k-Connected Spanning Subgraphs of Low Degree

Tomás Feder* Rajeev Motwani† An Zhu‡

Abstract

We consider the problem of finding a k-vertex (k-edge) connected spanning subgraph K of a given n-vertex graph G while minimizing the maximum degree d in K. We give a polynomial time algorithm for fixed k that achieves an $O(\log n)$-approximation. The only known previous polynomial algorithms achieved degree $d+1$ for optimum d if $k = 1$ (the case of spanning trees) and a factor $O(n^\delta)$ for any $\delta > 0$ if $k = 2$ for the case of 2-edge connectivity. Our result answers open problems of Ravi, Raghavachari, and Klein [12, 11] and Hochbaum [8]. The result extends to a weighted version with non-uniform degree bounds for different vertices. Our approach is based on an $O(\log n)$-approximation bound for a problem that combines set cover and degree minimization, thus extending the tight $\Theta(\log n)$ bound for set cover. We also consider extensions to finding k-connected subgraphs or minors of low degree spanning a large fraction of the vertices, for $k = 1, 2, 3$, with an application to finding long cycles in graphs.

1 Introduction

The problem of computing low degree subgraphs satisfying given connectivity properties of a given graph arises naturally in the design of communications networks. The simplest of these problems is the Minimum Degree Spanning Tree (MDST) problem. The input is an arbitrary graph G and the goal is to compute a spanning tree of G whose maximal degree is the smallest among all spanning trees of G. The MDST problem is a generalization of the Hamiltonian Path problem and is NP-hard. For any $k \geq 2$, the problem of deciding whether the optimal solution has maximal degree k is NP-complete [6]. The first result on approximation a minimum-degree spanning tree was that of Fürer and Raghavachari [3]. They gave a polynomial time approximation algorithm with ratio $O(\log n)$. In subsequent work, Fürer and Ragahavachari [4] improved their previous results and provided another polynomial time algorithm to approximate MDST problem to within one from optimal. Clearly no better approximation algorithms are possible for this problem.

Little success was obtained in generalizing this problem to larger degrees of connectivity. Ravi, Raghavachari, and Klein [12, 11] used local optimization techniques to obtain an approximation algorithm for the minimum-degree 2-edge-connected spanning subgraph problem. They obtained an algorithm that approximates the maximal degree d by $cd + O(\log_c d)$ for any $c \geq 1$ in time $n^{O(1)+\log_c n}$. Thus approximation factor $O(\log n/\log\log n)$ is achieved in time $n^{O(\log n/\log\log n)}$, and the best approximation factor achieved in polynomial time is $O(n^\delta)$ for any fixed $\delta > 0$. This leaves open the question of whether better approximation factors, either for this problem or for higher

*268 Waverley Street, Palo Alto, CA 94301. Email: `tomas@theory.stanford.edu`

†Department of Computer Science, Stanford University, Stanford, CA 94305. Supported in part by NSF Grants IIS-0118173, EIA-0137761, and ITR-0331640, and grants from Microsoft, SNRC, and Veritas. Email: `rajeev@cs.stanford.edu`

‡Department of Computer Science, Stanford University, Stanford, CA 94305. Email: `anzhu@cs.stanford.edu`

connectivity, can be obtained. The problem of minimizing the number of edges while achieving k-connectivity has also been extensively studied [1, 5, 7]. A general introduction to this subject area can be found in Chapters 6 and 7 in the book, *Approximation Algorithms for NP-Hard Problems*, edited by Hochbaum [8].

In this paper, we solve the open problem mentioned above [12, 11, 8] by designing an $O(k \log n)$-approximation algorithm (in polynomial time) that minimizes the maximum degree of k-edge-connected or k-vertex-connected spanning subgraphs for fixed k via a new approach. The basic approach comes from a general problem related to set cover, which we call the MINIMUM DEGREE HYPERGRAPH (MDH) problem. The MDH problem asks to select a set of hyperedges in a hypergraph G, where each edge e in G has a corresponding hyperedge f in an associated hypergraph H, and the aim is to cover the vertices in H with the hyperedges f while minimizing the maximal degree of vertices with the corresponding selected hyperedges g in G. We first give a combinatorial algorithm that achieves a bound of $O(s \log n)$ in polynomial time, where H has n vertices and the hyperedges in G have size at most s. We further improve this bound to $O((\log s / \log\log s) \log n)$ via randomization and $O(s^{1/r} \log n)$ after re-randomization for any integer constant r.

The results generalize to the case where the vertices in G and/or in H have possibly different degree upper and lower bounds given to be achieved within an approximation factor, and with possibly different degree contributions given for each hyperedge to each vertex in G and/or in H. In particular, this implies that if there exists a k-vertex (edge, respectively) connected spanning subgraph with degree d_i for each vertex v_i, then we can find in polynomial time a corresponding k-vertex (edge, respectively) connected spanning subgraph with degree $O(d_i \log n)$ for each vertex v_i.

We also consider the problem of minimizing the degree of a k-vertex (edge) connected subgraph (minor) that spans only r of the vertices, and give a polynomial time approximation algorithm with ratio $(n/r) \log^{O(1)} n$ for $k = 1$ and $k = 2$ in the case of subgraphs, and for $k = 3$ in the case of minors. We remark that this version of the problem is closely related to the problem of finding a long cycle in a graph, and leads to a result that finds a long cycle in a graph containing a large 3-cyclable minor of low degree, continuing the work of Feder and Motwani [2].

The rest of the paper is organized as follows. Section 2 presents a deterministic algorithm for the MDH problem, as well as a reduction from the k-vertex (edge) connected spanning subgraph problem to the MDH problem. Section 3 presents a randomized algorithm for the MDH problem with non-fixed degree upper bounds. Finally, Section 4 talks about finding k-vertex (edge) connected subgraph that spans a large fraction of the vertices, and applications to finding long cycles in graphs.

