Clique Problem

Let us start by defining terminology. A *graph* is a duple $G = (V, E)$ consisting of a vertex set V and an edge set $E \subseteq \{(u, v) \mid u, v \in V, u \neq v\}$. We will follow the convention $n = |V|$ and $m = |E|$. An *induced subgraph* $G' = (V', E')$ of G consists of a vertex set $V' \subseteq V$ and the maximal edge subset $E' \subseteq E$ such that the endpoints of each edge are in V', formally $E' = \{(u, v) \in E \mid u, v \in V'\}$. For the purpose of this project, we are concerned only with undirected, unweighted graphs.

A graph is *complete* if there exists an edge between every two distinct vertices.

A *clique* $G' = (V', E')$ in G is an induced subgraph which is also a complete graph. We say a clique is of size k if its vertex set contains k elements. The *Clique Problem*, which we refer to as CLIQUE, is the problem of determining for a graph G and a positive integer k whether there exists an induced subgraph G' satisfying the following two conditions:

1. **(Clique Condition)** G' is a clique in G
2. **(Size Condition)** G' is of size at least k

Note that for a clique G' in G, every induced subgraph G'' of G' is also a clique in G, so we could equivalently formulate CLIQUE with the more strict size condition "G' is of size exactly k".

While we are concerned mostly with the decision problem formulation of CLIQUE, it's worth discussing the closely related optimization version: Given a graph G', find a clique of maximum size. We define $\omega(G)$ to be the maximum clique size of a graph G. The optimization and decision versions are similar in complexity. If $\omega(G)$ is known, then the decision problem

is equivalent to checking if $k \leq \omega(G)$. Likewise, if we have an oracle for CLIQUE which provides the clique's vertex set as a certificate, then we can solve the optimization version in $O(\log n)$ calls to the oracle with a binary search of k.

The clique problem is of significance in complexity theory for a few reasons. Firstly, it is one of the classical NP-complete problems. CLIQUE is in NP since the vertex set of a k clique can serve as a $O(n)$ certificate from which it is easy to verify the Clique and Size properties. CLIQUE can be shown to be NP-hard through the two reductions,

$$\text{SAT} \leq_p \text{3SAT} \leq_p \text{CLIQUE} \tag{1}$$

Secondly, the hardness of CLIQUE is often exploited in proving the NP-hardness of VERTEX-COVER and other NP-hard problems which indicates it is a powerful and useful problem. Finally, CLIQUE is "hard" in the algorithmic sense that there does not exist an approximation algorithm which performs better than $n^{1-\epsilon}$[3]. This hardness of approximation sets CLIQUE apart from many other classical NP-complete problems.

2 Selection of a SAT Solver

Rather than performing a survey of different SAT solvers (which is already done each year on a global scale), we chose to focus our efforts one solver in particular. MiniSat is widely used, open source, minimalistic, and designed for integration, which makes it an good candidate to serve as the foundation for our CLIQUE solver. MiniSat works through conflict-driven learning and unit propagation[1].

3 Obtaining Data Sets

We want to analyze our clique solver on both real-world and randomly-generated graphs. We decide to go with Facebook friend networks as our source of real data. Vertices correspond to friends of a central user (without including the central user as a vertex) and edges correspond to friendships between users in the graph. A clique is intuitively a group of people who are all friends with each other. Running analyses on real data allows us to draw social insights from our application. For example, given a large clique, we can identify common traits between the constituents, such as membership in a club or a certain place of employment.

3.1 Social Network Data

The Facebook Graph API provides REST access to the information we require, including lists of friends and lists of mutual friends in easily parsable formats. However, since the release of version 2.0 of the API, lists only show friends who have actively logged into the developer's particular application (in this case our data collection tool)[2]. This severely restricts the usefulness of this data. One alternative approach to collecting the same data would be to write a web-scraper using a browser automation tool like Selenium. However, this solution begs complex ethical and legal implications since neither Facebook nor the users consent to this style of data collection.

Luckily, the Stanford Network Analysis Project provides a handful of anonymized, real-world datasets including graphs of Facebook friend data. While the anonymous nature of this data limits social insights that could come out of our analysis, the datasets will nonetheless allow us to test our solvers on real-world data.

