RESEARCH ARTICLE

Combinatorial algorithm for counting small induced graphs and orbits

Tomaž Hočevar*, Janez Demšar

Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia

* tomaz.hocevar@fri.uni-lj.si

Abstract

Graphlet analysis is an approach to network analysis that is particularly popular in bioinformatics. We show how to set up a system of linear equations that relate the orbit counts and can be used in an algorithm that is significantly faster than the existing approaches based on direct enumeration of graphlets. The approach presented in this paper presents a generalization of the currently fastest method for counting 5-node graphlets in bioinformatics. The algorithm requires existence of a vertex with certain properties; we show that such vertex exists for graphlets of arbitrary size, except for complete graphs and a cycle with four nodes, which are treated separately. Empirical analysis of running time agrees with the theoretical results.

Citation: Hočevar T, Demšar J (2017) Combinatorial algorithm for counting small induced graphs and orbits. PLoS ONE 12(2): e0171428. doi:10.1371/journal.pone.0171428

Editor: J. Alberto Conejero, IUMPA - Universitat Politecnica de Valencia, SPAIN

Received: June 2, 2016

Accepted: January 20, 2017

Published: February 9, 2017

Data Availability Statement: The source code and data are available from the http://github.com/thocevar/orca repository.

Funding: This work has been funded by the Slovenian research agency (https://www.arrs.gov.si/en/) grants J2-5480 and P2-0209. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Introduction

Analysis of networks plays a prominent role in various fields, from learning patterns [1] and predicting new links in social networks [2, 3], inferring gene functions from protein-protein interaction networks [4] in bioinformatics, to predicting various properties of chemical compounds (mutagenicity, boiling point, anti-cancer activity) [5] from their molecular structure in chemoinformatics. Many methods rely on the concept of node similarity, which is typically defined in a local sense, *e.g.* two nodes are similar if they share a large number of neighbours. Such definitions are insufficient for detecting the role of the node. A typical social structure includes hubs, followers, adversaries and intermediaries between groups. While local similarity definitions treat the hub and its adjacent nodes as similar, a role-based similarity would consider the hubs as similar disregarding their distance in the graph.

A popular approach in bioinformatics extracts the node's local topology by counting the small connected induced subgraphs (called *graphlets*) [6], which the node touches, and, when a more detailed picture is required, the node's position (*orbit*) [7] in those graphs. See the following paragraphs for a more formal definition. Fig 1 illustrates all four-node graphlets and orbits of their nodes. Most applications of graphlet and orbit counts are based on the assumption that the node's local network topology is somehow related to the functionality or some other property of the observed node in the network. Therefore, we can assume that nodes with similar signatures will have similar observed properties. This is the foundation for methods such as clustering of nodes, inference of certain node's properties etc. The nodes often correspond to proteins in the protein-protein interaction networks. However, the networks can model an arbitrary process. With the development of new technologies, these networks are becoming larger, which motivates the development of efficient subgraph counting algorithms.

Let $\mathcal{G}_k$ be a set of all non-isomorphic connected simple graphs (graphlets) on k nodes, and let $G \in \mathcal{G}_k$. The orbit of a vertex $v \in G$ is a set of all vertices $a(v)$, $a \in Aut(G)$. Let $\mathcal{O}_k$ be a set of orbits for all $v \in G$ and for all $G \in \mathcal{G}_k$. Pržulj [7] numbered the 30 graphs in $\mathcal{G}_2$, $\mathcal{G}_3$, $\mathcal{G}_4$ and $\mathcal{G}_5$ and the corresponding 73 orbits; we will use her enumeration in the examples in this paper.

Let $H = (V, E)$ be the host graph (network) and let $x \in H$. Vertex x participates in a number of subgraphs $G \in \mathcal{G}_k$ induced in H, in which it appears in different orbits $O_i \in \mathcal{O}_k$. Let o_i be the number of times x appears in orbit O_i in induced subgraphs from $\mathcal{G}_k$.

An example is shown in Fig 2. The orbit count o_{17} of vertex x is 9 since x appears in nine paths G_9 as the central vertex (note that the paths must be induced). Other orbit counts for G_9, o_{15} and o_{16}, are 0 and 4, respectively: x does not appear as the end vertex (O_{15}) of G_9 in H, but it appears four times in the role of the node between the center and the end (O_{16}). For a few more examples, $o_{44} = 1$, $o_{47} = 4$, and $o_{59} = 2$; all other orbit counts of 4-node graphlets are 0.

The *orbit count distribution* is a $|\mathcal{O}_k|$-dimensional vector of o_i for all $O_i \in \mathcal{O}_k$. The orbit count distribution represents a signature of the node: it contains a description of the node's neighborhood and the node's position ("role") within it. As such, this distribution is a useful feature vector for various network analysis tasks.

We will describe an algorithm for computation of orbit count distributions for all vertices $x \in V$ for subgraphs of arbitrary size k. The efficient implementation of the algorithm requires setting up a system of equations that relate subgraph nodes with specific properties; we will prove that such nodes exist for all $k \geq 3$ except complete graphs and the cycle on 4 nodes, which can be treated specifically. We will show—both theoretically as well as empirically—that

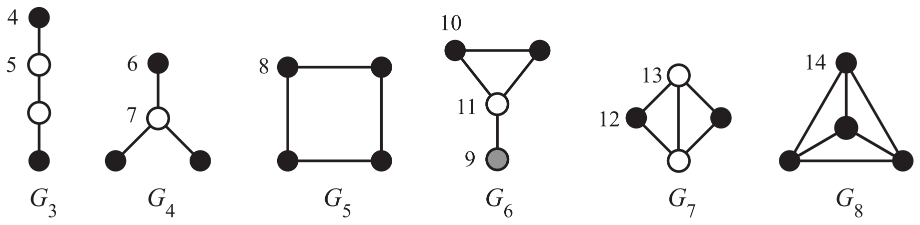

Fig 1. Four-node graphlets ($\mathcal{G}_4$). Vertices marked by the same color belong to the same orbit within a graphlet.

doi:10.1371/journal.pone.0171428.g001

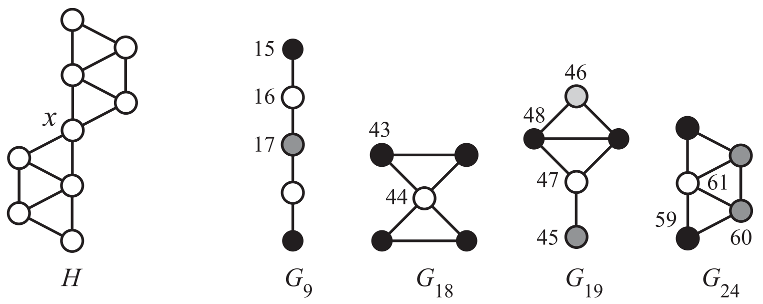

Fig 2. A host graph H and graphs G_9, G_{18}, G_{19} and G_{24} from $\mathcal{G}_5$. Graphs and orbits are numbered as in [7].

doi:10.1371/journal.pone.0171428.g002

Table 1. Notation.

