Finding hidden cliques in linear time

Uriel Feige, Dorit Ron

Finding hidden cliques in linear time

Uriel Feige[1†] and Dorit Ron[2]

[1] *Weizmann Institute of Science, Rehovot, Israel. Email: uriel.feige@weizmann.ac.il*
[2] *Weizmann Institute of Science, Rehovot, Israel. Email: dorit.ron@weizmann.ac.il*

In the hidden clique problem, one needs to find the maximum clique in an n-vertex graph that has a clique of size k but is otherwise random. An algorithm of Alon, Krivelevich and Sudakov that is based on spectral techniques is known to solve this problem (with high probability over the random choice of input graph) when $k \geq c\sqrt{n}$ for a sufficiently large constant c. In this manuscript we present a new algorithm for finding hidden cliques. It too provably works when $k > c\sqrt{n}$ for a sufficiently large constant c. However, our algorithm has the advantage of being much simpler (no use of spectral techniques), running faster (linear time), and experiments show that the leading constant c is smaller than in the spectral approach. We also present linear time algorithms that experimentally find even smaller hidden cliques, though it remains open whether any of these algorithms finds hidden cliques of size $o(\sqrt{n})$.

Keywords: random graphs, hidden clique

1 Introduction

The input graph G in the hidden clique model (a.k.a. planted clique) is chosen at random as follows. Given parameters n, k, first a random graph G' is chosen from the distribution $G_{n,1/2}$ (the graph has n vertices and every edge is included independently with probability $1/2$). Then a random set K of k vertices is selected, and for every two vertices $i, j \in K$, the edge (i, j) is added to the graph (if not already there).

It is well known that with high probability, the size of the maximum clique in G' is roughly $2 \log n$, and that if k is larger than this size, then most likely K is the unique maximum clique in G. Our goal is to develop algorithms that find K (with high probability over the choice of G) when k is sufficiently large. The problem becomes more difficult the smaller k is as a function of n. (This can be formally proved by employing operations that change the values of n and k, such as selecting at random a subgraph of G and searching for a maximum clique in the subgraph. Details are omitted.)

Kucera (1995) observed that when $k \geq c\sqrt{n \log n}$ for some sufficiently large constant c, then it is likely that the vertices of K are those with highest degree in G. In this case K can be found efficiently by first sorting all vertices of G in order of increasing degree, and then returning the k top vertices. If however $k \leq c\sqrt{n \log n}$ for a sufficiently small constant $c > 0$, Kucera's simple algorithm no longer works. In particular, in this case the highest degree vertex in G is unlikely to be from K.

[†]The author holds the Lawrence G. Horowitz Professorial Chair at the Weizmann Institute. Work supported in part by The Israel Science Foundation (grant No. 873/08).

Alon, Krivelevich and Sudakov in Alon et al. (1998) presented a spectral algorithm for finding hidden cliques of size $c\sqrt{n}$ for some sufficiently large constant c. Their algorithm can be explained as follows. Consider the order n matrix A' which is the ± 1 version of the adjacency matrix of $G'(V, E')$ – entries $A'_{ii} = 0$ and for $i \neq j$, entry $A'_{ij} = 1$ if $(i, j) \in E'$ and $A'_{ij} = -1$ if $(i, j) \notin E'$. It is well known that for G' selected from $G_{n,1/2}$ it is very unlikely that this matrix has an eigenvalue larger than $2\sqrt{n}$. On the other hand, for the corresponding ± 1 adjacency matrix A of G, the existence of the clique K forces an eigenvalue of value at least $k - 1$. Hence if $k \gg 2\sqrt{n}$, the value of the largest eigenvalue of A serves as a distinguisher between $G_{n,1/2}$ graphs and graphs with planted cliques. The corresponding eigenvector can serve as a starting point for finding the actual planted clique K, which indeed can be found (with high probability over the choice of input graph) using some extra work.

Alon et al. (1998) also observed that the leading constant in the value of $k = c\sqrt{n}$ can be improved as follows. Pick at random a constant number t of vertices from G. With probability roughly $(k/n)^t$ they all belong to K. Hence in an expected number of $(n/k)^t \simeq n^{t/2}$ independent attempts, we shall indeed pick t vertices from K. Consider now only the graph G_t induced on their common neighbors. With high probability it has only $n_t \simeq n/2^t$ vertices, it is still random, and it has a clique of size $c\sqrt{n}-t \simeq c2^{t/2}\sqrt{n_t}$. Hence with respect to G_t, it suffices to solve a hidden clique problem with more favorable parameters. As a consequence, it follows that for every $\epsilon > 0$ one can find planted cliques of size $\epsilon\sqrt{n}$ in time $n^{O(\log 1/\epsilon)}$.

So far, all attempts to develop polynomial time algorithms that find hidden cliques of size $o(\sqrt{n})$ failed. See for example Jerrum (1992) regarding the metropolis process, or Feige and Krauthgamer (2003) regarding Lovasz-Schrijver hierarchies of semidefinite programs. For approaches based on maximizing a cubic form, see Frieze and Kannan (2008).

Observe that one can find planted cliques in time $n^{O(\log n)}$. For example, one can systematically consider all sets of vertices of size $3 \log n$ until one finds a clique of size $3 \log n$. With high probability over the choice of G it must be the case that all these vertices belong to K, and that all their common neighbors are the remaining vertices of K.

Being a problem on random inputs that can be solved in time $n^{O(\log n)}$ but for which no polynomial time algorithm is known, there are interesting connections between the hidden clique problem and other problems of a similar computational complexity status. See Hazan and Krauthgamer (2009) for example.

