Proceedings of the Ninth International
Symposium on Combinatorial Search (SoCS 2016)

Finding Maximum k-Cliques Faster Using Lazy Global Domination

Ciaran McCreesh* and **Patrick Prosser**
University of Glasgow, Glasgow, Scotland

Abstract

A clique in a graph is a set of vertices, each of which is adjacent to every other vertex in this set. A k-clique relaxes this requirement, requiring vertices to be within a distance k of each other, rather than directly adjacent. In theory, a maximum clique algorithm can easily be adapted to solve the maximum k-clique problem, although large sparse k-clique graphs reduce to large dense clique graphs, which can be computationally challenging. We adapt a state of the art maximum clique algorithm to show that this reduction is in fact useful in practice, and introduce a lazy global domination rule which sometimes vastly reduces the search space. We include experimental results for a range of real-world and benchmark graphs, and a detailed look at random graphs. We also use thread-parallel search to solve some harder instances.

Introduction

A clique in a graph is a set of vertices, each of which is adjacent to every other vertex in the set. Finding a clique of maximum size in a graph is one of the basic NP-hard problems (Garey and Johnson 1990); applications include geometry, coding theory, computer vision and bioinformatics (Bomze et al. 1999; Butenko and Wilhelm 2006). However, when analysing real-world data, a clique may be too strong a requirement. A k-clique (or sometimes n-clique or s-clique—in an unfortunate clash of notation, "k-clique problem" is sometimes used elsewhere for the decision version of the clique problem, to distinguish it from the maximum clique problem) is a relaxed form of clique, where instead of requiring each pair of vertices to be directly adjacent, we only require that they be connected by a path of length at most k (Luce 1950). Thus a 1-clique *is* a clique, a 2-clique may be thought of as "a group of people, all of whom either know each other or have a mutual acquaintance", and so on. We illustrate this in Figure 1. Determining the size of a maximum k-clique is NP-hard for any fixed k (Bourjolly, Laporte, and Pesant 2002).

A related relaxation is a k-club, which tightens the requirement of a k-clique as follows (Mokken 1979). In a k-clique, each pair of vertices is connected by a path of length at most k, but that path may use any vertices in the original graph. In a k-club, each pair of vertices must be connected by a path of length at most k using only vertices that are also in the club. Thus the 2-clique in Figure 1 is *not* a 2-club (obviously, every k-club is a k-clique).

A recent survey by Shahinpour and Butenko (2013) discusses algorithms and results for k-clique and k-club problems. We adopt their notation of $\tilde{\omega}_k$ for the size of a maximum k-clique; the use of ω for the size of a maximum clique is standard. They note that "unlike the maximum clique problem, the maximum s-clique problem has not been the subject of extensive research and we are not aware of any computational results for this problem to date". This is in contrast to the k-club problem, for which a wide range of computational results are available (Bourjolly, Laporte, and Pesant 2000; 2002; Mahdavi Pajouh and Balasundaram 2012; Hartung, Komusiewicz, and Nichterlein 2012; Chang et al. 2013; Shahinpour and Butenko 2013; Wotzlaw 2014; Picker 2015; Carvalho and Almeida 2016).

A maximum clique algorithm can easily be adapted to find a maximum k-clique in a graph G by considering the graph G^k: this graph has the same vertex set as G, and edges between any two distinct vertices v_1 and v_2 iff there is a path of length at most k between v_1 and v_2 in G. It is easy to see that maximum cliques in G^k correspond with maximum k-cliques in G (Balasundaram, Butenko, and Trukhanov 2005). However, it is not obvious that this is a viable approach: even if G is sparse, G^k may not be, and the maximum clique problem on dense graphs can be very challeng-

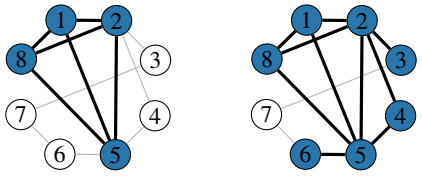

Figure 1: On the left, a graph, with its unique maximum clique $\{1, 2, 5, 8\}$ of size 4 highlighted. On the right, the same graph, with a maximum 2-clique $\{1, 2, 3, 4, 5, 6, 8\}$ of size 7 highlighted. This is not a 2-club, since the only path of length 2 between vertices 3 and 6 goes through vertex 7. A 3-clique covers the entire graph.

*This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/K503058/1]

ing computationally. Here we take a state-of-the-art maximum clique algorithm which is suitable for use on dense graphs (Prosser 2012), and investigate whether this approach is feasible in practice. We modify the algorithm to include a new lazy "global domination" inference step—this technique provides no benefit for typical maximum clique problems, but for maximum k-clique graphs it sometimes gives improvements of several orders of magnitude. We present computational results for the maximum k-clique problem on a range of benchmark and real-world graphs. We finish with a detailed look at random graphs.

Throughout, our graphs are finite, undirected, and contain no loops. If G is a graph with vertex set V and edge set E, we may write $V(G)$ to mean V. The *neighbourhood* of a vertex v in a graph G, written $N_G(v)$, is the set of vertices adjacent to v. The *degree* of a vertex is the cardinality of its neighbourhood. The density of a graph, denoted D, is the proportion of distinct pairs of vertices which have an edge between them. The subgraph *induced by* a set of vertices W is the subgraph with vertex set W, and all edges from the original graph that are between pairs of vertices in W. If A and B are sets of vertices, we write $A \setminus B$ for the set of vertices which are in A but not B, and we write $A + v$ and $A - v$ for $A \cup \{v\}$ and $A \setminus \{v\}$ respectively.

Algorithms

Our approach for finding a maximum k-clique is presented as Algorithm 1. Our first step (line 3) is to replace our input graph G with G^k. We may construct this graph using a bounded breadth-first search: we refer to Chang et al. (2013) for how to implement this quickly in practice.

Colouring The current state-of-the-art for the maximum clique problem on dense graphs, due to Tomita et al. (Tomita and Kameda 2007; Tomita et al. 2010), is to use branch and bound with a greedy graph colouring. A *colouring* of a graph is an assignment of colours to vertices, such that adjacent vertices are given different colours; if we can colour a graph using c colours, then the graph cannot contain a clique of size greater than c (each vertex in a clique must be given a different colour).

Obtaining a minimal colouring is NP-hard, but we may create a greedy colouring in polynomial time. This is done by the `colour` routine: we start the first colour (line 28), and while there are uncoloured vertices remaining (line 29), we try to give each vertex in turn the current colour (lines 30 to 36). When we cannot colour any further vertices, we start a new colour (line 37).