$H = (V, E)$	**host graph within which we count the graphlets and orbits**
n	number of nodes of H; $n = \|V\|$
e	number of H's edges; $e = \|E\|$
$d(v)$	degree of node v
d	maximal node degree in H; $d = \max_{v \in V} d(v)$
$N(v)$	set of neighbours of vertex $v \in V$
$N(v_1, v_2, \ldots, v_j)$	set of common neighbours of $v_1, v_2, \ldots, v_j$; $N(v_1, v_2, \ldots, v_j) = N(v_1) \cap N(v_2) \cap \ldots \cap N(v_j)$
$N(\mathcal{S})$	common neighbours of nodes in the set $\mathcal{S} \subset V$; $N(\mathcal{S}) = \cap_{v \in \mathcal{S}} N(v)$
$c(v)$, $c(v_1, v_2, \ldots, v_j)$, $c(\mathcal{S})$	number of common neighbours of vertex v, of vertices $v_1, v_2, \ldots, v_j$, and of vertices from set $\mathcal{S}$, respectively; that is, $c(v) = \|N(v)\|$, $c(v_1, v_2, \ldots, v_j) = \|N(v_1, v_2, \ldots, v_j)\|$, $c(\mathcal{S}) = \|N(\mathcal{S})\|$
$\mathcal{G}_k$	set of all graphlets with k nodes
G_a	graphlet a, according to some enumeration
O_i	orbit i, according to some enumeration
$o_i(v)$, o_i	the number of times the node v appears in an induced subgraph in orbit i; since v will be obvious, we will use the shorter notation o_i
$m(i)$	index of the graphlet containing the orbit O_i, *e.g.* $m(16) = 9$

doi:10.1371/journal.pone.0171428.t001

the algorithm's time complexity on sparse graphs is lower by an order of magnitude in comparison with enumeration-based approaches.

Preliminaries

Referring to graphlets, orbits, neighbours, etc., requires some notation which we summarize in Table 1 and use throughout this paper.

Let $K = (V_K, E_K)$ be a subgraph of $J = (V_J, E_J)$, and let $v \in V_K$. We will denote J's vertex that corresponds to v by v^J. If there are multiple isomorphic embeddings of K in J, v^J refers to one of them. Similarly, if $S \subseteq V_K$, then the corresponding vertices in J are denoted by S^J.

Related work

The most basic case of counting induced patterns in graphs is that of counting triangles. Itai and Rodeh [8] showed that this can be done faster than by exhaustive enumeration in $O(n^3)$ time. Raising the graph's adjacency matrix A to the third power gives the number of paths of length 3 between pairs of nodes. Element $A^3_{x,x}$ represents the number of paths of length 3 that start and finish in the node x, which corresponds to the number of triangles that include x. The total number of triangles is then $\frac{1}{6} \sum_{x \in G} A^3_{x,x}$. Note that the same triangle is counted twice for each of its three nodes. The time complexity of this procedure equals that of multiplying two matrices, which is faster than exhaustive enumeration of triangles in dense graphs. A natural extension of this result is to larger cliques. Nesetril and Poljak [9] studied the problem of detecting a clique of size k in a graph with n nodes. They showed that this problem can be solved faster than with the straight-forward $O(n^k)$ solution. Their approach reduces the original problem to detection of triangles in a graph with $O(n^{k/3})$ nodes. Since we can detect triangles faster than in $O(n^3)$ with fast matrix multiplication algorithms, we can also detect cliques of size k faster than $O(n^k)$.

Counting all non-induced subgraphs is as hard as counting all induced subgraphs because they are connected through a system of linear equations. Despite this it is sometimes beneficial to compute induced counts from non-induced ones. Rapid Graphlet Enumerator (RAGE) [10] takes this approach for counting four-node graphlets. Instead of counting induced subgraphs directly, it reconstructs them from counts of non-induced subgraphs. For computing the latter, it uses specifically crafted methods for each of the 6 possible subgraphs (P_4, claw, C_4, paw, diamond and K_4). The time complexity of counting non-induced cycles and complete graphs is $O(e \cdot d + e^2)$, while counting other subgraphs runs in $O(e \cdot d)$. However, the run-time of counting cycles and cliques in real-world networks is usually much lower.

Some approaches exploit the relations between the numbers of occurrences of induced subgraphs in a graph. Kloks *et al.* [11] showed how to construct a system of equations that allows computing the number of occurrences of all six possible induced four-node subgraphs if we know the count of any of them. The time complexity of setting up the system equals the time complexity of multiplying two square matrices of size n. Kowaluk *et al.* [12] generalized the result by Kloks to counting subgraph patterns of arbitrary size. Their solution depends on the size of the independent set in the pattern graph and relies on fast matrix multiplication techniques. They also provide an analysis of their approach on sparse graphs, where they avoid matrix multiplications and derive the time bounds in terms of the number of edges in the graph.

Floderus *et al.* [13] researched whether some induced subgraphs are easier to count than others as is the case with non-induced subgraphs. For example, we can count non-induced stars with k nodes, $\sum_{x \in V} \binom{c(x)}{k-1}$, in linear time. They conjectured that all induced subgraphs are equally hard to count. They showed that the time complexity in terms of the size of G for counting any pattern graph H on k nodes in graph G is at least as high as counting independent sets on k nodes in terms of the size of G.

Vassilevska and Williams [14] studied the problem of finding and counting individual non-induced subgraphs. Their results depend on the size s of the independent set in the pattern graph and rely on efficient computations of matrix permanents and not on fast matrix multiplication techniques like some other approaches. If we restrict the problem to counting small patterns and therefore treat k and s as small constants, their approach counts a non-induced pattern in $O(n^{k-s+2})$ time. This is an improvement over a simple enumeration when $s \geq 3$. Kowaluk *et al.* [12] also improved on the result of Vassilevska and Williams when $s = 2$. Alon et al. [15] developed algorithms for counting non-induced cycles with 3 to 7 nodes in $O(n^{\omega})$, where ω represents the exponent of matrix multiplication algorithms.

Alon *et al.* [16] introduced the color-coding technique for finding simple paths and cycles in graphs. Their technique is applicable not just to paths and cycles but also to other patterns with a small treewidth. The authors of [17] used such color-coding approach to approximate a 'treelet' distribution (frequency of non-induced trees) for trees with up to 10 nodes.

Recently, Melckenbeeck et al. [18] published a paper that describes how to generate systems of equations similar to those used in the ORCA algorithm [19] for arbitrarily large graphlets. However, the resulting equations do not satisfy the requirements needed for an efficient counting algorithm. Consider for example the equation $o_{50} + o_{55} = \sum_{P_7(x,a,b,c)}(c(a, b, c) - 1)$. There can be as many as $O(ed^2)$ sets of nodes $\{a, b, c\}$ with a nonzero number of common neighbours, which makes the computation of common neighbours the limiting factor in terms of space and time requirements. The method we present in this paper and the related proofs show how to avoid this issue and construct an efficient algorithm for arbitraty graphlet sizes.

Outline of the proposed algorithm

We will derive a system of linear equations that relate the orbit counts of a fixed node for graphlets with k vertices. The coefficients on the left-hand sides reflect the symmetries in the graphlets and do not depend on the host graph, so they are derived in advance. The right-hand sides are computed as sums over graphlets with $k - 1$ vertices induced in the host graph H, and the sums include terms that represent the number of common neighbours of certain vertices in the embeddings of graphlets in H.

The resulting system of equations will be triangular and have a rank of $|\mathcal{O}_k| - 1$. We can efficiently enumerate the complete graphlet, after which the system of equations for the remaining orbit counts can be solved using integer arithmetic, thus avoiding any numerical errors.

Original contributions

We already presented the original idea of the algorithm in a recent article in Bioinformatics [19], in which we focused on its use in genetics and avoided formal descriptions and analysis. In this paper we

1. present the algorithm more formally;
2. describe a general method for derivation of the system of equations relating the orbit counts;
3. generalize it to induced subgraphs of arbitrary size; in particular, we prove that the system of equations with the properties required for the efficient implementation of the algorithm exists for any $k \geq 4$;
4. provide worst time-complexity analysis and the analysis of the expected time complexity on random graphs;
5. empirically explore the efficiency of the orbit counting algorithm and compare it with the theoretical results.