1.1 Our results

A convenient aspect of the hidden clique problem is that the input distribution is well defined. Hence the performance of algorithms for this problem can be evaluated empirically, and this may help guide the design of improved algorithms. Of course, given an algorithm that seems to perform well empirically, one would still need theoretical analysis to show that its performance is indeed as suggest by the experimentation, since the results of experiments might be an artifact of the use of imperfect random generators in the generation of the input graph, or it might be the case that the true asymptotic behavior of the algorithm did not yet manifest itself on the sizes of input graphs that were tried.

We followed this experimental assisted approach, and tested several algorithms. Some of them appeared to perform better experimentally then the algorithm of Alon et al. (1998) that has the previous best theoretical performance guarantees. Of the algorithms that we tested, the simplest one is a linear time algorithm that we call LDR (for Low Degree Removal). For this algorithm we prove the following performance guarantees.

Theorem 1 *For a sufficiently large constant $c > 0$, the LDR algorithm solves the hidden clique problem when $k \geq c\sqrt{n}$ (with probability at least 2/3, where probability is taken over the choice of the random input graph G).*

Experimental results suggest that a constant such as $c = 1$ in Theorem 1 suffices. However, to keep the proof of Theorem 1 simple, we shall take c to be a large constant. Observe that once c is sufficiently large, one may hope that the LDR algorithm solves the hidden clique problem with probability that approaches 1 as n grows. Again, to keep the proof simple, we shall not attempt to prove such a stronger claim.

Among the linear time algorithms that we tried, the one with best performance (smallest value of leading constant c for which the hidden clique was found) is not LDR, but a more complicated algorithm that we call TPMR (for Truncated Power Method Removal). Experimental result are inconclusive as to whether LDR or TMPR can actually find hidden cliques of size $o(\sqrt{n})$. We leave this as an open question.

The manuscript is organized as follow. In Section 2 we present a high level discussion of approaches to finding hidden cliques. In Section 3 we present and analyze the LDR algorithm. In Section 4 we present two other algorithms that we tried, LDRM and TPMR, and discuss them briefly. The appendix contains some of our experimental results.

2 What does it take to find a planted clique?

Assume that one wishes to develop an algorithm that finds planted cliques whenever $k \geq f(n)$, where f is some function increasing more rapidly than $2 \log n$ (which is the size of the maximum clique in a random graph). What might serve as an intermediate step towards finding the planted clique K? We list several approaches:

1. Finding a random vertex from K. In this case, the neighborhood of this vertex is a random graph on roughly $n/2$ vertices that contains a planted clique of size $k - 1 \geq f(n/2)$. Hence the algorithm can be continued recursively.

2. Finding an arbitrary vertex from K. In this case, the neighborhood of this vertex is a graph on roughly $n/2$ vertices that contains a planted clique of size $k - 1 \geq f(n/2)$. However, the graph is not necessarily distributed as a uniform random graph, and this might interfere with recursive applications of the algorithm (especially at deep levels of the recursion). Nevertheless, one would expect (at every level i) the increase in the value of k_i compared to $f(n_i)$ to compensate for the distortion of the uniform distribution, and hence that being able to find an arbitrary vertex from the hidden clique to suffice in order to find the whole clique.

3. Find a collection of t vertices, most of which are from K. One can imagine that if t is large enough (e.g., $t > k/2$) then this gives so much information about K that K can be efficiently found. On the other hand, if t is small (say $t < \log n$), this becomes similar (though a bit better) than approach (4) below.

4. Find a collection of t vertices, at least one of which belongs to K. In this case, one can branch to t runs, at least one of which is similar to approach (2). Then, in $\ell = c \log_t n$ iterations, one generates n^c problems (still polynomial in n for constant c), and one of which has roughly $n/2^\ell$ vertices and a planted clique of size $k - \ell$. This may suffice to find the planted clique. For example, if for $k = \sqrt{n}/\log n$ one had $t = \log n$, then one can choose $\ell > \log \log n$ and presumably find the planted clique once $k_\ell > \sqrt{n_\ell}$.

5. Find a random vertex not in K. In this case, remove it. The resulting graph is random on $n-1$ vertices with a planted clique of size $k \geq f(n-1)$. Hence the algorithm can be repeated.

6. Find an arbitrary vertex not in K. In this case, remove it. The resulting graph has $n-1$ vertices with a planted clique of size $k \geq f(n-1)$. However, the graph is not random (though close to random), and it is not clear how many times the argument can be repeated before it breaks down. The gain in k_i compared to $f(n_i)$ might not be significant enough to compensate for the non-randomness.

Algorithms for finding planted cliques can be thought of as having a first phase in which all vertices are sorted in some order that is hopefully correlated with being in K. Thereafter, one of the approaches above is employed. The algorithm of Kucera (1995) sorts vertices by degrees, and then essentially follows approach (1) - it picks the k top vertices, hoping to capture exactly K. The spectral algorithm of Alon et al. (1998) sorts vertices by their value in an eigenvector of the adjacency matrix of G, and then follows approach (3). The approaches used in this manuscript are closer to (6). After the vertices are sorted (by degree in LDR, or by a more complicated order in TPMR), the vertices deemed least likely to be in K are removed. We are not guaranteed to remove only vertices not in K. Hence when the removal phase ends, the vertices that we remain with form only part of K, and this puts us in the situation of approach (2). In fact, the situation is better than (2) in that we have several vertices from K and not just 1, and this makes finding the rest of K easy.

Our experimental results helped guide us in the design of algorithms. They suggested that approach (6) is more effective in finding hidden cliques than approach (3). They also suggested that once approach (6) is used, there is no significant advantage of sorting vertices by eigenvalues (as in Alon et al. (1998)) rather than by degree (as in Kucera (1995)).

