Solving maximum clique in sparse graphs: an $O(nm + n2^{d/4})$ algorithm for d-degenerate graphs

Austin Buchanan, Jose L. Walteros, Sergiy Butenko, Panos M. Pardalos

October 20, 2013

Abstract

We describe an algorithm for the maximum clique problem that is parameterized by the graph's degeneracy d. The algorithm runs in $O(nm + nT_d)$ time, where T_d is the time to solve the maximum clique problem in an arbitrary graph on d vertices. The best bound as of now is $T_d = O(2^{d/4})$ by Robson. This shows that the maximum clique problem is solvable in $O(nm)$ time in graphs for which $d \leq 4 \log_2 m + O(1)$. The analysis of the algorithm's runtime is simple; the algorithm is easy to implement when given a subroutine for solving maximum clique in small graphs; it is easy to parallelize. In the case of Bianconi-Marsili power-law random graphs, it runs in $2^{O(\sqrt{n})}$ time with high probability. We extend the approach for a graph invariant based on common neighbors, generating a second algorithm that has a smaller exponent at the cost of a larger polynomial factor.

keywords: maximum clique, degeneracy, sparse graphs, fixed-parameter tractability, d-degenerate graphs

1 Introduction

We consider the maximum clique problem in sparse graphs. For notation and more information on this problem, see [3]. Our work is inspired by the near-optimal algorithm of [7] that lists all maximal cliques in a d-degenerate n-vertex graph in $O(dn3^{d/3})$ time. Since this algorithm enumerates all maximal cliques, it finds all maximum cliques as well. This shows that the maximum clique problem is fixed-parameter tractable when parameterized by degeneracy. Our task of finding a maximum clique is less involved and admits a better time bound.

Degeneracy is a common measure of the sparseness of a graph. Many real-life graphs are sparse, have low degeneracy, and follow a power-law degree distribution [6]. It is not uncommon to encounter graphs with millions of vertices but with degeneracy less than one hundred (see Table 2). Even some of the most

Table 1: Comparison of fastest known clique algorithms.

	arbitrary graphs	d-degenerate graphs
list all maximal cliques	$O(3^{n/3})$ by [5]	$O^*(3^{d/3})$ by [7]
find a maximum clique	$O(2^{n/4})$ by [10]	$O^*(2^{d/4})$ this paper

challenging benchmark instances for the maximum clique problem have $d \leq n/2$. In general, the degeneracy is sandwiched between the minimum degree and the maximum degree.

Definition 1 (degeneracy). *A graph is said to be d-degenerate if every (non-empty) subgraph has a vertex of degree at most d. The degeneracy of a graph is the smallest value of d such that it is d-degenerate.*

By [8], every d-degenerate graph admits an ordering of its vertices $(v_1, \ldots, v_n)$ such that each vertex v_i has at most d neighbors after it in the ordering, i.e., $|N(v_i) \cap \{v_i, \ldots, v_n\}| \leq d$. In fact, admitting such an ordering is equivalent to being d-degenerate. The degeneracy, as well as such an ordering, can be found in $O(m+n)$ time by iteratively removing a vertex of minimum degree [9]. In this paper, we will assume, w.l.o.g. that the input graph is connected, so $O(m+n) = O(m)$.

The community-degeneracy, defined below, can also be found in polynomial time by iteratively removing an edge whose incident vertices have the fewest common neighbors. In this paper we show that such an ordering can be found in $O(nm)$ time. We note that for most of the real-life instances appearing in [11], the community-degeneracy is halfway between the clique number and the degeneracy (or tighter). See Table 2 below.

Definition 2 (community-degeneracy). *A graph is said to be c-community-degenerate if every (non-edgeless) subgraph G' has an edge $\{u, v\}$ with $|N_{G'}(u) \cap N_{G'}(v)| \leq c$. The community-degeneracy of a graph is the smallest value of c such that it is c-community-degenerate.*

It is easy to see that a c-community-degenerate graph $G = (V, E)$ admits an ordering of its edges $(e_1, \ldots, e_m)$ such that each edge $e_i = \{u_i, v_i\}$ has $|N_{G[E_i]}(u_i) \cap N_{G[E_i]}(v_i)| \leq c$, where $G[E_i]$ is the *edge-induced* subgraph of $E_i = \{e_i, \ldots, e_m\}$.

