Technical Report TR-05-03
A New Approach for Solving the Maximum Clique Problem

P.J. Taillon
School of Computer Science
Carleton University
Ottawa, Ontario, Canada
ptaillon@scs.carleton.ca

ABSTRACT

We describe an improved exact algorithm for solving the MAXIMUM CLIQUE problem in a graph using a novel sampling technique combined with a FPT k-vertex cover algorithm. We also introduce a very effective heuristic for finding a maximum clique that combines our sampling approach with fast independent set approximation. In experiments using the DIMACS benchmark, the heuristic established new lower bounds for four instances and provides the first optimal solution for an instance unsolved until now. The heuristic competitively matched the accuracy of the current best exact algorithm in terms of correct solutions, while requiring a fraction of the runtime.

Categories and Subject Descriptors

G.2 [**Discrete Mathematics**]: Graph Theory

General Terms

Theory, Algorithms, Experimentation

Keywords

Complexity, Parameterized complexity, Graph algorithms, Maximum clique, Heuristics

1. INTRODUCTION

The problem of finding a *vertex cover* in a graph, $G = (V, E)$, that is to say a subset of vertices $VC \subseteq V$, $|VC| \leq K$, such that every edge is adjacent to one of the vertices in VC, is one of the original six problems shown to be NP-complete by Karp [21]. This problem, among its many applications, figures prominently in VLSI design, computational biology, to name a few.

Downey and Fellows [11, 12, 13, 14, 15] developed parameterized tractability as a framework for studying classes of problems that are NP-complete and yet for which there exist efficient algorithms that decide the problem in time bounded by an exponential function of some fixed parameter. Such problems are called *fixed-parameter tractable*, or FPT. Problems that are in the class FPT have the property that their instances can be reduced in polynomial time to instances of size bounded by a function of a fixed parameter, k, and thus FPT algorithms have a complexity described by $O(n^{O(1)} + f(k))$, where n is the size of the problem instance and f is an arbitrary function. For example, the best FPT algorithm that solves the k-VERTEX COVER problem, i.e., we wish to determine if a graph, $G = (V, E)$, has a vertex cover of size bounded by a constant, k, has complexity $O(k|V| + 1.2852^k)$[9]. Although there are many problems that are fixed-parameter tractable not all parameterized problems are in the class FPT. The W-hierarchy consists of classes of problems that are not likely to be fixed-parameter tractable, with the class $W[1]$ representing the lowest level of intractability [16, 15]. For example, k-INDEPENDENT SET is complete for $W[1]$.

A *clique* is a subset, $C \subseteq V$, such the vertices of C form a complete subgraph in G. The CLIQUE problem, like VERTEX COVER, is one of the original NP-complete problems. A wealth of different approaches have been used to tackle the problem, including graph coloring [32] and integer programming formulations [30], not to mention heuristics such as simulated annealing [20], neural networks [27], genetic algorithms [22], etc. We direct the reader to Pelillo [28], or Bomze, et al. [6] for thorough surveys. The *complement graph* of G is the graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(i,j) | i,j \in V, (i,j) \notin E\}$. C is a clique in G if and only if C is an independent set in $\overline{G}$. Finally, a graph G has a vertex cover of size k if and only if the complement graph $\overline{G}$ has a clique of size $n - k$. In parameterized complexity theory, k-CLIQUE is $W[1]$-complete [15]. Abhu-Khzam, Langston, Shanbhag [1], Abhu-Khzam, et al. [2], and later Baldwin, et al. [4], describe experiments that use a parallel FPT k-vertex cover implementation for solving the k-CLIQUE problem by determining a minimum vertex cover in the complemented input graph.

In what follows we summarize the four most recent exact algorithms for the problem. These contributions have been studied experimentally and form a performance standard against which we compare our heuristic algorithm. The algorithm of Wood [31] is based on a branch-and-bound approach that uses a fractional coloring procedure [3] as an upper bound heuristic. Östegard [23] uses a strategy that is similar to dynamic programming, where the problem is solved for subsequently increasing numbers of nodes. An optimal value from previous computations is used as the minimal value for the current problem. The algorithm of Fahle [17] introduces simple cost-based domain filtering [18]. The approach restricts the candidate set by eliminating vertices that can not be used to extend the clique under construction, or by fixing certain vertices that will be in the current clique. The algorithm of Regin [29], also based on constraint programming, proposed two new upper bounds on the largest clique and a new search strategy used to control the search

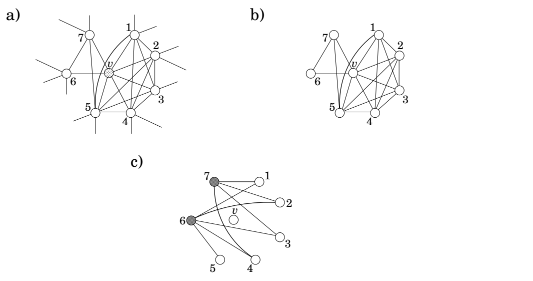

Figure 1: Determining a local maximum clique: (a) subgraph around vertex v, with local maximum clique $\{v, 1, 2, 3, 4, 5\}$; (b) extracted subgraph $G(N[v])$; (c) minimum vertex cover for $\overline{G}(N[v])$ is $\{6, 7\}$ with corresponding maximum independent set of $\{v, 1, 2, 3, 4, 5\}$.

for an optimal solution. It is currently the state-of-the-art for maximum clique algorithms.