3 The Low Degree Removal algorithm

We present here a simple algorithm for finding cliques. We call it the *Low Degree Removal* (LDR) algorithm. It's input is a graph G with vertices indexed 1 to n, and it works in two phases.

The removal phase.

1. Set $r = 0$ and $G_0 = G$.

2. If G_r is a clique, stop and return G_r.

3. Else, remove from G_r the vertex of lowest degree (breaking ties in favor of vertex with lower index). Call the remaining graph G_{r+1}.

4. Increment r and return to step 2.

Note that r counts the number of vertices that were removed from G in order to obtain G_r. The output G_r of the removal phase is necessarily a clique (of at least two vertices if G has an edge). The purpose of the second phase is to expand this clique greedily using the remaining vertices from G.

The inclusion phase.

1. For simplicity, rename those vertices of G that are not in the output G_r of the removal phase as $v_r, \ldots, v_1$. Set $t = r$ and K_t to be the set of vertices of G_r.
2. If v_t is connected (in G) to all vertices of K_t, then $K_{t-1} = K_t \cup v_t$. Else $K_{t-1} = K_t$.
3. Decrement t.
4. If $t = 0$ stop and return K_0. Else, return to step 2.

The output of the inclusion phase (and hence of LDR) is necessarily a clique.

It is pretty simple to implement LDR as a linear time algorithm (linear in n^2, the size of the adjacency matrix of G). We assume an adjacency matrix representation of G. For each vertex, its degree can be determined in time $O(n)$ by scanning the corresponding row in the matrix. Hence we can create a vector of all vertex degrees. Finding the minimum degree can then be done in time $O(n)$. When removing the vertex of minimum degree, updating the degrees of all other vertices takes time $O(n)$ (subtracting one from each neighbor of the removed vertex). Since less than n vertices are removed, the whole removal phase takes time $O(n^2)$, which is the size of the input matrix. Likewise, it is easy to see that the inclusion phase can be performed in time $O(n^2)$.

3.1 Intuition

This section presents some intuition regarding the performance of the algorithm LDR.

The key part of algorithm LDR is the removal phase. Once one has a good intuition for the probable outcome of the removal phase, one may design alternatives to the inclusion phase. The inclusion phase described for LDR works if all vertices that remain in G_r after the removal phase belong to K, and moreover, no vertex not in K is a neighbor of all vertices of G_r. However, if instead the likely outcome would have been judged to be different but still correlated with K, then one could imagine other continuations to the LDR algorithm (similar to approach (4) in Section 2).

In G' (random graph with no planted clique) the distribution of degrees is essentially the normal distribution with mean $n/2$ and standard deviation $\sqrt{n}/2$. Hence the fraction of vertices of degree greater than $n/2 + a\sqrt{n}$ (for some $a > 0$, or smaller than $n/2 - a\sqrt{n}$) is $e^{-\Omega(a^2)}$, and the maximum deviation from the average is $O(\sqrt{n \log n})$. The tail of the normal distribution is such that the highest degree vertex is likely to be unique, and likewise for the second highest degree vertex.

If the removal phase is run on G' one may expect that vertices will be removed roughly in order of their initial degree, with some random fluctuations. As an example for these fluctuations, if the two highest degree vertices do not share an edge, then necessarily the output will not include at least one of them, and will include at least one vertex of lower degree. By the time half the vertices have been removed, one expects the distribution of degrees to no longer resemble the bell shaped normal distribution. Instead, it is likely that the distribution of degrees will be skewed more heavily towards the larger degrees than towards the lower degrees. For example, the deviation of the highest degree vertex from the median degree will be larger than the deviation of the lowest degree vertex.

Consider now what happens in the graph G that includes the planted clique K. All vertices of K receive a boost in their degree, increasing it by roughly $k/2$. If $k > c\sqrt{n \log n}$ (for large enough $c > 0$) then they become the vertices of highest degree. In this case, the intuition that the removal phase roughly respects

the initial order of degrees implies that at the end of the removal phase, we are likely to remain with K. However, if $k < c\sqrt{n \log n}$ (for small enough $c > 0$), then the nature of the tail of the normal distribution is such that the vertex of highest degree in G is likely not to be from K (and similarly, only very few of the vertices in the high end of the degree spectrum of G belong to K). If indeed the removal phase would respect the original order of degrees, the final G_r would not contain vertices of K. However, as we shall see, it is likely that all vertices of G_r are from K even when $k = c\sqrt{n}$ for a sufficiently large constant c. How can we explain this?

Assume that $k = \sqrt{n}$. Assume also for simplicity that initially the removal phase follows the initial degree order perfectly. That is, that the first $n/2$ vertices to be removed are those of lowest initial degree. Then a large fraction of the vertices of K, say 84%, still remain (because the boost of $k/2$ received in the degree is one standard deviation of the normal distribution, and roughly 34% of the vertices of K are up to one standard deviation below the median degree). At that point the size of G decreased to $n' = n/2$, whereas the size of K decreased to $k' \simeq 0.84k > k/\sqrt{2}$. Now if the remaining graph at this stage would be a random graph on n' vertices ($G_{n',1/2}$) with a random planted clique of size k' then we would have reached improved parameters compared to the original parameters of the input (we would have $k' = cn'$ for some $c > 1$), and we could continue the argument recursively (this time considering the next $n'/2$ vertices that are removed). In the recursive process the relative rate at which the vertices of K are removed would be sufficiently smaller than the rate at which other vertices are removed to ensure that eventually only vertices of K remain.

