Efficient Algorithms for Clique Problems[☆]

Virginia Vassilevska

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA

Abstract

The k-clique problem is a cornerstone of NP-completeness and parameterized complexity. When k is a fixed constant, the asymptotically fastest known algorithm for finding a k-clique in an n-node graph runs in $O(n^{.792k})$ time (given by Nešetřil and Poljak). However, this algorithm is infamously inapplicable, as it relies on Coppersmith and Winograd's fast matrix multiplication.

We present good *combinatorial* algorithms for solving k-clique problems. These algorithms do not require large constants in their runtime, they can be readily implemented in any reasonable random access model, and are very space-efficient compared to their algebraic counterparts. Our results are the following:

- We give an algorithm for k-clique that runs in $O(n^k/(\varepsilon \log n)^{k-1})$ time and $O(n^\varepsilon)$ space, for all $\varepsilon > 0$, on graphs with n nodes. This is the first algorithm to take $o(n^k)$ time and $O(n^c)$ space for c independent of k.
- Let k be even. Define a *k-semiclique* to be a k-node graph G that can be divided into two disjoint subgraphs $U = \{u_1, \ldots, u_{k/2}\}$ and $V = \{v_1, \ldots, v_{k/2}\}$ such that U and V are cliques, and for all $i \leq j$, the graph G contains the edge $\{u_i, v_j\}$. We give an $\tilde{O}(k2^k n^{k/2+1})$ time algorithm for determining if a graph has a k-semiclique. This yields an approximation algorithm for k-clique, in the following sense: if a given graph contains a k-clique, then our algorithm returns a subgraph with at least 3/4 of the edges in a k-clique.

Key words: algorithms, combinatorial problems, graph algorithms, clique

[☆] This research was sponsored by the National Science Foundation under contracts no. CCR-0122581, no. CCR-0313148, and no. IIS-0121641. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Email addresses: `virgi@cs.cmu.edu` (Virginia Vassilevska)

URL: `http://www.cs.cmu.edu/~virgi` (Virginia Vassilevska)

1. Introduction

The k-clique problem is one of the fundamental problems in computer science. In parametrized complexity k-clique plays a central role: one of the major problems in this area is to resolve whether $W[1] = FPT$, and k-clique is $W[1]$-complete [9]. Furthermore, results by Chen *et al.* [6] imply that if there is an $n^{o(k)}$ algorithm for k-clique [1], then many of the classical NP-complete problems are solvable in subexponential time.

The naive algorithm for k-clique runs in $O(n^k)$ time by examining all k-tuples of vertices. In 1978, Itai and Rodeh [12] showed that a 3-clique (triangle) in an n-node graph can be found in the same time as $n \times n$ Boolean matrix multiplication. Nowadays this means that there is an $O(n^{2.376})$ algorithm [8] for 3-clique; we denote this exponent by ω. Nešetřil and Poljak [18] showed how to generalize Itai and Rodeh's [12] reduction to obtain an $O(n^{\omega k})$ time algorithm for $3k$-clique[2]. Since that result very little progress has been made. In 2004 Eisenbrand and Grandoni [10] used rectangular matrix multiplication to obtain improved running times for some values of k, *e.g.* for $k \geq 5$ such that $k = 2 \mod 3$.

Although these algebraic matrix product approaches yield good theoretical time bounds, the known fast matrix multiplication algorithms that run in $O(n^{3-\varepsilon})$ time for $\varepsilon > 0$ ([21, 8]) are very inefficient in practice. Another issue is that all of the above mentioned algorithms for k-clique use $n^{\Omega(k)}$ space. In 1984 ([16], p.46), László Babai asked whether there exists an $o(n^k)$ algorithm for k-clique using $O(n^c)$ space for some constant c independent of k. This question has until now remained unanswered. In fact, the problem of designing better time and space efficient k-clique algorithms has been recently reraised by Woeginger [22].