The key step in Tomita et al.'s algorithms is to produce a constructive colouring—the `colour` routine does not just return the number of colours used. Instead, it returns a pair of arrays, *order* and *bounds*. The *order* array contains vertices, in the order in which they were coloured. The ith entry of the *bounds* array contains the colour number used for the ith vertex in *order*. We illustrate this in Figure 2. Crucially, *bounds* is non-decreasing (i.e. $bounds[i+1] \geq bounds[i]$), and we may colour the subgraph induced by the first i vertices of *order* using $bounds[i]$ colours.

Algorithm 1: Solving the maximum k-clique problem.

```
1  maxKClique :: (Graph G, Integer k) → Vertex Set
2  begin
3      G ← G^k
4      permute G into non-increasing degree order
5      global C* ← ∅
6      expand(∅, V(G))
7      return C* (unpermuted)
8  expand :: (Vertex Set C, Vertex Set P)
9  begin
10     (order, bounds) ← colour(P)
11     v_rej ← unset
12     for i ← |P| downto 1 do
13         if |C| + bounds[i] ≤ |C*| then return
14         if v_rej ≠ unset then
15             P ← P \ dominated(v_rej)
16         v ← order[i]
17         if v ∈ P then
18             C' ← C + v
19             if |C'| > |C*| then C* ← C'
20             P' ← P ∩ N_G(v)
21             if P' ≠ ∅ then expand(C', P')
22             P ← P − v
23         v_rej ← v
24 colour :: (Vertex Set P) → (Vertex Array, Int Array)
25 begin
26     (order, bounds) ← ([], [])
27     uncoloured ← P
28     currentColour ← 1
29     while uncoloured ≠ ∅ do
30         colourable ← uncoloured
31         while colourable ≠ ∅ do
32             v ← the first vertex of colourable
33             append v to order
34             append currentColour to bounds
35             uncoloured ← uncoloured − v
36             colourable ← colourable \ N_G(v)
37         currentColour ← currentColour + 1
38     return (order, bounds)
39 dominated :: (Vertex v) → Vertex Set
40 begin
41     return {w ∈ V(G) : N_G(w) − v ⊆ N_G(v) − w}
```

The order in which vertices are selected for colouring can have a large effect upon performance. Various initial vertex orderings have been considered for the maximum clique problem (Prosser 2012; Segundo, Lopez, and Batsyn 2014). Here we will colour vertices in a static non-increasing degree order, which we do by permuting the graph at the top of search (line 4). We will *not* be using a dynamic tie-breaking mechanism: although doing so can sometimes be beneficial for small dense graphs in a maximum clique context (Tomita

et al. 2010), for the larger graphs we will be considering here the cubic cost is prohibitively expensive. For the same reason, and additionally because our colour classes typically contain many vertices, we use a simple greedy colouring and do not use a colour repair step (Tomita et al. 2010) or stronger MaxSAT-based inference (Segundo, Nikolaev, and Batsyn 2015; Li, Jiang, and Xu 2015).

Branching and recursing We may now describe the main recursive part of the algorithm. If v is a vertex, then a clique in G^k either contains only v and vertices adjacent to v, or does not contain v. This allows us to grow cliques by repeatedly picking a vertex, and branching upon whether or not to include it. Our growing clique is stored in the variable C, which is initially empty (line 6). We also track which vertices may still be added to C in the variable P, which initially contains every vertex (line 6). The `expand` procedure picks a vertex v (line 16), then considers adding v to C (lines 18 to 21): we create a new P' from P (line 20) by rejecting vertices which are not adjacent to v (and thus every vertex in P' is adjacent to *every* vertex in C). If vertices remain in P', we recurse (line 21). We then take the opposite branch choice, and consider rejecting from P and C (line 22). We then loop, and pick a new v.

Integrating the colour bound We keep track of the best solution we have found so far, which we call the *incumbent*; this is stored in $C^\star$, which is initially empty (line 5). Whenever we find a new clique, we compare its size to that of $C^\star$, and if it is better, the incumbent is unseated (line 19). Now we may make use of the colour bound. At the start of the recursive procedure (line 10), we use `colour` to produce a constructive greedy colouring of the subgraph induced by P into the array $order$, with the colour numbers placed in $bounds$. When selecting v, we iterate over $bounds$ from right to left (line 12). Now on line 13 we know that the largest possible clique we could find at the current location has size no greater than $|C| + bounds[i]$, so if this cannot unseat the incumbent then we may abandon search and backtrack.

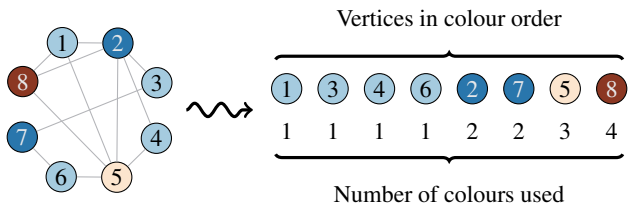

Figure 2: The graph on the left has been coloured greedily, using four colours: vertices 1, 3, 4 then 6 were given the first colour, then vertices 2 then 7 were given the second colour, then vertex 5 was given the third colour, and vertex 8 the fourth colour. The $order$ array, on top, contains the vertices in the order they were coloured; the ith entry of the $bounds$ array, below, contains the number of colours used to colour the first i vertices of $order$.

Lazy global domination Aside from the G^k step, what we have described so far is a standard maximum clique algorithm, and all we have done is opted out of certain more computationally expensive inference steps (more complicated initial vertex orderings, and cubic colourings). If we ignore the lines shown in blue, we obtain the maximum clique algorithm variation that Prosser (2012) calls "MCSa1". Now we will introduce a new lazy global domination rule which performs additional inference during search. This rule is not specific to the maximum k-clique problem, and is also valid for the maximum clique problem.

Let v and w be distinct vertices in a graph G. We say that v *dominates* w if the neighbourhood of w, excluding v, is a (possibly non-strict) subset of the neighbourhood of v, excluding w. From a maximum clique perspective, this means that v is "better than" w. If v and w are adjacent, any clique containing w may always be extended by the inclusion of v; if v and w are non-adjacent, replacing w with v in any clique containing w cannot reduce the amount by which the clique may be grown.

Suppose a graph does contain one or more pairs of dominating vertices. We could make use of this fact during search in at least two ways. Firstly, when accepting a vertex w, we may also unconditionally accept any vertex v which both dominates and is adjacent to w. Secondly, when rejecting a vertex v, we may also unconditionally reject any vertex w which is dominated by v. We could also choose to calculate domination globally (i.e. with respect to G^k, or even the original G), or locally (i.e. with respect to the subgraph of G^k induced by $C \cup P$).