2. MAXIMUM CLIQUE ALGORITHMS

In this section, we introduce a new graph sampling technique that we combine with a k-vertex cover algorithm to develop a very fast algorithm for a finding maximum clique in a graph. This approach will greatly reduce the computational complexity of the search in that we will not be required to search an entire complemented graph instance. We will later describe a heuristic for the maximum clique problem that is not based on determining a vertex cover, but instead on calculating an independent set. In experiments this heuristic proves to be very accurate and extremely fast compared to state-of-the-art algorithms.

The algorithms exposited in this section are based on the following theorem. Let $G = (V, E)$ be a graph and assume $v \in V$ is a member of a clique, $C \subseteq V$, of G. To identify all the vertices in C it is sufficient to examine only the subgraph $\overline{G}(N[v])$.

THEOREM 1. *Let $G = (V, E)$ be a graph and assume $v \in V$ is a member of a clique, $C \subseteq V$, of G. If C is the largest clique of which v is a member, then the maximum independent set of the complement of the subgraph $G(N[v])$ is exactly C.*

PROOF. The proof follows from the definition of a clique and the relationship between the MINIMUM VERTEX COVER, MAXIMUM INDEPENDENT SET and MAXIMUM CLIQUE problems. □

2.1 Vertex cover-based maximum clique algorithm

We can in fact derive an algorithm whereby we extract the subgraph $G(N[v])$, form its complement, $\overline{G}(N[v])$, and determine a minimum vertex cover for this subgraph. A maximum independent set for $\overline{G}(N[v])$ is the largest of all cliques in G of which v is a member (see Figure 1). Although determining such a local maximum clique remains of exponential time complexity, the benefit arises from limiting the size of the complemented subgraph that is searched. If our candidate vertex v is a member of the maximum clique then we can determine the maximum clique without having to search the entire complemented graph $\overline{G} = (V, \overline{E})$.

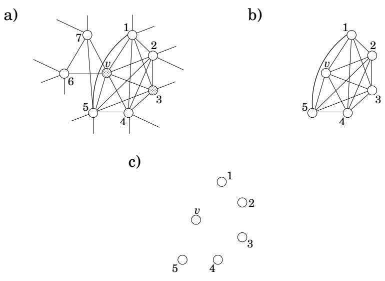

Figure 2: Determining a local maximum clique with 2-level sampling, assuming sample vertices $v, 3$: (a) subgraph around vertex v, with local maximum clique $\{v, 1, 2, 3, 4, 5\}$; (b) extracted subgraph $G(N[v] \cap N[3])$; (c) corresponding maximum independent set of $\{v, 1, 2, 3, 4, 5\}$.

2.1.1 Algorithm description

The algorithm consists of the following steps:

1. Given an input graph, $G = (V, E)$, randomly choose some vertex, v, $\deg(v) > 2$, and extract the subgraph $G(N[v])$;
2. generate the complement of subgraph $G(N[v])$, denoted $\overline{G}(N[v])$;
3. determine a minimum vertex cover of subgraph $\overline{G}(N[v])$;
4. generate the maximum clique by extracting the corresponding maximum independent set;
5. repeat at Step (1) for some fixed number of iterations.

For some instances, the subgraph $G(N[v])$ extracted in Step (2) may be almost as large as the graph G. In such a case, we modify the algorithm as follows:

1. Given an input graph, $G = (V, E)$, randomly choose some vertex, v, $\deg(v) > 2$, and extract the subgraph $G(N[v])$;
2. (a) randomly choose a vertex, $z \in G(N(v))$, $\deg(z) > 2$, and extract the subgraph $G(N[v] \cap N[z])$;
   (b) generate the complement of subgraph $G(N[v] \cap N[z])$, i.e., $\overline{G}(N[v] \cap N[z])$;
   (c) determine a minimum vertex cover of subgraph $\overline{G}(N[v] \cap N[z])$;
   (d) generate the maximum clique by extracting the corresponding maximum independent set;
   (e) repeat at Step (a) for some fixed number of iterations;
3. repeat at Step (1) for some fixed number iterations.

This two-level sampling assumes that if the initial guess of v was correct then we are presented with even better probability of guessing a vertex that is also in the clique, $z \in N(v)$, within the smaller subgraph. The approach accelerates the process of calculating the minimum vertex cover by further restricting the size of the complemented subgraph (see Figure 2). In the case of the heuristic algorithm, it helps generate more precise solutions by limiting the subset of vertices that are inspected by the heuristic.

If we iterate through all vertices we can find the maximum clique for the graph. On the other hand, one can compromise between accuracy and speed: the more random guesses that are made, the more confident one becomes that the largest clique has been (or will be) found. As we will see, this contrasts with other search algorithms that can fail to find a clique within some allotted time, or can not improve on the largest clique no matter how much time is allowed.

