Finding and counting small induced subgraphs efficiently

Ton Kloks

Department of Applied Mathematics

Charles University

Prague, Czech Republic

kloks@kam.ms.mff.cuni.cz

Dieter Kratsch and Haiko Müller

Fakultät für Mathematik und Informatik

Friedrich-Schiller-Universität Jena

07740 Jena, Germany

{kratsch,hm}@minet.uni-jena.de

Abstract

We give two algorithms for listing all simplicial vertices of a graph running in time $O(n^\alpha)$ and $O(e^{\frac{2\alpha}{\alpha+1}}) = O(e^{1.41})$, respectively, where n and e denote the number of vertices and edges in the graph and $O(n^\alpha)$ is the time needed to perform a fast matrix multiplication. We present new algorithms for the recognition of diamond-free graphs ($O(n^\alpha + e^{3/2})$), claw-free graphs ($O(e^{\frac{\alpha+1}{2}}) = O(e^{1.69})$), and K_4-free graphs ($O(e^{\frac{\alpha+1}{2}}) = O(e^{1.69})$). Furthermore, we show that *counting* the number of K_4's in a graph can be done in time $O(e^{\frac{\alpha+1}{2}})$. For all other graphs on four vertices we can count within $O(n^\alpha + e^{1.69})$ time the number of occurences as induced subgraph.

1 Introduction

The first problem we consider is the problem of finding all simplicial vertices of a graph. This is of interest, since the complexity of many problems (e.g., CHROMATIC NUMBER, MAXIMUM CLIQUE) can be reduced by first removing simplicial vertices from the graph.

We first show an algorithm with time $O(n^\alpha)$, which is the time needed to compute the square of an $n \times n$ 0/1-matrix, where n is the number of vertices of the graph. ($\alpha < 2.376$.) Then, using an idea of Alon, Yuster and Zwick in [1] we obtain an alternative algorithm with time $O(e^{\frac{2\alpha}{\alpha+1}})$, where e is the number of edges in the graph.

A very basic problem in theoretical computer science is the problem of finding a triangle in a graph. In 1978 Itai and Rodeh presented two solutions for this problem. The first algorithm has a time of $O(n^\alpha)$. The second algorithm needs $O(e^{3/2})$.

In [2] this result was refined to $O(ea(G))$, where $a(G)$ is the arboricity of the graph. Since $a(G) = O(\sqrt{e})$ in a connected graph (see [2]), this extends the result of Itai and Rodeh. These were for almost ten years the best known algorithms.

A drastic improvement was made recently by Alon et al. In [1] they showed the following surprisingly elegant and easy result. Deciding whether a directed or an undirected graph G contains a triangle, and finding one if it does, can be done in $O(e^{\frac{2\alpha}{\alpha+1}}) = O(e^{1.41})$.

For finding a certain induced connected subgraph with four vertices there are not many non-obvious results known. Two notable exceptions are the recognition of paw-free graphs and the recognition of P_4-free graphs. A *paw* is the graph consisting of a triangle and one pendant vertex. Using a characterization of Olariu [9], the class of paw-free graphs can be recognized in $O(n^\alpha)$ time. P_4-free graphs (cographs) can even be recognized in linear time [3]. We present new efficient algorithms for all other induced connected subgraphs on four vertices.

First we adopt the idea of [1] to obtain an efficient algorithm that checks if a graph contains a diamond and finds one if it does. The time of our algorithm is $O(n^\alpha + e^{3/2})$.

Using standard techniques, it is easy to see that connected claw-free graphs can be recognized in $O(n^{\alpha-1}e)$ time. We show that there is also a $O(e^{\frac{\alpha+1}{2}}) = O(e^{1.69})$ recognition algorithm for claw-free graphs.

In [1] an easy $O(n^\alpha)$ algorithm counting the number of triangles in a graph is given. Furthermore the authors ask for an efficient algorithm counting the K_4's. We give an $O(e^{\frac{\alpha+1}{2}}) = O(e^{1.69})$ algorithm counting the K_4's. Moreover we show that for any fixed graph H on four vertices the number of copies of H in a given graph G can be counted in time $O(n^\alpha + e^{\frac{\alpha+1}{2}})$.