The Stanford Datasets provides us with ten graphs of varying sizes. We label these graphs `facebook.a` through `facebook.j`.

3.2 Analyzing the Data

Before we run and evaluate our clique solver, we wish to analyze the graph datasets. We leverage "igraph", an open source graph analysis library with a Python wrapper to compute $\omega(G)$. igraph's $\omega(G)$ routine is based off of an algorithm due to Tsukiyama *et al.*[8] and has a worst-case runtime of $O\left(3^{n/3}\right)$[4]. This implementation will serve as a point of comparison for the performance of our CLIQUE solver.

4 Reduction Strategy

Given a graph G and a positive integer k, we want to produce an instance of SAT such that the SAT system is satisfiable if and only if our graph contains a clique G' of at least size k. Every SAT instance consists of variables and clauses. Since a clique is defined by its vertex set and a solution to SAT is defined by a set of variable assignments, we naturally correspond vertices to variables. For all $v \in V$, let x_v represent the variable indicating whether $v \in V'$.

Given the vertices are represented by variables, we need some way of encoding edges in our SAT system. The Size Condition in CLIQUE is independent of the edge set E and only depends on the truth assignments of variables in the vertex set V'. Therefore, we must reference our edge set E in terms of the Clique Condition.

We restate our Clique Condition as:

$$\forall\ u, v \in V, \qquad u \in V' \wedge v \in V' \implies (u, v) \in E \tag{2}$$

$$\forall\ u, v \in V, \qquad x_u \wedge x_v \implies (u, v) \in E \tag{3}$$

If we take the contrapositive, we get

$$\forall\ u, v \in V, \qquad (u, v) \notin E \implies \neg x_u \vee \neg x_v \tag{4}$$

$$\forall\ u, v \in V, \qquad (u, v) \in \overline{E} \implies \neg x_u \vee \neg x_v \tag{5}$$

$$\forall\ (u, v) \in \overline{E}, \qquad \neg x_u \vee \neg x_v \tag{6}$$

where $\overline{E}$ is the complement edge set, or formally $\overline{E} = \{(u, v) \mid u, v \in V,\ u \neq v\} \setminus E$. This last statement of the clique condition can easily be encoded in SAT since $\neg x_u \vee \neg x_v$ is a CNF clause and $\overline{E}$ is an enumerable set of size $O(n^2 - m)$ over which we can add clauses.

The Size Condition is a bit trickier. We must add variables and clauses such that at least k of our vertex variables must be true. If we number our vertices $1 \ldots n$ and consider our

vertex variables as binary indicator variables, we want

$$\sum_{i=1}^{n} x_i \geq k \tag{7}$$

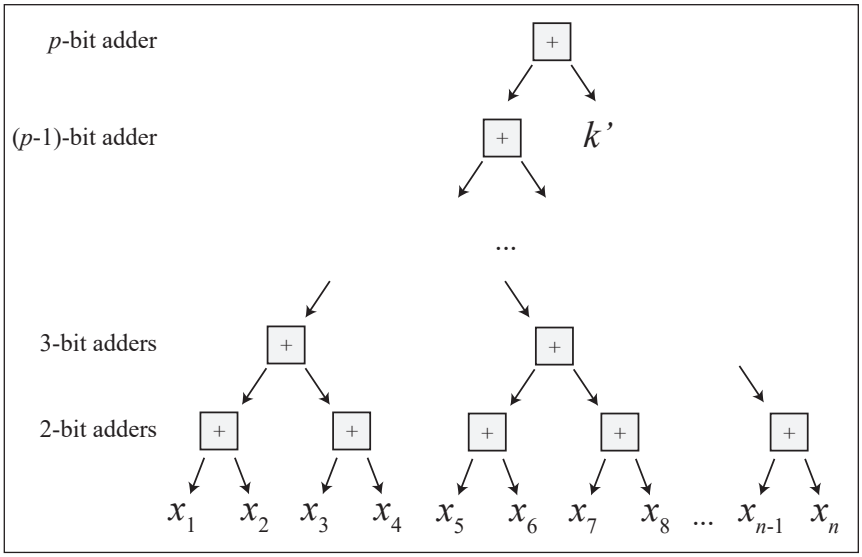
The computation on the left hand side can be represented as a combinational digital logic circuit with n single-bit inputs and one $(\lfloor \log_2 n \rfloor + 1)$-bit output. Let us define k' to be the least natural number such that $k + k'$ is a power of 2, say 2^p. Adding k' to both sides, we get