Combinatorial, nonalgebraic algorithms based on preprocessing and table look-up offer less impressive asymptotic improvements but are much more practical than algorithms based on algebraic fast matrix multiplication. Using a "Four Russians" type approach for combinatorial matrix multiplication [3, 20, 4, 5] one can obtain an $O(n^k/(k \log^2 n))$ time algorithm for k-clique which hides no enormous constant factors in its runtime. Nevertheless, even this algorithm is space-inefficient – it requires $\Omega(n^{2k/3})$ space.

In this paper we answer Babai's question in the affirmative. We give a combinatorial algorithm for k-clique which runs in $O(n^k/(\varepsilon \log n)^{k-1})$ time for every fixed $k \geq 3$ and uses $O(n^\varepsilon)$ space, for any constant $\varepsilon > 0$ independent of k. This algorithm is not only space-efficient but also: (a) it beats the runtimes of the current best combinatorial algorithms for finding k-clique for $k > 3$, and (b) it implies an algorithm for maximum node-weighted k-clique within *the same* runtime.

[1] Here, k is assumed to be any unbounded function of n.

[2] Interestingly, there are $O(2^{\delta n})$ algorithms ([11, 19]) for finding a *maximum* clique for small δ even though the maximum clique can be really large, *e.g.* $\Omega(n)$.

In the second part of the paper we investigate the problem of finding so called k-semicliques in a graph. We define a k-semiclique to be a graph on k nodes, for even k, composed of two disjoint $\frac{k}{2}$-cliques $U = \{u_1, \ldots, u_{k/2}\}$ and $V = \{v_1, \ldots, v_{k/2}\}$ so that in addition to the clique edges there is an edge (u_i, v_j) for all $j \geq i$. Semicliques are generalizations of the so called diamond graphs, 4-cliques missing one edge. Possible applications of semicliques can be found in databases or privacy where one wishes to find two cliques (perhaps representing crime organizations) which are very connected to one another.

We give an algorithm for finding a (not necessarily induced) k-semiclique in a given graph in $O(n^{1+k/2})$ time. The case $k = 4$ (diamonds) has been studied before. It is known (e.g. [13, 10]) that one can find an induced diamond in $O(n^3)$ time. Our result can be seen as an extension of this running time for all even k for not necessarily induced subgraphs. Our result also yields an $O(n^{1+k/2})$ time approximation algorithm for k-clique, in the following sense: if the graph contains a k-clique, then our algorithm returns a subgraph with at least $2\binom{k/2}{2} + \binom{k/2+1}{2} = 3k^2/8 - k/4$ edges, *i.e.* at least $3/4$ of the edges in a k-clique.

Preliminaries. All graphs we consider are undirected, simple, and connected, unless otherwise noted. We let m denote the number of edges and n the number of vertices. We denote the degree of a node x in a graph by $deg(x)$. For a graph $G = (V, E)$ and $u, v \in V$, $E(u, v)$ is defined to be 1 when $(u, v) \in E$ and 0 otherwise. All logarithms are base 2. For a positive integer n, we use $[n]$ to mean $\{1, \ldots, n\}$. For a time function $f(n)$, $\tilde{O}(f(n))$ denotes $O(f(n)\text{polylog } n)$.

2. Algorithm for k-Clique

We begin with a small-space combinatorial algorithm which given a node-weighted graph finds a k-clique of maximum weight sum. The basic idea used in the algorithm is to reduce the problem to finding maximum node-weighted $(k-1)$-cliques in many small $O(\log n)$ size subgraphs, and then attempt to complete these cliques by adding an extra node. Our algorithm proceeds in iterations so that each iteration reuses the space used by the previous ones.