2 Deterministic Algorithms for Fixed Degree Bound

We first formally define the MINIMUM DEGREE HYPERGRAPH (MDH) problem. An instance of the MDH problem consists of two hypergraphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$, where the hyperedges of G are paired up with the hyperedges of H. The goal is to select a set of hyperedges in H to form a set cover of the vertices in H and so that the corresponding set of selected hyperedges in G give the minimum possible maximum degree for vertices in G. If all hyperedges in G share a vertex in G, then this is the SET COVER problem, and the optimal polynomial time algorithm achieves a $\Theta(\log |V(H)|)$-approximation [10, 13].

We can formulate the MDH problem as an integer program, as follows:

$$
\begin{array}{rll}
\text{Minimize:} & d & \\
\text{Subject to:} & \sum\limits_{u_i \in S_e} x_e \le d & \forall u_i \in V(G) \\
& \sum\limits_{v_j \in T_e} x_e \ge 1 & \forall v_j \in V(H) \\
& x_e \in \{0,1\} &
\end{array}
$$

where x_e is an integer variable corresponding to whether hyperedge $e \in E(G)$ (and its corresponding edge $e \in E(H)$) is chosen, S_e is the set of vertices in G corresponding to hyperedge e, and T_e is the set of vertices in H corresponding to hyperedge e.

The problem can be generalized by requiring various proportional degree factors $\alpha_i \ge 1$ for each vertex u_i in G, various proportional degree contributions $1 \le a_{ei} \le \alpha_i$ for S_e to vertex u_i, and similarly, various proportional multiple covering amounts $\beta_j \ge 1$ for each vertex v_j in H, and various proportional multiple covering contributions $1 \le b_{ej} \le \beta_j$ for T_e to vertex v_j. This leads to the following integer program.

$$
\begin{array}{rll}
\text{Minimize:} & d \ge 1 & \\
\text{Subject to:} & \sum\limits_{u_i \in S_e} a_{ei} x_e \le \alpha_i d & \forall u_i \in V(G) \\
& \sum\limits_{v_j \in T_e} b_{ej} x_e \ge \beta_j & \forall v_j \in V(H) \\
& x_e \in \{0,1\} &
\end{array}
$$

We consider two parameters $s = \max_e |S_e|$ and $m = \sum_j \beta_j$. Here we obtain a $O(s \log n)$-approximation with a deterministic combinatorial algorithm under a specific additional assumption, $a_{ei} = a_e$ for all $e \in E(G), i \in V(G)$. In the next section, we shall improve this bound with a randomized algorithm that achieves an $O((\log s / \log\log s) \log m)$-approximation, by means of linear programming and randomized rounding, which also gives an $O(s^{1/r} \log m)$ deterministic approximation result after de-randomization for any integer constant r.

Theorem 1 *Suppose $a_{ei} = a_e$, $\forall e \in E(G), \forall i \in V(G)$. Then there is a deterministic polynomial time algorithm for the MDH problem that achieves a maximum degree $2(ds+1)\log m$ for an optimum degree of d, or $(ds+1)\log m$ if further $a_e = 1$ for all $e \in E(G)$, for an approximation factor of $O(s \log m)$.*

Proof. We adopt a greedy approach. In each phase $i \ge 0$ (with $m_0 = m$), greedily select a maximal set of hyperedges in G, so that each chosen hyperedge e, which contributes $h_e = \sum_{v_j \in T_e} \min(\beta_j, b_{ej})$ towards $m_i = \sum_j \beta_j$, is picked to give the largest possible value for h_e/a_e[1]. Hyperedges are selected until each e that has not been chosen has a vertex u_i in G for which the bound α_i has been reached. Only hyperedges e such that this bound α_i has not been reached for any u_i in e may be chosen, so the degree bound for the selected edges in each phase in G may not exceed $\alpha_i + a_e \le 2\alpha_i$, and may not exceed α_i if all $a_e = 1$. When we may no longer choose any e, the chosen edges have covered a total $h = \sum h_e$ out of m_i. We then set $m_{i+1} = m_i - h$ and continue until all the β_j's become 0.

We wish to bound the amount of m_{i+1} with respect to m_i. Let GE denote the set of the hyperedges selected greedily during phase i. The remaining $m_i - h$ at the end of phase i can be covered with the edges in the optimal solution. Assume a hyperedge e in the optimal solution could cover h_e with respect to m_{i+1}, and let u_i be the vertex for which α_i has been reached using the

[1] Once some hyperedge is chosen, we properly subtract the amount b_{ej} from each affected β_j, or set $\beta_j = 0$ if $b_{ej} > \beta_j$.

edges in GE. Then for each $e' \in GE$ that was used to reach α_i we have $h_{e'}/a_{e'} \geq h_i \geq h_e/a_e$ since the algorithm was greedy. Of the $h_{e'}$ achieved by e', we may assign $h_{e'}/s$ to u_i by the definition of s. The edges $e \notin GE$ in the optimal solution containing u_i have a combined contribution towards m_{i+1} no more than $\sum h_e = \sum a_e(h_e/a_e) \leq \sum a_e h_i \leq d\alpha_i h_i \leq \sum d a_{e'}(h_{e'}/a_{e'}) = \sum d h_{e'}$, a factor of ds from the assigned $h_{e'}/s$. Thus $m_{i+1} = m_i - h \leq dsh$, or $m_{i+1} \leq m_i(1 - 1/(ds+1))$, indicating the total number of phases is bounded by $(ds+1)\log m$.

Since each phase contributes at most $2\alpha_i$ for u_i, or at most α_i for the case where all $a_e = 1$, this gives the bounds $2\alpha_i(ds+1)\log m$ and $\alpha_i(ds+1)\log m$ respectively. $\square$

Next we show how to reduce the k-vertex (edge) connected spanning subgraph problem to the MDH problem. We introduce an intermediate Minimum Degree Spanning Subgraph (MDSS) problem. An instance of the MDSS problem is a graph G and a collection of r subgraphs G_i of G. The aim is to select a subgraph K of G so that the subgraph of G_i induced by the edges of G_i in K spans G_i, for each G_i, and the maximum degree d of K is minimized. As before, this may be generalized to having amounts $\alpha_i \geq 1$ at each vertex u_i in G, so that the degree at u_i to be satisfied is $\alpha_i d$, and we may also assume that edge e contributes $1 \leq a_{ei} \leq \alpha_i$ to the degree at u_i.