$$\sum_{i=1}^{n} x_i + k' \geq k + k' \tag{8}$$

This expression is equivalent to checking whether the p-th bit of the sum on the left hand size is equal to 1, which can be expressed as a one-literal clause.

We implement the sum using an adder tree. The leaves of the tree are the vertex variables and k' while the internal nodes are binary adders of varying size. The j-bit binary adders are constructed from full adders, which are comprised of NAND and NOR gates, which are constructed from boolean clauses with additional variables.

Figure 1: Adder Tree Schematic



We can run this reduction on the Facebook datasets with $k = \omega(G)$ to see the size of the resulting SAT systems.

Figure 2: Number of SAT Literals and Variables when $k = \omega(G)$

Name	n	m	Variables	Clauses
`facebook.a`	333	5038	2286	60199
`facebook.b`	1034	53498	7193	531836
`facebook.c`	224	6384	1559	26819
`facebook.d`	150	3386	1041	12909
`facebook.e`	168	3312	1154	16091
`facebook.f`	61	540	424	2931
`facebook.g`	786	28048	5457	312068
`facebook.h`	747	60050	5220	265496
`facebook.i`	534	9626	3644	149737
`facebook.j`	52	292	355	2287

While the adder tree introduces many additional variables and clauses, the set of added variables only has one satisfying assignment for each assignment of the vertex variables. MiniSat should be able to pick assignments to the vertex variables and then solve the intermediate variables in the adder tree via unit propagation[7].

5 Evaluation

To test the correctness of our clique solver, we verify it produces the same values for $\omega(G)$ as the igraph script. A proof of $\omega(G)$ with our CLIQUE solver consists of showing that the program succeeds when $k = \omega(G)$ and that the program fails when $k = \omega(G) + 1$,. In other words, there exists a clique of size $\omega(G)$ but no clique of size larger than $\omega(G)$. We first run our igraph script on each `Facebook` graph and get results for all values except `facebook.h`. igraph's ω function runs on `facebook.h` for at least 30 CPU minutes. Since this is far more than the computation time required for any of the other graphs, we classify igraph's ω algorithm as not terminating on `facebook.h` within a reasonable length of time.

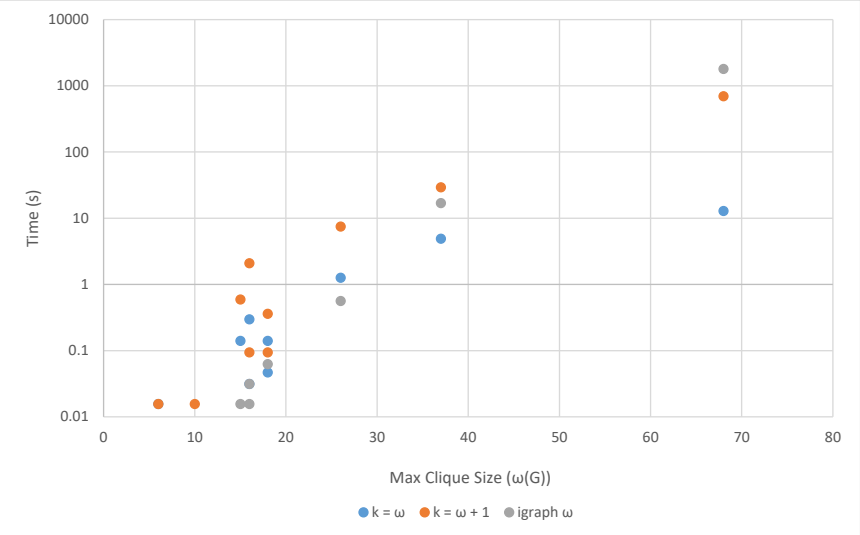
Next, we run our CLIQUE solver on the same set of graphs for our previously obtained values of $\omega(G)$ and $\omega(G) + 1$. We compute $\omega($`facebook.h`$)$ using our solver in about 15 minutes.

