A Dynamic Programming Approach for Timing and Designing Clique Algorithms

Wendy Myrvold * Tania Prsa † Neil Walker ‡

November 7, 1997

Abstract

Traditionally, practical clique algorithms have been compared based on their performance on various random graphs. We propose a new testing methodology which permits testing to be completed in a fraction of the time required by previous methods. In addition, the range of testing can be extended to include problems that could not be attempted in the past because of being too time consuming. We accomplish this by applying the approach that makes dynamic programming a very effective algorithmic technique: we use tabulated estimates for the time required to solve subproblems rather than timing each exactly. Our computational experiments validate this approach.

The next step is to use this workbench to develop fast new algorithms for the maximum clique problem. A *mixed algorithm* is an algorithm which applies different strategies for the subproblems that arise. Using the dynamic programming approach again for timing, we mechanize the process of developing table driven algorithms which apply the best possible strategy to each subproblem. We demonstrate the utility of this approach by creating new algorithms for random graphs with varying densities. Our resulting clique algorithms are extremely simple, yet significantly faster than the best approach found in the literature to date.

1 Introduction

An *undirected graph* G consists of a set V of *vertices* and a set E of unordered pairs of vertices called *edges*. The number of vertices is called the *order* of the graph, and the number of edges is the *size*. Given a graph $G = (V, E)$, a *clique* in the graph G is a set of pairwise adjacent vertices. A *maximum clique* in G is the largest such set of vertices. This well-known problem has been shown to be NP-complete [10]. In spite of this inherent

*University of Victoria, wendym@csr.UVic.ca, Supported by NSERC.

†Financial Models Company, Inc., tprsa@fmco.com

‡University of Victoria, nwalker@csr.UVic.ca

difficulty there are many algorithms which perform well in practice, even on fairly large graphs.

One of the earliest published results concerning practical clique algorithms is the 1970 paper by Auguston and Minker [1]. They produced computational results comparing two algorithms. In 1973, Bron and Kerbosch [8] produced a branch and bound algorithm called CACM457. The branch and bound technique is in fact the basis for most recent clique algorithms, including Balas and Yu [5], Balas and Niehaus [3], Balas and Xue [4], Bar-Yehuda et. al. [6], Carraghan and Pardalos [9], Mannino and Sassano [11], and Babel [2]. The most notable of these are the algorithms by Balas and Yu [5] and Babel [2]. Balas and Yu use a greedy colouring scheme together with properties of a maximally triangulated induced subgraph (MTIS) to achieve tighter bounds on the maximum clique size. Their algorithm was one of the fastest until 1990, when Babel devised an algorithm which also employs a colouring, this time the DSATUR colouring method due to Brelaz [7].

Various actions can radically affect performance of clique algorithms. For example, sorting vertices by degree before applying some algorithms can have a big effect. However, there are times when it is faster not to sort. Ideally, authors would try various combinations of options for their algorithms and then would indicate the best choices depending on the graph order and edge density. This has not been common practice however, likely because of the huge amount of computation time it would require to test all combinations.

Instead of timing the algorithms exactly (which would take too long), we applied the techniques of dynamic programming to quickly obtain an estimate for the running time of our recursive algorithms as follows. The objective is to build a table of estimated average running times for fixed density and varying numbers of vertices. For small problems we run the actual algorithms on several random graphs and store the average running times in a table. Then for larger problems, we run and time the work done at the top level of recursion, but instead of making recursive calls for various subgraphs of the graph we look up their times in the table, summing them with the time for the preprocessing. In this way the times are built from the bottom up, giving accurate estimates for a wide variety of problems in a fraction of the time it would take to run the algorithms to completion. For each subproblem, the time estimate is looked up in constant time whereas it would take exponential time to compute the time exactly.

Our new dynamic programming methodology allows extensive testing to be completed very quickly. Furthermore, it allows for the selection of an appropriate algorithm in the case when you have a big problem for which you have available perhaps many days cpu time, but not many years. This is important because the estimated times can differ very dramatically. For example, on a 250 vertex random graph with edge density 90%, we have the algorithm GREEDY (algorithms are defined in Section 2) with decreasing sort requiring only an estimated 19 days and 9 hours, while DFMAX with no sorting requires around 1,360 years (see Table 1).