The problem is of course that the intermediate graphs that remain (e.g., with $n/2$ vertices) are no longer random graphs (in the uniform sense). Instead, as we observed earlier, their degree distribution is skewed towards the higher degrees. Moreover, the degrees of vertices of K that survive might be skewed towards the lower degrees (as a fair fraction of them survived by getting a boost in degree that made them barely cross the original median degree). Hence for the recursive argument to work, one needs to show that the gain in the leading constant from 1 to c is sufficiently large to outweigh the skewness in the distribution of degrees. Something along these lines can indeed be done – see Section 3.2.

One may ask whether the LDR algorithm can detect planted cliques of size $o(\sqrt{n})$. This appears unlikely to us. In the appendix (Section A) we provide an informal argument to justify our intuition.

3.2 Analysis

Here we prove Theorem 1. We shall use the following notation.

$n_r = n - r$ denotes the number of vertices in G_r.

K_r denotes the set of vertices from K in G_r, and k_r denotes their number.

F_r denotes the set of other vertices in G_r, and $f_r = n_r - k_r$ denotes their number. We use F to denote F_0.

All probabilities are taken over the choice of G (the algorithm LDR is deterministic). The term *with high probability* denotes probability that tends to 1 as n grows. We shall use standard probabilistic inequalities (Markov, Chernoff, etc.) without stating them.

Proposition 2 *Let c_0 be some sufficiently large constant. Then with high probability, for every vertex $v \in G$, its degree into K in G' is at most $k/2 + c_0\sqrt{k \log n}$.*

Proof: Standard combination of Chernoff bound and union bound. □

Lemma 3 *Let c_1 be some sufficiently large constant. Then (with high probability) at every step r in which $f_r \geq \sqrt{n}$, the vertex of lowest degree in G_r has degree at most $n_r/2 + c_1\sqrt{n}$.*

Proof: An equivalent method for generating G is by choosing the set K first, making it into a clique, and then every remaining pair of vertices are connected by an edge independently with probability 1/2. Consider the situation after choosing K. For arbitrary values f_r in the range $\sqrt{n} \leq f_r \leq n - k$ and k_r in the range $0 \leq k_r \leq k$, pick an arbitrary set of f_r vertices from F and an arbitrary set of k_r vertices from K. Now, the expected number of endpoints of edges touching F_r is exactly $f_r(n_r - 1)/2$, the standard deviation is $O(\sqrt{f_r n_r})$, and the probability of deviating from the expectation by more than $\alpha\sqrt{n}\sqrt{f_r n_r}$ is at most 2^{-2n} (when $\alpha > 0$ is a sufficiently large constant). To compute the average degree in F_r, one needs to divide the number of endpoints by f_r, and hence with probability at least $1 - 2^{-2n}$, there is some vertex in F_r of degree at most $\frac{n_r - 1}{2} + \alpha\sqrt{n}\sqrt{n_r/f_r} \leq n_r/2 + \alpha\sqrt{(c+1)n}$ (we use here the fact that $f_r > \sqrt{n}$ and $k_r \leq c\sqrt{n}$). Taking a union bound over all permissible values of f_r and k_r, and then over all possible sets F_r and K_r (regardless of whether they actually appear in the algorithm LDR), the above holds simultaneously for all such choices, with probability at least $1 - 2^{-n}$. Setting $c_1 \geq \alpha\sqrt{c+1}$ the lemma is proved. □

For the next lemma we shall use the following definitions. Consider an arbitrary vertex $v \in K$. Let x_r be a random variable indicating whether in step r of the removal phase, the vertex removed was from F_r. Let y_i be the random variable indicating whether in the first step r at which $\sum_{j=1}^{r} x_j = i$, the vertex removed (that must be from F_r since $x_r = 1$ at this point) is a neighbor of v in G' (and hence in G). Let r_v denote the step in which v itself was removed (or the last step of the removal phase if v was not removed). Let $s_i(v) = \sum_{j=1}^{i} y_i - i/2$ denote the *surplus* of neighbors of v removed compared to non-neighbors up to the point at which i vertices from F where removed (defined only for values of i for which $i \leq \sum_{j=1}^{r_v} x_j$).

Lemma 4 *Let $c_2 > 0$ be some sufficiently large constant. Consider an arbitrary vertex $v \in K$. Then its maximum surplus satisfies:*

$$Pr[\max_{i \leq \sum_{j=1}^{r_v} x_j} s_i(v) \geq c_2\sqrt{n}] \leq 1/100$$

In other words, the surplus for v is unlikely to ever exceed $O(\sqrt{n})$.

Proof: Switch the order in which the random events that generate G are revealed to LDR. First, pick all edges of G' at random except those edges connected to v. Then pick K to include v, and complete it into a clique. Now generate a *tentative run* of the removal phase, without yet knowing which vertices of F are neighbors of v, and not allowing v to be removed during the tentative run. For the tentative run, one will need to know whether v is a neighbor of vertices (of F) that are candidates for removal. Only at the point in which the need arises, we will decide at random whether the candidate vertex u is a neighbor of v or not. At that point we say that (u, v) is *exposed* (either as being an edge or not). Hence the tentative run proceeds as follows.

1. If G_r is a clique, stop.

2. If G_r is not a clique, pick the vertex $u \neq v$ that in the current G_r has lowest degree (counting edges to v *only* in cases that they were already exposed, and breaking ties in favor of the vertex of lower

index). Now, in order to make the tentative run consistent with the true run, there are several cases to consider:

(a) If (u, v) was previously exposed (one such case is when $u \in K$ and then we know that (u, v) is an edge), remove u.