Theorem 2 *The MDSS problem can be approximated in polynomial time to achieve maximum degree at most $2(2d+1)\log((|V(G)|-1)r)$ when the optimum degree is d and if $a_{ei} = a_e$, $\forall e \in E(G), \forall i \in V(G)$; and at most $(2d+1)\log((|V(G)|-1)r)$ if $a_e = 1$, $\forall e \in E(G)$. This gives an $O(\log(|V(G)|r))$ approximation factor.*

Proof. During the duration of the algorithm, we will need to consider at most $|V(G_i)|-1$ cuts per G_i to be traversed in order to obtain a spanning tree of G_i with $|V(G_i)|-1$ edges, for a total of at most $(|V(G)|-1)r$ cuts. At any stage of the algorithm, we may let H be the hypergraph whose vertices are the different cuts out of different components of selected edges in G_i, where the hyperedges of H correspond to edges $e \in E(G)$, and the elements of such hyperedges in H are the cuts traversed by e. We may thus apply Theorem 1 to G and H, with H updated every time a new edge is added to K to reflect the changing cut requirements. The result follows with $s = 2$ and $m = (|V(G)|-1)r$ from Theorem 1. $\square$

For the problem of k-vertex (edge, respectively) connected spanning subgraph, the G_i's are the graphs obtained from G by removing $k-1$ vertices (edges, respectively), with $r = \binom{|V(G)|}{k-1}$ ($r = \binom{|E(G)|}{k-1}$, respectively). The following answers an open problem of Hochbaum's book [8] and of Ravi, Raghavachari, and Klein's paper [12, 11].

Theorem 3 *For any constant k, there is a polynomial time algorithm that finds a k-vertex-(or edge)-connected spanning subgraph H of maximum degree $O(dk\log n)$, where the optimum maximum degree is d, for an $O(k\log n)$ approximation factor. This approximation factor for d extends to the case the degree at a vertex u_i must be at most $\alpha_i d$ and the contribution of each edge e to this degree is $a_{ei} = a_e$.*

For the special case of 2-vertex connectivity, we could further improve the above bound under certain conditions, via the following.

Theorem 4 *There is a polynomial time algorithm that finds a 2-vertex-connected spanning subgraph H of maximum degree $d+1+2df$ for optimum degree d, provided a 2-edge-connected spanning subgraph H' of maximum degree f can be found.*

Proof. First find a spanning tree H'' of degree at most $d+1$. Each time a vertex v splits H'' (that is $H''-v$ has at least two components), split v into two adjacent vertices, one for each of the two parts joined at v. Finding a 2-edge-connected spanning subgraph on this new graph takes care of each such edge v_1v_2, with increase in degree $2f$. This needs to be done d times to fully take care of v, for an increase of $2df$ and total degree $d+1+2df$. □

The current best known estimate for f is $cd+O(\log_c n)$, for any $c \geq 1$, with running time $n^{O(\log_c n)}$ [12, 11]. Thus improves the bound of Theorem 3 only if $d \leq \log n$. In particular, if d is constant, we can achieve degree $O(\log n/\log\log n)$ in time $n^{O(\log n/\log\log n)}$ from Theorem 4, versus degree $O(\log n)$ in polynomial time from Theorem 3. Note that the earlier best know polynomial time approximation algorithm for the case of a 2-edge connected spanning subgraph gives an approximation of n^δ for any chosen $\delta > 0$, by letting $c = n^\delta$, as opposed to the much smaller $O(\log n)$ bound of Theorem 3.

3 The General Case

We now generalize Theorem 1 by removing the assumption that $a_{ei} = a_e$ there stated.

Theorem 5 *There is a randomized polynomial time algorithm for the general MDH problem that achieves an $O((\log s/\log\log s)\log m)$ approximation factor, and also achieves an $O(s^{1/r}\log m)$ approximation factor after de-randomization for any integer constant $r \geq 2$.*

Proof. We consider the integer program corresponding to the problem, replace the integer constraint $x_e \in \{0,1\}$ with the linear constraint $0 \leq x_e \leq 1$, and solve the linear program to obtain values for the x_e and for $d \geq 1$. We may assume that $d = 1$ by setting the new α_i to be $\alpha_i d$. Next, we select hyperedge e with probability x_e, independently for each e at random. We shall show that with probability at least $1-1/(cs)$ for any chosen constant c, for a constraint corresponding to u_i and bound α_i, if we set x_e to 1 or 0 depending on whether e is chosen or not, the constraint will be satisfied with the bound α_i replaced by a new bound $O((\log s/\log\log s)\alpha_i)$. Furthermore, if the x_e's are not chosen as mutually independent but pairwise independent, the same assertion holds but with a weaker new bound $O(\sqrt{s}\alpha_i)$. This weaker bound is improved to $O(s^{1/r}\alpha_i)$ when the x_e's are chosen r-wise independent for $r \geq 2$. We then set an x_e that had value 1 (i.e., chosen by the random process) to 0 if the new bound in each of these two cases is not satisfied. This happens with probability at most $1/(cs)$ per constraint for the u_i, and therefore with probability at most $1/c$ per x_e, since each x_e participates in at most s such constraints. Finally, after this modification of the x_e's, for a constraint corresponding to v_j and bound β_j, the constraint with the modified bound β_j/c' is satisfied with some constant probability at least $1/c''$ for some constants c' and c''. Therefore repeating this process an expected $O(c'c'')$ number of times leads to satisfying in one such trial a portion $\Omega(m/(c'c''))$ of $m = \sum_j \beta_j$. We then start the process over with m replaced by $m - \Omega(m/(c'c''))$ and corresponding adjustments of the β_j. Thus after $O(\log m)$ such rounds the bound m is satisfied. Adding all the rounds satisfies the constraints with for the u_i and bound α_i with a bound $O((\log s/\log\log s)\alpha_i\log m)$, or with the bound $O(s^{1/r}\alpha_i\log m)$ if only r-wise independence is used. Furthermore, r-wise independence for $r \geq 2$ only requires a polynomial size sample space that can be determined ahead of time, so in the latter case the algorithm can be de-randomized. It thus only remains to prove the probabilistic claims to complete the proof of this theorem.