2.2 Heuristic maximum clique algorithm

As with most heuristics, one is willing to trade-off the cost of attempting to find an optimal solution for an algorithm of lower complexity that returns possibly suboptimal solution quickly. Having established a viable means of finding a maximum clique in a graph in the Section 2.1, we introduce a fast and effective heuristic for finding a large clique in a graph. As we will see in Section 2.4, although state-of-the-art algorithms may attempt to determine a maximum clique through exhaustive search, in several cases they do not return a solution in a reasonable amount of time.

2.2.1 *Algorithm description*

The algorithm consists of the same steps as shown in Section 2.1, except that we use the heuristic below to calculate a maximum independent set (cf. ([25, 9])) in Step 2(c) for subgraph $\overline{G}(N[v] \cap N[z])$ rather than determining a minimum vertex cover. We assume the complemented subgraph $\overline{G}(N[v] \cap N[z])$ has already been generated.

1. Select vertex, $w \in N[v] \cap N[z]$, where w has minimum degree;
2. add w to the independent set;
3. remove $N[w]$ from $\overline{G}(N[v] \cap N[z])$;
4. repeat at Step (1) until $\overline{G}(N[v] \cap N[z])$ has no edges.

2.3 Analysis

Let $G = (V, E)$ be a graph with n vertices and let $0 < \alpha \leq 1$. The algorithm has time complexity $O(\alpha n \cdot T(n_i))$, where $i \in V$ is the vertex with the largest degree, $n_i = |N[i]|$, and $T(\cdot)$ represents the cost of finding a clique.

The analysis for the exact solution and the heuristic is essentially the same except for the cost $T(n_i)$ to find the maximum independent set in the complement subgraph. Determining the (exact) maximum independent set takes $O(\overline{k}n_i + c^{\overline{k}})$ using a k-vertex cover algorithm solving for minimum k and extracting the corresponding maximum independent set. The constant c depends on the underlying vertex cover algorithm. Finding the independent set heuristically requires $\lceil (n_i - 1)/d \rceil$ steps, where d is the maximum degree taken over all the vertices in the subgraph $\overline{G}(N[i] \cap N[j])$, $i \neq j$.

The impact of sampling is potentially quite significant, in that solving several small local minimum vertex cover instances is far and away quicker than solving one single instance on the full graph. We restrict our analysis to the algorithm presented in Section 2.1 that guesses one vertex at a time. The analysis can be easily extended to the case using two-level sampling. Recall the relationship between a minimum vertex cover, a maximum independent set, and a maximum clique. A set of vertices, $C \subseteq V$ with $m = |C|$, is a maximum clique in G if and only if C is a maximum independent set in $\overline{G}$. A minimum vertex cover for $\overline{G}$ of size $\overline{k}$ induces a independent set of size $n - \overline{k}$. We know $m = n - \overline{k}$ in $\overline{G}$ and thus $\overline{k} = n - m$. Now consider a complemented subgraph, $\overline{G}(N[v_i])$, and let $n_i = |N[v_i]|$ for some $v_i \in V$. Assuming our hypothesis that $v_i \in C$ then the maximum clique size is fixed for both G and $G(N[v_i])$ implying that the minimum cover size for the subgraph is $\overline{k}_i = n_i - m$. We conclude that determining a minimum vertex cover in $\overline{G}$ requires $O((n - m)n + c^{n-m})$ time, compared to that of finding a minimum cover in $\overline{G}(N[v_i])$ with complexity $O((n_i - m)n_i + c^{n_i - m})$, where $n_i \leq n$.

2.4 Experimental results

In general, algorithms for finding maximum cliques rely on massive experimentation to determine their effectiveness. To expedite our experiments, we make use of two separate platforms. Our first experimental platform consisted of a shared-memory SunFire 6800, configured with 20 900MHz UltraSPARC-III processors and 20GB of memory. Our second experimental platform consisted of a 2.0GHz Intel Xeon processor with 512MB RAM and 60GB of disk storage. Execution time was measured as wall clock time in seconds and includes the time taken to read the input graph from a file and output the solution size.

A *run* consists of a single execution of the code, given some sampling parameters, out of which the size of the largest clique found is output. An *experiment* consists of some number of runs on a given instance and taking the largest clique produced out of all the runs. During a run on instance, $G_i = (V_i, E_i)$, the code generates a random sample set of vertices, $S_1 \subseteq V_i$, $|S_1| = \alpha_1 |V_i|$. For each vertex $v \in S_1$, we generate a second random sample set, $S_2 \subseteq N(v)$, $|S_2| = \alpha_2 |S_1|$. Finally, for each $z \in S_2$ we extract the subgraph induced by $G(N[v] \cap N[z])$, complement the subgraph, and extract the maximum independent set. This clique is then checked in the original graph to ensure its correctness.