2 Listing all simplicial vertices

Definition 1 *A vertex x in a graph G is called a* simplicial vertex *if its neighborhood $N(x)$ is complete.*

For many problems (e.g., coloring) simplicial vertices can be safely removed, tackling the problem on the reduced graph. It is therefore of interest to find these simplicial vertices quickly.

For any graph $G = (V, E)$ and $X \subseteq V$ we denote by $G[X]$ the subgraph of G *induced* by X. We denote by $N[x]$ the *closed* neighborhood of x, i.e., $N[x] = \{x\} \cup N(x)$. Let $d(x)$ be the degree of x, i.e., $d(x) = |N(x)|$.

Lemma 1 *A vertex x is simplicial if and only if for all neighbors y of x, $N[x] \subseteq N[y]$.*

Proof. The 'only if' part is obvious. Assume x is a vertex such that for every neighbor y, $N[x] \subseteq N[y]$. Assume x is not simplicial. Let y and z be two non adjacent neighbors of x. Then $z \in N[x]$, but $z \notin N[y]$, contradicting $N[x] \subseteq N[y]$. □

Corollary 1 *A vertex x is simplicial if and only if for all neighbors y of x, $|N[x] \cap N[y]| = |N[x]|$.*

Now let A be the 0/1-adjacency matrix of $G = (V, E)$ with 1's on the diagonal, i.e., $A_{x,x} = 1$ for all x and for $x \neq y$, $A_{x,y} = 1$ if x and y are adjacent in G and $A_{x,y} = 0$ otherwise. Hence A is a symmetric $n \times n$ 0/1-matrix. Consider A^2. The following is a key observation. For all $x, y \in V$:

$$\left(A^2\right)_{x,y} = |N[x] \cap N[y]|$$

(In particular $(A^2)_{x,x} = d(x) + 1$).)

Theorem 1 *There exists an $O(n^\alpha)$ algorithm which finds a list of all simplicial vertices of a graph G with n vertices.*

Proof. We assume that we have for each vertex a list of its neighbors. Construct the 0/1-adjacency matrix A with 1's on the diagonal. Compute A^2 in time $O(n^\alpha)$. By Corollary 1 a vertex x is simplicial if and only if $(A^2)_{x,y} = (A^2)_{x,x}$ holds for all $y \in N(x)$. Hence for each vertex x this test can be performed in time $O(d(x))$. The result follows. □

The next algorithm is based on a technique presented in [1]. Let D be some integer (to be determined later). We call a vertex x of *low degree* if $d(x) \leq D$. A vertex which is not of low degree, is said to be of *high degree*. Our algorithm consists of four phases.

Phase 1 Search all simplicial vertices that are of low degree.

Phase 2 Mark all vertices of high degree that have a low degree neighbor.

Phase 3 Remove all low degree vertices from the graph. Call the resulting graph G^*.

Phase 4 Perform a matrix multiplication for the 0/1-adjacency matrix of G^* with 1's on the diagonal. Make a list of all simplicials of G^* which are *not* marked in Phase 2, using the algorithm of Theorem 1.

Correctness follows from the following observation.

Lemma 2 *If a vertex x of high degree is simplicial, then all its neighbors are of high degree.*

Proof. Assume a vertex x of high degree is simplicial. Then for every neighbor y of x, $d(y)+1 = |N[y]| \geq |N[x]| = d(x)+1 > D+1$, by Lemma 1. Hence y is also of high degree. □

Theorem 2 *There exists an $O(e^{\frac{2\alpha}{\alpha+1}})$ time algorithm to compute a list of all simplicial vertices of a graph.*

Proof. Let $D = e^{\frac{\alpha-1}{\alpha+1}}$. We assume we have for every vertex a list of its neighbors. Let L be the set of vertices of low degree and let H be the set of vertices of high degree.

The first phase of the algorithm can be implemented as follows. For each $x \in L$, we check if every pair of its neighbors is adjacent. Hence the first phase can be performed in time proportional to $\sum_{x \in L} d(x)^2 \leq 2De$.

Phase 2 and Phase 3 can clearly be implemented in linear time.

Now notice that $2e \geq \sum_{x \in H} d(x) \geq |H|D$, hence the number of vertices of G^* is at most $2e/D$. Computing the square of the 0/1-adjacency matrix for G^* with 1's on the diagonal can be performed in time $O((2e/D)^\alpha)$. Hence the total time needed by the algorithm is $O(eD + (2e/D)^\alpha) = O(e^{\frac{2\alpha}{\alpha+1}})$ by our choice of D. □

3 Diamond-free graphs

In this section we consider the recognition of diamonds in graphs.