Detecting whether one vertex dominates another may be done in linear time (we discuss this further below), but finding all vertices dominated by a particular vertex is quadratic, and finding all dominations is cubic. This is a heavy price to pay, if there are no dominating vertices. This is why such a rule has not previously been used in the maximum clique context: in the authors' experience, most graphs typically considered for the maximum clique problem do not contain dominating vertices, and those that do are too easy computationally for the step to be worthwhile.

However, some of the graphs we consider in the following section *do* contain dominating vertices, and although the maximum clique problem is trivial on these graphs, the maximum k-clique problem is not for some values of k. Preliminary experiments suggested that the use of a domination rule could be extremely beneficial in certain circumstances, but that in cases where it had little effect, doing such a calculation introduced a substantial penalty to runtimes. Moreover, even in graphs where dominating vertices are present, knowing this fact is sometimes not useful: it is common for an optimal solution to be found straight away, and for the bound to be strong enough to prove optimality immediately, so no branching occurs.

This motivates the design of a lazy global domination rule. We perform our domination checks globally, with respect to G^k (which may contain more dominating vertices than G), and we remember and reuse the results of any domination checks we perform. We also only perform inference on the "reject" case, to avoid introducing any cost when a

solution is found and proven optimal without branching.

The lines marked in blue in Algorithm 1 show how this is done. When a vertex v_{rej} is rejected, we remove from P any vertex that is dominated (with respect to G^k) by v_{rej}. This is line 15; the set of dominated vertices calculated here should be cached. One might expect that this calculation would appear after line 22. However, this introduces a cost if the bound allows the next choice of v to be eliminated. Thus we simply remember that we have rejected v by storing it in v_{rej} (line 23), and lazily postpone the filtering until after the bound has been checked.

Finally, note that we do not perform a new colouring when we reject dominated vertices—doing so typically does not lead to a smaller bound, since most colour classes contain many vertices. Thus when we select a v from $order$, it is now possible that v has already been rejected. We check for this on line line 17.

Bitset encoding San Segundo et al. (2011; 2013) observed that the performance of Tomita's algorithms could be enhanced substantially by using a bitset encoding to obtain a form of SIMD-like parallelism, without altering the steps taken. We have taken such an approach here too, although we do not describe it explicitly—when permuting G on line line 4, the graph should be re-encoded as an array of adjacency bitsets. (It is not helpful to do this before constructing G^k.) Now the intersection on line 20 becomes a simple bitwise "and" operation, and the intersection with complement on line 36 is a bitwise "and not" operation. This is beneficial when testing for dominance, too: each bit in the dominated set on line 15 may be determined by a bitwise "and not", unsetting a bit, and testing whether the result is empty; the set difference is again a bitwise "and not" operation.

Experimental Results

We now give experimental results on a range of standard benchmarks, and on real-world and random graphs. Experiments were run on a machine with dual 3.1GHz Intel E5-2687W v3 processors and 256GBytes RAM; single-threaded runtimes are given, except in Table 5, where 20 threads were used (our machine does not have hyper-threading enabled). Our software was implemented in C++, using C++11 native threads, and was compiled using GCC 5.1.0. The time taken to read in the graph from a file is excluded, but preprocessing time (including the construction of G^k and the bitset encoding) is included. We use the term *nodes* to refer to the number of recursive calls made by the branch-and-bound part of the algorithm.

Real-World Graphs

We begin with a selection of real-world and standard benchmark graphs. We look at k equal to 2, 3 and 4 in every case—this is a standard practice for the k-club problem.

Erdős collaboration graphs In Table 1 we present experimental results from Erdős collaboration graphs from the Pajek dataset by Vladimir Batagelj and Andrej Mrvar[1]. We were able to solve all of these problems in under four minutes (and all but three in under four seconds) when using the domination rule. However, using the unmodified maximum clique algorithm, three of these results did not finish running within one hour. Note that for $k = 4$, a k-clique covers all of "Erdos02".

				Unmodified		With Domination	
Instance	k	D	$\tilde{\omega}_k$	Nodes	Time	Nodes	Time
Erdos971	2	0.09	42	42	0.0	42	0.0
$\|V\|$=472	3	0.31	117	121	0.0	119	0.0
$\|E\|$=1314	4	0.56	235	468	0.0	468	0.0
Erdos972	2	0.01	258	258	0.5	258	0.5
$\|V\|$=5488	3	0.09	517	537	0.8	521	0.8
$\|E\|$=8972	4	0.35	1509	1.1×10^7	>1 h	8197	3.8
Erdos981	2	0.09	43	43	0.0	43	0.0
$\|V\|$=485	3	0.31	123	358	0.0	354	0.0
$\|E\|$=1381	4	0.57	245	246	0.0	246	0.0
Erdos982	2	0.01	274	274	0.6	274	0.6
$\|V\|$=5822	3	0.09	547	555	0.9	547	0.9
$\|E\|$=9505	4	0.35	1594	1.1×10^7	>1 h	618826	207.3
Erdos991	2	0.09	43	44	0.0	44	0.0
$\|V\|$=492	3	0.31	126	375	0.0	374	0.0
$\|E\|$=1417	4	0.57	246	491	0.0	491	0.0
Erdos992	2	0.01	277	277	0.6	277	0.6
$\|V\|$=6100	3	0.09	562	573	0.9	562	0.8
$\|E\|$=9939	4	0.35	1643	1.0×10^7	>1 h	202543	69.4
Erdos02	2	0.02	508	508	0.8	508	0.8
$\|V\|$=6927	3	0.20	1014	1022	1.3	1015	1.3
$\|E\|$=11850	4	1.00	6927	6927	11.9	6927	10.9

Table 1: Experimental results for Erdős collaboration graphs. For each graph, we consider k equal to 2, 3 and 4. In each case we show the density of G^k, the size of a maximum k-clique, and then for both the unmodified algorithm and the algorithm with our lazy global domination step, the number of nodes required, and the runtime in seconds.

In several cases, the algorithm found and proved an optimal solution immediately ($\tilde{\omega}_k$ is equal to the number of search nodes). This illustrates the necessity of laziness: if we simply computed dominating pairs upfront, we would be paying a cubic preprocessing cost for an algorithm which is effectively quadratic in practice.

By comparing these results with the k-club results of Chang et al. (2013), we see that in all but four cases the k-clique and k-club numbers are equal; all of these differences occur when $k = 4$. (Chang et al. did not investigate the "Erdos02" graph, but Wotzlaw confirmed privately that the k-clique and k-club numbers are the same here too.) On the other hand, the k-clique numbers are sometimes much easier to find, both algorithmically and computationally.