Definition 3 (induced subgraph). *Consider a graph $G = (V, E)$. Given a subset $S \subseteq V$ of vertices, we denote the* ***vertex-induced*** *subgraph by $G[S] = (S, E \cap (S \times S))$. Given a subset $E' \subseteq E$ of edges, we denote the* ***edge-induced*** *subgraph by $G[E'] = (V', E')$ where $V' = \{v \in V : \exists \{v, w\} \in E'\}$.*

It is also easy to see that $c \leq d-1$, as any d-degenerate-ordering immediately gives a $(d-1)$-community-degenerate ordering (replace the vertex v_1 in the vertex-ordering by the edges incident to v_1, replace the vertex v_2 by the edges incident to v_2 but not to v_1, etc). In general, the community-degeneracy c is sandwiched between $\min_{\{u,v\} \in E} |N(u) \cap N(v)|$ and $\max_{\{u,v\} \in E} |N(u) \cap N(v)|$.

Table 2: Comparing degeneracy d and community-degeneracy c versus clique number ω on some real-life graphs from [1, 11].

Graph	n	m	ω	c	d
as-22july06	22,963	48,436	17	15	25
kron_g500-simple-logn16	65,536	2,456,071	136	283	432
citationCiteseer	268,495	1,156,647	13	11	15
ldoor	952,203	22,785,136	21	19	34
in-2004	1,382,908	13,591,473	489	487	488
cage15	5,154,859	47,022,346	6	4	25
uk-2002	18,520,486	261,787,258	944	942	943

In arbitrary graphs, the degeneracy d and the community-degeneracy c can be $\Omega(n)$; the complete graph on n vertices has $d = n-1$ and $c = n-2$. However, these graph invariants behave much differently on power-law graphs. A graph is said to be power-law if the number of vertices with degree q is proportional to $q^{-\alpha}$, where $\alpha \in (1, 3)$ is a constant. Under the Bianconi-Marsili power-law random graph model, bounds on d (and therefore c) can be shown to hold with high probability. It has been shown [2] that $d = O(n^{1/(2\alpha)})$ whenever $1 < \alpha \leq 2$ and $d = O(n^{(3-\alpha)/4})$ whenever $2 < \alpha < 3$. This shows that, with high probability, our algorithm that is parameterized by degeneracy runs in $2^{O(\sqrt{n})}$ time in power-law random graphs ($\alpha > 1$). The time bound improves as α increases, running in time $2^{O(n^{1/4})}$ for $\alpha > 2$. We also note that a different power-law random graph model, the Barabási-Albert model, creates graphs with bounded degeneracy [7].

Furthermore, the degeneracy d can be arbitrarily larger than the community-degeneracy c. This is the case for the class of hypercube graphs. These graphs are triangle-free and hence have community-degeneracy $c = 0$. However, the degeneracy of the d-dimensional hypercube graph is d.

2 The algorithms

Lemma 1. *Let $(v_1, \ldots, v_n)$ be any vertex-ordering of an n-vertex graph $G = (V, E)$. Denote by $\omega(G)$ the size of a maximum clique in a graph G. Then,*

$$\omega(G) = 1 + \max_{1 \leq i \leq n} \omega(G[S_i]), \tag{1}$$

where $S_i = N(v_i) \cap \{v_i, \ldots, v_n\}$.

Proof. First see that $\omega(G) \geq 1 + \omega(G[S_i])$ for any vertex $v_i \in V$; take a maximum clique in $G[S_i]$ and add v_i. Now we show the reverse inequality. Let S be a maximum clique in G and let $v_{i^*} \in S$ be its earliest vertex in the vertex-ordering. Then $S \subseteq S_{i^*} \cup \{v_{i^*}\}$ and $\omega(G[S_{i^*}]) \geq \omega(G) - 1$. □

Let T_d denote the time to solve the maximum clique problem in an arbitrary d-vertex graph. Note that $T_d = O(2^{d/4})$ by the well-cited but unpublished paper [10] or $T_d = O(1.2114^d)$ by the peer-reviewed [4]. Either (or any other clique algorithm) can be used as MaxCliqueSubroutine($\cdot$) in Algorithms 1 and 3.

```
Data: A graph $G = (V, E)$
Result: The clique number $\omega(G)$
compute a degeneracy ordering $(v_1, \ldots, v_n)$ of $G$;
for $i = 1, \ldots, n$ do
    $S_i \leftarrow N(v_i) \cap \{v_i, \ldots, v_n\}$;
    // call Robson's algorithm [10]
    $\omega(G[S_i]) \leftarrow$ MaxCliqueSubroutine($G[S_i]$);
end
return $\omega(G) = 1 + \max_{1 \leq i \leq n} \omega(G[S_i])$;
```

Algorithm 1: A maximum clique algorithm parameterized by degeneracy.