It is difficult to directly compare the algorithm of Section 2.1 with the results in [1, 2, 4], as they have implemented a different kernelization algorithm. Instead we use a readily-available FPT k-vertex cover algorithm described in [7] and compare it against itself, i.e., we measure the performance of our k-vertex cover code running on a complemented graph instance versus our new maximum clique algorithm. This provides a relative performance comparison that carries over to any k-vertex cover algorithm. Because of the time required to calculate an exact minimum vertex cover, we restrict our experiment to consisting of one run for the parallel k-vertex cover algorithm using 9 processors on the first platform. This is compared to one run of the maximum clique algorithm on the same input with sampling parameters $\alpha_1 = 0.10$ and $\alpha_2 = 0.05$. We justify using the parallel platform in this series because the minimum vertex cover of a complemented graph can become significantly large and we wish to conduct the experiments in a reasonable time.

Given the shorter execution time of the heuristic algorithm, each experiment is repeated sequentially 50 times on each graph instance using our second platform. We consider an experiment to consist of 10 runs of the heuristic, and the time for an experiment is taken as the average time of the 10 runs. We take as sampling parameters $\alpha_1 = \alpha_2 = 0.10$. We compare the heuristic performance with implementations of the algorithms of Wood [31], Östegard [23], Fahle [17], and

Graph	$\|K\|$	Density	$\|V\|$	$\|E\|$
Somatostatin	14	0.22	559	33652
WW	22	0.45	425	40182
Thrombin (10)	24	0.19	646	40717

Table 1: Graph instances derived from experimental biological data.

Graph	$\|K\|$	Density	$\|V\|$	$\|E\|$
globin10	12	0.16	972	75,386
globin15	23	0.32	972	149,473
pp_sh2-3	3	0.05	148	494
pp_sh2-10	17	0.27	726	69,982
sh2-5	7	0.08	839	26,612
sh2-10	23	0.37	839	129,697

Table 2: Graph instances derived from experimental biological data.

Regin [29]. Our platform closely resembles that of Regin[1] so we can draw directly upon his observations to evaluate our algorithm against previous work. In Section 2.4.3 we present a table adapted from [29] that includes measurements for all four algorithms.

2.4.1 Data sets

We used two different types of graphs for our experiments with the objective of studying the performance of the algorithms using both real-world and public domain benchmark data. The first set of graphs, Table 1, comprise a small subset selected from our experiments in [7, 8]. The second set, Table 2, are those cited in Baldwin, et al. [4] from which the authors derived phylogenetic trees based on proteins domains. The third set comprise the atendentious DIMACS benchmark [10].

2.4.2 FPT k-vertex cover maximum clique algorithm performance

The objective of this experiment is to demonstrate the efficacity of the algorithm from Section 2.1. We selected the instances *WW*, *Somatostatin*, and *Thrombin*, and used the parallel k-vertex cover code described in [7, 8], running on 9 processors.

For each graph instance we measure the total execution time, the time required before a clique matching the optimal size is found, and the number of times a clique matching the optimal size is found. We emphasize the relative speed with which an optimal clique is found rather than the actual run-times. In Table 3 we observe that the sampling algorithm completed in a shorter time than that required to find a minimum vertex cover in a complemented graph. The experiment with the *Thrombin* instance highlights one of the benefits of the new maximum clique algorithm. The new algorithm generated several cliques of size 23 within 48 hours of starting and ultimately found several cliques matching the optimal size before termination. In contrast, the algorithm searching for a minimum vertex cover in the complemented graph had in that same time only processed two cover instances (sizes $k = 623$ and $k = 621$). The execution on the complemented graph was terminated after 2 weeks, never having completed its search.

2.4.3 Maximum clique heuristic performance

The software we developed for these experiments used libraries of graph and set manipulation routines made available by Östegard [24].

We see from Table 4 that the heuristic performed excellently on real-world data, with the exception instance *Sh2-10*. In the latter case the average clique size produced by the heuristic was very close to the optimal.

We follow the methodology of Regin for testing the performance of the heuristic algorithm against the DIMACS clique benchmark set, i.e., code execution on a given instance taking longer than 14,400 seconds is terminated. Table 5 summarizes the performance data for Wood, Östegard, Fahle, and Regin's algorithms for the benchmark (adapted from [29]). The timing measurements have been scaled appropriately to compensate for different processor speeds.

We divide the datasets in two groups, based on the amount of time they require to complete the series of experiments within the 4 hour time limit. For each graph instance we measure the average clique size found during the 50 experiments, the average time of all the runs, the average time for the experiments, and a count of the number of times that an experiment found a clique that matched the largest known size. Table 6 consists of instances that were sampled with $\alpha_1 = \alpha_2 = 0.10$. The same methodology is used to test the algorithm on instances in Table 2. As the instances in Table 7 were large, we limited the sampling to $\alpha_1 = 0.01$ and $\alpha_2 = 0.05$.

In comparing the results in Table 5 and Table 6 we see the heuristic algorithm consistently outperforms the algorithms of Wood, Östegard, and Fahle, and is faster than that of Regin while producing a competitive number of optimal or similar answers. In general, when the heuristic fails it does so for the same instances as does the current best algorithm, while generating comparable suboptimal answers in a fraction of the time. For example, the solution to the *Brock800_1* instance matches the current best suboptimal answer, but required much less time to compute (80 seconds). Another such instance is *P_hat700-3*, where the heuristic found a solution matching the best lower bound in less than a minute. Regin's algorithm exceeded the 4 hour time limit while finding a solution of the same size. There are many cases where the heuristic finds the largest clique extremely quickly, whereas the algorithm of Regin either fails to terminate within the 4 hour time limit (e.g., *Johnson32-2-4*, *Keller5*) or takes noticeably longer (e.g., *San400_0.9_1*, *Sanr400_0.7*).