Theorem 2.1 *Let $\varepsilon > 0$ and $k \geq 3$. Let $g(k) = (2(k-1))^{k-1}$. Let $G = (V, E)$ be a graph with arbitrary real weights on its nodes. There is an algorithm for maximum node-weighted k-clique that runs in $O\left(g(k) \cdot \frac{n^k}{(\varepsilon \log n)^{k-1}}\right)$ time and uses $O(kn^\varepsilon)$ space.*

Proof. Let $G = (V, E)$, $\varepsilon > 0$ and k be given. Let $\varepsilon' = \varepsilon/(2(k-1))$. We begin with an algorithm which finds a k-clique in an unweighted graph and then explain how to modify this algorithm so it finds a maximum node-weighted clique in a node-weighted graph.

Partition the nodes into $n/(\varepsilon' \log n)$ parts of $\varepsilon' \log n$ nodes each. The partition is chosen by grouping consecutive nodes according to the order of the columns in the adjacency matrix. This ensures that any k chunks of a row corresponding to k of the $(\varepsilon' \log n)$-size parts can be concatenated in $O(k)$ time, assuming constant time look-ups and an $O(\log n)$-word RAM.[3]

We will process all $(k-1)$-tuples of parts as follows. Fix a $(k-1)$-tuple of parts. Obtain the union U of these parts, and create a look-up table T_U with $(\varepsilon'(k-1)\log n)$-bit keys. For all $2^{\varepsilon'(k-1)\log n}$ subsets of U determine whether the corresponding induced subgraph of G contains a $(k-1)$ clique. Store the result for each subset (a $(k-1)$-clique, if found) in T_U, with the binary $(\varepsilon'(k-1)\log n)$-length vector representing the subset as the key. When a new union is processed, the look-up table information is overwritten.

Now, for every node $v \in V$, concatenate the portions of the neighborhood vector for v corresponding to U in $O(k)$ time to obtain a $(\varepsilon'(k-1)\log n)$-bit key. Look up in T_U using this key whether the induced neighborhood of v in U contains a $(k-1)$-clique. If this is so, return the union of the clique and v. Creating all tables T_U takes

$$O((n/(\varepsilon' \log n))^{k-1} \cdot n^{\varepsilon'(k-1)} \cdot (\varepsilon'(k-1)\log n)^{k-1}) = O(n^{(1+\varepsilon')(k-1)} \cdot (k-1)^{k-1})$$

time, and $O(k \log n 2^{\varepsilon'(k-1)\log n})$ space over all. The entire procedure (assuming $O(1)$ time look-ups in T_U) takes $O(n \cdot (n/(\varepsilon' \log n))^{k-1}))$ time, *i.e.* $O(n^{(1+\varepsilon')(k-1)} \cdot (k-1)^{k-1} + n^k/(\varepsilon' \log n)^{k-1})$ time overall. Since we set $\varepsilon' = \varepsilon/(2(k-1))$, the runtime becomes $O((2(k-1))^{k-1} \cdot n^k/(\varepsilon \log n)^{k-1})$. The space usage becomes asymptotically

$$k \log n 2^{\varepsilon'(k-1)\log n} = kn^{\varepsilon/2} \log n = O(kn^{\varepsilon}).$$

To modify the algorithm so that a maximum weight clique can be found, for every $(\varepsilon'(k-1)\log n)$-node subgraph of a union of $(k-1)$-tuples of node partitions, compute the maximum weight $(k-1)$ clique and store that clique in the look-up table for the union. Since the algorithm goes over all possible choices for a k-th node, the maximum weight clique can be returned. $\square$

We can use the algorithm of Theorem 2.1 to obtain an algorithm with a running time depending on the number of edges in the graph. We use an idea used in many other results (*e.g.* [2]) – either the clique has all high degree vertices, or it has a low degree vertex. In both cases we can reduce the problem to finding a clique in a strictly smaller subgraph.