The remainder of the paper is composed of two parts. In the next section we show a technique for building the system of equations with desired properties, and in the following section we present an algorithm based on them and analyze its time- and space-complexity.

Relations between orbit counts

We will show how to construct linear relations between a chosen orbit count o_i and some orbits belonging to graphlets with a larger number of edges. We will illustrate the procedure on figures showing the derivation of the following Eq (1) that relates the count for orbit 59 and counts for orbits 65, 68 and 70.

$$o_{59} + 4o_{65} + 2o_{68} + 6o_{70} = \sum_{\substack{x_1, x_2, x_3 : \\ x_1 < x_2 \land x_3 \notin N(x), \\ H[\{x, x_1, x_2, x_3\}] \cong G_7}} [(c(x_1, x_3) - 1) + (c(x_2, x_3) - 1)] \tag{1}$$

Derivation of general relations between orbit counts

Let orbit O_i appear in a connected simple k-node graphlet $G_a = (V_G, E_G)$ ($a = m(i)$). We denote the G_a's node that is in orbit O_i by x; if there are multiple such nodes, we pick one. Next, we choose a node $y \neq x$, such that $G' = G_a \backslash \{y\}$ is still a connected graph; we will impose additional constraints on y later to ensure an efficient implementation of the algorithm. According to our notation, $x^{G'}$ is the node in G' that corresponds to x in G_a; let O_m be its orbit. We label the remaining $k - 2$ nodes with $x_1, x_2, \ldots, x_{k-2}$ (Fig 3).

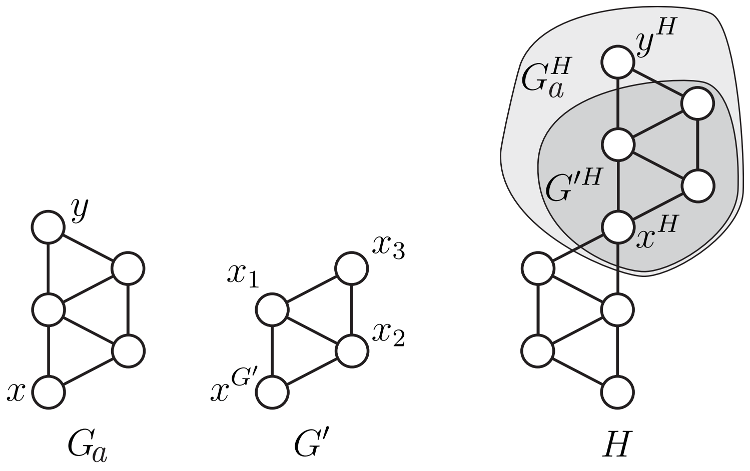

Fig 3. Reducing the *k*-node graphlet G_a to a $k-1$ node graphlet G'. O_{59} appears in $G_a = G_{24}$; the node in O_{59} is labelled *x*. Removal of node *y* results in $G' = G_7$. The node in G' that corresponds to *x*, $x^{G'}$, belongs to O_{12}. We assigned labels x_1, x_2 and x_3 to the remaining nodes of G'. Embeddings of G_a and G' in *H* are referred to as G_a^H and G'^H, respectively. The nodes corresponding to *x* and *y* are marked by x^H and y^H.

doi:10.1371/journal.pone.0171428.g003

We now go in the opposite direction: starting with G', we consider its possible extensions to G_a. Let $E \subset V_{G'}$ be a set of nodes such that adding a new vertex *y* connected to all vertices in *E* yields G_a with *x* in orbit O_i (Fig 4). Let $\mathcal{E}$ be a set of all such subsets *E*.

Let G'^H be some particular occurrence of G' in *H*. To count o_i for the node x^H (the node in *H* to which *x* maps), we need to explore the extensions of G'^H to G_a^H. A necessary (but insufficient) condition to put x^H into O_i is that the additional node *y* is a common neighbour of all vertices E^H for one of $E^H \in \mathcal{E}^{\mathcal{H}}$ (with respect to the particular occurrence of G' in *H*). There are at most

$$\sum_{E \in \mathcal{E}} (c(E^H) - c(E^{G'})) \tag{2}$$

candidate nodes *y*; $c(E^{G'})$ represents the number of neighbours of *E* that are already in G' (*i.e.* x_i) and cannot be mapped to *y*. Eq (2) represents the term in the sum in the right side of the

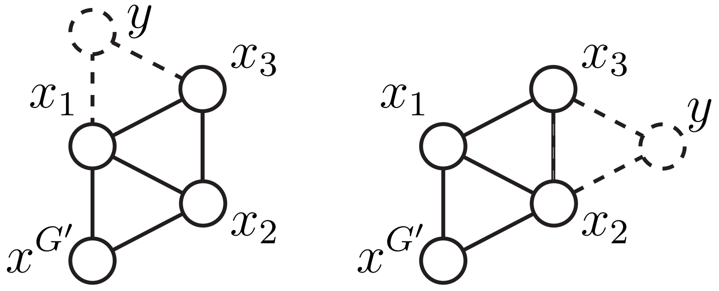

Fig 4. Extensions of the $k-1$-node graphlet G' to G_a. G_7 can be extended to G_{24} by attaching *y* to either x_1 and x_3 or to x_2 and x_3, hence $\mathcal{E} = \{\{x_1, x_3\}, \{x_2, x_3\}\}$. With respect to Eq (2), x_1 and x_3 (as well as x_2 and x_3) have one common neighbour in G', so $c(E^{G'}) = 1$ and $\sum_{E \in \mathcal{E}} (c(E^H) - c(E^{G'})) = (c(x_1, x_3) - 1) + (c(x_2, x_3) - 1)$. The right side in Eq (1) sums this over all unique occurrences of $G' = G_7$ with *x* in O_{12} within *H*.

doi:10.1371/journal.pone.0171428.g004

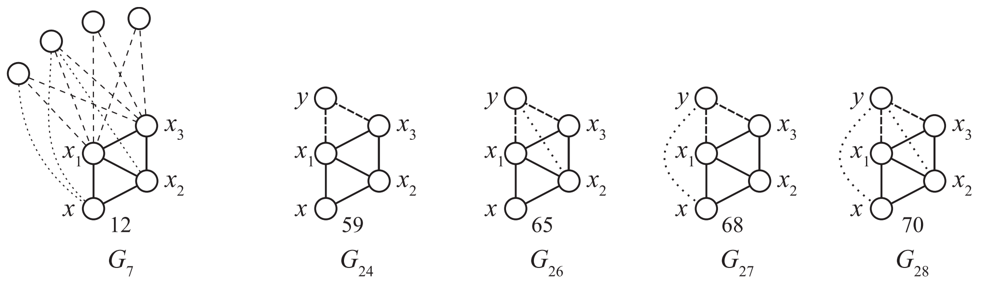

Fig 5. Extended graphlets with extra edges. Dashed lines represent edges required by condition $y \in N(x_1, x_3)$. Dotted lines represent possible extra edges that make the resulting induced graph isomorphic to G_{26}, G_{68} or G_{70} instead of G_{24}, with x in orbits o_{65}, o_{68} or o_{70} instead of o_{59}; these orbits appear on the left-hand side of Eq (1).

doi:10.1371/journal.pone.0171428.g005

relation. To compute the total orbit count o_i for x, we sum Eq (2) over all occurences of G' in H (Fig 4).