Definition 2 *A* diamond *is a graph isomorphic to the graph depicted in Figure 1 on the left.*

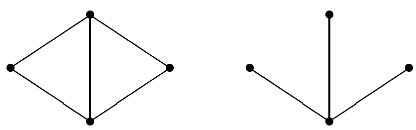

Figure 1: diamond (left) and claw (right)

For the following result, see also [11].

Lemma 3 *A graph G is diamond-free if and only if for every vertex x, the graph $G[N(x)]$ is a disjoint union of cliques.*

Proof. Consider a vertex x of degree 3 in a diamond. Then the neighborhood of x contains a P_3. Hence $G[N(x)]$ cannot be a disjoint union of cliques. Conversely, if for some vertex x, $G[N(x)]$ is not a disjoint union of cliques, then $G[N(x)]$ must contain a P_3. It follows that G contains a diamond. $\square$

Corollary 2 MAXIMUM CLIQUE *is solvable in time* $O(n(n+e))$ *for diamond-free graphs.*

Our algorithm for checking if a graph is diamond-free works as follows. Again, let D be some number. We partition the vertices into vertices of low degree, i.e., vertices of degree at most D and vertices of high degree. Let L be the set of low degree vertices, and H the set of high degree vertices.

Phase 1 Check if there is an induced diamond with a vertex of degree 3 which is of low degree in G.

Phase 2 Check if there is an induced diamond with a vertex of degree 2 which is of low degree in G.

Phase 3 If no diamond is found in the two previous steps, then remove all vertices of low degree from the graph (otherwise stop). Call the resulting graph G^*.

Phase 4 Check if G^* has a diamond.

We can implement this algorithm as follows. We assume we have the 0/1-adjacency matrix A with 0's on the diagonal. For each vertex x of low degree, construct adjacency lists for $G[N(x)]$. This can be accomplished in time $O(d(x)^2)$. Compute the components of $G[N(x)]$ (for example using depth first search). Check if each component is a clique. This can be done in $O(d(x)^2)$ time. Notice that if some component in $N(x)$ is *not* a clique, a P_3, and hence a diamond, can easily be computed in $O(d(x)^2)$ time (for example if y and z are non adjacent vertices in the same component, then start a breadth first search from y). It follows that Phase 1 can be implemented to run in $\sum_{x \in L} d(x)^2 \leq De$ time.

Next consider Phase 2. For each vertex x of low degree, we now have the cliques in the neighborhood (as a result of the previous phase). First compute A^2 in time $O(n^\alpha)$. Let C be a clique in $N(x)$. For each pair y, z

in C, we check whether $(A^2)_{y,z} > |C| - 1$. If this is the case, then y and z have a common neighbor outside $N[x]$. Hence in that case we find a diamond. (Producing the diamond can then be done in linear time, if all adjacency lists are sorted.) Hence, Phase 2 can be implemented to run in time $(De + n^\alpha)$.

Finally, assume that neither Phase 1 nor Phase 2 produces a diamond. Compute G^*. The number of vertices in G^* is at most $2e/D$. Now, simply repeat the procedure described in Phase 1, for *all* vertices of G^*. This takes time proportional to $\sum_{x \in H} d_H(x)^2 = O(e|H|) = O(e^2/D)$.

Hence the total running time is bounded by $O(De + e^2/D + n^\alpha)$. Taking $D = \sqrt{e}$ we obtain following theorem.

Theorem 3 *There is an $O(e^{3/2} + n^\alpha)$ algorithm that checks if a graph has a diamond and produces one if it does.*

4 Claw-free graphs

The importance of claw-free graphs follows from matching properties, line graphs, hamiltonian properties and the polynomial time algorithm for computing the independence number [4, 7]. However, although many characterizations are known, there is no fast recognition algorithm known. Even the extensive survey on claw-free graphs in [4] mentiones only a $O(n^{3.5})$ recognition algorithm. We present in this section an $O(e^{\frac{\alpha+1}{2}}) = O(e^{1.69})$ recognition algorithm.