Theorem 1. *Algorithm 1 solves the maximum clique problem in d-degenerate graphs in $O(nm + nT_d) = O(nm + n2^{d/4})$ time. Using n processors, this reduces to $O(m + T_d)$ time.*

Proof. The degeneracy ordering $(v_1, \ldots, v_n)$ can be found in $O(m)$ time [9]. The for-loop can be run in parallel. It is clear that S_i can be found in $O(n)$ time. We argue that $G[S_i]$ can be created in $O(m)$ time. For each edge e, check (in constant time by storing S_i as a boolean n-array) if its incident vertices are after v_i in the vertex-ordering. If so, add edge e to $G[S_i]$. See that S_i has at most d vertices by the degeneracy ordering, so $\omega(G[S_i])$ can be computed in $T_d = O(2^{d/4})$ time. Thus, using $p \leq n$ processors, the algorithm runs in $O(m + \frac{n}{p}(m + T_d))$ time. Correctness of the algorithm follows by Lemma 1. □

Corollary 1. *The maximum clique problem is solvable in $O(nm)$ time in the class of graphs for which $d \leq 4 \log_2 m + O(1)$.*

We now move on to the second maximum clique algorithm (Algorithm 3), based on the community-degeneracy. The algorithm achieves a smaller exponent at the cost of a larger polynomial factor (m instead of n). First we give an efficient algorithm for computing community-degeneracy.

```
Data: A graph $G = (V, E)$
Result: A community-degeneracy edge-ordering $(e_1, \ldots, e_m)$
$H \leftarrow G$;
for $i = 1, \ldots, m$ do
    find an edge $e = \{u, v\}$ in $H$ that minimizes $|N_H(u) \cap N_H(v)|$;
    $e_i \leftarrow e$;
    $H \leftarrow H - e$;
end
return $(e_1, \ldots, e_m)$;
```

Algorithm 2: An algorithm for finding a community-degeneracy edge-ordering.

Lemma 2. *Algorithm 2 finds a community-degeneracy ordering and can be implemented to run in $O(nm)$ time.*

Proof. The correctness is clear, so we are left with devising data structures that achieve $O(nm)$ time. The idea is to mimic the data structures used in the degeneracy algorithm [9], but instead of using an adjacency list of the neighbors for each vertex, we have an intersection list for each edge $e = \{u, v\}$ that lists the vertices from $N(u) \cap N(v)$. Clearly, the intersection list of an edge contains at most $n - 2$ vertices, and since $N(u) \cap N(v)$ can be obtained in $O(n)$ time, generating all such lists takes $O(nm)$ time.

Let $D_H(e) = |N_H(u) \cap N_H(v)|$. To execute the steps in the for-loop, instead of updating the intersection lists at each iteration, it is possible to keep track of the edges that have not been removed using a bucket structure. This structure is comprised of $n-1$ buckets (namely, $\{0, 1, \ldots, n-2\}$) and an array of headers pointing to one of the elements in each bucket. The buckets are stored as linked lists and are initialized while creating the intersection lists. At each iteration of the for-loop, if edge e is still in H, it is stored in bucket $D_H(e)$. Identifying the edge to be removed can be done in $O(n)$ time by searching for the first nonempty bucket. Furthermore, whenever an edge $e = \{u, v\}$ is removed from H, the algorithm scans the intersection list of e checking if edges $e' = \{u, w\}$ and $e'' = \{v, w\}$ remain in H, for all w in the list. If e' or e'' were not removed before, they are relocated to buckets $D_H(e') - 1$ and $D_H(e'') - 1$, respectively. Scanning the intersection list takes $O(n)$ time, and any edge relocation in a linked list takes constant time. Thus, since the algorithm removes all m edges and every iteration takes $O(n)$ time, the algorithm runs in $O(nm)$ time. □

Lemma 3. *Let $(e_1, \ldots, e_m)$ be any edge-ordering of an m-edge graph $G = (V, E)$ with $m \geq 1$. Denote by $\omega(G)$ the size of a maximum clique in a graph G. Then,*

$$\omega(G) = 2 + \max_{1 \leq i \leq m} \omega(G_i), \tag{2}$$

where $e_i = \{u_i, v_i\}$, $E_i = \{e_i, \ldots, e_m\}$, $S_i = N_{G[E_i]}(u_i) \cap N_{G[E_i]}(v_i)$, and $G_i = (S_i, E_i \cap (S_i \times S_i))$.