A *pure* recursive clique algorithm applies the same approach at all levels of recursion. A *mixed* strategy varies the tactics employed. Mixed algorithms which use one strategy at the top level of recursion and one other strategy everywhere else have been employed by previous authors. For example, Balas and Yu [5] compute a MTIS to aid in clique finding but only at the top level of recursion. Further, it is common to presort vertices by degree at the top level only. There has been no systematic attempt so far to automate the process of developing the best mixed algorithm for a class of problems. The second step of our research is to use the dynamic programming approach for estimating clique times to automate the process of creating a table driven mixed algorithm which applies the best mixed strategy to each subproblem.

To validate this work, we developed mixed algorithms for random graphs with particular densities and compared the running times to Babel's algorithm [2]. Although our final algorithms are extremely simple, they are significantly faster for all the cases considered.

Section 2 describes the clique algorithms used. Then, a brief summary of the computational results is given in Section 3. We conclude with suggestions for extending our work.

2 Template for the Clique Algorithms

All the algorithms we tested fit into the framework described below. The ordering and restriction strategies are described underneath. All of these ideas have appeared earlier in papers by previous authors.

Clique(G, k)
Input: Graph G and integer k.
Output: **yes** if G has a clique of order k, and **no** otherwise.
1. Order the vertices of the graph as v_0, v_1, ... , v_{n-1}.
2. Apply a restriction rule to partition the vertices into two sets:
   $R=$ u_0, u_1, ... , u_{r-1} and $S=$ u_r, u_k+1, ... , u_{n-1}.
3. Now search the neighbourhoods of the vertices in S:
   For i from $n-1$ down to r do:
      Set N(v_i) equal to the graph induced by the neighbours of vertex i in G.
      If Clique(N(v_i), k-1) return(yes).
      Else set $G = G - v_i$.
4. Return(no)

The ordering rules used in step 1 are one of:
NOT: The vertices are not sorted.
INCREASING: Sort from minimum to maximum degree.
DECREASING: Sort from maximum to minimum degree.
INDUCED: Sort so that the minimum degree vertex is last. Update degrees to reflect deletion of this vertex from the graph before continuing.

The restriction rules used in step 2 are one of:
DFMAX: S consists of all but the first $k-1$ vertices.

Ordering/ Restrictions	*No. Vertices*	*10%*	*50%*	*70%*	*90%*
Not GREEDY	250	0.05406s	16.98s	58min,4.4s	5y,100d
	500	0.36880s	19min,43.4s	49d,2h	>99m.yrs
Not DFMAX	250	0.04573s	27.41s	3h,23min	1,360y
	500	0.29273s	32min,39.4s	157d,19h	>2b.yrs
Not DENSE	250	0.05418s	37.44s	3h,15min	20y,170d
	500	0.35514s	43min,43.3s	135d,17h	>177m.yrs
Decreasing GREEDY	250	0.13685s	23.56s	26min,48s	19d,9h
	500	0.87650s	23min,7.4s	12d	69,876y
Decreasing DFMAX	250	0.11236s	44.87s	1h,37min	16y,162d
	500	0.75072s	39min,21.9s	41d,20h	>13m.yrs
Decreasing DENSE	250	0.12817s	1min,17.8s	4h,3min	20d,18h
	500	0.86133s	1h,17min	133d,22h	236,339y

s = seconds, min = minutes, h = hours, d = days, y = years
m.yrs = million years, b.yrs = billion years

Table 1: Estimated Times for Random Graphs with 250 and 500 Vertices

DENSE: S consists of v_{n-1} and each of its non-neighbours.
GREEDY: Colour the vertices using the greedy colouring algorithm (this algorithm assigns to each vertex the first available colour which does not appear on some neighbour of the vertex). Then set S to be the vertices not coloured by one of the first $k-1$ colours.

The correctness of these approaches is based first on the observation that if G has a k-clique, then for a fixed vertex v, either v is in a k-clique or it is not. If v is in a k-clique, then a $(k-1)$-clique is found when searching the neighbourhood of v. Otherwise, there is a k-clique in $G - v$. The second important point is that for each of these restriction rules, every k-clique is guaranteed to contain at least one vertex from the set S.