Definition 3 *A* claw *is a graph isomorphic to the graph depicted in Figure 1 on the right. A graph is* claw-free *if it does not have an induced subgraph isomorphic to a claw. We denote the vertex of degree 3 in a claw as the* central vertex.

Let G be a graph with n vertices and e edges. We start with an easy observation.

Lemma 4 *If G is claw-free then every vertex has at most $2\sqrt{e}$ neighbors.*

Proof. Consider a vertex x. If G is claw-free, then $\overline{G[N(x)]}$ is triangle-free. Let $p = |N(x)|$. By Turán's theorem, a graph with p vertices and without triangles can have at most $\frac{p^2}{4}$ edges (see, e.g., [10, 5]). Hence, there must be at least $\frac{1}{2}p(p-1) - \frac{1}{4}p^2 = \frac{1}{4}p^2 - \frac{1}{2}p$ edges in $G[N(x)]$. Then, (adding the edges incident with x), $G[N[x]]$ must contain at least $\frac{1}{4}p^2 - \frac{1}{2}p + p \geq \frac{1}{4}p^2$ edges. This can be at most the number of edges of G. Hence $p \leq 2\sqrt{e}$. $\square$

Our algorithm for recognizing claw-free graphs works as follows. First we check whether every vertex has at most $2\sqrt{e}$ neighbors. If there is a vertex with more than $2\sqrt{e}$ neighbors, there must be a claw with this vertex as the central vertex by Lemma 4.

If every vertex has at most $2\sqrt{e}$ neighbors, we perform a fast matrix multiplication for each neighborhood to check if the complement of such a neighborhood contains a triangle. This step of the algorithm can be performed in time proportional to $\sum_x d(x)^\alpha \leq (2\sqrt{e})^{\alpha-1} \sum_x d(x) \leq 2^\alpha e^{\frac{\alpha+1}{2}}$.

This proves the following theorem.

Theorem 4 *There is an $O(e^{\frac{\alpha+1}{2}})$ algorithm to check whether a connected graph is claw-free.*

5 Counting the number of K_4's

In this section we first describe an algorithm that decides whether a connected graph has a K_4 as induced subgraph and outputs one if there exists one. Moreover, we show how to extend it to an algorithm that counts the number of occurences of K_4 as an induced subgraph of the given connected graph. Both algorithms have time $O(e^{\frac{\alpha+1}{2}}) = O(e^{1.69})$. The first one improves upon an $O(e^{1.89})$ algorithm mentioned in [1].

As in the preceeding sections the vertex set of the input graph G is partitioned into the sets L and H. The vertices of L are those with degree at most D. The recognition algorithm works as follows.

Phase 1 For each vertex $x \in H$ compute the square of the adjacency matrix of $G[N(x) \cap H]$ to decide if there is a triangle contained in $G[N(x) \cap H]$.

This can be done in time $\sum_{x \in H} d_H(x)^\alpha = O(e^\alpha / D^{\alpha-1})$.

Phase 2 For each vertex $x \in L$ compute the square of the adjacency matrix of $G[N(x)]$ to decide whether $G[N(x)]$ contains a triangle.

This can be done in time $\sum_{x \in L} d(x)^\alpha = O(D^{\alpha-1}e)$.

Theorem 5 *There is an $O(e^{\frac{\alpha+1}{2}})$ algorithm that checks whether a connected graph has a K_4 and outputs one if it does.*

Proof. G has a K_4 if and only if the algorithm finds a triangle in $G[N(x) \cap H]$ in Phase 1 or a triangle in $G[N(x)]$ in Phase 2. The time bound follows by taking $D = \sqrt{e}$. $\square$

In [1] an easy $O(n^{\alpha+1})$ algorithm for counting the K_4's in a graph is given and the authors ask whether it can be improved to an $o(n^{\alpha+1})$ algorithm. We present an $O(e^{(\alpha+1)/2})$ algorithm that counts the number of K_4's in a given connected graph. Note that this is at least as good as the algorithm of [1] for all graphs. However it is an $o(n^{\alpha+1})$ algorithm only for sparse graphs.