Clique graphs In Table 2 we present results from the "clique" graphs from the Second DIMACS implementation challenge[2]. These graphs were designed to test maximum

[1] http://vlado.fmf.uni-lj.si/pub/networks/data/

[2] http://dimacs.rutgers.edu/Challenges/

				Unmodified		With Domination	
Instance	k	D	$\tilde{\omega}_k$	Nodes	Time	Nodes	Time
c-fat200-1	2	0.13	18	41	0.0	35	0.0
$\|V\|$=200	3	0.19	24	74	0.0	48	0.0
$\|E\|$=1534	4	0.24	30	134	0.0	65	0.0
c-fat200-2	2	0.27	35	35	0.0	35	0.0
$\|V\|$=200	3	0.39	46	488	0.0	102	0.0
$\|E\|$=3235	4	0.50	57	1496	0.0	128	0.0
c-fat200-5	2	0.71	87	11513	0.0	257	0.0
$\|V\|$=200	3	1.00	200	200	0.0	200	0.0
$\|E\|$=8473	4	1.00	200	200	0.0	200	0.0
c-fat500-1	2	0.06	21	52	0.0	43	0.0
$\|V\|$=500	3	0.09	28	28	0.0	28	0.0
$\|E\|$=4459	4	0.11	35	35	0.0	35	0.0
c-fat500-2	2	0.12	39	134	0.0	79	0.0
$\|V\|$=500	3	0.17	52	52	0.0	52	0.0
$\|E\|$=9139	4	0.22	65	65	0.0	65	0.0
c-fat500-5	2	0.31	96	10133	0.1	196	0.0
$\|V\|$=500	3	0.44	128	128	0.1	128	0.1
$\|E\|$=23191	4	0.56	159	1.5×10^9	>1 h	326	0.1
c-fat500-10	2	0.62	189	1.1×10^9	>1 h	560	0.1
$\|V\|$=500	3	0.87	252	252	0.1	252	0.1
$\|E\|$=46627	4	1.00	500	500	0.1	500	0.1
p-hat300-1	2	1.00	299	299	0.0	299	0.0
$\|V\|$=300	3	1.00	300	300	0.1	300	0.0
$\|E\|$=10933	4	1.00	300	300	0.0	300	0.0

Table 2: Experimental results for DIMACS clique graphs with diameter greater than two.

clique implementations. Nearly all of these graphs have diameter 2, so a 2-clique covers the entire graph—we have ignored these. The only exceptions are the "c-fat" family (all of which are trivial for a maximum clique solver), and one of the "p_hat" graphs.

With the domination rule, we solve all of these problems within a tenth of a second. Without, two of the results take over an hour, and the rest remain trivial. Note that in several cases, for some values of k a k-clique covers the entire graph. Again using Chang et al.'s results, we see that for the first six graphs in this table the k-clique and k-club numbers are the same for each value of k (Chang et al. did not investigate "c-fat500-10" or "p-hat300-1").

Partitioning graphs Table 3 presents results from the smallest 20 partitioning graphs from the 10th DIMACS Implementation Challenge[3]. Many of these graphs are considerably larger than those typically considered for the maximum clique problem, and we might expect our $O(|V|^2)$ memory requirements to cause problems. Nonetheless, with the domination rule there is only one instance which we were unable to solve within an hour (and without the domination rule, there are two).

On the other hand, we sometimes see a significant cost where the domination rule does not help, and where the proof of optimality is not immediate: in "3elt" and "4elt", our runtimes can nearly double, and for "cs4" and "cti" the

[3]http://staffweb.cms.gre.ac.uk/~wc06/partition/

				Unmodified		With Domination	
Instance	k	D	$\tilde{\omega}_k$	Nodes	Time	Nodes	Time
3elt	2	0.00	10	340	0.3	340	0.6
$\|V\|$=4720	3	0.01	16	1582	0.4	1582	1.0
$\|E\|$=13722	4	0.01	27	911	0.4	911	0.7
4elt	2	0.00	11	486	3.8	486	5.5
$\|V\|$=15606	3	0.00	20	717	3.9	717	6.1
$\|E\|$=45878	4	0.00	36	345	3.9	345	4.4
add20	2	0.04	124	124	0.1	124	0.2
$\|V\|$=2395	3	0.25	671	671	0.2	671	0.2
$\|E\|$=7462	4	0.67	1454	1454	0.5	1454	0.5
add32	2	0.00	32	32	0.4	32	0.4
$\|V\|$=4960	3	0.01	99	286	0.4	194	0.4
$\|E\|$=9462	4	0.03	268	268	0.4	268	0.5
bcsstk29	2	0.01	72	9752	3.7	963	8.7
$\|V\|$=13992	3	0.02	132	2.7×10^7	1576.2	7781	11.4
$\|E\|$=302748	4	0.04	210	3.2×10^7	>1 h	21689	16.9
bcsstk30	2	0.01	219	224	14.5	219	14.5
$\|V\|$=28924	3	0.03	496	509	17.8	496	17.6
$\|E\|$=1007284	4	0.05	843	854	23.7	845	23.3
bcsstk31	2	0.00	189	189	20.2	189	20.0
$\|V\|$=35588	3	0.01	278	605	21.8	369	22.8
$\|E\|$=572914	4	0.02	428	119640	97.9	6588	29.8
bcsstk33	2	0.03	141	141	1.3	141	1.3
$\|V\|$=8738	3	0.08	228	26033	6.3	1744	3.8
$\|E\|$=291583	4	0.15	435	2.0×10^6	380.4	52779	17.6
crack	2	0.00	10	2894	1.5	2894	7.8
$\|V\|$=10240	3	0.00	17	4996	1.7	4987	10.4
$\|E\|$=30380	4	0.01	31	2173	1.7	2173	7.0
cs4	2	0.00	6	5780	8.2	5780	73.4
$\|V\|$=22499	3	0.00	12	7812	8.4	7812	78.5
$\|E\|$=43858	4	0.00	18	29032	9.5	29032	136.3
cti	2	0.00	7	8918	4.5	8918	58.0
$\|V\|$=16840	3	0.00	15	6406	4.8	6406	45.5
$\|E\|$=48232	4	0.01	26	62316	7.0	62316	113.6
data	2	0.01	18	638	0.1	617	0.2
$\|V\|$=2851	3	0.02	32	4982	0.1	4913	0.2
$\|E\|$=15093	4	0.04	52	40095	0.4	36089	0.4
fe-4elt2	2	0.00	13	61	1.7	61	2.1
$\|V\|$=11143	3	0.00	20	389	1.8	389	3.0
$\|E\|$=32818	4	0.01	32	448	1.8	446	3.0
fe-pwt	2	0.00	16	95	21.8	95	22.6
$\|V\|$=36519	3	0.00	29	167	21.0	167	24.4
$\|E\|$=144794	4	0.00	52	224	21.1	224	23.9
fe-sphere	2	0.00	7	14173	4.5	14173	53.0
$\|V\|$=16386	3	0.00	12	34328	4.9	34328	111.1
$\|E\|$=49152	4	0.00	19	73632	6.3	73632	148.7
memplus	2	0.02	574	574	5.8	574	5.7
$\|V\|$=17758	3	0.26	8057	8061	49.4	8058	48.2
$\|E\|$=54196	4	0.74	8963	8963	95.5	8963	92.4
uk	2	0.00	5	433	0.3	433	0.7
$\|V\|$=4824	3	0.00	8	1891	0.4	1891	0.9
$\|E\|$=6837	4	0.01	14	2168	0.4	2168	1.0
vibrobox	2	0.02	121	302	2.9	302	3.5
$\|V\|$=12328	3	0.08	408	1984	8.9	1984	10.0
$\|E\|$=165250	4	0.26	$\geq$1094	7.9×10^6	>1 h	7.4×10^6	>1 h
whitaker3	2	0.00	9	1222	1.3	1222	4.8
$\|V\|$=9800	3	0.00	15	3724	1.4	3724	7.4
$\|E\|$=28989	4	0.01	23	6530	1.5	6530	9.5
wing-nodal	2	0.01	29	648	2.1	648	3.5
$\|V\|$=10937	3	0.02	54	13091	4.2	13039	15.2
$\|E\|$=75488	4	0.04	114	6.0×10^7	1981.3	6.0×10^7	1908.9