Consider the constraint $\sum_{u_i \in S_e} a_{ei}x_e \leq \alpha_i$. If the x_e's are r-wise independent, then the probability that $x_{e_1},\ldots,x_{e_r}$ are all set to 1 is $x_{e_1}\cdots x_{e_r}$. Suppose some variables x_e are set to 1 for a set

E of hyperedges e with probability p, and we have $\sum_{u_i \in E} a_{ei} \geq t\alpha_i$ with $t \geq 2r^2$. This contributes $p\sum_{\{e_1,\ldots,e_r\}\subseteq E} a_{e_1 i}\cdots a_{e_r i} \geq p(\sum_{e\in E} a_{ei})^r/r! - \alpha_i r(\sum_{e\in E} a_{ei})^{r-1}/(r-1)! \geq p(t\alpha_i)^r(1-r^2/t)/r! \geq p(t\alpha_i)^r/(2r!)$ to the sum $\sum_{\{e_1,\ldots,e_r\}\subseteq S_e} a_{e_1 i}x_{e_1 i}\cdots a_{e_r i}x_{e_r} \leq (\sum_{e\in S_e} a_{ei}x_e)^r \leq \alpha_i^r/r!$. Adding over all such cases with corresponding probabilities p, we have $\sum p \leq \alpha_i^r/((t\alpha_i)^r/2) \leq 2/t^r$. Setting $t = s^{1/r}$, we thus have that with probability at least $1-1/(cs)$, $\sum_{u_i\in S_e} a_{ei}x_e \leq O(s^{1/r})\alpha_i$.

The stronger bound under the assumption of mutual independence is obtained via Chernoff bound. We simplify the constraint as follows $\sum_{u_i\in S_e} b_{ei}x_e \leq 1$, where $0 \leq b_{ei} = a_{ei}/\alpha_i \leq 1$. We thus have random variables Y_e, where $Y_e = b_{ei}$ with probability x_e, and 0 with probability $1-x_e$. We replace Y_e with random variables Z_e, where $Z_e = 1$ with probability $b_{ei}x_e$, and 0 with probability $1-b_{ei}x_e$. We have that for any convex function f, $E[f(Y_e)] \leq E[f(Z_e)]$. Define random variables $Y = \sum Y_e$ and $Z = \sum Z_e$. We know that $E[Y] = E[Z] = \mu \leq 1$. Thus, it suffices to obtain the tail bound on Y applying Chernoff bound on Z. In particular, we have that

$$P[Y > (1+\delta)\mu] < \left[\frac{e^\delta}{(1+\delta)^{1+\delta}}\right]^\mu.$$

Setting $\delta = O(\log s/(\mu\log\log s))$, we have $P[Y > O(\log s/\log\log s)] < 1/(cs)$.

Consider the constraint $\sum_{v_j\in T_e} b_{ej}x_e \geq \beta_j$. For each integer $p \geq 0$, define $q'_{ep} = \min(\max(b_{ei} - p, 0), 1)$, Thus $b_{ei} = \sum_p q'_{ep}$, with $q'_{ep} = 0$ or $q'_{ep} = 1$, and $0 \leq p \leq b_{ei} \leq \beta_j{}^2$. For each $0 \leq p \leq \beta_j$, consider the sum $r'_p = \sum_e q'_{ep}x_e$. Divide the summands into at most $2r'_p + 1$ groups with at most one group adding to less than $1/2$ and all the others between $1/2$ and 1. The total number of groups adding to at least $1/2$ is thus at least $\sum_p(r'_p - 1/2) \geq (\sum_p r'_p) - \beta_j/2 = (\sum_{p,e} q'_{ep}x_e) - \beta_j/2 = (\sum_e b_{ei}x_e) - \beta_j/2 \geq \beta_j/2$. If we consider one such group, with the assumption of pairwise independence, the probability to obtain at least one $x_e = 1$ is at least $\sum_e x_e - \sum_{\{e,e'\}} x_e x_{e'} \geq \sum_e x_e - (\sum_e x_e)^2/2 \geq (\sum_e x_e)/2 \geq 1/4$. The probability that this $x_e = 1$ that occurs with probability at least $1/4$ will be changed to $x_e = 0$ is at most $s(1/(cs)) \leq 1/c$ by the preceding argument, and therefore the probability that it will still be $x_e = 1$ is at least $1/4 - 1/c \geq 1/5$. The expected total number of groups containing at least one $x_e = 1$ out of $\beta_j/2$ groups is thus at least $(\beta_j/2)/5 \geq \beta_j/10$, while the probability that it will be at least $\beta_j/20$ is at least $1/10$, otherwise expected value $\beta_j/10$ cannot be obtained with maximum value $\beta_j/2$. Therefore a portion $\beta_j/20$ of the bound β_j is satisfied with probability at least $1/10$ under the assumption of pairwise independence, hence a constant fraction of $m = \sum_j \beta_j$ is satisfied at each round, for a total of $O(\log m)$ rounds.

The de-randomization of pairwise independent x_e is as follows. We may assume that each x_e is of the form $x_e = q_e/p$ for some prime $p = c''n$ for some sufficiently large constant c'', where n is the number of variables x_e, and $0 \leq q_e \leq p$ is integer, with only a small rounding effect on the preceding argument. We may then choose the sample space to be uniform on $\{(x,y) \in (\{0,\ldots,p-1\})^2\}$, and set $x_e = 1$ for $e = 1,\ldots,n$ if $x + ey + t = 0$ modulo p for some $0 \leq t < q_e$. Thus the probability that $x_e = 1$ is $(q_e p)/p^2 = q_e/p$, and the probability that $x_e = 1$ and $x_{e'} = 1$ is $(q_e q_{e'})/p^2 = (q_e/p)(q_{e'}/p)$, giving pairwise independence. Testing all p^2 possible elements in this sample space de-randomizes the algorithm. A similar de-randomization applies to r-wise independent random variables with $r \geq 2$ an integer constant. □

Similar to Theorem 2 and 3, we conclude the following.