We distinguish five different types of a K_4, depending on the number of low and high vertices, respectively, in the K_4. We denote by $L(i)$ the set of K_4's with exactly i vertices of low degree. Clearly, each K_4 is of exactly one of the types $L(0)$, $L(1)$, $L(2)$, $L(3)$ and $L(4)$. The counting algorithm determines the number of K_4's in G from each type, denoted by $\#(L(4))$, $\#(L(3))$, $\#(L(2))$, $\#(L(1))$ and $\#(L(0))$, respectively. The algorithm starts as the recognition algorithm, however in each phase it counts the corresponding number of triangles using the $O(n^\alpha)$ algorithm given in [1].

Phase 1 For each vertex $x \in H$ compute the square of the adjacency matrix of $G[N(x) \cap H]$. Compute the sum of the number of triangles contained in $G[N(x) \cap H]$ taken over all $x \in H$. This number is exactly $4\#(L(0))$. The time is $\sum_{x \in H} d_H(x)^\alpha = O(e^\alpha / D^{\alpha-1})$.

Phase 2 For each vertex $x \in L$ compute the square of the adjacency matrix of $G[N(x)]$. Compute the sum of the number of triangles contained in $G[N(x)]$ taken over all $x \in L$. This number is equal to $4\#(L(4)) + 3\#(L(3)) + 2\#(L(2)) + \#(L(1))$ since each K_4 is counted exactly i times, $i \in \{1, 2, 3, 4\}$, if and only if it contains i low vertices.

Phase 2 can be done in time $\sum_{x \in L} d(x)^\alpha = O(D^{\alpha-1}e)$.

The next two phases compute $\#(L(4))$ and $\#(L(1))$, respectively, in a similar fashion.

Phase 3 For every $x \in L$ compute $G[N(x) \cap L]$. Compute the sum of the number of triangles in $G[N(x) \cap L]$. This number is equal to $4\#(L(4))$. This takes time $\sum_{x \in L} d(x)^\alpha = O(eD^{\alpha-1})$.

Phase 4 For every $x \in L$ compute $G[N(x) \cap H]$. Compute the sum of the number of triangles in $G[N(x) \cap H]$. This number is equal to $\#(L(1))$. This takes time $\sum_{x \in L} d(x)^\alpha = O(eD^{\alpha-1})$.

Finally we are going to count the triangles in the neighbourhood of a vertex in a sligthly different way.

Phase 5 For every low vertex x compute the adjacency matrix $A(x)$ of $G[N(x)]$ and then compute its square $A(x)^2$. Then $(A(x)^2)_{i,j}$ is exactly the number of common neighbours of $i, j \in N(x)$ belonging to $N(x)$. We compute the sum over all $(A(x)^2)_{i,j}$ for which $\{i, j\} \in E$, $i < j$ and $i, j \in H$. Moreover, we sum these values over all low vertices x. Hence, we only count those K_4 of type $L(2)$ or $L(1)$. The final value we get is exactly $2\#(L(2)) + 3\#(L(1))$. This can be done in time $\sum_{x \in L} d(x)^\alpha = O(eD^{\alpha-1})$.

Theorem 6 *There is an $O(e^{(\alpha+1)/2}) = O(e^{1.69})$ time algorithm counting the number of K_4's in a given graph.*

Proof. Using all the values we computed by the algorithm it is easy to compute $\#(L(0)) + \#(L(1)) + \#(L(2)) + \#(L(3)) + \#(L(4))$ i.e., the number of K_4's occuring as induced subgraph in G. The time bound follows by taking $D := \sqrt{e}$. $\square$

6 Recognizing K_ℓ's

Nešetřil and Poljak gave in [8] an $O(n^{\alpha\lfloor \ell/3 \rfloor + i})$ time algorithm that decides whether a given graph contains a K_ℓ. Here i is the remainder of ℓ by division by 3 which we will also denote by $i = \mathrm{mod}(\ell, 3)$. As mentioned in [1] it is easy to see that the method of [8] can also be used for counting K_ℓ's within the same time bound. Alon, Yuster and Zwick mention in [1] that combining this K_ℓ counting algorithm of [8] with their methods leads to $O(n\, d(G)^{\alpha\lfloor(\ell-1)/3\rfloor+i})$ and $O(e\, d(G)^{\alpha\lfloor(\ell-2)/3\rfloor+i})$ time algorithms for counting the number of K_ℓ's in a given graph where $d(G)$ denotes the degeneracy of the input graph.