Most importantly, the heuristic improves on the best lower bound for four as yet unsolved instances, and solves a fifth instance optimally. For the instance *Brock800_3*, it determined a clique of size 22. The previous best lower bound for these graphs was 20. For the instances *Brock800_2* and *Brock800_4* the heuristic found cliques of size 21 in each graph. The heuristic found a clique of size 55 in the *Keller6* instance, where previously the best suboptimal clique size was 54. For the instance *P_hat1500-2* the heuristic found a clique matching the optimal size of 65. The best lower bound until now was size 63 as determined by Regin.

We observe two cases of oversampling. When solving the *Hamming10-2* instance (Table 6), the algorithm found a

[1]His platform was an Intel Pentium IV, 2.0GHz, 512MB RAM.

Graph	$\|K\|$	k-VC: $\overline{G}$	k-VCMax. clique: two-level sampling (10%, 5%)		
		Total time (hr)	Total time (hr)	Largest first found (hr)	No. of largest
Somatostatin	14	02:18:37	00:29:40	00:03:03	7
WW	22	43:34:15	19:33:12	02:45:01	21
Thrombin (10)	24	DNF	198:42:11	116:39:45	10

Table 3: Summary of k-vertex cover-based maximum clique algorithm results using biological datasets.

Graph	$\|K\|$	Avg. $\|K\|$	Avg. run time (s)	Avg. exp. time (s)	Largest found
globin10	12	12.00	4.52	45.22	50
globin15	23	22.57	29.87	298.70	50
pp_sh2-3	3	1.81	0.01	0.10	50
pp_sh2-10	17	16.92	6.81	68.06	50
sh2-5	7	6.26	0.27	2.72	48
sh2-10	23	22.00	28.91	289.10	11

Table 4: Summary of heuristic algorithm results using biological datasets.