Condition $y \in N(E^H)$ (for some $E^H \in \mathcal{E}^H$) is not sufficient. Node y can also be connected to any of the other $k - 1 - |E|$ ($E \in \mathcal{E}$) nodes in G'^H, resulting in $2^{k-1-|E|}$ possible graphlets and orbits for x. The counts for these orbits are summed on the left side of the relation (Fig 5).

While adding the orbit counts on the left-hand side, we need to account for the over-counts, that is, the number of times that Eq (2) counts the same occurrence of $G^* = G_{m(p)}$ (the graphlet containing the orbit p) within H. G^* is obtained by extending G' with $y \in N(E)$. The coefficients on the left-hand side thus equals the number of ways in which G' can be extended to G^* with a fixed node x (Fig 6).

In general, we have to consider all induced occurrences of G' in G^* (with a fixed point x), which is the same as considering nodes $z \in V_{G^*}$ whose removal results in G' with $x^{G'}$ in orbit O_p. For every such case we increase the coefficient by the number of extensions $E \in \mathcal{E}$ such that node z is connected to the extension nodes, i.e. $N(z) \supseteq E^{G^*}$.

The general procedure for relating the orbit count o_i with counts of orbits with higher indices is outlined in Algorithm 1.

Algorithm 1 Derive an equation for orbit O_i.

```
function EQUATION($O_i$)
  $G_a \leftarrow G_{m(i)}$                          ▷ Let $G_a$ be the graphlet that contains $O_i$
  $x \in O_i$                                        ▷ and $x$ one of the nodes in $O_i$.
  $y \leftarrow$ SELECTY($x$)                        ▷ Alg. 2—Pick node $y$ such that $y \neq x$
  $G' \leftarrow G_a \setminus y$                    ▷ and $G'$ is a connected graph.
  $r \leftarrow \sum_{G'^H:\ x^H \in O_i}$ (Eq (2))  ▷ The right side of equation sums over.
                                                     ▷ all occurrences of $G'$ in the host graph.
  for $p \in \mathcal{O}$                            ▷ Construct left side of the equation.
    $G^* = G_{m(p)}$                                 ▷ Graphlet containing orbit $O_p$.
    $f_p \leftarrow 0$                               ▷ Overcount coefficient of orbit $O_p$.
    for $z \in G^*$: $(G^* \setminus z) \cong G'$    ▷ Is $z$ in the same orbit as $y$
                                                     ▷ given a fixed point $x$?
      $f_p \leftarrow f_p + |\{E \in \mathcal{E} : N(z) \supseteq E\}|$   ▷ By how many extensions?
    endfor
  endfor
  $l \leftarrow \sum_p f_p \cdot o_p$                ▷ Left side is a weighted sum of orbit counts.
  return equation $l = r$
end function
```

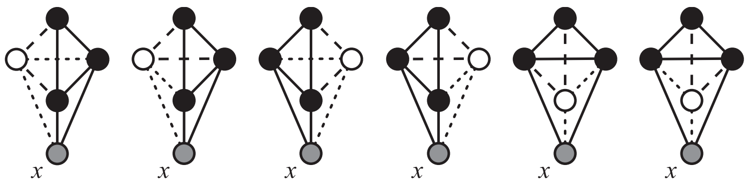

Fig 6. Symmetries and coefficients. The coefficient at o_{70} in Eq (1) is 6 because each induced embedding of $G^* = G_{m(70)} = G_{28}$ in H is counted six times. First, there are three different choices for y (the white node), therefore each embedding of G_{28} results in three corresponding appearances of graphlet G_7 (solid edges) with x in orbit O_{12}. For each occurrence, both extensions (dashed edges) lead to x in orbit O_{70}.

doi:10.1371/journal.pone.0171428.g006

Additional constraints on selection of y

In the preceding derivation, the only limitation on selection of vertex y was that the remaining graphlet is still connected. Different choices of y yield different equations. With the coefficients independent of the host graph and known in advance, the time consuming part of using these equations to calculate orbit counts is the computation of the right-hand side terms. To speed it up, we impose some additional constraints on the choice of the node y: the restraints will be such that the right-hand sides will contain only the counts $c(S)$ in which either $|S| < k - 2$, or equal $|S| = k - 2$ with the nodes in S forming a connected subgraph of G_k. This will allow pre-calculation and caching of all $c(S)$ needed for computation of right-hand sides.

For efficient precomputation, vertex $y \neq x$ must meet the following criteria:

1. $d(y) \leq k - 2$,
2. $G \backslash \{y\}$ is a connected graph,
3. if $d(y) = k - 2$, the neighbours of y induce a connected graph,

where $d(y)$ represents the degree of y.

We will prove that such a vertex exists in any graphlet $k \geq 4$ and all possible x, except for complete graphlets (all vertices violate the first condition) and for the cycle on four points, C_4 (all vertices violate the last condition).

Let L_i represent the set of vertices at a distance i from x (see Fig 7). Let l_i be the vertex in L_i with the smallest degree. Let L_u be the last non-empty set, and, accordingly, l_u the vertex with the smallest degree among the vertices farthest from x. We will show that l_u fulfils the conditions in most cases, except in some for which we can use l_{u-1}.

Algorithm 2 Selection of node y.

```
function SELECTY($x$)
  $u$← distance to the furthest node from $x$
  $L_u$← set of nodes at distance $u$
  $L_{u-1}$← set of nodes at distance $u-1$
  if $l_u$ satisfies the criteria then
    $y$← lowest degree node from $L_u$
  else
    $y$← lowest degree node from $L_{u-1}$
  endif
  return $y$
end function
```

Each node $v \in L_i$ $(i > 0)$ has at least one neighbour in L_{i-1}, since the first node in the shortest path from v to x belongs to L_{i-1}. Consequently, all L_i for $i \leq u$ are non-empty. Note also

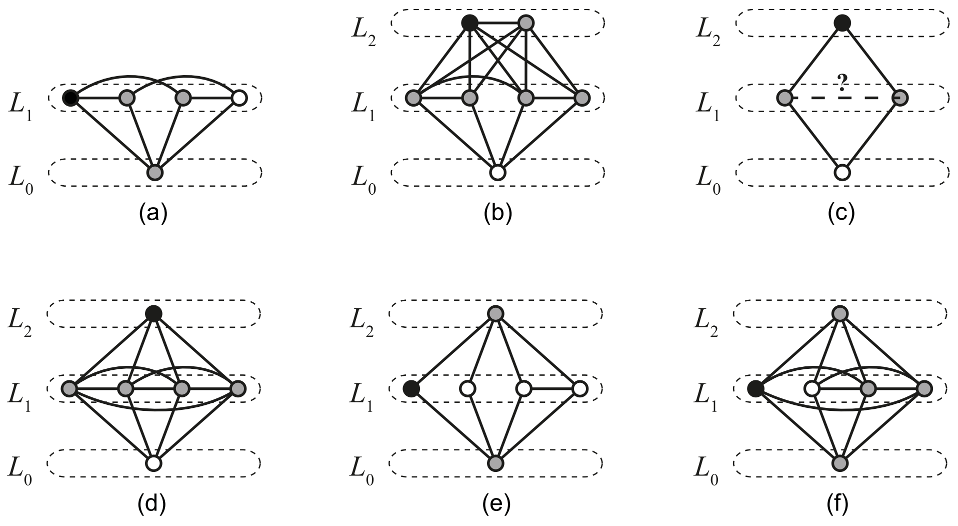

Fig 7. Illustrations of different cases. Node at the bottom level L_0 represents node x. The selected node y is colored black and its neighbours are indicated with a gray color. (a) $u = 1$. (b) $u = 2 \wedge |L_2| > 1$. (c) $u = 2 \wedge |L_2| = 1 \wedge k = 4$. (d) $u = 2 \wedge |L_2| = 1 \wedge k \geq 5 \wedge d(l_1) = k - 1$. (e) $u = 2 \wedge |L_2| = 1 \wedge k \geq 5 \wedge d(l_1) < k - 2$. (f) $u = 2 \wedge |L_2| = 1 \wedge k \geq 5 \wedge d(l_1) = k - 2$.

doi:10.1371/journal.pone.0171428.g007

that vertices from L_i are adjacent only to vertices L_{i-1}, L_i and L_{i+1} since any edge from L_i to L_j with $j < i - 1$ would imply a shorter path from the node in L_i to x.