[2] In this proof only we can assume that a_{ei} and α_i's are integers. Since originally $a_{ei}, \alpha_i \geq 1$, we can simply round them to the next integer while loosing only a constant factor in terms of approximation.

Theorem 6 *The MDSS problem can be approximated in polynomial time to achieve maximum degrees at most $O(\alpha_i \log(|V(G)|r))$ when there exists a solution with maximum degrees α_i for weights a_{ei}, giving an $O(\log(|V(G)|r))$ approximation factor.*

Proof. Follows from Theorems 5 as in Theorem 2. Except that we don't constantly update H when edges are selected. Thus, for a particular G_i, it could happen that all the cuts are covered, however, the resulting graph does not connect G_i. However, in the worst case, the added edges form a matching and reduce the number of cuts by at least a half. □

Theorem 7 *For any constant k, there is a polynomial time algorithm that finds a k-vertex (edge) connected spanning subgraph H of degrees $O(\alpha_i k \log n)$, where there exists a solution with degree bounds α_i for weights a_{ei}, giving an $O(k \log n)$ approximation factor.*

4 Large 2-Connected Subgraphs of Low Degree

In their study of the problem of finding large cycles in a graph, Feder and Motwani [2] obtained the intermediate result that if a graph G with n vertices has a cycle K with k vertices, then for all $r \geq 1$ we can find in G a subgraph containing a 2-connected component that has at least $k(1-1/r)$ of the vertices of K, with vertex degree at most $(rn/k)\log^c n$, for some constant $c \geq 1$ in polynomial time [2]. The proof generalizes to the following seven results, and in particular allow us to find large cycles in graphs with a large 3-cyclable minor of small degree. This extends the result of [2] that shows that if a graph has a cycle of length k, then for all $r \geq 1$ we can find in polynomial time an almost 3-cyclable minor with at least $k(1-1/r)$ vertices and with degrees at most $(rn/k)\log^c n$, for some constant $c \geq 1$, and thus a cycle of length $k^{\Omega(1/(\log(n/k)+\log\log n))}$.

Theorem 8 *If a connected graph G with n vertices and m edges has a subgraph that is a tree K with k vertices of vertex degree at most d, then for all $r \geq 1$ we can find in G a tree that has at least $k(1-1/r)$ of the vertices of K, with vertex degree at most $O((drn/k)\log^2 n)$, in $O(nm)$ time. This gives a polylogarithmic degree approximation if r and n/k are bounded by a polylogarithm.*

Proof. Let $t = r\log n$. We consider a series of $\log n$ phases $i = 0, 1, \ldots, \log n$ during which we select edges to form a subgraph H. In phase i, we consider components of H that have between 2^i and 2^{i+1} vertices. By the end of the phase, these components will be combined into components with at least 2^{i+1} vertices, or not considered as part of H. The number of vertices that are thus removed from H but belong to K in each phase will be at most k/t, and the degree increase in each phase will be at most dtn/k. Thus over all $\log n$ phases we have removed from H a total of $(k/t)\log n = k/r$ vertices that belong to K, and the maximum degree is at most $(dtn/k)\log n = (drn/k)\log^2 n$, as required in Theorem 8.

Consider H as selected at the beginning of phase i. The components that have between 2^i and 2^{i+1} can be joined in pairs by a maximal collection of disjoint paths that do not go through H and are added to H. Eventually, we are left with such components that can not be combined in pairs by paths not going through H. Suppose there are at least k/t vertices of K that belong to such components. Then the number of such components that contain vertices of K is at least $s = \lceil k/(t2^{i+1})\rceil$. These components can be joined together by the tree K of maximum degree d. These s components have s representative vertices in K, that can be joined in pairs as disjoint paths, so that every time a path is chosen, joining two consecutive leaves that are among these s vertices with common ancestor v, at most $d-1$ other leaf descendants from v that are among the s vertices are discarded. Thus at least $2s/d$ of these s vertices can be paired by disjoint paths. These disjoint paths necessarily go through other components of H that have at least 2^{i+1} vertices, so we can set a flow problem where the sources are the components having between 2^i and 2^{i+1} vertices and the sinks are the components having at least 2^{i+1} vertices, thus obtaining at least $\lceil s/d \rceil \geq \lceil k/(td2^i) \rceil$ paths. Since the total number of such components to be joined is at most $n/2^i$, the computation of disjoint paths by flow happens at most $(n/2^i)/(k/(td2^i)) = dtn/k$ times, for an increase of degree at most dtn/k during a phase. When such a flow can no longer be found, the phase is complete, which happens only when fewer than k/t vertices of K belong to such components that have between 2^i and 2^{i+1} vertices. The time complexity is dominated by pushing $O(n)$ units of flow in $O(m)$ time each, for a total $O(nm)$ time. □

Theorem 9 *If a 2-edge-connected graph G with n vertices and m edges has a 2-edge-connected subgraph K with k vertices of vertex degree at most d, then for all $r \geq 1$ we can find in G a 2-edge-connected subgraph that contains at least $k(1-1/r)$ of the vertices of K, while having vertex degree at most $O((drn/k)\log^4 n)$, in $O(nm)$ time. This gives a polylogarithmic degree approximation if r and n/k are bounded by a polylogarithm.*

Proof. Let $t = r\log^2 n$. We initially find the tree of degree $O((drn/k)\log^2 n)$ from Theorem 8. We shall gradually add paths to this tree H until H becomes 2-edge-connected. At any stage, H consists of a collection of disjoint 2-edge-connected components joined by single edges in a tree structure. The degree of a component is the number of these edges coming out of the component. Consider removing all paths joining leaf components to components of degree at least 3, going through components of degree 2. If we repeatedly remove these paths, the number of paths going through components of degree 2 in H is halved each time, so we may remove such hanging paths at most $\log n$ times. This sets up $\log n$ phases, where each phase considers the hanging paths of components.