Proof. First see that $\omega(G) \geq 2 + \omega(G_i)$ for any edge $e_i = \{u_i, v_i\} \in E$; take a maximum clique in G_i and add vertices u_i and v_i. Now we show the reverse inequality. Let S be a maximum clique in G and let $e_{i^*} = \{u_{i^*}, v_{i^*}\}$ be the earliest edge of $G[S]$ in the edge-ordering. Then every vertex in $S \setminus \{u_{i^*}, v_{i^*}\}$ belongs to G_{i^*} and every pair of vertices in $S \setminus \{u_{i^*}, v_{i^*}\}$ is adjacent in G_{i^*}, so $\omega(G_{i^*}) \geq \omega(G) - 2$. □

```
Data: A graph $G = (V, E)$
Result: The clique number $\omega(G)$
compute a community-degeneracy edge-ordering $(e_1, \dots, e_m)$ of $G$;
for $i = 1, \dots, m$ do
    $\{u_i, v_i\} \leftarrow e_i$;
    $E_i \leftarrow \{e_i, \dots, e_m\}$;
    $S_i \leftarrow N_{G[E_i]}(u_i) \cap N_{G[E_i]}(v_i)$;
    $G_i \leftarrow (S_i, E_i \cap (S_i \times S_i))$;
    // call Robson's algorithm [10]
    $\omega(G_i) \leftarrow$ MaxCliqueSubroutine$(G_i)$;
end
return $\omega(G) = 2 + \max_{1 \le i \le m} \omega(G_i)$;
```

Algorithm 3: A maximum clique algorithm parameterized by community-degeneracy.

Theorem 2. *Algorithm 3 solves the maximum clique problem in c-community-degenerate graphs in $O(m^2 + mT_c) = O(m^2 + m2^{c/4})$ time. Using m processors, this reduces to $O(nm + T_c)$ time.*

Proof. The community-degeneracy ordering $(e_1, \dots, e_m)$ can be found in $O(nm)$ time by Lemma 2. The for-loop can be run in parallel. The set S_i can be found in $O(m)$ time. It consists of those vertices $w \in V$ such that the edges $\{u_i, w\}, \{v_i, w\}$ occur after edge $e_i = \{u_i, v_i\}$ in the edge-ordering. We argue that the subgraph G_i can be found in $O(m)$ time. Check each edge e that appears after e_i in the edge-ordering. If both of its incident vertices belong to S_i then add e to G_i. See that S_i has at most c vertices by the community-degeneracy ordering, so $\omega(G_i)$ can be computed in $T_c = O(2^{c/4})$ time. Thus, using $p \le m$ processors, the algorithm runs in $O(nm + \frac{m}{p}(m + T_c))$ time. Correctness of the algorithm follows by Lemma 3. □

3 Conclusion

We have provided two algorithms for the maximum clique problem in sparse graphs. The algorithms give theoretical evidence that the maximum clique problem can be solved more quickly in sparse graphs than in arbitrary graphs. It remains to be seen which of the two algorithms performs better in practice. In this work, we focus on the worst-case running time of these algorithms; their computational performance will be investigated in a subsequent paper.

Incidentally, this paper gives algorithms for dense instances of maximum independent set or of vertex cover. We suspect that many other algorithms for NP-hard graph problems can be parameterized by degeneracy or co-degeneracy (the degeneracy of the complement graph). This seems to be a promising area for algorithms research due to the ubiquity of low-degeneracy power-law graphs in practice.

Acknowledgements

This material is based upon work supported by the AFRL Mathematical Modeling and Optimization Institute. Partial support by AFOSR under grants FA9550-12-1-0103 and FA8651-12-2-0011 is also gratefully acknowledged.

References

[1] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner. *Graph partitioning and graph clustering*, volume 588. American Mathematical Society, 2013.

[2] G. Bianconi and M. Marsili. Emergence of large cliques in random scale-free networks. *Europhysics Letters*, 74(4):740, 2006.

[3] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of combinatorial optimization*, pages 1–74. Springer, 1999.

[4] N. Bourgeois, B. Escoffier, V. T. Paschos, and J. M. M. van Rooij. Fast algorithms for max independent set. *Algorithmica*, 62(1–2), 382–415, 2012.

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

[6] F. Chung. Graph theory in the information age. *Notices of the AMS*, 57(6):726–732, 2010.

[7] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. *Algorithms and Computation*, pages 403–414, 2010.

[8] D. R. Lick and A. T. White. k-degenerate graphs. *Canad. J. Math*, 22:1082–1096, 1970.

[9] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983.

[10] J. M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical report, LaBRI, Université de Bordeaux I, 2001.

[11] A. Verma, A. Buchanan, and S. Butenko. Solving the maximum clique and vertex coloring problems on very large sparse networks. Working paper, Department of Industrial and Systems Engineering, Texas A&M University, College Station, TX, 2012.