Theorem 2.2 *Let $k \geq 5$ and $1 > \varepsilon > 0$. There is a function $\gamma(k)$ only depending on k and an algorithm for k-clique that runs in $O(km^{\varepsilon/2})$ space and $O\left(\gamma(k) \cdot \frac{m^{\frac{k}{2}}}{(\varepsilon \log m)^{(k-2-\frac{1}{k-1})}}\right)$ time on graphs with $m \geq 2$ edges.*

Proof. Let D be a parameter. Let $g(k) = (2(k-1))^{k-1}$ as in Theorem 2.1. Consider the set S of all nodes of degree at least $2D$. There are at most m/D

[3]The same can be accomplished on a pointer machine using some tricks.

such nodes. Use the algorithm of Theorem 2.1 to find a k-clique contained in S if one exists in $O(g(k)(m/D)^k/(\varepsilon\log(m/D))^{k-1})$ time. If no k-clique is found in S, any k-clique in the graph must contain a node of degree $< 2D$. Go through all edges (u,v) incident to low degree nodes, and look for a $(k-2)$-clique in the intersection of the neighborhoods of u and v, again using Theorem 2.1. This takes time $O(mg(k-2)(2D)^{k-2}/(\varepsilon\log 2D)^{k-3})$.

To minimize the final running time of the clique algorithm, as k is fixed, we can set

$$m(2D)^{k-2}/(\varepsilon\log 2D)^{k-3} = \Theta((m/D)^k/(\varepsilon\log(m/D))^{k-1}),$$

and hence $D = \sqrt{m}/(\varepsilon\log m)^{\frac{1}{k-1}}$ suffices. The runtime of the first part of the procedure becomes

$$O(g(k)2^{k-1}m^{k/2}/(\varepsilon\log m)^{(k-2-\frac{1}{k-1})}).$$

For $k \geq 5$, $m \geq 2$ and $\varepsilon < 1$ the runtime of the second part becomes

$$O\left(\frac{g(k-2)2^{k-2}m^{k/2}}{\varepsilon^{k-2-\frac{1}{k-1}}(\log^{1-\frac{1}{k-1}} m)(\log\frac{\sqrt{m}}{(\varepsilon\log m)^{\frac{1}{k-1}}})^{k-3}}\right) \leq O\left(\frac{(g(k-2)2^{3k-8})\cdot m^{k/2}}{(\varepsilon\log m)^{k-2-\frac{1}{k-1}}}\right).$$

If we set $\gamma(k) = g(k-2)2^{3k-8} + g(k)2^{k-1}$, we obtain the claimed runtime $O(\gamma(k)m^{k/2}/(\varepsilon\log m)^{(k-2-\frac{1}{k-1})})$. The space usage is $O(k(m/D)^{\varepsilon})$ in the first part of the procedure, and $O((k-2)D^{\varepsilon})$ in the second part. The space usage in the first part dominates and is $O(km^{\varepsilon/2})$. □

3. Finding k-semicliques

In this section, we present a combinatorial algorithm for detecting a k-semiclique in an arbitrary graph that runs in $\tilde{O}(n^{k/2+1})$ time for any fixed k. Our algorithm is inspired by Yates' algorithm ([15], pp.501–502) for computing the Fourier coefficients of an n-variable function in $O(n2^n)$ time. The basic idea of our algorithm is to enumerate all $k/2$-cliques, and try to find a disjoint pair of such cliques that have the appropriate edges between them. To check the edges, we can use a dynamic programming recurrence that slowly "replaces" a vertex from one $k/2$ clique with a vertex from the other $k/2$ clique, checking that the edge relations hold at each step. However, we also need to ensure that our algorithm is actually checking two *disjoint* $k/2$-cliques; since we are replacing some vertices with others, we could lose track of what nodes participate in the two $k/2$ cliques. To do this, we use a randomization trick.

Let $G = (V,E)$ be a graph, and let $n = |V|$. We begin by performing a step that is reminiscent of Alon, Yuster, and Zwick's color-coding [1] and its younger relative, "divide-and-color" [14, 7]. Choose a permutation $\pi : V \to [n]$, uniformly at random. Let U, W be two $\frac{k}{2}$-cliques in G such that $U \cup W$ is a k-semiclique. A permutation of V is *good* if for every $u \in U$ and $w \in W$, $\pi(u) < \pi(w)$.