Lemma 1 *A vertex $v \in L_i$ can have a degree of at most k–i.*

The vertex v is not adjacent to any vertex in L_j, where $0 \leq j < i - 1$. Since L_j are non-empty, there are at least $i - 1$ non-adjacent vertices, so the degree of v is at most $k - 1 - (i - 1) = k - i$.

As a consequence, if $d(l_u) = k - 2$, then $u \leq 2$.

Lemma 2 *If $d(v) = k - 2$ and $v \in L_2$, then v is adjacent to all vertices except x.*

A vertex in L_2 is not adjacent to x by definition of L_2, and there are no loops, so to have a degree of $k - 2$ it must be adjacent to all other vertices.

Lemma 3 *If G is not a complete graph, then $d(l_u) \leq k - 2$*

For $u > 1$, the lemma follows directly from Lemma 1, so we only need to prove it for $u = 1$. For contrapositive, assume that $d(l_1) = k - 1$. Since l_1 has the smallest degree in L_1, all vertices in L_1 have a degree of $k - 1$. Furthermore, x has a degree of $k - 1$ since all vertices in L_1 are adjacent to it by definition of L_1. Hence, G is a complete graph.

The last lemma ensures that the farthest vertex with the lowest degree, l_u, fulfills the first condition. It also fulfills the second one: all vertices are connected to x with the shortest paths of lengths at most u, which cannot include l_u, thus the removal of l_u keeps them connected (at least) via x.

We will prove that l_u also fulfills the third condition, except for one special case ($d(l_u) = k - 2$ and $u = 2$ and $|L_2| = 1$ and $k \geq 5$ and $d(l_1) \leq k - 2$), in which we choose another suitable vertex. We will consider six different cases, which are (except for the trivial first case) illustrated in Fig 7.

1. $d(l_u) < k - 2$: Condition (iii) does not apply.
2. $d(l_u) = k - 2$ **and** $u = 1$: Since all vertices except x are in L_1, they are adjacent to x (Fig 7a). x itself is among the neighbours of l_1, hence neighbours of l_1 are connected through x.

3. $d(l_u) = k - 2$ **and** $u = 2$ **and** $|L_2| > 1$: Since l_2 is the vertex with the smallest degree in L_2, all vertices in L_2 must have a degree of $k - 2$ and are adjacent to all vertices except x by Lemma 2 (Fig 7b). The neighbour set of l_2 is $L_1 \cup L_2 \backslash l_2$. Since $|L_2| > 1$, there exists a vertex $v \in L_2$ s.t. $v \neq l_2$. v is adjacent to all nodes from $L_2 \cup L_1$, therefore $L_1 \cup L_2 \backslash l_2$ is connected.

4. $d(l_u) = k - 2$ **and** $u = 2$ **and** $|L_2| = 1$ **and** $k = 4$: L_1 contains two vertices; both are adjacent to x by definition of L_1 and to l_2 since $d(l_2) = k - 2 = 2$. l_2 is not adjacent to x by definition of L_2. This leaves only two possible graphs, the cycle C_4 and a diamond (Fig 7c). For the former, the vertex with the required properties does not exist. For the diamond, l_u fulfills all three conditions.

5. $d(l_u) = k - 2$ **and** $u = 2$ **and** $|L_2| = 1$ **and** $k \geq 5$ **and** $d(l_1) = k - 1$: The neighbour set of l_2 is the entire L_1 (Fig 7d). Since the smallest degree in L_1 is $k - 1$, L_1 is a complete graph and therefore connected.

6. $d(l_u) = k - 2$ **and** $u = 2$ **and** $|L_2| = 1$ **and** $k \geq 5$ **and** $d(l_1) \leq k - 2$: The graph consists of $L_0 = \{x\}$, $L_2 = \{l_2\}$, and of L_1 with at least 3 vertices since $k \geq 5$ (Fig 7e). All nodes in L_1 are adjacent to x by definition of L_1 and to l_2 by Lemma 2 since we assume $d(l_2) = k - 2$.
In this case, l_u does not always fulfil the conditions, so we choose the lowest degree vertex from L_1, l_1. It fulfils the condition (i) by assumptions of this special case. As for condition (ii), the graph $G \backslash l_1$ is still connected since all points in L_1 are adjacent to x. Since $|L_1| \geq 3$ and $d(l_u) = d(l_2) = k - 2$, vertices x and l_2 are connected through the remaining vertices in $L_1 \backslash l_1$. Condition (iii) needs to be verified just for the case when $d(l_1) = k - 2$ (Fig 7f). The neighbours of l_1 include x, l_2 and all vertices from L_1 except one. Since $|L_1| \geq 3$, L_1 must include at least one other neighbour of l_1, which thus connects x and l_u.

We have covered all possible cases: the degree of l_u cannot exceed $k - 2$ due to Lemma 3 (assuming the graph is not complete), and when $d(l_u) = k - 2$, u cannot exceed 2 due to Lemma 1.

We have proven that the vertex with the smallest degree in L_u, l_u, fulfills the given conditions in all cases except when $d(l_u) = k - 2$ and $u = 2$ and $|L_2| = 1$ and $k \geq 5$ and $d(l_1) \leq k - 2$. In the latter case, the conditions are fulfilled by l_1. Complete graphlets and C_4 are handled differently.

Equation for a cycle on 4 nodes

A cycle on 4 nodes, C_4, is treated separately since there is no suitable node y with the required properties. For C_4 (O_8) we choose one of the nodes adjacent to x for the role of y, resulting in

$$2o_8 + 2o_{12} = \sum_{\substack{x_1, x_2 : \\ x, x_2 \in N(x_1), \\ H[\{x, x_1, x_2\}] \cong G_1}} [c(x, x_2) - 1]. \tag{3}$$

Note that this choice violates the third condition that the neighbours of y should induce a connected graph. The Eq (3) contains a term on the right side that corresponds to the number of common neighbours of node x and some other node x_2 at distance 2 from x. The algorithm stores precomputed values for all such pairs x and x_2, which would require $O(nd^2)$ space and increase the algorithm's space complexity. However, we can still handle this case without consequences for the time and space complexity. We achieve this by reusing $O(n)$ space and recomputing the number of common neighbours every time the algorithm starts a computation of orbit counts for a different node of interest x. This optimization is necessary to keep the space requirement at $O(nd)$ for counting four-node graphlets and does not impact the time complexity.

System of equations

In the constructed system of equations, each orbit is related to orbits from graphlets with higher number of edges. This yields a triangular system of equations: we have one equation for every orbit O and these equations include as terms only the orbit O and other orbits belonging to graphlets with a larger number of edges (*e.g.*, the orbit 59 in Eq (1) is related to orbits 65, 68 and 70).