Table 3: Results for smaller DIMACS partitioning graphs.

slowdown is sometimes over a factor of ten. Thus laziness can be costly when the rule is used, but useless.

Five of these graphs were considered for the k-club problem by Wotzlaw (2014). In every case, the k-clique and k-club numbers are the same. However, the k-clique number was again consistently much easier to find.

Clustering graphs Table 4 presents results from the smallest 20 clustering graphs from the 10th DIMACS Implementation Challenge[4]. Again, from a maximum clique perspective these would be considered unusually large graphs. However, only five were unsolvable within an hour (plus a further two when the domination rule was not used), and over half of the problems took under two seconds.

Seven of these graphs were considered for the k-club problem by Wotzlaw (2014). In these cases, the 2-clique and 2-club numbers are the same, except for "football" where the 2-club number is 16 but the 2-clique number is 17; for $k = 3$ and $k = 4$ there are some differences. There is a large difference in computational difficulty between the k-clique and k-club problems: for "polblogs" with $k = 3$ and $k = 4$, Wotzlaw was unable to prove optimality within an hour, but we required less than two seconds to do so. In both of these cases the k-clique and k-club numbers are the same.

Parallel Search

Parallel search is an active research area for maximum clique algorithms (McCreesh and Prosser 2013; Depolli et al. 2013; McCreesh and Prosser 2015; Segundo, Lopez, and Pardalos 2016). The idea is to work with a shared incumbent, and to speculatively explore portions of the search tree in parallel using multiple threads, in the hopes that most of the speculative work will contribute to the optimality proof.

We selected the four graphs which we were unable to solve (for some values of k), along with two of the challenging graphs which required substantial amounts of search (at least half a million nodes) to solve, and repeated the experiments using parallel search with 20 threads and the domination rule (with the laziness made thread-safe) enabled, using the resplitting strategy described by McCreesh and Prosser (2015). The results are shown in Table 5. Parallel search allowed us to close two more instances, and gave improved bounds on the four remaining instances within 12 hours. However, to get sufficient work balance, we had to allow for work splitting to depth 10 rather than depth 3. A close inspection of the search patterns showed that simpler static decomposition approaches (Depolli et al. 2013; Segundo, Lopez, and Pardalos 2016) would give little to no speedup on many harder k-clique instances, unless it were somehow possible to generate $\mathcal{O}(|V|^{10})$ subproblems.

For the easier instances where the sequential runtime is known, the parallel search did more work—this is to be expected, since the work distribution approach we used recomputed parts of the search space (to allow for more efficient mutable data structures to be used during search), and speculative parallelism is unlikely to contribute to the solution when the sequential search tree is small. However, despite

[4]http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml

				Unmodified		With Domination	
Instance	k	D	$\tilde{\omega}_k$	Nodes	Time	Nodes	Time
adjnoun	2	0.50	50	50	0.0	50	0.0
$\|V\|$=112	3	0.91	83	164	0.0	164	0.0
$\|E\|$=425	4	0.99	107	107	0.0	107	0.0
as-22july06	2	0.04	2391	2391	11.7	2391	11.3
$\|V\|$=22963	3	0.36	8455	374427	>1 h	94497	898.2
$\|E\|$=48436	4	0.79	14911	14911	252.0	14911	250.4
astro-ph	2	0.01	361	365	5.0	362	5.0
$\|V\|$=16706	3	0.10	1553	1567	15.3	1560	12.5
$\|E\|$=121251	4	0.35	$\geq$4040	1.0×10^6	>1 h	1.1×10^6	>1 h
celegans-meta	2	0.44	238	238	0.0	238	0.0
$\|V\|$=453	3	0.89	371	371	0.0	371	0.0
$\|E\|$=2025	4	0.98	432	432	0.0	432	0.0
celegansneural	2	0.55	135	135	0.0	135	0.0
$\|V\|$=297	3	0.95	245	245	0.0	245	0.0
$\|E\|$=2148	4	1.00	295	295	0.0	295	0.0
cond-mat	2	0.00	108	108	4.5	108	4.5
$\|V\|$=16726	3	0.01	250	1403	5.1	844	5.6
$\|E\|$=47594	4	0.05	720	7.3×10^6	>1 h	674453	280.6
cond-mat-2003	2	0.00	203	204	15.7	204	16.0
$\|V\|$=31163	3	0.02	$\geq$629	6.3×10^6	>1 h	6.1×10^6	>1 h
$\|E\|$=120029	4	0.12	$\geq$2605	1.9×10^6	>1 h	1.8×10^6	>1 h
cond-mat-2005	2	0.00	279	279	26.8	279	26.4
$\|V\|$=40421	3	0.03	$\geq$1060	2.0×10^6	>1 h	1.9×10^6	>1 h
$\|E\|$=175691	4	0.16	$\geq$4185	562816	>1 h	585596	>1 h
dolphins	2	0.32	14	14	0.0	14	0.0
$\|V\|$=62	3	0.59	30	30	0.0	30	0.0
$\|E\|$=159	4	0.77	40	40	0.0	40	0.0
email	2	0.09	72	72	0.0	72	0.1
$\|V\|$=1133	3	0.45	233	19031	0.3	19031	0.4
$\|E\|$=5451	4	0.86	654	6854	0.3	6676	0.4
football	2	0.45	17	147	0.0	145	0.0
$\|V\|$=115	3	0.95	69	70	0.0	70	0.0
$\|E\|$=613	4	1.00	115	115	0.0	115	0.0
hep-th	2	0.00	51	51	1.0	51	1.0
$\|V\|$=8361	3	0.01	125	239	1.2	176	1.3
$\|E\|$=15751	4	0.04	347	158164	21.6	23714	4.7
jazz	2	0.69	103	107	0.0	107	0.0
$\|V\|$=198	3	0.95	174	174	0.0	174	0.0
$\|E\|$=2742	4	0.99	192	192	0.0	192	0.0
karate	2	0.61	18	18	0.0	18	0.0
$\|V\|$=34	3	0.86	25	25	0.0	25	0.0
$\|E\|$=78	4	0.99	33	33	0.0	33	0.0
lesmis	2	0.43	37	37	0.0	37	0.0
$\|V\|$=77	3	0.85	58	58	0.0	58	0.0
$\|E\|$=254	4	0.99	75	75	0.0	75	0.0
netscience	2	0.01	35	35	0.0	35	0.0
$\|V\|$=1589	3	0.01	54	54	0.0	54	0.0
$\|E\|$=2742	4	0.02	85	85	0.0	85	0.0
PGPgiantcompo	2	0.00	206	206	1.7	206	1.7
$\|V\|$=10680	3	0.02	423	843	2.6	841	2.7
$\|E\|$=24316	4	0.07	1161	1161	5.9	1161	6.0
polblogs	2	0.27	352	352	0.1	352	0.1
$\|V\|$=1490	3	0.58	776	2210	1.2	2177	1.2
$\|E\|$=16715	4	0.66	1127	1537	2.0	1166	1.9
polbooks	2	0.37	28	28	0.0	28	0.0
$\|V\|$=105	3	0.64	54	54	0.0	54	0.0
$\|E\|$=441	4	0.86	68	68	0.0	68	0.0
power	2	0.00	20	20	0.4	20	0.4
$\|V\|$=4941	3	0.00	30	30	0.4	30	0.4
$\|E\|$=6594	4	0.01	61	61	0.4	61	0.4