A phase that deals with hanging paths of components first begins by consider each such path, one at a time. It starts with the leaf component and finds the path not going through H that connects to the latest component on the path of components starting at this leaf, thus forming a single 2-edge connected component that constitutes a new leaf, and the operation is performed again. Since vertices in the earlier leaf are not used to start a new path, this increases degrees by at most 2. When the leaf component can no longer be connected, we proceed to the parent of the leaf component as we had with the leaf. In the end, we have reduced the number of components on the path as much as possible by paths not going through H, and increased the degree by at most 2.

The hanging paths of components are then taken care in $\log n$ sub-phases by considering paths of components with between $n/2^{i+1}$ and $n/2^i$ vertices in phases $0 \leq i < \log n$. For each such path of components, we consider the components containing the bottom half of vertices closer to the leaf. As in Theorem 8, we first connect these to one another by disjoint paths not going through H, and when this is no longer possible, find disjoint paths for the remaining ones joining them to the rest of the graph. If at least k/t vertices of K are in these bottom halfs, then at least $s = \lceil (k/t)/(n/2^i) \rceil$ such bottom halfs are involved, and as in Theorem 8, there are at least s/d such disjoint paths within K. A flow operation finds s/d such paths as before, and since the number of such bottom halfs is at most 2^{i+1}, the number of flow operations in each sub-phase is at most $2^{i+1}d/s = 2dtn/k$, for degree increase at most $2dtn/k$. After each such sub-phase, either a bottom half was joined in the sub-phase, so the remaining hanging path of components has at most half as many vertices, or the bottom half is ignored and we consider the half as many vertices of the top half for the next sub-phase. Since each sub-phase increases degree by at most $2dtn/k$, and the total number of sub-phases is $\log^2 n$ because there are $\log n$ sub-phases within $\log n$ phases, the total degree increase is at most $(2dtn/k)\log^2 n = O((drn/k)\log^4 n)$. Each sub-phase leaves out k/t vertices of K, for a total of $(k/t)\log^2 n = k/r$ vertices of K left out. The time complexity is dominated by pushing $O(n)$ units of flow in $O(m)$ time each, for a total $O(nm)$ time. □

Theorem 10 *If a 2-vertex-connected graph G with n vertices and m edges has a 2-vertex-connected subgraph K with k vertices of vertex degree at most d, then for all $r \geq 1$ we can find in G a 2-vertex-connected subgraph that has at least $k(1 - 1/r)$ of the vertices of K, with vertex degree at most $O((drn/k)\log^6 n)$, in $O(nm)$ time. This gives a polylogarithmic degree approximation if r and n/k are bounded by a polylogarithm.*

Proof. Let $t = r\log^3 n$. We proceed as in Theorem 9, with the difference that the tree of 2-connected components is viewed as a tree with vertices corresponding both to the actual components and the vertices joining components, as vertices instead of edges join 2-connected components. We define phases and sub-phases in an analogous way. The algorithm only differs when examining hanging paths of components in order to join the bottom halves of these paths to the rest of the graph by a maximum flow defining disjoint paths. Here the separating vertices joining the bottom half to the top half, and joining the top half to the rest of the graph, must not be used in joining the bottom half, but may be used by the bottom half of other paths, so we may not use a single flow computation. Instead, we divide the at most 2^i bottom halfs into groups of size 2^j with $1 \leq j \leq i$, and attempt in sub-sub-phase j flows joining the bottom of one part of size 2^{j-1} to the top of the other part of size 2^{j-1} in successive flows giving disjoint paths. Each sub-sub-phase may not join in flows at most of $(k/t)/(dn/2^i)$ bottom halfs, or $(k/t)/(dn/2^j)$ bottom halfs per group. Since the number of bottom halves per group is 2^j, the degrees increase by at most dtn/k in each sub-sub-phase, or $(dtn/k)\log^3 n = O((drn/k)\log^6 n)$ over all $\log^3 n$ sub-sub-phases. The k/t vertices of K that may not be joined in a sub-sub-phase give a total of $(k/t)\log^3 n = k/r$ vertices over all $\log^3 n$ sub-sub-phases. The time complexity is dominated by pushing $O(n)$ units of flow in $O(m)$ time each, for a total $O(nm)$ time. □

Say that a 2-edge-connected graph H *represents* a 2-edge-connected graph K if K is obtained from H by repeatedly replacing some vertices x of degree two and their incident edges $(x, y), (x, z)$ with a single edge (y, z).

Theorem 11 *If a graph G with n vertices and m edges has a 2-edge-connected subgraph H that represents a 3-edge-connected graph K with k vertices of vertex degree at most d, then for all $r \geq 1$ we can find in G a subgraph H' representing a 3-edge-connected graph K' with least $k(1 - 1/r)$ vertices and with vertex degree at most $O((drn/k)\log^4 n)$, in $O(nm)$ time. This gives a polylogarithmic degree approximation if r and n/k are bounded by a polylogarithm.*

Proof. We initially find the 2-edge-connected subgraph K' of degree $O((drn/k)\log^4 n)$ from Theorem 9. We then proceed as in Theorem 9 to find the subgraph H representing a 3-edge-connected subgraph K. The proof finds pairs of edges separating K' organized in a tree structure. The root corresponds to any such pair of edges (e, f) separating K' into two components K'_1 and K'_2. In general, given a component K'_i connected to the rest of the graph by a pair of edges (e_i, f_i), we identify maximal components K'_j contained in K'_i and connected to the rest of the graph by a pair of edges (e_j, f_j), and select a maximal collection of disjoint such K'_j as children of K'_i. This gives to all such K'_j a tree structure T, and we proceed as in Theorem 9 by considering $\log n$ phases of hanging paths of T, where the last phase reaches the root, and subdivide each phase by $\log n$ sub-phases. The only difference with the proof of Theorem 9 is that each chosen path leaving K_j goes to the other side of the pair of edges (e_j, f_j) instead of going to the other side of a single edge e_j. The proof is thus completed as in Theorem 9. When a component K'_j remains separated by a pair of edges (e_j, f_j), the component K'_j is removed and replaced with a single edge combining e_j and f_j in K'; the paths used to join different components are taken as individual edges in K'. The time complexity is dominated by pushing $O(n)$ units of flow in $O(m)$ time each, for a total $O(nm)$ time. □