(b) If (u, v) was not previously exposed, expose it. If after that u remains of lowest degree (in particular, this must happen when (u, v) is not an edge), remove u. Otherwise, return to step (1).

When the tentative run stops, all edges to v have been exposed. Hence now one is in a position to perform a true run of the removal phase. If v is not removed in the true run, then the true run is identical to the tentative run. (This is a consequence of using a fixed rule for breaking ties, in particular, the rule of breaking them in favor of the vertex of lower index.) If v is removed in the true run, then the true run and the tentative run are identical up to the point in which v is removed. Beyond that point, we no longer care about either of these two runs.

Now, consider the development of surplus of v with respect to the tentative run, up to step r_v (up to that point the tentative run and true run are identical, and the lemma is not concerned with the surplus after point r_v). Vertices in K do not contribute to the surplus (by definition). As for vertices $u \in F$, up to step r we exposed the status of (u, v) for many candidates, but not all of them have been removed. Note however that each exposed (u, v) has probability exactly $1/2$ of being an edge (independent of all other events) and all those exposed and not removed have (u, v) as an edge. It follows that the process of building up surplus is stochastically dominated by a random sequence of ± 1. Moreover, the length of this sequence is at most n. By standard bounds (that follow from the Optional Stopping Theorem for Martingales), the probability that a random sequence ever builds up a surplus of $c_2\sqrt{n}$ (for sufficient large $c_2 > 0$) is at most $1/100$. □

Corollary 5 *With probability at least 9/10, there are at most $k/10$ vertices from K that ever reach a surplus of $c_2\sqrt{n}$ during the removal phase of algorithm LDR.*

Proof: Follows from the results of Lemma 4 and the use of Markov's inequality. □

We note that Corollary 5, with its simple proof that only uses Markov's inequality, is the reason why in Theorem 1 the probability is a fixed constant (rather than asymptotically 0 or 1). Possibly, the statement of Corollary 5 can be strengthened.

We can now prove Theorem 1.

Proof: Follow the (probable) evolution of the removal phase until step $\hat{r}$ satisfying $f_{\hat{r}} = \sqrt{n}$. (Observe that the removal phase cannot possibly end before $f_r = \sqrt{n}$ unless G' has a clique of size $\sqrt{n}$, which is highly unlikely.) Let us analyze now the value of $k_{\hat{r}}$, by upper bounding the number of vertices from K that are not in $K_{\hat{r}}$. Recall that $k = c\sqrt{n}$ for some sufficiently large constant c (larger than all other constants appearing in the proof, to be chosen later). In our proof we shall assume that all events that happen with high probability actually happen. (This assumption can be made as we shall only consider a constant number of such events.)

Let L be the set of vertices from K that in G' have degree not more than $n/2 - c_3\sqrt{n}$ (where c_3 is a sufficiently large constant). The expected number of such vertices is at most $k/100$, and hence with

probability at least $9/10$ there are at most $k/10$ such vertices. We do not assume that any of the vertices of L get to reach $K_{\hat{r}}$.

By Proposition 2, every vertex of K gained at least $k/2 - c_0\sqrt{k \log n}$ in its degree when K was made into a clique. For sufficiently large n we have that $k/2 \gg c_0\sqrt{k \log n}$, and for simplicity of notation (and with only negligible effect on the bounds) we shall assume that the increase in degree is $k/2$. Hence all vertices of $K \setminus L$ have degree at least $n/2 + (c/2 - c_3)\sqrt{n}$ in G. Let S be the set of vertices of K that suffered a surplus of $c_2\sqrt{n}$ or more in the removal phase. By Corollary 5, with probability at least $9/10$ we have that $|S| \leq k/10$. We do not assume that any of the vertices of S get to reach $K_{\hat{r}}$.

For all vertices of $K \setminus (L \cup S)$, their initial degree in G was at least $n/2 + (c/2 - c_3)\sqrt{n}$, their surplus is at most $c_2\sqrt{n}$, and hence at every step r their degree in G_r is at least $n_r/2 + (c/2 - c_2 - c_3 - c/5)\sqrt{n}$ (where the term $(c/5)\sqrt{n} = k/5 = k/10 + k/10$ is an upper bound on the effect of L and S not reaching $G_{\hat{r}}$). If this degree is at least $n_r/2 + c_1\sqrt{n}$, then these vertices cannot be removed. (At every step r, the vertex removed has degree at most $n_r/2 + c_1\sqrt{n}$, by Lemma 3.) If follows that if $3c/10 \geq c_1 + c_2 + c_3$ then $k_{\hat{r}} \geq 4k/5$. Recall that Lemma 3 required that $c_1 \geq \alpha\sqrt{c+1}$. Hence if c is large enough so as to satisfy $3c \geq 10(\alpha\sqrt{c+1} + c_2 + c_3)$ all required relations between the various parameters can be made to hold.

Let us now analyze what happens in the removal phase after step $\hat{r}$. At this point, all vertices of K_r have degree at least $4k/5$. All vertices of F_r have degree at most $f_r + k/2 + c_0\sqrt{k \log n}$ (by Proposition 2), which is smaller than $4k/5$ (when c is sufficiently large, using also the fact that $f_r \leq \sqrt{n}$). Hence no vertex of $K_{\hat{r}}$ will be removed, and all vertices of $F_{\hat{r}}$ will be removed. This establishes that the removal phase ends with at least $4k/5$ vertices from K and no vertex from F.

As to the inclusion phase, the fact that $k_r \geq 4k/5$ together with Proposition 2 implies that no vertex of F will be included. On the other hand, all vertices of $K \setminus K_r$ will be included, and hence the output of LDR will be the hidden clique K.