Table 4: Results for smaller DIMACS clustering graphs.

				Sequential		Parallel	
Instance	k	D	$\tilde{\omega}_k$	Nodes	Time	Nodes	Time
astro-ph	2	0.01	361	362	5.0	17227	4.9
$\|V\|$=16706	3	0.10	1553	1560	12.5	32614	9.5
$\|E\|$=121251	4	0.35	≥4125	1.1×10^6	> 1 h	3.1×10^8	> 12 h
cond-mat	2	0.00	108	108	4.5	16833	4.5
$\|V\|$=16726	3	0.01	250	844	5.6	17933	5.4
$\|E\|$=47594	4	0.05	720	674453	280.6	22604	8.7
cond-mat-2003	2	0.00	203	204	16.0	31424	14.8
$\|V\|$=31163	3	0.02	634	6.1×10^6	> 1 h	3.4×10^8	8131.3
$\|E\|$=120029	4	0.12	≥2606	1.8×10^6	> 1 h	5.1×10^8	> 12 h
cond-mat-2005	2	0.00	279	279	26.4	40701	24.2
$\|V\|$=40421	3	0.03	1060	1.9×10^6	> 1 h	1.0×10^9	69836.7
$\|E\|$=175691	4	0.16	≥4271	585596	> 1 h	1.2×10^8	> 12 h
vibrobox	2	0.02	121	302	3.5	12699	3.4
$\|V\|$=12328	3	0.08	408	1984	10.0	17246	6.5
$\|E\|$=165250	4	0.26	≥1107	7.4×10^6	> 1 h	1.8×10^9	> 12 h
wing-nodal	2	0.01	29	648	3.5	11580	2.9
$\|V\|$=10937	3	0.02	54	13039	15.2	23709	17.9
$\|E\|$=75488	4	0.04	114	6.0×10^7	1908.9	113619	18.4

Table 5: Experimental results for multi-threaded search on harder instances, using 20 threads. For each graph, we consider k equal to 2, 3 and 4. In each case we show the density of G^k, the size of a maximum k-clique, and then for both the sequential algorithm and the parallel algorithm (with lazy global domination in both cases), the number of nodes required, and the runtime in seconds.

the extra work, and despite having to introduce overhead into early stages of the algorithm to allow for work stealing, and despite having more processing power but not more memory bandwidth, in no cases were the parallel runtimes substantially longer than the sequential runtimes (although in many cases they were not substantially better either). We also did not parallelise the construction of G^k or the preprocessing stage of the algorithm, which in many cases dominated the runtime.

As well as improving the results on the instances we could not solve sequentially, parallel search sometimes gave large improvements for the easier instances. For "wing-nodal" with $k = 4$, the parallel run did much less work than the sequential run: a speedup of over 100 was obtained from 20 cores. A similar effect occurred with "cond-mat" and $k = 4$. This is because the work splitting mechanism we used explicitly diversifies at the top of search first, where branching heuristics are least likely to be correct, which leads to an initial incumbent being found faster—this is in line with recent observations that tailored work stealing should be favoured over randomised work stealing for combinatorial search problems (Chu, Schulte, and Stuckey 2009; McCreesh and Prosser 2015).

Random Graphs

An Erdős-Rényi random graph $G(n, p)$ has n vertices, and an edge between each distinct pair of vertices with probability p, chosen independently. We now investigate the size of a maximum k-clique in such graphs, and the complexity of finding it. In each case, we use an average over 100 samples for every point. We do not use the domination rule for these experiments: the probability of random graphs having dominating vertices is very low.

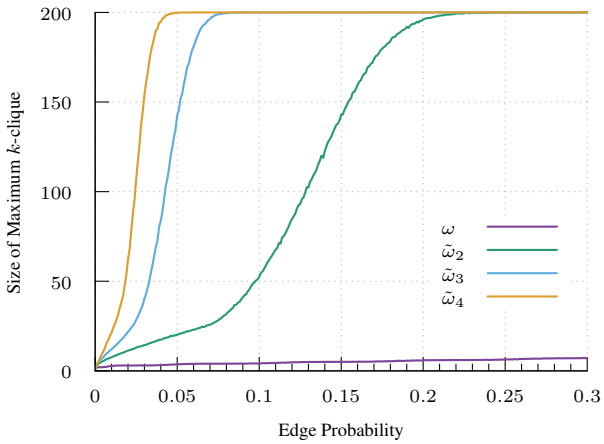

Figure 3: Values of $\tilde{\omega}_k$ for random graphs $G(200, p)$, with varying edge probabilities. We see that even for very low edge probabilities, a maximum k-clique quickly covers the entire graph when $k > 1$.