3 Computational Results

Due to space limitations, it is impossible to completely describe our computational results. A summary of the main points is given in this section.

The dynamic programming approach to timing the clique algorithms was compared to actual average times. While the results were not perfect, they did provide an accurate indicator of the best algorithm to choose. Some estimated times are given in Table 1. A sample comparison of estimated average times to actual average times is given in Figure 1.

The best pure and mixed strategies are indicated in Figure 2. For testing purposes, random graphs were generated by selecting p and then including each possible edge with probability p. The results given are for p equal to 0.10, 0.30, 0.50, 0.70, and 0.90. The orders of the graphs considered range from 10 to 1000 vertices in steps of 10. The times

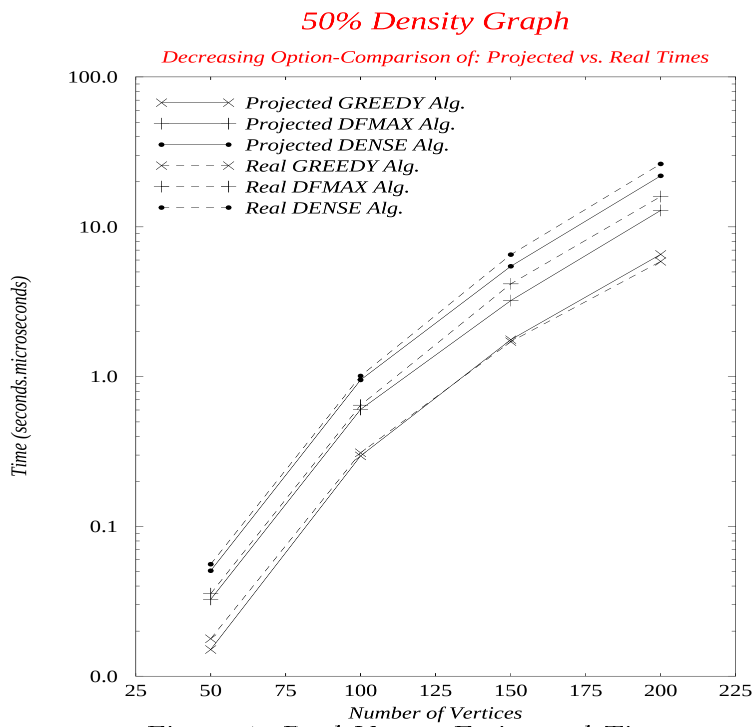

Figure 1: Real Versus Estimated Times

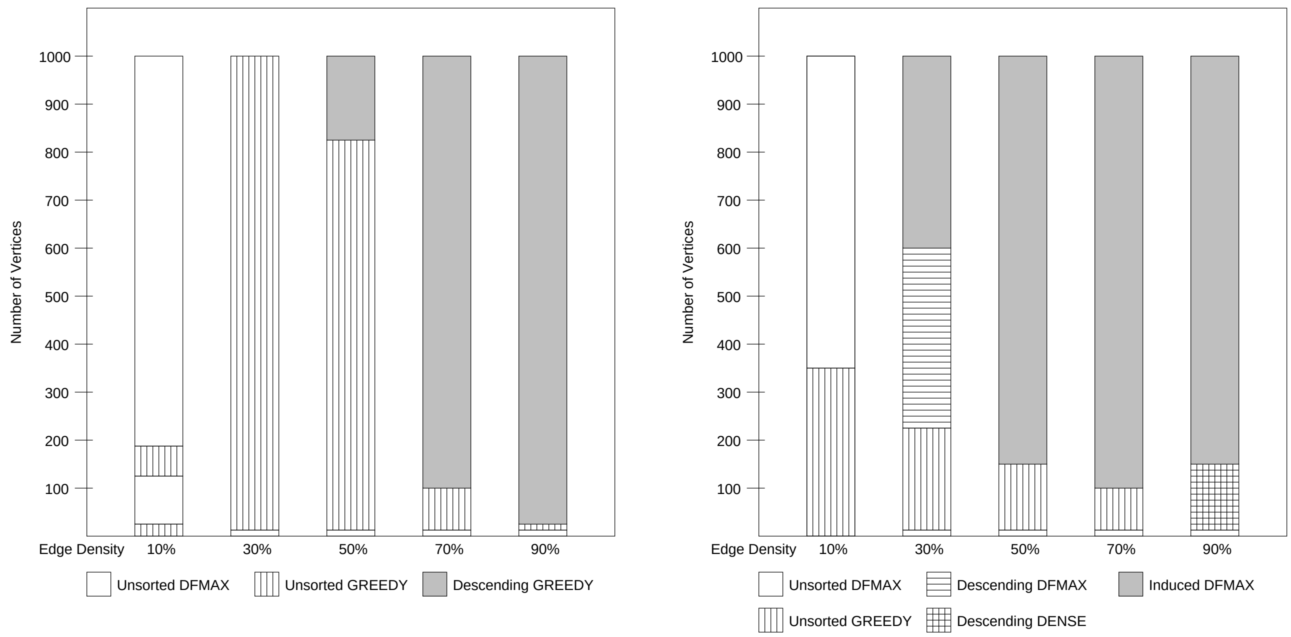

Figure 2: The fastest pure and mixed approaches

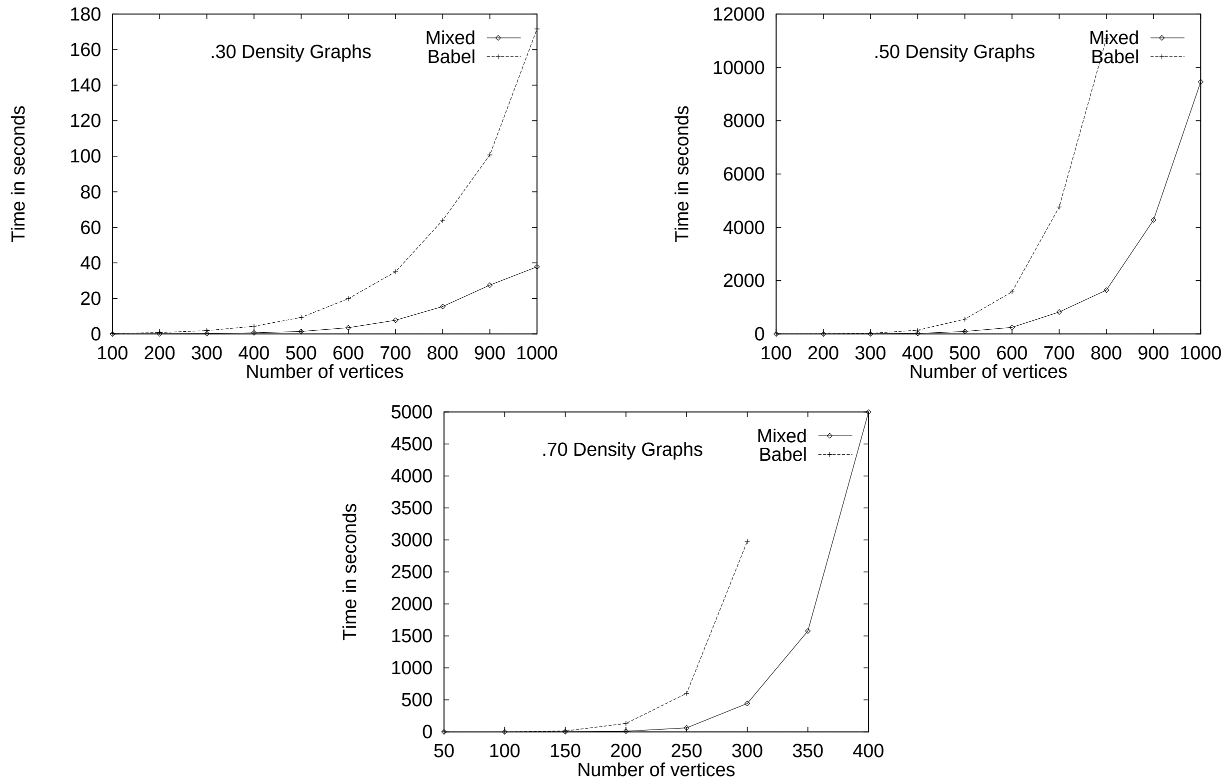

Figure 3: Our Mixed Algorithms versus Babel's Algorithm

represent the average of five random trials in each instance.