The system has $\mathcal{O}_k - 1$ linear equations for $\mathcal{O}_k$ orbit counts. To solve it, one orbit count must be enumerated directly. The networks that we encounter in practical applications are usually sparse, which makes the complete graphlet (clique) a good candidate. Because of its rarity and symmetricity, we can efficiently restrict the enumeration.

Enumerating the orbit in the graph with the largest number of edges also simplifies solving the given triangular system of equations.

Extension to edge orbits

Edge orbits (Fig 8) can be defined in a similar way as node orbits. We can use the same approach for setting up the corresponding system of equations. Since the system does not refer to a single x but to an edge (x_a, x_b), the selected node y must not coincide with either of these endpoints. We can set $x = x_a$ and show that we can always choose a node $y \neq x_b$ with the required properties.

Since x_b is always in L_1, we have to analyze only the cases where we choose y from L_1. This happens in cases 1, 2, and 6 from the proof in section about additional constraints on y. We need to check that there at least two suitable vertices for y, so if one of them is x_b, the other is chosen as y.

1. $d(l_u) < k - 2$: We need to consider only the case when $u = 1$. Since $d(l_u) < k - 2$, all vertices in L_1 satisfy condition (i). The remaining graph is connected through x (condition (ii)), and condition (iii) does not apply.

2. $d(l_u) = k - 2$ **and** $u = 1$: Recall that the graph is not complete. Since all vertices in L_1 are connected to x, there must be at least one pair in L_1 that is not connected and thus has a degree of $k - 2$. These two vertices satisfy condition (i). Conditions (ii) and (iii) again hold since all vertices are connected through x.

6. $d(l_u) = k - 2$ **and** $u = 2$ **and** $|L_2| = 1$ **and** $k \geq 5$ **and** $d(l_1) \leq k - 2$: Since the vertex in L_2 has a degree of $k - 2$, it is connected to all vertices in L_1; nodes in L_1 are connected to x. The

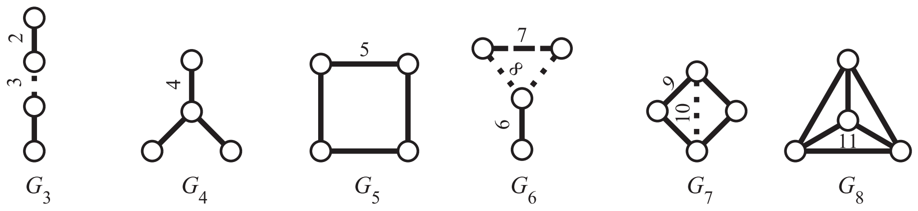

Fig 8. Edge orbits in four-node graphlets. Different styles of lines within a graphlet indicate edge orbits.

doi:10.1371/journal.pone.0171428.g008

nodes in L_1 do not induce a complete graph ($d(l_1) \leq k - 2$), so there must again exist a disconnected pair in L_1, which satisfies all conditions like in above case 2.

For C_4, one of the nodes in L_1 is x_b and the other is y.

Algorithm

Coefficients on the left-hand side of the relations are related to symmetry properties of the graphlets and not to the graph H. The terms on the right sides of equations depend on the host graph H. Their computation requires enumeration of all graphlets of size $k - 1$ and adding up their possible extensions.

The first step is pre-computation and storing of $c(\mathcal{S})$ for all subsets S with up to $k - 3$ vertices and for all connected subsets of $k - 2$ vertices. These conditions match the criteria for selection of y, so the precomputed values $c(\mathcal{S})$ represent the terms in the sum on the right-hand sides of equations.

This is followed by direct enumeration of cliques with k vertices. This enumeration does not have to be extremely fast, but just fast enough not to dominate the time complexity of the entire graphlet counting algorithm. For this purpose we can employ some of the approaches to listing cliques [20, 21].

Following this precomputation, the next two steps are repeated for each vertex $x \in V$.

Computation of sums on the right-hand sides of equations. Computation is implemented as enumeration of $(k - 1)$-node graphlets touching x, as specified by the conditions under the sums. For each graphlet, the terms in the sum consist of the counts $c(\mathcal{S})$ precomputed in the previous step.

For $k \leq 5$, the number of graphlets with $k - 1$ nodes is small, so it is feasible to design efficient individual procedures for enumerating them. These procedures involve early pruning of non-viable candidates and completely avoiding any isomorphism testing. Medium-sized graphlets ($k = 5$ or 6) require graphlet recognition of enumerated connected subgraphs, however these patterns can be efficiently distinguished with the use of some trivial invariants such as a *degree sequence*. Enumeration of larger graphlets would benefit from efficient methods for isomorphism testing.

Solving the system of equations. The system is triangular, with each equation relating one orbit to those with larger number of edges, Since the count for the highest orbit, which belongs to the clique, is computed by direct enumeration, the system can be solved by decreasing orbit indices.

All orbit counts, coefficients and free terms are integers, thus the computation is numerically stable.

Time- and space-complexity

We will analyze the worst-case complexity and the expected complexity on random Erdős-Rényi graphs, followed by empirical verification.

Worst-case complexity. We will evaluate the worst-case time complexity of the algorithm in terms of the number of nodes (n) and the maximum degree of a node (d) in the host graph. We treat the size of the graphlets, k, as a constant. We assume that the graph is stored as a list of adjacent nodes together with a hash table for checking whether two nodes are connected in constant time. The algorithm consists of four steps.

Precomputation of common neighbours. We need to precompute the number of common neighbours of sets of $k - 3$ or fewer nodes and of connected sets of $k - 2$ nodes to efficiently construct right sides of our equations. To achieve this we enumerate all subsets of $k - 2$ or fewer neighbours for every node. This results in time complexity $O(nd^{k-2})$. Storing the number of common neighbours of sets of at most $k - 3$ nodes with the above method requires $O(nd^{k-3})$ space. Because we request that in the case of $k - 2$ nodes, these nodes induce a connected subgraph, we can limit their number to the number of $(k - 2)$-node induced connected subgraphs, which is also $O(nd^{k-3})$.

Enumeration of cliques. We will refer to the time complexity of counting k-node cliques in this step as $O(T_k)$. A worst-case time complexity is $O(nd^{k-1})$ and requires constant space. However, this enumeration can be implemented very efficiently in practical applications on sparse networks that contain few cliques.

Enumerating all $(k - 1)$-node graphlets and counting their extensions. This step computes the right sides of the system of equations. It requires constant space, since the space is reused for each vertex, and runs in $O(nd^{k-2})$ time needed for enumeration of $k - 1$-node graphlets.

Solving the system of equations. The system of equations is independent of the host graph and requires constant time and space.

Overall, the algorithm has a $O(nd^{k-2} + T_k)$ time complexity while requiring $O(nd^{k-3})$ space. In the worst case, the time complexity is the same as that of a simple exhaustive enumeration method, $O(T_k) = O(nd^{k-1})$. However, the term T_k is much smaller in practice.

Expected time complexity in random graphs. Although the worst-case time complexity of the algorithm is equal to that of brute-force enumeration, the actual performance on real-world networks and on random graphs is much better. We analyzed the expected time complexity on random Erdős-Rényi graphs with n nodes and edge probability p. Throughout this analysis we will assume that $np > 1$, otherwise the graph is likely to have more than one component which can be processed independently.