The probability that the assumptions of the analysis fail to hold is at most $1/10 + 1/10 + o(1) < 1/3$. □

We note that the proof of Theorem 1 works almost without change also if the removal phase in LDR is changed so that instead of removing a vertex of lowest degree, one removes a vertex of degree not larger than the average, or not larger than $n_r/2 + c_1\sqrt{n}$.

4 Some other algorithms

A variation of the LDR algorithm is LDRM (RM can stand for "remove many") which is similar to LDR, except that the number of vertices removed at each step is not 1, but rather a $p = 1/10$ fraction of the remaining vertices. This principle of removing a constant fraction of the vertices in each step is useful in algorithms (such as TPMR below) in which updating the sorted order of vertices after a removal is computationally expensive, and hence we do not wish to do this $\Omega(n)$ times. For LDR, updating the sorted order is not expensive, but nevertheless we ran also the version LDRM so as to see to what extent this modification affects the size of cliques that one can find. Our empirical finding is that the size of cliques is not as small as in LDR, but the qualitative behavior or LDR and LDRM appears similar.

An algorithm that was quantitatively better than LDR (though again, qualitatively similar) is the *Truncated Power Method Removal* (TPMR) algorithm. It is based on the well known power method for finding

the eigenvector which corresponds to the second largest eigenvalue. [(i)] If sufficiently many iterations are performed the (approximate version of the) eigenvector is indeed reached. However, we perform only L iterations, were L is a small number. Hence in a sense the vector that we reach is a blend between the eigenvector and our starting vector, which is the vector of degrees. The rational for using a blend between these vectors is that we view them as almost independent sources of information about the planted clique K, and hence the combination of the two is more informative than each one of them alone. We ran TPMR with the following values for the parameters: $L = 6$, $\omega = 0.5$ and $p = 1/10$. Lower values of p make the algorithm slower, though it manages to find somewhat smaller planted cliques. Like LDR, the algorithm TPMR works in two phases.

The removal phase.

1. Set $r = 0$ and $G_0 = G$.
2. If G_r is a clique, stop and return G_r.
3. Else
   (a) *Initialize vector.* Set $l = 0$ and set V_r^l to be a vector of $|G_r|$ entries, each entry is initialized with the degree of the corresponding vertex in G_r.
   (b) *Multiply by adjacency matrix.* Calculate $V^{\mathrm{PM}} = A_r\, V_r^l$, where A_r is the adjacency matrix of G_r.
   (c) *Compute average entry.* Calculate $\overline{V^{\mathrm{PM}}} = \sum_{i=1}^{|G_r|} V_i^{\mathrm{PM}}/|G_r|$.
   (d) *Normalize vector to be orthogonal to (approximate) largest eigenvector.* Calculate $V^{\mathrm{NPM}} = (V^{\mathrm{PM}} - \overline{V^{\mathrm{PM}}})/\sqrt{|G_r|}$.
   (e) *Do not jump – move slowly.* $V_r^{l+1} = \omega V^{\mathrm{NPM}} + (1-\omega)V_r^l$.
   (f) *Repeat.* Increment l. If $l < L-1$ return to step (b).
4. Sort V_r^{L-1} in descending order and remove from G_r the vertices corresponding to the last p-fraction of entries in the sorted V_r^{L-1}. Call the remaining graph G_{r+1}.
5. Increment r and return to step (2).

The inclusion phase. Same as the inclusion phase of LDR.

References

N. Alon, M. Krivelevich, and B. Sudakov. Finding a large hidden clique in a random graph. *Random Structures and Algorithms*, 13:457–466, 1998.

[(i)] The underlying matrix that is considered for TPMR is the 0/1 adjacency matrix of the graph. Similarly, Alon et al. (1998) use the second eigenvector of this matrix in their algorithm. It was only for the sake of simplicity, and with no significant effect on their results, that in the introduction our description of the algorithm of Alon et al. (1998) referred instead to the largest eigenvalue of the ± 1 adjacency matrix.

U. Feige and R. Krauthgamer. The probable value of the lovasz–schrijver relaxations for maximum independent set. *SIAM J. Comput.*, 32(2):345–370, 2003.

A. Frieze and R. Kannan. A new approach to the planted clique problem. *FSTTCS*, pages 187–198, 2008.

E. Hazan and R. Krauthgamer. How hard is it to approximate the best nash equilibrium? *SODA*, pages 720–727, 2009.

M. Jerrum. Large cliques elude the metropolis process. *Random Structures and Algorithms*, 3:347–359, 1992.

L. Kucera. Expected complexity of graph partitioning problems. *Discrete Applied Math.*, 57:193–212, 1995.

A Does LDR find cliques of size $o(\sqrt{n})$?

Here we provide informal justification (though this is not a proof) that LDR is unlikely to find hidden cliques of size $o(\sqrt{n})$.

Assume that $k = \epsilon\sqrt{n}$, where $\epsilon > 0$ is a sufficiently small constant. Initially, give the degree of every vertex of K a boost of $2k$ and not just $k/2$. Now consider the first $n/2$ steps of the removal phase. We still expect them to remove roughly the $n/2$ vertices of lowest degree. The extra large boost in degree received by vertices of K allows us to assume that half the boost of a vertex is lost in the first $n/2$ steps (similar to the assumption that roughly half the degree of a vertex is lost), and hence with respect to the boosted degree, the rate of loss in degree of vertices in K and not in K is similar. If ϵ is sufficiently small, the boost in degree helped only a small $O(\epsilon)$ fraction of vertices of K cross the median degree. Hence when $n' = n/2$ vertices remain in G_r, the number of vertices that remain from K is $k' \leq (1/2 + O(\epsilon))k < k/\sqrt{2}$. Hence now $k' = \epsilon' n'$ with $\epsilon' < \epsilon$, and in a recursive application of this argument no vertex of K will remain in the final G_r. Of course, the simplistic recursive argument is not correct due to the skewness in degrees. However, the discussion below indicates that the skewness should decrease rather than increase the likeliness of vertices of K to survive.