DIMACS		Wood		Östegard		Fahle		Regin	
Graph	$\|K\|$	$\|K\|$	Time (s)	$\|K\|$	Time (s)	$\|K\|$	Time (s)	$\|K\|$	Time (s)
brock200_1	21	21	53.68	21	18.10	21	92.97	21	10.72
brock200_2	12	12	0.26	12	0	12	0.31	12	0.29
brock200_3	15	15	2.57	15	0.15	15	2.23	15	0.86
brock200_4	17	17	6.20	17	0.33	17	8.18	17	2.13
brock400_1	27		fail		fail	≥ 24	fail	27	11,340.80
brock400_2	29		fail		fail	≥ 29	fail	29	7,910.60
brock400_3	31		fail		fail	≥ 24	fail	31	4,477.23
brock400_4	33		fail		fail	≥ 25	fail	33	6,051.77
brock800_1	**23**		**fail**		**fail**	$\geq$ **21**	**fail**	$\geq$ **21**	**fail**
brock800_2	**24**		**fail**		**fail**	$\geq$ **20**	**fail**	$\geq$ **20**	**fail**
brock800_3	**25**		**fail**		**fail**	$\geq$ **20**	**fail**	$\geq$ **20**	**fail**
brock800_4	**26**		**fail**		**fail**	$\geq$ **20**	**fail**	$\geq$ **20**	**fail**
c-fat200-1	12	12	0	12	0	12	0	12	0
c-fat200-2	24	24	0	24	0	24	0	24	0
c-fat200-5	58	58	0	58	2.60	58	0	58	0
c-fat500-1	14	14	0	14	0.02	14	0	14	0
c-fat500-10	126	126	0	126	0.02	126	0.02	126	0.04
c-fat500-2	26	26	0	26	0.03	26	0	26	0
c-fat500-5	64	64	0	64	3,480.21	64	0.02	64	0
hamming10-2	512	512	0	512	0.84	512	5.16	512	1.04
hamming10-4	≥ 40		fail		fail	≥ 32	fail	≥ 40	fail
hamming6-2	32	32	0	32	0	32	0	32	0
hamming6-4	4	4	0	4	0	4	0	4	0
hamming8-2	128	128	0	128	0	128	0.07	128	0
hamming8-4	16	16	5.28	16	0.28	16	6.11	16	4.19
johnson16-2-4	8	8	13.05	8	0.09	8	7.91	8	3.80
johnson32-2-4	$\geq$ **16**		**fail**		**fail**	$\geq$ **16**	**fail**	$\geq$ **16**	**fail**
johnson8-2-4	4	4	0	4	0	4	0	4	0
johnson8-4-4	14	14	0	14	0	14	0.03	14	0
keller4	11	11	1.23	11	0.17	11	2.53	11	0.50
keller5	**27**		**fail**		**fail**	$\geq$ **25**	**fail**	$\geq$ **27**	**fail**
keller6	$\geq$ **59**		**fail**		**fail**	$\geq$ **43**	**fail**	$\geq$ **54**	**fail**
MANN_a9	16	16	0	16	0	16	0	16	0
MANN_a27	126	126	46.95		fail	126	10,348.87	126	18.48
MANN_a45	345		fail		fail	≥ 331	fail	≥ 345	fail
MANN_a81	$\geq 1,100$		fail		fail	≥ 996	fail	$\geq 1,100$	fail
p_hat300-1	8	8	0.10	8	0	8	0.07	8	0.11
p_hat300-2	25	25	0.67	25	0.33	25	3.01	25	0.59
p_hat300-3	36		fail		fail	36	856.67	36	40.71
p_hat500-1	9	9	0.91	9	0.10	9	0.60	9	2.30
p_hat500-2	36	36	17.81	36	142.93	36	203.93	36	32.69
p_hat500-3	50		fail		fail	≥ 48	fail	50	12,744.70
p_hat700-1	11	11	2.69	11	0.22	11	2.67	11	6.01
p_hat700-2	44		fail		fail	44	2,086.63	44	255.79
p_hat700-3	$\geq$ **62**		**fail**		**fail**	$\geq$ **54**	**fail**	$\geq$ **62**	**fail**
p_hat1000-1	10	10	18.88	10	1.95	10	16.43	10	27.80
p_hat1000-2	46		fail		fail	≥ 44	fail	46	16,845.70
p_hat1000-3	≥ 68		fail		fail	≥ 50	fail	≥ 66	fail
p_hat1500-1	12		fail		fail	12	119.77	12	480.84
p_hat1500-2	$\geq$ **65**		**fail**		**fail**	$\geq$ **52**	**fail**	$\geq$ **63**	**fail**
san1000	15	15	43.59	15	0.17	15	3,044.09	15	102.80
san200_0.7_1	30	30	0.06	30	0.19	30	1.57	30	0.36
san200_0.7_2	18	18	0.03	18	0	18	0.66	18	0.37
san200_0.9_1	70	70	0.77	70	0.09	70	62.61	70	1.04
san200_0.9_2	60	60	70.13	60	1.43	60	1,930.90	60	2.62
san200_0.9_3	44		fail		fail	44	194.96	44	182.70
san400_0.5_1	13	13	0.75	13	0	13	6.74	13	1.19
san400_0.7_1	40	40	13.25		fail	40	425.99	40	23.28
san400_0.7_2	30	30	415.12	30	168.70	30	159.72	30	67.53
san400_0.7_3	22		fail		fail	22	617.07	22	273.23
san400_0.9_1	**100**		**fail**		**fail**	**100**	**7,219.53**	**100**	**1,700.00**
sanr200_0.7		18	22.50	18	4.70	18	24.99	18	4.30
sanr200_0.9			fail		fail	≥ 41	fail	42	150.08
sanr400_0.5		13	22.55	13	2.21	13	23.09	13	17.12
sanr400_0.7			**fail**		**fail**	**21**	**15,925.00**	**21**	**3,139.11**

Table 5: Performance comparison of recent maximum clique algorithms.

clique matching the largest known size very quickly, but it continued to search through its rather large sample set, incurring an unnecessarily long runtime. In the case of the *MANN_a81* instance (Table 7), the heuristic found a relatively large clique in one run within the time limit, although the experiment exceeded the time limit.