The precomputation consists of iterating over central nodes, enumerating all sets of $l \leq k - 2$ neighbours and incrementing the number of common neighbours of the leaf nodes. The l nodes have to be connected to the central node, which happens with probability p^l. The expected time complexity of this step is $O(n \sum_{l=1}^{k-2} n^l p^l)$. Assuming $np > 1$, we can simplify it to $O(n^{k-1} p^{k-2})$.

In the second step, the algorithm enumerates all subgraphs with $k - 1$ nodes. It does so incrementally by first enumerating smaller connected subgraphs of size l and extending them to larger connected subgraphs. The expected time complexity is therefore proportional to the expected number of connected subgraphs with $l \leq k - 1$ nodes. We need to estimate the probability that a set of l nodes induces a connected subgraph. We can view the process of building every such subgraph by consecutively attaching a new node to at least one of the existing nodes. This of course will overestimate the number of connected subgraphs by some constant because every such subgraph can be built in several different orders of attaching nodes. The probability that an edge exists from some newly added node to at least one of the i existing nodes is $1 - (1 - p)^i$. The expected number of enumerated subgraphs is therefore $O(\sum_{l=1}^{k-1} n^l \prod_{i=1}^{l-1} (1 - (1 - p)^i)) = O(\sum_{l=1}^{k-1} n^l p^{l-1})$. Assuming $np > 1$, the expected time complexity is $O(n^{k-1} p^{k-2})$.

The total expected time complexity for setting-up the system of equations in Erdős-Rényi graphs with n nodes and edge probability p is thus $O(n^{k-1} p^{k-2})$. In practice, we observe graphlets with 4 and 5 nodes. The expected time complexities for these cases are $O(n^3 p^2)$ and $O(n^4 p^3)$, respectively.

Table 2. Comparison of run times for counting node- and edge-orbits of 4-node graphlets.

	four-node graphlets			
edges [thousands]	**50**	**100**	**150**	**200**
bruteforce [s]	27.24	236.83	811.52	1876.98
node-orbits [s]	0.70	2.40	6.16	14.01
edge-orbits [s]	0.69	2.33	6.12	14.21

doi:10.1371/journal.pone.0171428.t002

Empirical evaluation of time complexity. We evaluated the performance of our algorithm for counting 4- and 5-node graphlets on random Erdős-Rényi graphs.

We measured the time needed for counting node- and edge-orbits with the Orca algorithm and compared it to a bruteforce enumeration. For the latter we used an implementation from GraphCrunch. Orca outperforms exhaustive enumeration by an order of magnitude (Tables 2 and 3). The running times for counting node-orbits and edge-orbits are practically the same in the case of counting 4- or 5-node graphlets in random graphs with 1 000 nodes and of increasing density. The size of the graphs ($n = 1000$) was chosen arbitrarily to put the run times in the range of a couple of seconds. In the remainder of this section we focus on counting node-orbits.

Second, we compare the running times of counting orbits of 4-node and 5-node graphlets on random graphs with 10 000 nodes and up to 800 000 edges. These graphs are sparse as the number of edges represents only about 1.6% of all possible edges; as such they represent a realistic case of large network analysis. A logarithmic plot of execution times in Fig 9 shows a polynomial dependence on the size of graphlets. Dotted lines correspond to a bruteforce enumeration method and solid lines to our Orca algorithm. The line corresponding to bruteforce counting of 4-node graphlets aligns nicely with counting 5-node graphlets using Orca. This reflects the enumeration of 4-node graphlets used to compute 5-node graphlet counts. The only difference is by a constant factor.

The slope of the Orca's lines should be 2 and 3, respectively, according to the expected time complexities ($O(n^3 p^2)$, $O(n^4 p^3)$). However, this is clearly not the case in Fig 9. Further experiments show that this is the result of CPU cache misses when accessing the precomputed lookup tables. We performed a similar experiment with disabled CPU cache. Because of the slowdown, we decreased the number of nodes to 1 000 and maintained average degree of nodes. The measurements with disabled CPU cache in Fig 10 line up with the expected slopes of 2 and 3.

Finally, we probed for the region in which the enumeration of cliques begins dominating the time complexity. We performed the experiment for counting 4-node graphlets in graphs with 1 000 nodes and increasing edge probabilities p. In Fig 11 the plot follows a straight line up to around $p = 0.07$ and another steeper line from $p = 0.3$ onwards. This is consistent with

Table 3. Comparison of run times for counting node- and edge-orbits of 5-node graphlets.

	five-node graphlets				
edges [thousands]	**5**	**10**	**15**	**20**	**25**
bruteforce [s]	0.87	12.75	61.57	191.94	461.53
node-orbits [s]	0.23	1.03	2.93	6.85	13.88
edge-orbits [s]	0.22	0.91	2.54	5.90	11.78

doi:10.1371/journal.pone.0171428.t003

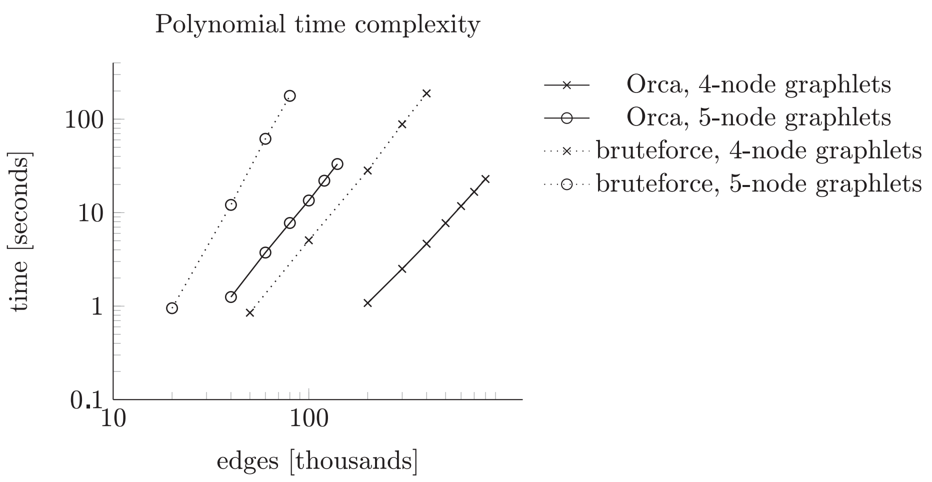

Fig 9. Polynomial time complexity. Comparison of counting times for orbits of 4- and 5-node graphlets on sparse graphs with 10000 nodes and varying densities.

doi:10.1371/journal.pone.0171428.g009

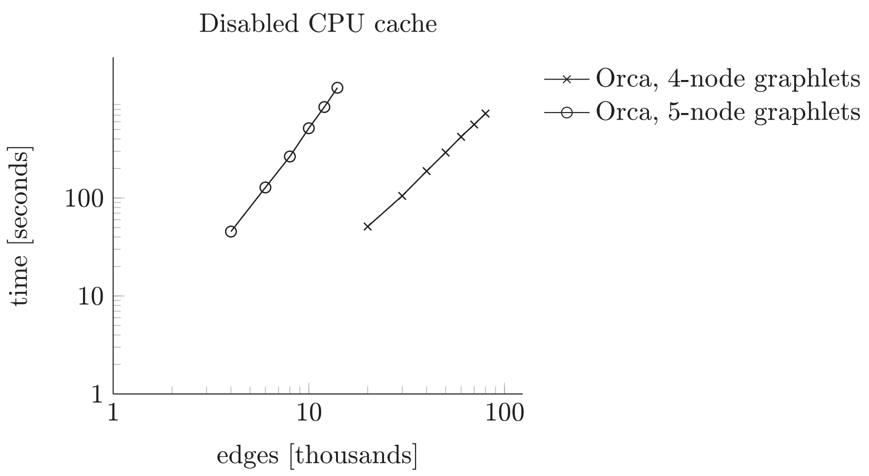

Fig 10. Running times with disabled CPU cache.

doi:10.1371/journal.pone.0171428.g010

the contribution of the step of enumerating cliques. Random sparse graphs contain fewer cliques whose enumeration is efficient and does not significantly affect the running time. However, as the graphs become denser, this becomes the bottleneck of the algorithm.