Claim 1 *A random permutation is* good *with* $1/\binom{k}{k/2}$ *probability.*

Proof. The number of good permutations is $\binom{n}{k} \cdot (k/2)! \cdot (k/2)! \cdot (n-k)!$, as a good permutation can be selected by choosing a set S of k integers that U and V will be mapped to, choosing a permutation on the $\frac{k}{2}$ smallest integers in S and a permutation on the $\frac{k}{2}$ largest integers in S, then choosing an arbitrary permutation on the rest. Hence, the probability is $(k/2)!(k/2)!/k! = 1/\binom{k}{k/2}$. □

From here on, we presume that we have found a good permutation π. By repeating the procedure $O(2^k)$ times, this presumption is true with high probability. The permutation choice can be easily derandomized by using (n,k)-universal sets to pick the most significant bit of each node label. Naor *et al.* [17] show that in linear time one can construct a (n,k)-universal collection of size $2^k k^{O(\log k)} \log n$, and hence we would incur only a logarithmic overhead in our running time.

Refer to each vertex v in the graph by its index $\pi(v) \in [n]$. Build a table K of $n^{k/2}$ bits, indexed by tuples of the form $(i_1, \ldots, i_{k/2})$ with $i_j \in [n]$, $i_j < i_{j+1}$, so that $K(i_1, \ldots, i_{k/2}) = 1$ if and only if the vertices $i_1, \ldots, i_{k/2}$ form a $k/2$-clique. Clearly this table can be built in $O(k^2 n^{k/2})$ time. Now define functions C_j, for $j = 0, \ldots, k/2$, as follows:

$$C_j(i_1, \ldots, i_{\frac{k}{2}}) = \bigvee_{i'_j \in [n], i'_j > i_1, \ldots, i_{\frac{k}{2}}} \left(\prod_{\ell=1}^{j} E(i'_j, i_\ell) \right) C_{j-1}(i_1, \ldots, i_{j-1}, i'_j, i_{j+1}, \ldots, i_{\frac{k}{2}}).$$

$$C_0(i_1, \ldots, i_{\frac{k}{2}}) = K(i_1, \ldots, i_{\frac{k}{2}}).$$

For all $i_1, \ldots, i_{\frac{k}{2}} \in [n]$, and all $j = 0, \ldots, \frac{k}{2}$, the values $C_j(i_1, \ldots, i_{\frac{k}{2}})$ can be computed in $O(k\binom{n}{k/2}n)$ time, by a simple dynamic programming strategy. In particular, note that if all values for the function C_{j-1} have been computed, then a particular value $C_j(i_1, \ldots, i_{k/2})$ can be obtained in $O(n)$ time. Finally, check if there is a tuple $(i_1, \ldots, i_{k/2})$ satisfying both $C_{k/2}(i_1, \ldots, i_{k/2}) = 1$ and $K(i_1, \ldots, i_{k/2}) = 1$. If so, output *yes, there is a k-semiclique.* If no tuples satisfy the property, output *no.* If we detect a semiclique for some $S = \{i_1, \ldots, i_{k/2}\}$, a semiclique can be found by just going through all $k/2$-cliques and checking whether they form a semiclique with S in overall extra $O(k^2 n^{k/2})$ time.

It is easy to verify the following lemma by induction.

Lemma 3.1 *For $i_1, \ldots, i_{k/2} \in [n]$, let $S_j(i_1, \ldots, i_{k/2})$ denote the set of all j-tuples $i'_j, \ldots, i'_1 \in [n]$, such that $i'_j > i_1, \ldots, i_{k/2}$, and $i'_\ell < i'_{\ell-1}$ for all $\ell = 2, \ldots, j$. Then*

$$C_j(i_1, \ldots, i_{k/2}) = \bigvee_{(i'_j, \ldots, i'_1) \in S_j(i_1, \ldots, i_{k/2})} K(i'_1, \ldots, i'_j, i_{j+1}, \ldots, i_{k/2}) \prod_{p=1}^{j} \left(\prod_{\ell=1}^{p} E(i'_p, i_\ell) \right).$$