When $k = \epsilon\sqrt{n}$, remove all vertices whose degree is smaller than the average degree by $\sqrt{n}/\epsilon$ or more (perhaps with some leading constant). The fraction of vertices of K removed is less than half the fraction of other vertices removed. Hence hypothetically (if skewness had no effect), repeating such a process until $n/2$ vertices remain will leave at least $3k/4$ vertices from K. However, we argued above that only $(1/2 + O(\epsilon))k$ vertices from K are likely to remain, implying (if the argument above was correct) that the effect of the skewness works against vertices of K (at least in the initial steps of the removal phase).

B Some experimental results

Plotting the experimental results for LDRM and TMPR as in Figures 1 and 2 (with vertical axis as success probability and horizontal axis as size of hidden clique parameterized as a power of n) suggests that LDRM and TMPR may be able to find hidden cliques of size n^δ with $\delta < 1/2$. However, we have no theoretical analysis to support this extrapolation and suspect that it is not correct. Plotting the experimental results as in Figures 3 and 4, with size of hidden clique parameterized by leading constant in front of $\sqrt{n}$, suggest that this leading constant is smaller than 1 and slowly decreasing as n grows. Hence the

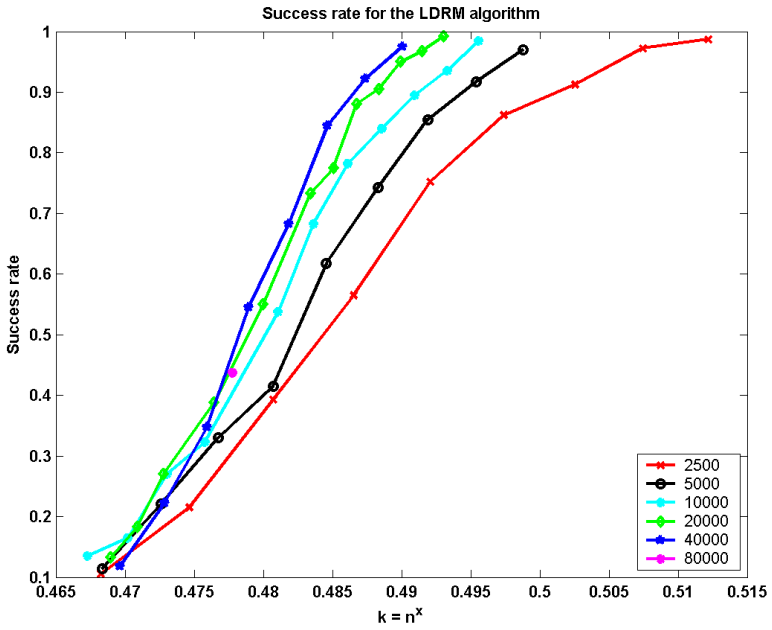

Fig. 1: The success rate for the LDRM algorithm is plotted (for graph sizes from 2500 to 80000) versus the clique size k which is given as a power of n, i.e., $k = n^x$. Each result is averaged over 400 independent runs.

experiments support the possibility that the algorithms can find hidden cliques of size $o(\sqrt{n})$, though we suspect that eventually the leading constant converges to some value above 1/2.

The random number generator. The results for Figures 1, 2, 3 and 4 were all obtained on a unix machine with the standard C++ random number generator which produces a number between 0 and 2^{31}. We have compared some of our results with a different random number generator by feeding our algorithm with random matrices produced by a FORTRAN program using the well known Numerical Recipes *ran2*, which is a multiplicative congruential algorithm which also includes random shuffling of the generated sequence with a cycle of 2^{59}. We got similar results which indicate that the results are not sensitive to the used random number generator, see Table 1.

Sorting vertices. In our experiments, we tried sorting vertices by their degrees, by certain eigenvectors, and by various blending of these approaches, such as TPMR. We made some comparisons checking the correlations of these sorted orders with membership in the planted clique. Table 2 presents some of our empirical results.

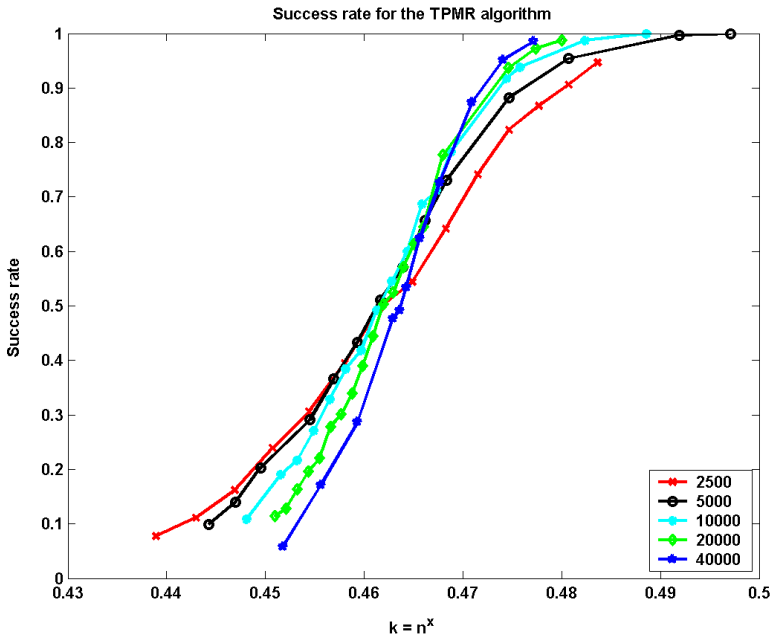

Fig. 2: The success rate for the TPMR algorithm is plotted (for graph sizes from 2500 to 40000) versus the clique size k which is given as a power of n, i.e., $k = n^x$. Each result is averaged over 1600 independent runs.