Final remarks

The source code of the algorithm in C++, which computes the node and edge orbits for $k = 4$ and $k = 5$, along with the randomly generated data used in experiments is available at https://github.com/thocevar/orca. The corresponding R package `orca` is also available on CRAN. Parts of this algorithm that have been presented previously are also already included in the GraphCrunch package [22].

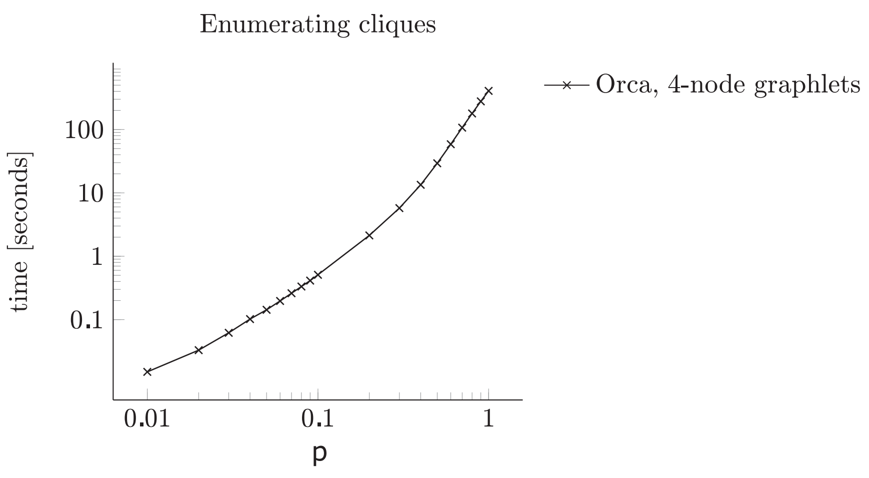

Fig 11. Enumerating cliques. The effect of enumerating cliques on running times in random graphs of increasing density.

doi:10.1371/journal.pone.0171428.g011

Author Contributions

Conceptualization: TH JD.

Data curation: TH.

Formal analysis: TH JD.

Funding acquisition: JD.

Investigation: TH.

Methodology: TH.

Project administration: JD.

Resources: JD.

Software: TH.

Supervision: JD.

Validation: TH.

Visualization: TH JD.

Writing – original draft: TH JD.

Writing – review & editing: TH JD.

References

1. Kuramochi M, Karypis G. Frequent subgraph discovery. In: Proceedings 2001 IEEE International Conference on Data Mining. IEEE Comput. Soc; 2001. p. 313–320. Available from: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=989534
2. Backstrom L, Leskovec J. Supervised random walks. In: Proceedings of the fourth ACM international conference on Web search and data mining—WSDM'11. New York, New York, USA: ACM Press; 2011. p. 635–644. Available from: http://portal.acm.org/citation.cfm?doid=1935826.1935914

3. Liben-Nowell D, Kleinberg J. The link-prediction problem for social networks. Journal of the American Society for Information Science and Technology. 2007; 58(7):1019–1031. doi: 10.1002/asi.20591

4. Milenković T, Przulj N. Uncovering biological network function via graphlet degree signatures. Cancer informatics. 2008; 6:257–273. PMID: 19259413

5. Ralaivola L, Swamidass SJ, Saigo H, Baldi P. Graph kernels for chemical informatics. Neural Networks. 2005; 18(8):1093–1110. doi: 10.1016/j.neunet.2005.07.009 PMID: 16157471

6. Przulj N, Corneil DG, Jurisica I. Modeling interactome: scale-free or geometric? Bioinformatics. 2004; 20(18):3508–3515. doi: 10.1093/bioinformatics/bth436 PMID: 15284103

7. Przulj N. Biological network comparison using graphlet degree distribution. Bioinformatics. 2007; 23 (2):177–183. doi: 10.1093/bioinformatics/btl301 PMID: 17237089

8. Itai A, Rodeh M. Finding a Minimum Circuit in a Graph. SIAM Journal on Computing. 1978; 7(4):413–423. doi: 10.1137/0207033

9. Nešetřil J, Poljak S. On the complexity of the subgraph problem. Commentationes Mathematicae Universitatis Carolinae. 1985; 26(2):415–419.

10. Marcus D, Shavitt Y. RAGE – A rapid graphlet enumerator for large networks. Computer Networks. 2012; 56(2):810–819. doi: 10.1016/j.comnet.2011.08.019

11. Kloks T, Kratsch D, Müller H. Finding and counting small induced subgraphs efficiently. Information Processing Letters. 2000; 74(3-4):115–121. doi: 10.1016/S0020-0190(00)00047-8

12. Kowaluk M, Lingas A, Lundell EM. Counting and Detecting Small Subgraphs via Equations. SIAM Journal on Discrete Mathematics. 2013; 27(2):892–909. doi: 10.1137/110859798

13. Floderus P, Kowaluk M, Lingas A, Lundell Em. Induced Subgraph Isomorphism: Are Some Patterns Substantially Easier Than Others? In: 18th Annual International Computing and Combinatorics Conference; 2012. p. 37–48. Available from: http://link.springer.com/10.1007/978-3-642-32241-9_4

14. Vassilevska V, Williams R. Finding, minimizing, and counting weighted subgraphs. In: Proceedings of the 41st annual ACM symposium on Symposium on theory of computing—STOC'09. New York, New York, USA: ACM Press; 2009. p. 455–464. Available from: http://portal.acm.org/citation.cfm?doid=1536414.1536477

15. Alon N, Yuster R, Zwick U. Finding and counting given length cycles. Algorithmica. 1997; 17(3):209–223. doi: 10.1007/BF02523189

16. Alon N, Yuster R, Zwick U. Color-coding. Journal of the ACM. 1995; 42(4):844–856. doi: 10.1145/210332.210337

17. Alon N, Dao P, Hajirasouliha I, Hormozdiari F, Sahinalp SC. Biomolecular network motif counting and discovery by color coding. Bioinformatics. 2008; 24(13):241–249. doi: 10.1093/bioinformatics/btn163

18. Melckenbeeck I, Audenaert P, Michoel T, Colle D, Pickavet M. An Algorithm to Automatically Generate the Combinatorial Orbit Counting Equations. PLOS ONE. 2016; 11(1):1–19. doi: 10.1371/journal.pone.0147078 PMID: 26797021

19. Hocevar T, Demsar J. A combinatorial approach to graphlet counting. Bioinformatics. 2014; 30(4):559–565. doi: 10.1093/bioinformatics/btt717 PMID: 24336411

20. Bron C, Kerbosch J. Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM. 1973; 16(9):575–577. doi: 10.1145/362342.362367

21. Eppstein D, Löffler M, Strash D. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In: Algorithms and Computation. vol. 6506 of Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010. p. 403–414. Available from: http://www.springerlink.com/index/10.1007/978-3-642-17517-6

22. Milenković T, Lai J, Pržulj N. GraphCrunch: A tool for large network analyses. BMC Bioinformatics. 2008; 9(1). doi: 10.1186/1471-2105-9-70 PMID: 18230190