Graph	$\lvert K\rvert$	Avg. $\lvert K\rvert$	Avg. run time (s)	Avg. exp. time (s)	Largest found
brock200_1	21	20.45	0.56	5.64	50
brock200_2	12	10.91	0.15	1.52	48
brock200_3	15	14.04	0.28	2.82	49
brock200_4	17	15.97	0.38	3.80	41
brock400_1	27	24.08	8.17	81.68	0
brock400_2	29	24.35	8.17	81.70	17
brock400_3	31	24.61	8.13	81.32	22
brock400_4	33	26.49	8.19	81.92	47
brock800_1	**23**	**20.81**	**225.41**	**2,254.14**	**0**
brock800_2	**24**	**20.78**	**223.14**	**2,231.40**	**0**
brock800_3	**25**	**21.13**	**245.43**	**2,454.32**	**0**
brock800_4	**26**	**20.55**	**221.80**	**2,218.06**	**0**
c-fat200-1	12	12.00	0.014	0.14	50
c-fat200-2	24	23.91	0.03	0.30	50
c-fat200-5	58	58.00	0.13	1.28	50
c-fat500-1	14	14.00	0.03	0.30	50
c-fat500-10	126	126.00	2.63	26.34	50
c-fat500-2	26	26.00	0.10	1.04	50
c-fat500-5	64	64.00	0.61	6.10	50
hamming10-2	512	512.00	1,900.45	19,004.48	50
hamming10-4	$\geq$ 40	36.00	439.04	4,390.44	0
hamming6-2	32	32.00	0.03	0.26	50
hamming6-4	4	4.00	0.01	0.08	50
hamming8-2	128	128.00	4.10	40.98	50
hamming8-4	16	16.00	0.77	7.74	50
johnson16-2-4	8	8.00	0.09	0.90	50
johnson32-2-4	**$\geq$ 16**	**16.00**	**25.30**	**253.04**	**50**
johnson8-2-4	4	4.00	0.01	0.08	50
johnson8-4-4	14	14.00	0.02	0.22	50
keller4	11	11.00	0.20	2.00	50
keller5	**27**	**27.00**	**106.67**	**1,066.74**	**50**
MANN_a9	16	16.00	0.01	0.14	50
MANN_a27	126	125.00	24.95	249.48	0
p_hat300-1	8	7.89	0.10	1.04	50
p_hat300-2	25	24.93	0.84	8.42	50
p_hat300-3	36	34.69	2.83	28.32	26
p_hat500-1	9	9.00	0.60	6.00	50
p_hat500-2	36	35.66	6.59	65.92	50
p_hat500-3	50	49.02	21.81	218.10	37
p_hat700-1	11	10.29	1.94	19.38	50
p_hat700-2	44	43.99	24.74	247.44	50
p_hat700-3	**$\geq$ 62**	**60.26**	**4.50**	**44.96**	**25**
p_hat1000-1	10	10.00	6.81	68.14	50
p_hat1000-2	46	45.72	91.62	916.20	50
p_hat1000-3	$\geq$ 68	65.08	331.23	3,312.30	0
p_hat1500-1	12	11.22	35.58	355.76	45
p_hat1500-2	**65**	**64.09**	**514.32**	**5,143.22**	**34**
san1000	15	10.48	5.09	50.94	29
san200_0.7_1	30	29.72	0.47	4.72	50
san200_0.7_2	18	17.12	0.47	4.72	50
san200_0.9_1	70	70.00	1.18	11.76	50
san200_0.9_2	60	59.98	1.10	10.96	50
san200_0.9_3	44	42.14	1.06	10.58	50
san400_0.5_1	13	11.60	2.46	24.60	50
san400_0.7_1	40	39.94	6.63	66.26	50
san400_0.7_2	30	29.62	6.46	64.56	50
san400_0.7_3	22	20.31	6.43	64.28	50
san400_0.9_1	**100**	**100.00**	**16.42**	**164.18**	**50**
sanr200_0.7	18	17.77	0.45	4.54	50
sanr200_0.9	42	41.37	1.08	10.76	50
sanr400_0.5	13	12.58	1.91	19.12	50
sanr400_0.7	**21**	**20.82**	**6.46**	**64.56**	**50**

Table 6: Summary of heuristic algorithm results using DIMACS benchmark.

3. CONCLUSION

We describe an improved exact algorithm for solving the Maximum Clique problem in a graph using a novel sampling technique combined with the FPT k-vertex cover algorithm. We also introduced a very effective heuristic for finding a maximum clique that combines our sampling approach with fast independent set approximation. In experiments using the DIMACS benchmark, the heuristic established new lower bounds for four instances and provides the first optimal solution for an instance unsolved until now. The heuristic competitively matched the accuracy of the current best exact algorithm in terms of correct solutions, while requiring a fraction of the runtime.

The research presented here opens several exciting avenues for future exploration, which we are currently pursuing. We note that Pardalos and Xue [26] identify the following three issues as fundamental to branch-and-bound algorithms for the Maximum Clique problem: determination of good lower bounds, determination good upper bounds, and suitable subproblem generation. The results in this paper address the first and, to some extent, third of these issues. Because of the speed of the heuristic, it could be used as a preprocessing phase for other clique algorithms, by providing a relatively accurate lower bound for the largest clique in the graph. Our sampling strategy provides a natural means of breaking the original instance into subproblems. We are also investigating how our maximum clique algorithms could be augmented with various clique size-bounding heuristics as pruning strategies to improve performance. Finally, we are investigating the use of our code to solve an instance of *Keller7* using a parallel approach based on the implementation described in this paper. This graph has $14,190$ vertices, $87,091,347$ edges, and is conjectured to have a maximum clique of size somewhere between 123 and 127 vertices.

4. ACKNOWLEDGMENTS

The author wishes to thank Robert C. Kane for his helpful technical support. Thanks go also to the High-Performance Computing Virtual Laboratory (HPCVL) for the use of its excellent resources.

5. REFERENCES

[1] F.N. Abu-Khzam, M.A. Langston, P. Shanbhag. "Scalable parallel algorithms for difficult combinatorial problems: A

Graph	$\|K\|$	Avg. $\|K\|$	Avg. run time (s)	Avg. exp. time (s)	Largest found
keller6	$\geq$ **59**	**55.00**	**1,007.27**	**10,072.70**	**0**
MANN_a45	345	342.00	132.74	1,327.40	0
MANN_a81	$\geq 1,100$	1,096.00	13,961.40	139,614.00	0

Table 7: Summary of heuristic algorithm results using large DIMACS instances.

case study in optimization". In *Proceedings of the International Conference on Parallel and Distributed Computing and Systems*, November, 2003.

[2] F.N. Abu-Khzam, R.L. Collins, M.R. Fellows, M.A. Langston, W.H. Suters, C.T. Symons. "Kernelization algorithms for the vertex cover problem: Theory and experiments". In *Proceedings of the ACM-SIAM Workshop on Algorithm Engineering and Experiments*, January, 2004.