We show that the algorithm of [8] can also be used to design another type of algorithm that recognizes K_ℓ's by generalizing the $O(e^{\frac{2\alpha}{\alpha+1}})$ time triangle recognition algorithm of [1] and the $O(e^{\frac{\alpha+1}{2}})$ time K_4 recognition algorithm of section 5.

As in the preceeding sections the vertex set of the input graph G is partitioned into the sets L and H. The vertices of L are those with degree at most D. The algorithm recognizing K_ℓ's works as follows.

Phase 1 For each vertex $x \in L$ compute the adjacency matrix of $G[N(x)]$ and decide whether $G[N(x)]$ contains a $K_{\ell-1}$ by applying the $K_{\ell-1}$ recognition algorithm of [8].

Phase 2 Compute the adjacency matrix of $G[H]$ and decide whether there is a K_ℓ contained in $G[H]$ by applying the K_ℓ recognition algorithm of [8].

Theorem 7 *There is an $O(e^{\frac{1}{2}(\alpha\lfloor\frac{\ell}{3}\rfloor+i)})$ time algorithm deciding whether a given graph contains a K_ℓ if $i = \mathrm{mod}(\ell, 3) \in \{1, 2\}$.*
There is an $O(e^{\beta\frac{\beta-\alpha+2}{2\beta-\alpha+1}})$, where $\beta = (\alpha\ell)/3$, time algorithm that decides whether a given graph contains a K_ℓ if ℓ is a multiple of 3.

Proof. The adjacency matrices of $G[N(x)]$, $x \in L$, are $d(x)\times d(x)$ matrices with $d(x) \leq D$. Hence the time of Phase 1 is

$$\sum_{x\in L} O(d(x)^{\alpha\lfloor(\ell-1)/3\rfloor+\bmod(\ell-1,3)}) = \\ O(e\,D^{\alpha\lfloor(\ell-1)/3\rfloor+\bmod(\ell-1,3)-1})$$

Since $|H| \leq 2e/D$ the running time of Phase 2 is $O((e/D)^{\alpha\lfloor\ell/3\rfloor+\bmod(\ell,3)})$.

Now we have $\alpha\lfloor(\ell-1)/3\rfloor + \mathrm{mod}(\ell-1, 3) = \alpha\lfloor l/3\rfloor + \mathrm{mod}(\ell, 3) - 1$ if ℓ is not a multiple of 3 and $\alpha\lfloor(\ell-1)/3\rfloor + \mathrm{mod}(\ell-1, 3) = \beta - \alpha + 2$ if ℓ is a multiple of 3.

Suppose ℓ is not a multiple of 3, i.e., $i = \mathrm{mod}(\ell, 3) \in \{1, 2\}$. Then we choose $D := \sqrt{e}$ implying that the time of the K_ℓ recognition algorithm is $O(e^{\frac{1}{2}(\alpha\lfloor\ell/3\rfloor+i)})$.

Suppose ℓ is a multiple of 3. Choosing $D := e^{\frac{\beta-1}{2\beta-\alpha+1}}$ we get the stated time of the K_ℓ recognition algorithm. □

It is worth mentioning that our algorithms are at least as fast as the algorithms of Nešetřil and Poljak on all input graphs if ℓ is not a multiple of

3. If ℓ is a multiple of 3 then this is not the case in general. However, notice that the time of the algorithm given in [8] is better only when the number of edges in the graph exceeds $\Omega\left(n^{2-\frac{3-\alpha}{\beta-\alpha+2}}\right) = \Omega\left(n^{2-\Theta\left(\frac{1}{\ell}\right)}\right)$. Hence, for large values of ℓ, the algorithm of [8] is faster only for very dense graphs.

It is an interesting open problem whether there is an $O(e^{\alpha\ell/6})$ algorithm for recognizing whether a graph contains a K_ℓ, if ℓ is a multiple of 3, in particular whether there is a $O(e^{\alpha/2})$ algorithm for recognizing triangles.

7 Other small subgraphs

We show that for any connected subgraph H on four vertices there is a $O(n^\alpha + e^{(\alpha+1)/2})$ algorithm counting the number of occurences of H in the given graph G. More precisely, on input $G = (V, E)$ we compute the cardinality of $\{W : W \subseteq V \text{ and } G[W] \text{ is isomorphic to } H\}$ for a fixed graph H on four vertices. The connected graphs on four vertices are K_4, $K_4 - e$ (the diamond), C_4 (the square), P_4, $\overline{P_3 + K_1}$ (the paw) and $K_{1,3}$ (the claw).