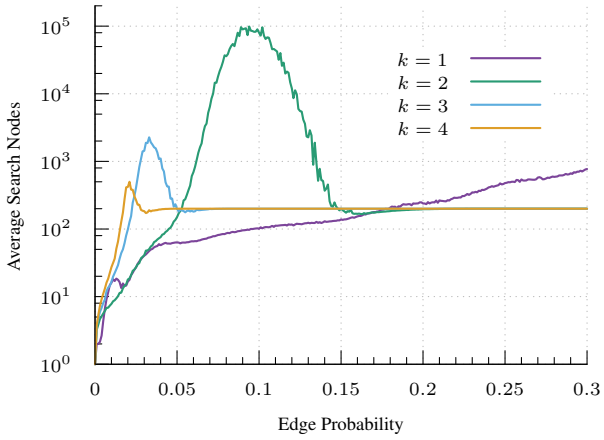

Figure 4: Search space size for random graphs $G(200, p)^k$, with varying edge probabilities. We see that 4-clique is easier than 3-clique in practice, which in turn is easier than 2-clique. (The complexity peak for maximum clique occurs at around edge probability 0.9, and requires approximately 15 million search nodes.)

In Figure 3 we illustrate the average value of $\tilde{\omega}_k$ in $G(200, p)$ for different values of k, and a range of values of p for the x-axis. We see that even for very low edge probabilities, a maximum k-clique quickly covers the entire graph. (This is in contrast to the maximum clique problem, where a maximum clique does not even cover a quarter of the graph for edge probabilities below 0.75.) In Figure 4 we show the average size of the search space (number of nodes, or recursive calls made) for the same problem. We see that

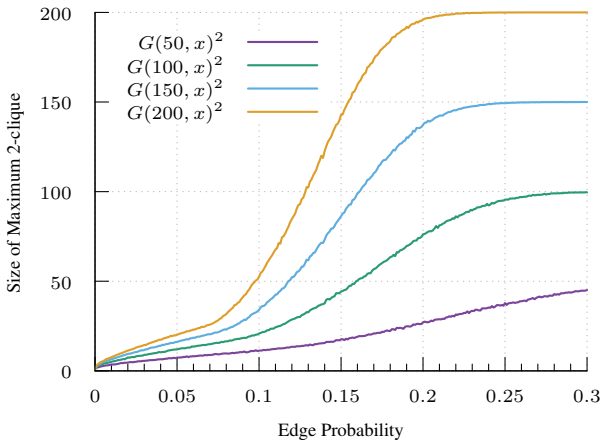

Figure 5: The size of a maximum 2-clique in random graphs $G(n,p)$ with varying edge probabilities, and different values of n. For $G(50,p)$, a 2-clique has size average 50 from $p = 0.42$ onwards.

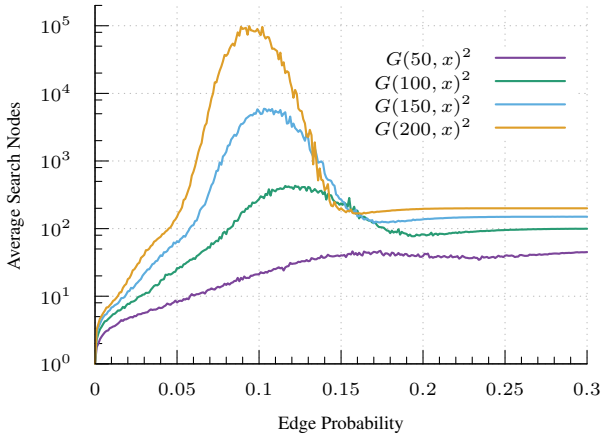

Figure 6: The search space size for the maximum 2-clique problem in random graphs $G(n,p)$ with varying edge probabilities, and for different values of n. As n increases, the complexity peak grows and moves slowly to the left.

there is a complexity peak for each k, although the peak is much smaller for $k = 4$ than it is for $k = 3$, which is in turn much smaller than it is for $k = 2$. The peak also occurs for lower edge probabilities as k increases. For contrast, for the maximum clique problem, the peak occurs at around edge probability 0.9, and is two orders of magnitude larger.

In Figures 5 and 6 we show the effect of changing n and fixing $k = 2$. As n increases from 50 to 200, the complexity peak becomes much more pronounced, and shifts slightly towards the left (lower edge probabilities).

Conclusion

We have shown that using a maximum clique algorithm to solve the maximum k-clique algorithm for a graph G by considering G^k in place of G is feasible in practice. This is despite G^k potentially being dense even if G is sparse—this ruled out the use of maximum clique algorithms which are designed for sparse graphs (Pattabiraman et al. 2013; Segundo, Lopez, and Pardalos 2016), and we were working with graphs with many more vertices than is typical for dense maximum clique algorithms.

We introduced a new lazy global domination rule. This was sometimes extremely beneficial, leading to exponential reductions in the search space—without this rule, we would have been unable to solve nine of the problem instances we considered, and many others would have taken much longer. However, even with laziness there is still sometimes a cost to pay when this rule does nothing. This rule is thus harmful (although only by a polynomial factor) for the graphs typically considered for the maximum clique problem, and we see the benefit of tailoring algorithms to the problem being solved. We suggest that a similar rule may also be useful for the maximum k-club problem.

We were able to use parallelism to close two further instances, although this required a much finer level of task granularity than usual. We suspect further progress could be made by tailoring the initial vertex ordering based upon what we know about the graphs, or by increasing the number of recursive calls per second by making the colouring stage cheaper, for example by reusing colour classes (Nikolaev, Batsyn, and Segundo 2015).

Quite often, we saw k-clique numbers and k-club numbers being the same. However, solving the maximum k-clique problem is much easier, both in terms of the algorithm and computationally. Thus it is worth checking whether the simpler model would be sufficient for practical applications before trying to solve the k-club problem.

In random graphs, we saw that $G(n,p)^k$ is easier than $G(n,p')$ with some higher probability p'. We also saw that as k increases, the problem gets easier—this was not typically the case for some of the real world graphs.

Our results suggest that k is a very coarse grained parameter. We saw that often a 2-clique or 3-clique would cover the entire graph. In these circumstances the increased restrictions for k-club are of no benefit. It is not obvious if somehow allowing a "fractional" value of k could give more fine-grained control. Thus it may be worth considering other clique relaxations not based upon distance (although other models also have problems: a density-based relaxation known as quasi-clique, for example, can allow vertices with only a single edge to be added to a "clique" (Abello, Resende, and Sudarsky 2002)).