The new mixed algorithms were timed and the performance was compared to an implementation of Babel's algorithm obtained from Babel on random graphs having p (the probability that an edge is included) equal to 0.30 0.50, and 0.70 (Figure 3). We chose Babel's algorithm [2] for comparison purposes as it appears to be the fastest published algorithm to date. Both programs were compiled and run on the same machine (a Sun running at 166 MHz). For each order tested, the running times presented in these figures reflect the average over five random trials. For comparison purposes, the benchmarking algorithm of Applegate and Johnson and the mixed algorithm for random graphs of 50% edge density were run on the DIMACS standard random graphs and these times are given in Table 2.

Because Babel's algorithm was significantly slower, the timing experiments could not be completed in a reasonable amount of time for some of the larger orders and higher densities. In all cases, the new mixed algorithms indicate a substantial improvement over Babel's algorithm in spite of their simplicity.

Test Graph	*Benchmark*	*Mixed Time*
r100.5.b	0.00 s	0.02 s
r200.5.b	1.00 s	0.45 s
r300.5.b	6.00 s	3.65 s
r400.5.b	50.00 s	24.06 s
r500.5.b	156.00 s	66.85 s

Table 2: Times to Solve DIMACS Benchmark Graphs

4 Conclusions and Future Research

We have shown that our automated process for developing a clique algorithm is an effective means for developing faster clique algorithms. It is unique in that it makes use of any number of strategies and combines them in what (at least by our estimates) is the most effective way. Attempts to create mixed algorithms in the past have not used such extensive testing and validation mainly because without our estimation process it was prohibitively time consuming.

Some obvious bounding techniques such as finding a MTIS [5], or fractional colouring [4], or the DSATUR colouring due to Brelaz [7] and used by Babel [2] have yet to be considered. Further, there is room for some new ideas as well.

Only very limited testing has been done so far on graphs which are not random. For certain classes with predictable structures, we can proceed as for random graphs. Otherwise, taking random samples from the subproblems encountered could provide an effective alternative.

Finally, it would be interesting to apply these ideas to create recursive algorithms for other NP-complete questions. We expect that there are other such problems where these ideas will improve on the fastest algorithms currently available.

References

[1] J.G. Augustson and J. Minker. An analysis of some graph theoretical cluster techniques. *J. ACM*, 17(4):571–588, 1970.

[2] L. Babel. Finding maximum cliques in arbitrary and in special graphs. *Comput.*, 15:321–341, 1991.

[3] E. Balas and W. Niehaus. Finding large cliques in arbitrary graphs by bipartite matching. In Davis S. Johnson and Michael A. Trick, editors, *Cliques, Colouring, and Satisfiability, Second DIMACS Implementation Challenge, Oct. 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 29–52, 1996.

[4] E. Balas and J. Xue. Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring. *Second DIMACS Challenge, CLIQUE Papers*, Oct. 1993. 12 pages.

[5] E. Balas and C. S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.*, 15(4):1054–1068, 1986.

[6] R. Bar-Yehuda, V. Dabholkar, K. Govindarajan, and D. Sivakumar. Randomized local approximations. *Second DIMACS Challenge, CLIQUE Papers*, Oct. 1993. 12 pages.

[7] D. Brelaz. New methods to color vertices of a graph. *Comm. of the ACM*, 22:251–256, 1979.

[8] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph H. *Comm. ACM*, 16(9):575–577, 1973.

[9] R. Carraghan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Op. Res. Letters*, 9:375–382, 1990.

[10] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., San Fransico, 1979.

[11] C. Mannino and A. Sassano. An exact algorithm for the maximum cardinality stable set problem. *Technical Report IASI-CNR-334*, pages 1–16, April 1992.