Theorem 8 *Let $\tilde{H}$ be a connected graph on four vertices such that there is an $O(t(n,e))$ time algorithm counting the number of $\tilde{H}$'s in a given graph. Then there is an $O(n^\alpha + t(n,e))$ time algorithm counting the number of H's for all connected graphs H on four vertices.*

Proof. The number of occurences of the connected subgraphs on four vertices in a graph G fullfill the following system of linear equations. and K, D, S, P, Q and Y denote the number of subgraphs in G isomoprphic to K_4, diamond, square, P_4, paw and claw, respectively.

$$\begin{aligned}
\sum_{(x,y)\in E} \binom{(A^2)_{x,y}}{2} &= 6K + D \\
\sum_{(x,y)\notin E} \binom{(A^2)_{x,y}}{2} &= D + 2S \\
\sum_{(x,y)\in E} (AC)_{x,y}(CA)_{x,y} &= 4S + P \\
\sum_{x\in V} (A^3C)_{x,x} &= 4D + 2P + 4Q
\end{aligned}$$

$$\sum_{(x,y)\in E}\left(\binom{(AC)_{x,y}}{2}+\binom{(AC)_{y,x}}{2}\right)$$
$$= \quad Q+6Y$$

On left hand side, A denotes the adjacency matrix of the graph G, C the adjacency matrix of $\overline{G}$, both with zeros on the main diagonal. We are able to compute the values on left sides of these equations in time $O(n^\alpha)$.

Let us explain how to see the first equation. We consider an edge $(x,y)\in E$ such that $\binom{(A^2)_{x,y}}{2}=k$. Then there exist exactly k pairs of different vertices p and q such that $p,q\in N(x)\cap N(y)$, i.e., p, q, x and y induce either a K_4 (if $(p,q)\in E$) or a diamond (if $(p,q)\notin E$). Summing up we count each K_4 exactly six times, since every edge of K_4 will play the role of (x,y), and every diamond exactly once, where x and y are the vertices of degree three in the diamond. The equation follows. Similar observations give the other equations.

On right side we need the number of $\tilde{H}$'s for one subgraph $\tilde{H}$ to be able to solve the system and find the numbers of H's for all the other subgraphs H. $\square$

A similar theorem can be shown for all graphs on four vertices by extending the system of linear equations. This and Theorem 6 imply

Corollary 3 *For any graph H on four vertices there is a $O(n^\alpha+e^{\frac{\alpha+1}{2}})$ algorithm counting the H's in a given graph G.*

References

[1] Alon, N., R. Yuster and U. Zwick, Finding and counting given length cycles, *Algorithmica* **17** (1997), pp. 209–223.

[2] Chiba, N. and T. Nishizeki, Arboricity and subgraph listing algorithms, *SIAM J. Comput.* **14**, (1985), pp. 210–223.

[3] Corneil, D. G., Y. Perl and L. K. Stewart, A linear recognition algorithm for cographs, *SIAM J. Comput.* **4**, (1985), pp. 926–934.

[4] Faudree, R., E. Flandrin and Z. Ryjáček, Claw-free graphs–A survey, *Discrete Mathematics* **164**, (1997), pp. 87–147.

[5] Harary, F., *Graph Theory*, Addison Wesley, Publ. Comp., Reading, Massachusetts, (1969).

[6] Itai, A. and M. Rodeh, Finding a minimum circuit in a graph, *SIAM J. Comput.* **7**, (1978), pp. 413–423.

[7] Minty, G. J., On maximal independent sets of vertices in claw-free graphs, *J. Combin. Theory B* **28**, (1980), pp. 284–304.

[8] Nešetřil, J. and S. Poljak, On the complexity of the subgraph problem, *Commentat. Math. Univ. Carolin.* **14** (1985), pp. 415–419.

[9] Olariu, S., Paw-free graphs, *Inform. Process. Lett.* **28**, (1988), pp. 53–54.

[10] Turán, P., Eine Extremalaufgabe aus der Graphentheorie, *Mat. Fiz. Lapok* **48**, (1941), pp. 436–452.

[11] Tucker, A., Coloring perfect $K_4 - e$-free graphs, *J. Combin. Theory B* **42**, (1987), pp. 313–318.