Theorem 12 *If a graph G with n vertices and m edges has a 2-vertex-connected subgraph H that represents a 3-vertex-connected graph K with k vertices of vertex degree at most d, then for all $r \geq 1$ we can find in G a subgraph H' representing a 3-vertex-connected graph K' with at least $k(1-1/r)$ vertices and with vertex degree at most $O((drn/k)\log^6 n)$, in $O(nm)$ time. This gives a polylogarithmic degree approximation if r and n/k are bounded by a polylogarithm.*

Proof. We initially find the 2-vertex-connected subgraph K' of degree $O((drn/k)\log^6 n)$ from Theorem 10. We then proceed as in Theorem 10 to find the subgraph H representing a 3-vertex-connected subgraph K. The proof finds pairs of vertices separating K' organized in a tree structure. The root corresponds to any such pair of vertices (u, v) separating K' into components K'_i In general, given a component K'_i connected to the rest of the graph by a pair of vertices (u_i, v_i), we identify maximal components K'_j contained in K'_i and connected to the rest of the graph by a pair of vertices (u_j, v_j), and select a maximal collection of disjoint such K'_j as children of K'_i. This gives to all such K'_j a tree structure T, and we proceed as in Theorem 10 by considering $\log n$ phases of hanging paths of T, where the last phase reaches the root, subdivide each phase by $\log n$ sub-phases, and further subdivide each sub-phase into $\log\log n$ sub-sub-phases. The only difference with the proof of Theorem 10 is that each chosen path leaving K_j goes to the other side of the pair of vertices (u_j, v_j) instead of going to the other side of a single vertex u_j. The proof is thus completed as in Theorem 10. When a component K'_j remains separated by a pair of vertices (u_j, v_j), the component K'_j is removed and replaced with a single edge joining u_j and v_j in K'; the paths used to join different components are taken as individual edges in K'. The time complexity is dominated by pushing $O(n)$ units of flow in $O(m)$ time each, for a total $O(nm)$ time. □

A graph K is *3-cyclable* if for every triple u, v, w of vertices in K, the graph K has a cycle going through u, v, w. Note that a cycle K in G is a 3-cyclable subgraph K of G of maximum degree 2.

Feder and Motwani [2] considered a weaker notion. Let G be a 2-connected graph. If G has two vertices u, v such that $G - \{u, v\}$ is not connected and has connected components R_i, then we may decompose G into graphs G_i with vertices $V(R_i) \cup \{u, v\}$ and the edges of R_i, the edges joining the vertices of R_i to u and v, plus the edge (u, v). We may then further decompose the graphs G_i similarly, thus obtaining a tree decomposition of G into graphs G_i such that each G_i is either (1) 3-connected, (2) a cycle, or (3) a multigraph consisting of two vertices u, v joined by multiple parallel edges. In this tree decomposition, the root graph G_0 has children G_i corresponding to edges uv in G_0, and similarly for the children G_i, until the leaf graphs G_j are reached. See Hopcroft and Tarjan [9] for an algorithm to obtain such a decomposition.

We say that G is *almost 3-cyclable* if for every graph G_j in this tree decomposition other than the root graph G_0, if the edge uv in G_i corresponds to the parent G_i of G_j, then (1) there is at most one edge incident to u in G_j other than uv that corresponds to a child graph $G_{j'}$ of G_j; (2) there is at most one edge incident to v in G_j other than uv that corresponds to a child graph $G_{j'}$

of G_j; (3) if three edges uv, e, f separate G_j into two parts, then at most one of u, v corresponds to a child graph $G_{j'}$ of G_j. Clearly if one of (1),(2),(3) is not satisfied for some G_j, then G is not 3-cyclable; thus 3-cyclable graphs are almost 3-cyclable graphs as well.

Theorem 13 *If a graph G with n vertices and m edges has a 2-vertex-connected subgraph H that represents a 3-cyclable graph K with k vertices of vertex degree at most d, then for all $r \geq 1$ we can find in G a subgraph H' representing an almost 3-cyclable graph K' with at least $k(1-1/r)$ vertices and with vertex degree at most $O((drn/k)\log^6 n)$, in $O(nm)$ time. This gives a polylogarithmic degree approximation if r and n/k are bounded by a polylogarithm.*

Proof. We proceed as in Theorem 12 by fist finding a 2-vertex-connected subgraph as in Theorem 10, and then proceeding to $\log n$ phases for the hanging paths of the tree decomposition, with each phase grouped into $\log n$ sub-phases and each sub-phase grouped into $\log n$ sub-sub-phases. Again, as in Theorem 12, we attempt in each sub-sub-phase to link the bottom half of the hanging paths of G_j so that the G_j in the top halfs are not separated by just 2 vertices. However, we measure the number of vertices to be obtained for a solution from such a G_j not by the total number of vertices in G_j, but by the number of vertices that would have to be removed to satisfy conditions (1),(2),(3) if G_j remained only 2-connected, that is, if we counted only the vertices corresponding to only one edge incident to u other than uv, counted only the vertices corresponding to only one edges incident to v other than uv, and for every pair of edges e, f such that uv, e, f separate G_j counted only the vertices for one of e, f.