Lemma 3.1 implies that $C_j(i_1, \ldots, i_{k/2}) = 1$ if and only if there exists a j-set $S_j = \{i'_1, \ldots, i'_j\}$ such that

- all elements of S_j appear in π after all elements of $\{i_1, \ldots, i_{k/2}\}$; S_j is hence disjoint from $\{i_1, \ldots, i_{k/2}\}$,
- for each $\ell < j$, i'_ℓ has edges to all i_p with $p \leq \ell$, and
- $S_j \cup \{i_{j+1}, \ldots, i_{k/2}\}$ forms a clique.

In particular, this implies that $C_{k/2}(i_1, \ldots, i_{k/2})$ is nonzero iff there is a $k/2$-tuple $(i_{k/2+1}, \ldots, i_k)$ such that, if $\{i_1, \ldots, i_{k/2}\}$ is a $k/2$-clique, then the k-set $\{i_1, \ldots, i_k\}$ is a k-semiclique. The correctness of our algorithm is immediate.

Theorem 3.1 *For any fixed integer k and a given graph $G = (V, E)$, one can find a k-semiclique subgraph in G, if one exists, in $O(n^{\frac{k}{2}+1})$ time with high probability, or deterministically in $O(n^{\frac{k}{2}+1} \log n)$ time.*

Acknowledgments. The author is extremely grateful to Ryan Williams for his assistance and valuable ideas, especially in the semiclique part of the paper.

References

[1] N. Alon, R. Yuster, and U. Zwick. Color-coding. *JACM*, 42(4):844–856, 1995.

[2] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[3] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev. On economical construction of the transitive closure of an oriented graph. *Soviet Math. Dokl.*, 11:1209–1210, 1970.

[4] J. Basch, S. Khanna, and R. Motwani. On diameter verification and boolean matrix multiplication. *Report No. STAN-CS-95-1544, Department of Computer Science, Stanford University (1995)*, 1995.

[5] T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. In *Proc. SODA*, pages 514–523, 2006.

[6] J. Chen, X. Huang, I. A. Kanj, and G. Xia. Linear FPT reductions and computational lower bounds. In *Proc. STOC*, pages 212–221, 2004.

[7] J. Chen, S. Lu, S. Sze, and F. Zhang. Improved algorithms for path, matching, and packing problems. In *Proc. SODA*, pages 298–307, 2007.

[8] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Computation*, 9(3):251–280, 1990.

[9] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM J. Comput.*, 24(4):873–921, 1995.

[10] F. Eisenbrand and F. Grandoni. On the complexity of fixed parameter clique and dominating set. *Theor. Comp. Sci.*, 326(1-3):57–67, 2004.

[11] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: A simple $O(2^{0.288n})$ independent set algorithm. In *Proc. SODA*, pages 18–25, 2006.

[12] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7(4):413–423, 1978.

[13] T. Kloks, D. Kratsch, and H. Müller. Finding and counting small induced subgraphs efficiently. *Inf. Proc. Letters*, 74(3-4):115–121, 2000.

[14] J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Divide-and-color. In *WG*, pages 58–67, 2006.

[15] D. Knuth. *The art of computer programming, 3rd edition, vol. 2, Seminumerical Algorithms*. Addison-Wesley, 1997.

[16] Miscellaneous Authors. Queries and problems. *SIGACT News*, 16(3):38–47, 1984.

[17] M. Naor, L. J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Proc. FOCS*, pages 182–191, 1995.

[18] J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Commentationes Math. Universitatis Carolinae*, 26(2):415–419, 1985.

[19] J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.

[20] W. Rytter. Fast recognition of pushdown automaton and context-free languages. *Information and Control*, 67(1–3):12–22, 1985.

[21] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13:354–356, 1969.

[22] G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems. In *Proc. IWPEC, LNCS 3162*, pages 281–290, 2004.