Figure 3: igraph and CLIQUE Solver Performance on Facebook Datasets

				igraph	CLIQUE Solver	
Name	n	m	ω	ω time	$k = \omega$ time	$k = (\omega + 1)$ time
`facebook.a`	333	5038	15	0.0156	0.141	0.594
`facebook.b`	1034	53498	37	16.859	4.906	29.297
`facebook.c`	224	6384	18	0.0625	0.141	0.359
`facebook.d`	150	3386	18	0	0.0469	0.0938
`facebook.e`	168	3312	16	0.0156	0.0313	0.0938
`facebook.f`	61	540	10	0	0	0.0156
`facebook.g`	786	28048	26	0.563	1.266	7.484
`facebook.h`	747	60050	68	-	12.813	694.594
`facebook.i`	534	9626	16	0.0313	0.297	2.0938
`facebook.j`	52	292	6	0	0.0156	0.0156

For each graph, we plot the maximum clique size versus the execution time for igraph's ω calculation, for our solver with $k = \omega$, and for our solver with $k = (\omega + 1)$.

Figure 4: Clique Size vs. Computation Time for Facebook Datasets



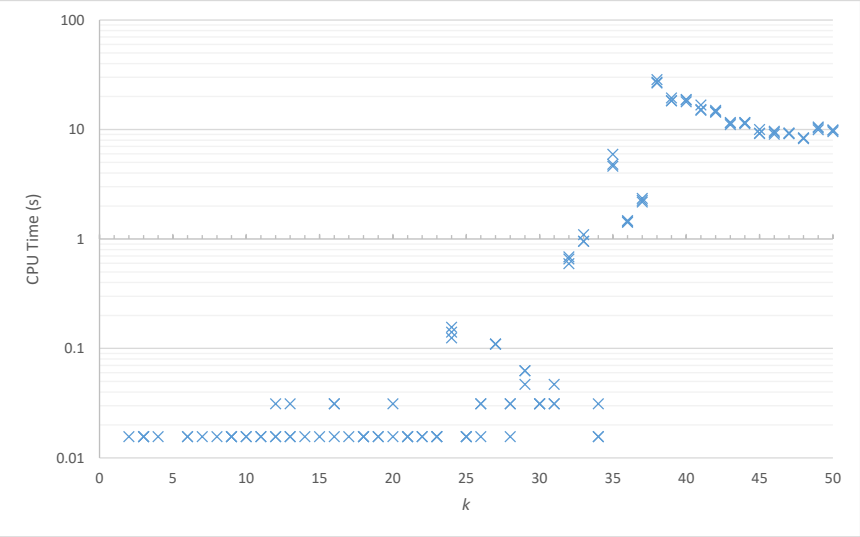
From this small set of data, we see a rough exponential relationship between clique size and run time for both our solver and igraph. This is expected since both programs have exponential bounds.

In addition to computation time, MiniSat keeps track of other useful metrics. By default, it reports the number of restarts, conflicts, decisions, propagations, and conflict literals at the end of execution. The number of conflicts in particular ends up being a fair indicator of problem complexity and is strongly associated with execution time. This figure is also telling of how MiniSat responds to a particular problem since the conflicts correspond to pieces of information MiniSat is able to learn as it progresses through its search.

5.1 Effects of k on Runtime

We can see for a fixed graph what effect varying k has on our solver's execution time. Intuitively, very small cliques, e.g. $k = 2$ or $k = 3$ should come up very quickly in the MiniSAT's search of the solution space, whereas finding cliques of size close to $\omega(G)$ should require more searching and time based on the hardness of the problem. We fix the graph `facebook.b`, which has one of the higher ω values ($\omega(G) = 37$) of our dataset. Then we run our solver three times for each value of k between 1 and 50 and plot the results.