Before considering connecting the bottom halfs of paths among one another and to the rest of the graphs, we process each path individually as in Theorem 12, by going up the path from the leaves G_j, joining G_j together as much as possible. In addition, to attempting to join G_j as close as possible to the leaf as possible to an ancestor on the path as in Theorem 12, we consider taking care of each G_j encountered by going up the path individually, in order to partially satisfy conditions (1),(2),(3) within G_j. For condition (3), the pairs of edges e_i, f_i that together with uv separate G_j into two parts decompose G_j into subgraphs $H_1, \ldots, H_s$ such that uv joins H_1 to H_s, and e_i, f_i join H_i to H_{i+1}, and each e_i, f_i corresponds to a path of edges in a child of G_j. We may thus consider the sequence $H_1, (e_1, f_1), H_2, (e_2, f_2), H_3, \ldots, H_{s-1}, (e_{s-1}, f_{s-1}), H_s$, and proceed as follows. First join H_1 to the last item in this sequence by a path, and if this last item is (e_i, f_i) then join H_1 to the last possible edge on the two paths for e_i and for f_i. Then proceed similarly starting anywhere up to this last item, which is now part of H_1. If it is not possible to proceed from H_1, we proceed through the edges in the paths for e_1, f_1 by connecting them to the last possible edge similarly. Each time we add intermediate vertices of maximum degree 3, the degree of the vertices from which the paths are started increases by 2, and the degree of the vertices where the paths are ended increases by 1, for a total increase of 3 in the degrees. For the remaining (e_i, f_i) that cannot be taken care of in this way, we will have to remove the vertices corresponding to e_i or f_i, whichever has the least number of vertices. This takes care of condition (3).

For conditions (1) or (2) of G_j, say condition (1), we consider each descendant $G_{j'}$ of G_j attached at the vertex u from the edge uv in G_j corresponding to the parent of G_j. In each $G_{j'}$, a path starting at u with edges $e_1, \ldots, e_a$ is subdivided into new graphs G_i corresponding to paths $e_1, \ldots, e_i$ for $1 \leq i \leq a$. Thus each edge e other than uv incident to u in G_j corresponds to a path of graphs $G_{e1}, \ldots, G_{ea}$ from the leaves going up to G_j, and we proceed from the leaf G_{e1} to connect to the highest G_{ei} and so on up the path, as from before, to reduce the number of G_{ei} on the path corresponding to e. Then we proceed to join the paths of corresponding to different e to one another and to the rest of G_j, by considering the bottom halfs of such paths, again by counting the number of vertices that might be included from each G_{ei} by joining it to the rest of G_j. If G_j has n_j vertices not in the child G_j, then instead of attempting to join at least $(k/t)/(dn/2^i)$ bottom halves per flow as before in sub-sub-phases, we consider joining at least $(n_j/n)(k/t)/(dn_j/2^i)$ bottom halves per flow, or $(n_j/n)(k/t)/d$ vertices, so the degree again goes up by at most $2dtn/k$, and the number of vertices not joined is at most $\sum_j (n_j/n)(k/t) \leq k/t$ over all G_j. Here this is incurred in $\log n$ sub-phases corresponding to the different values of $1 \leq 2^i \leq n_j$ for each G_j. This completes the cases for conditions (1),(2) and (3) and the construction of the amost 3-cyclable subgraph. The time complexity is dominated by pushing $O(n)$ units of flow in $O(m)$ time each, for a total $O(nm)$ time. □

Theorem 14 *If a graph G with n vertices has a 2-vertex-connected subgraph H that represents a 3-cyclable graph K with k vertices of vertex degree at most d, then for we can find in $O(n^3)$ time a cycle in G of length at least $k^{1/(c(\log d + \log(n/k) + \log\log n))}$ for some constant $c > 0$.*

Proof. Feder and Motwani [2] showed that if G of maximum degree d has an almost 3-cyclable minor with n vertices, then we can find in $O(n^3)$ time a cycle in G of length at least $n^{1/(c \log d)}$ for some constant $c > 0$. Applying this result to the almost 3-cyclable graph K' with at least $n' \geq k/2$ vertices and maximum degree $d' \leq d(n/k) \log^{c'} n$ from Theorem 13 yields the corresponding $k^{1/(c \log d')}$ bound on the length of the cycle found in K' and thus in G. □

References

[1] J. Cheriyan and R. Thurimella, "Approximating minimum-size k-connected spanning subgraphs via matching." *SIAM J. Comput 30-2* (2000) pp. 528–560.

[2] T. Feder and R. Motwani, "Finging Large Cycles in Hamiltonian Graphs." In *Proceedings of the ACM-SIAN Symposium on Discrete Algorithms* (2005), to appear.

[3] M. Fürer and B. Raghavachari, "An NC approximation algorithm for the minimum degree spanning tree problem." In *Proceedings of the 28th Annual Allerton Conference on Communication, Control and Computing* (1990) pp. 274–281.

[4] M. Fürer and B. Raghavachari, "Approximating the minimum degree spanning tree to within one from the optimal degree." In *Proceeding of the 3rd ACM-SIAM Symposium on Discrete Algorithms* (1992) pp. 317-324.

[5] H.N. Gabow, M.X. Goemans, E. Tardos, and D.P. Williamson, "Approximating the smallest k-edge connected spanning subgraph by LP-rounding." In *Proceedings of the 1th ACM-SIAM Symposium on Discrete Algorithms* (2005) pp. 562-581.

[6] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness. W.H. Freeman, 1979.

[7] P. Gubbala and B. Raghavachari, "Finding k-connected subgraphs with minimum average weight." In *Proceedings Latin American Theoretical Informatics* (LATIN 2004) pp. 212–221.

[8] D.S. Hochbaum, Approximation Algorithms for NP-Hard Problems. PWS Publishing Company, Boston, 1995.

[9] J.E. Hopcroft and R.E. Tarjan. "Dividing a graph into triconnected components." *SIAM Journal on Computing* 2–3 (1973), pp. 135–158.

[10] D.S. Johnson, "Approximation algorithms for combinatorial problems." *J. Comput. System Sci.* 9 (1974) pp. 256–278.

[11] P.N. Klein, R. Krishnan, B. Raghavachari, and R. Ravi, "Approximation algorithms for finding low-degree subgraphs." *Networks* (to appear).

[12] R. Ravi, B. Raghavachari, and P. Klein, "Approximation through local optimality: designing networks with small degree." In *Proceedings of the 12th Conference on Foundations of Software Tech. and Theoret. Comp. Sci., Lect. Notes in Comp. Sci. 652* (1992) pp. 279–290.

[13] R. Raz and S. Safra, "A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP." In *Proc. 29th Ann. ACM Symp. on Theory of Comp.* (1997) pp. 475–484.