		% success	
n	k	C++	FORTRAN
2500	44	0.489	0.479
2500	45	0.549	0.574
2500	47	0.736	0.726
2500	51	0.932	0.924
5000	60	0.488	0.487
5000	61	0.531	0.548
10000	83	0.491	0.548
10000	84	0.568	0.570

Tab. 1: Comparison between different random number generators for various clique sizes and graph sizes. Each result is averaged over 1000 independent runs.

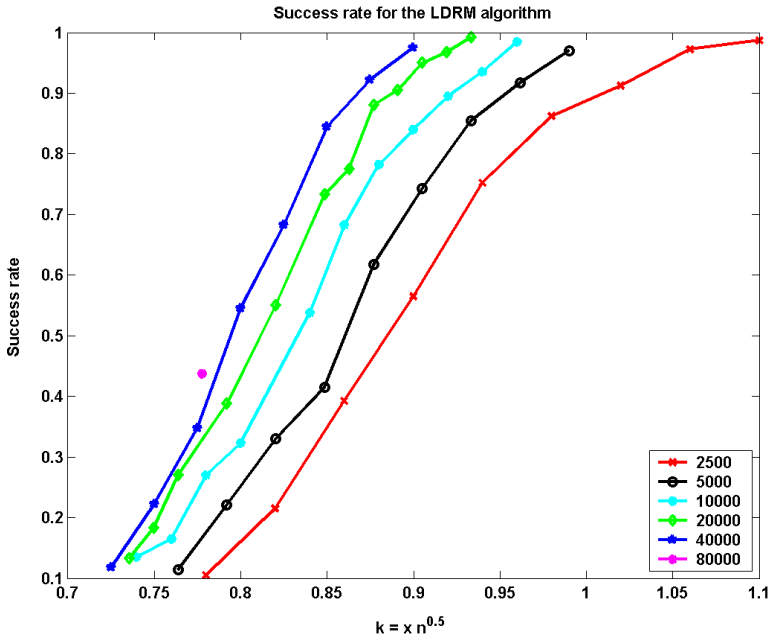

Fig. 3: The success rate for the LDRM algorithm is plotted (for graph sizes from 2500 to 80000) versus the clique size k which is given as a fraction of $n^{0.5}$, i.e., $k = xn^{0.5}$. Each result is averaged over 400 independent runs.

n	k	Degree	TPMR	EV
1600	33	30.80	31.81	28.93
3600	49	45.81	47.39	42.91
6400	64	59.36	61.66	55.27
10000	79	-	75.99	66.34

Tab. 2: One removal step algorithm which compares for various graph sizes n and clique sizes k three vectors: the vector of degree, the vector resulting from one step of the TPMR algorithm and the eigenvector which corresponds to the second largest eigenvalue of the $0/1$ version of the adjacency matrix of each graph. Each entry in the **Degree** and **TPMR** columns is averaged over 1000 cases and in the **EV** column over 100. Each number describes the averaged number of clique indices *left* after removing half of the vertices with smaller entries in the sorted vectors. Using the vector of degrees is better than using the eigenvector, and the TPMR vector is even better.

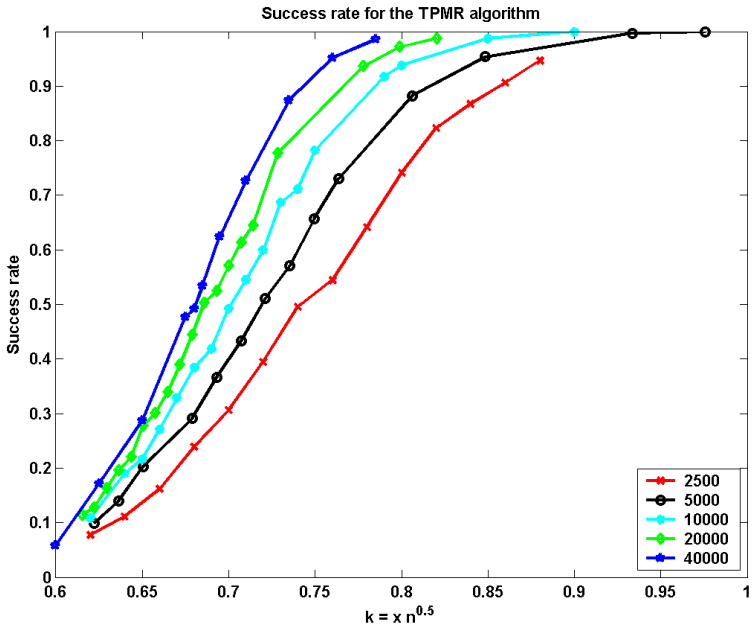

Fig. 4: The success rate for the TPMR algorithm is plotted (for graph sizes from 2500 to 40000) versus the clique size k which is given as a fraction of $n^{0.5}$, i.e., $k = xn^{0.5}$. Each result is averaged over 1600 independent runs.