[3] E. Balas, J. Xue. "Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring". In *Algorithmica*, Vol.15, pp.397–412, 1996.

[4] N.E. Baldwin, R.L. Collins, M.R. Leuze, M.A. Langston, C.T. Symons, B.H. Voy. "High-performance computational tools for motif discovery". In *Proceedings of the IEEE International Workshop on High Performance Computational Biology*, April, 2004.

[5] R. Bar-Yehuda, V. Dabholkar, K. Govindarajan, D. Sivakumar. "Randomized local approximations with applications to the MAX-CLIQUE problem". Technical Report 93-30, University at Buffalo, 1993.

[6] I.M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo. "The maximum clique problem". In *Handbook of Combinatorial Optimization*, Du, Pardalos (Eds.), Vol.A, Kluwer, pp.1–74, 1999.

[7] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, P.J. Taillon. "Solving large FPT problems on coarse grained parallel machines". In *Journal of Computer and System Sciences*, Vol.67, No.4, pp.691–706, 2003.

[8] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, P.J. Taillon. "A parallel FPT application for clusters". In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, 2003, pp.70–77, 2003.

[9] J. Chen, I.A. Kanj, W. Jia. "Vertex cover: Further observations and further improvements". In *Proceedings of the 25th International Workshop on Graph-Theoretical Concepts in Computer Science (WG'99)*, LNCS 1665, pp.313–324, 1999.

[10] DIMACS clique benchmarks. *ftp://dimacs.rutgers.edu/pub/challenge/graph/*, 1993.

[11] R.G. Downey, M.R. Fellows. "Fixed-parameter tractability and completeness." In *Congressus Numerantium*, Vol.87, pp.161–187, 1992.

[12] R.G. Downey, M.R. Fellows. "Parameterized computational feasibility". In *Feasible Mathematics II*, Clote, Remmel (Eds.), Birkhauser, pp.219–244, 1995.

[13] R.G. Downey, M.R. Fellows."Fixed parameter tractability and completeness I: Basic theory". In *SIAM Journal of Computing*, Vol.24, pp.873–921, 1995.

[14] R.G. Downey, M.R. Fellows. "Fixed parameter tractability and completeness II: Completeness for $W[1]$". In *Theoretical Computer Science A*, Vol.141 , pp.109–131, 1995.

[15] R.G. Downey, M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.

[16] R.G. Downey, M.R. Fellows, K.W. Regan. "Parameterized Circuit Complexity and the W Hierarchy". In *Theoretical Computer Science A*, Vol.191, pp.91–115, 1998.

[17] T. Fahle. "Simple and fast: Improving a branch-and-bound algorithm for maximum clique". In *10th Annual European Symposium (ESA'02)*, pp.485–498, 2002.

[18] F. Focacci, A. Lodi, M. Milano. "Cost-based domain filtering". In *Proceedings of CP'99*, LNCS 1713, pp. 189–203, 1999.

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

[20] S. Homer, M. Peinado. "Experiments with polynomial-time CLIQUE approximation algorithms on very large graphs". In *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, Johnson, Trick, (Eds.), American Mathematical Society, DIMACS Vol.26, pp.147–167, 1996.

[21] R.M. Karp. "Reducibility among combinatorial problems". In *Complexity of Computer Computations*, Miller, Thatcher (Eds.), Plenum Press, pp.85–103, 1972.

[22] E. Marchiori. "A simple heuristic based genetic algorithm for the maximum clique problem". In *Proceedings of the ACM Symposium on Applied Computing (SAC'98)*, pp.366–373, 1998.

[23] P.R.J. Östegard. "A new algorithm for the maximum-weight clique problem". In *Nordic Journal of Computing*, Vol.8, No.4, pp.424–436, 2001.

[24] P.R.J. Östegard. Private communication, 2004.

[25] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[26] P.M. Pardalos, J. Xue. "The maximum clique problem". In *Journal of Global Optim.*, Vol.4, pp.301–328, 1994.

[27] M. Pelillo. "Relaxation labeling networks for the maximum clique problem". In *Journal of Artificial Neural Networks*, Vol.2, pp.313–328, 1995.

[28] M. Pelillo. "Heuristics for maximum clique and independent set". In *Encyclopedia of Optimization*, Floudas, Pardalos (Eds.), Kluwer Academic, Vol.2, pp.411–423, 2001.

[29] J.-C. Regin. "Solving the maximum clique problem with constraint programming". In *Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2003.

[30] N.Z. Shor. "Dual quadratic estimates in polynomial and Boolean programming". In *Computational Methods in Global Optimization*, Pardalos, Rosen (Eds.), Ann. Oper. Res., Vol.25, pp.163–168, 1990.

[31] D.R. Wood. "An algorithm for finding maximum cliques in a graph". In *Operations Research Letters*, Vol.21, pp.211–217, 1997.

[32] J. Xue. *Fast Algorithms For Vertex Packing and Related Problems*. Ph.D. Thesis, GSIA, Carnegie Mellon University, 1991.