References

Abello, J.; Resende, M. G.; and Sudarsky, S. 2002. Massive quasi-clique detection. In Rajsbaum, S., ed., *LATIN 2002: Theoretical Informatics*, volume 2286 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 598–612.

Balasundaram, B.; Butenko, S.; and Trukhanov, S. 2005. Novel approaches for analyzing biological networks. *Journal of Combinatorial Optimization* 10(1):23–39.

Bomze, I. M.; Budinich, M.; Pardalos, P. M.; and Pelillo, M.

1999. The maximum clique problem. *Handbook of Combinatorial Optimization (Supplement Volume A)* 4:1–74.

Bourjolly, J.-M.; Laporte, G.; and Pesant, G. 2000. Heuristics for finding k-clubs in an undirected graph. *Computers & Operations Research* 27(6):559 – 569.

Bourjolly, J.-M.; Laporte, G.; and Pesant, G. 2002. An exact algorithm for the maximum k-club problem in an undirected graph. *European Journal of Operational Research* 138(1):21 – 28.

Butenko, S., and Wilhelm, W. E. 2006. Clique-detection models in computational biochemistry and genomics. *European Journal of Operational Research* 173(1):1–17.

Carvalho, F. D., and Almeida, M. T. 2016. The triangle k-club problem. *Journal of Combinatorial Optimization* 1–33.

Chang, M.-S.; Hung, L.-J.; Lin, C.-R.; and Su, P.-C. 2013. Finding large k-clubs in undirected graphs. *Computing* 95(9):739–758.

Chu, G.; Schulte, C.; and Stuckey, P. J. 2009. Confidence-based work stealing in parallel constraint programming. In Gent, I. P., ed., *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, 226–241. Springer.

Depolli, M.; Konc, J.; Rozman, K.; Trobec, R.; and Janezic, D. 2013. Exact parallel maximum clique algorithm for general and protein graphs. *Journal of Chemical Information and Modeling* 53(9):2217–2228.

Dhaenens, C.; Jourdan, L.; and Marmion, M., eds. 2015. *Learning and Intelligent Optimization - 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*, volume 8994 of *Lecture Notes in Computer Science*. Springer.

Garey, M. R., and Johnson, D. S. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Hartung, S.; Komusiewicz, C.; and Nichterlein, A. 2012. Parameterized algorithmics and computational experiments for finding 2-clubs. In Thilikos, D., and Woeginger, G., eds., *Parameterized and Exact Computation*, volume 7535 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 231–241.

Li, C.; Jiang, H.; and Xu, R. 2015. Incremental maxsat reasoning to reduce branches in a branch-and-bound algorithm for maxclique. In Dhaenens et al. (2015), 268–274.

Luce, R. 1950. Connectivity and generalized cliques in sociometric group structure. *Psychometrika* 15(2):169–190.

Mahdavi Pajouh, F., and Balasundaram, B. 2012. On inclusionwise maximal and maximum cardinality k-clubs in graphs. *Discrete Optimization* 9(2):84 – 97.

McCreesh, C., and Prosser, P. 2013. Multi-threading a state-of-the-art maximum clique algorithm. *Algorithms* 6(4):618–635.

McCreesh, C., and Prosser, P. 2015. The shape of the search tree for the maximum clique problem and the implications for parallel branch and bound. *TOPC* 2(1):8.

Mokken, R. 1979. Cliques, clubs and clans. *Quality and Quantity* 13(2):161–173.

Nikolaev, A.; Batsyn, M.; and Segundo, P. S. 2015. Reusing the same coloring in the child nodes of the search tree for the maximum clique problem. In Dhaenens et al. (2015), 275–280.

Pattabiraman, B.; Patwary, M. M. A.; Gebremedhin, A. H.; Liao, W.; and Choudhary, A. N. 2013. Fast algorithms for the maximum clique problem on massive sparse graphs. In Bonato, A.; Mitzenmacher, M.; and Pralat, P., eds., *Algorithms and Models for the Web Graph - 10th International Workshop, WAW 2013, Cambridge, MA, USA, December 14-15, 2013, Proceedings*, volume 8305 of *Lecture Notes in Computer Science*, 156–169. Springer.

Picker, M. 2015. Algorithms and experiments for finding robust 2-clubs. Master's thesis, Technische Universität Berlin.

Prosser, P. 2012. Exact algorithms for maximum clique: a computational study. *Algorithms* 5(4):545–587.

San Segundo, P.; Matia, F.; Rodriguez-Losada, D.; and Hernando, M. 2013. An improved bit parallel exact maximum clique algorithm. *Optimization Letters* 7(3):467–479.

San Segundo, P.; Rodríguez-Losada, D.; and Jiménez, A. 2011. An exact bit-parallel algorithm for the maximum clique problem. *Comput. Oper. Res.* 38(2):571–581.

Segundo, P. S.; Lopez, A.; and Batsyn, M. 2014. Initial sorting of vertices in the maximum clique problem reviewed. In Pardalos, P. M.; Resende, M. G. C.; Vogiatzis, C.; and Walteros, J. L., eds., *Learning and Intelligent Optimization - 8th International Conference, Lion 8, Gainesville, FL, USA, February 16-21, 2014. Revised Selected Papers*, volume 8426 of *Lecture Notes in Computer Science*, 111–120. Springer.

Segundo, P. S.; Lopez, A.; and Pardalos, P. M. 2016. A new exact maximum clique algorithm for large and massive sparse graphs. *Computers & OR* 66:81–94.

Segundo, P. S.; Nikolaev, A.; and Batsyn, M. 2015. Infra-chromatic bound for exact maximum clique search. *Computers & OR* 64:293–303.

Shahinpour, S., and Butenko, S. 2013. Distance-based clique relaxations in networks: s-clique and s-club. In Goldengorin, B. I.; Kalyagin, V. A.; and Pardalos, P. M., eds., *Models, Algorithms, and Technologies for Network Analysis*, volume 59 of *Springer Proceedings in Mathematics & Statistics*. Springer New York. 149–174.

Tomita, E., and Kameda, T. 2007. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global Optimization* 37(1):95–111.

Tomita, E.; Sutani, Y.; Higashi, T.; Takahashi, S.; and Wakatsuki, M. 2010. A simple and faster branch-and-bound algorithm for finding a maximum clique. In Rahman, M., and Fujita, S., eds., *WALCOM: Algorithms and Computation*, volume 5942 of *Lecture Notes in Computer Science*. Berlin, Heidelberg: Springer Berlin Heidelberg. 191–203.

Wotzlaw, A. 2014. On Solving the Maximum k-club Problem. *ArXiv e-prints*.