Figure 5: k vs. Computation Time



We observe that for $k < 24$, the computation time stays at or around 0. For $24 \leq k \leq \omega = 37$, there is a positive association but the the CPU time does not increase monotonically as we increase k. For example, the average computation time for $k = 35$ is 5.125 seconds while the average computation time for $k = 36$ is 1.443 seconds, despite the fact that it is strictly more difficult to find a clique of size 36 than to find one of size 35. This insight, combined with the fact that there is little deviation in computation time across trials for a fixed k, suggests that the value of k, encoded in our reduction via k' and the bit width of the adder tree root, plays a large role in guiding the behavior of MiniSat.

Between $k = \omega = 37$ and $k = \omega + 1 = 38$, the average computation time jumps from 2.26 seconds to 27.4 seconds. This sudden increase between ω and $\omega + 1$ appears consistently in Figure 3 as well. One possible reason is that when $k = \omega + 1$, there are many assignments which come close to satisfying the Size Condition, all of which must be ruled out in order to conclude no satisfying assignment exists.

For $\omega + 1 = 38 \leq k \leq 50$, the computation time steadily decreases. This aligns with out intuition, because as $k > \omega$ increases, the number of possible satisfying assignments to check, $\binom{n}{k}$, decreases. By extension, the solver concludes that the system is unsatisfiable for $k = n = 1024$ in only 0.203 seconds.

6 Conclusion

In this project, we constructed a reduction for CLIQUE $\leq_p$ SAT, implemented it in C++, interfaced our reduction with MiniSat, and then tested and evaluated the resulting CLIQUE solver. Our construction operating on top of a general purpose SAT solver had comparable performance to igraph's specialized $\omega(G)$ routine. This speaks to the power and generality of MiniSat as a computational tool.

Our analysis was heavily constrained by the availability of good graph data. We tried to construct random graphs using the Erdős - Renyi model but found that the uniform distribution of edges between pairs of vertices results in very small clique sizes, which limits the graphs' usefulness to our analysis. We also looked at a slightly more complex graph generation model consisting of vertex typing and type-wise connection probabilities. While graphs of this model yielded significantly larger clique sizes, they still failed to mimic the eccentricity distributions of social media graphs and were difficult to generate for varying parameters of n and m.

Given more time, we would have considered other graph generation models to aid in analysis. One family of graph models, the Kronecker Product Graph Model construction, has been very successful in mimicking social network graphs[5], including Facebook friend graphs[6]. Given the ability to generate these social network graphs of arbitrary size, we could have further tested the limitations and runtime of our solver.

7 Resources

The code and data involved in this project is publicly available at `https://github.com/moward/CliqueSolver`.

References

[1] Niklas Een and Niklas Sörensson. *An Extensible SAT-solver [ver 1.2]*. 2003.

[2] Facebook. *Facebook Bug Report 1502515636638396*. May 2014. URL: https://developers.facebook.com/bugs/1502515636638396/.

[3] Johan Hastad. "Clique is hard to approximate within $n^{1-\epsilon}$". In: *Acta Mathematica* 182.1 (1999), pp. 105–142. ISSN: 1871-2509. DOI: 10.1007/BF02392825. URL: http://dx.doi.org/10.1007/BF02392825.

[4] igraph. *Source code for clique.c*. Mar. 2016. URL: https://github.com/igraph/igraph/blob/master/src/cliques.c.

[5] Jure Leskovec et al. "Kronecker Graphs: An Approach to Modeling Networks". In: *J. Mach. Learn. Res.* 11 (Mar. 2010), pp. 985–1042. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1756006.1756039.

[6] S Moreno, P Robles, and J Neville. "Block Kronecker Product Graph Model". In: *Proceedings of the 11th Workshop on Mining and Learning with graphs*. 2013.

[7] Mate Soos. *MiniSat FAQ*. URL: https://www.msoos.org/minisat-faq/.

[8] Shuji Tsukiyama et al. "A New Algorithm for Generating All the Maximal Independent Sets". In: *SIAM Journal on Computing* 6.3 (1977), pp. 505–517. DOI: 10.1137/0206036. eprint: http://dx.doi.org/10.1137/0206036. URL: http://dx.doi.org/10.1137/0206036.