



HAL
open science

A General Purpose Algorithm for Counting Simple Cycles and Simple Paths of Any Length

P.-L Giscard, Nils Kriege, Richard Wilson

► **To cite this version:**

P.-L Giscard, Nils Kriege, Richard Wilson. A General Purpose Algorithm for Counting Simple Cycles and Simple Paths of Any Length. *Algorithmica*, 2019, 81 (7), pp.2716-2737. 10.1007/s00453-019-00552-1 . hal-03597440

HAL Id: hal-03597440

<https://hal.science/hal-03597440v1>

Submitted on 4 Mar 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A GENERAL PURPOSE ALGORITHM FOR COUNTING SIMPLE CYCLES AND SIMPLE PATHS OF ANY LENGTH*

PIERRE-LOUIS GISCARD[†], NILS KRIEGE[‡], AND RICHARD C. WILSON[†]

Abstract. We describe a general purpose algorithm for counting simple cycles and simple paths of any length ℓ on a (weighted di)graph on N vertices and M edges, achieving a time complexity of $O(N + M + (\ell^\omega + \ell\Delta)|S_\ell|)$. In this expression, $|S_\ell|$ is the number of (weakly) connected induced subgraphs of G on at most ℓ vertices, Δ is the maximum degree of any vertex and ω is the exponent of matrix multiplication. We compare the algorithm complexity both theoretically and experimentally with most of the existing algorithms for the same task. These comparisons show that the algorithm described here is the best general purpose algorithm for the class of graphs where $(\ell^{\omega-1}\Delta^{-1}+1)|S_\ell| \leq |\text{Cycle}_\ell|$, with $|\text{Cycle}_\ell|$ the total number of simple cycles of length at most ℓ , including backtracks and self-loops. On Erdős-Rényi random graphs, we find empirically that this happens when the edge probability is larger than circa $4/N$. In addition, we show that some real-world networks also belong to this class. Finally, the algorithm permits the enumeration of simple cycles and simple paths on networks where vertices are labeled from an alphabet on n letters with a time complexity of $O(N + M + (n^\ell \ell^\omega + \ell\Delta)|S_\ell|)$. A Matlab implementation of the algorithm proposed here is available for download.

Key words. Simple cycles; simple paths; self-avoiding walks; self-avoiding polygons; elementary circuits; connected induced subgraphs; networks; graphs; digraphs; labeled graphs

AMS subject classifications. 68Q25, 68W40, 05C30, 05C38, 05C22

1. Introduction. Counting Hamiltonian cycles and, more generally, all simple cycles passing through a given vertex is a #P-complete problem [41, 8]. The same classification holds for the problem of counting simple paths with fixed endpoints. Unsurprisingly, the best existing algorithms for counting such cycles have time complexities $O(2^N \text{poly}(N))$, which scales exponentially with the number N of vertices on the graph. Under the exponential time hypothesis [23], this exponential scaling is, in principle, the best possible.

Although evaluating the time complexity of an algorithm in the worst case scenario is of paramount importance for the classification of algorithmic performance, it is of little relevance to applications which differ significantly from this scenario. This is precisely the case when counting or enumerating simple cycles or simple paths. Real-world networks, be they from sociology, biology or chemistry, are typically very sparse. At the opposite, the worst case scenarios for this task—the complete graphs—are dense and counting or finding cycles and paths of any kind on them presents no interest, in particular since everything is already known analytically. An algorithm counting simple cycles and paths that is especially tailored for sparse graphs is therefore highly desirable. In particular, we expect the graph sparsity, or some quantity related to it, to be a relevant parameter when qualifying the complexity of such an algorithm.

In addition to these considerations, we observe that one rarely needs to count *all* simple cycles or paths. Rather it is typically sufficient to count only those whose

*Submitted to the editors DATE.

Funding: P.-L. Giscard acknowledges financial support from the Royal Commission for the Exhibition of 1851. N. Kriege is supported by the German Science Foundation (DFG) within the Collaborative Research Center SFB 876 “Providing Information by Resource- Constrained Data Analysis”, project A6 “Resource-efficient Graph Mining”.

[†]Department of Computer Science, University of York (pierre-louis.giscard@york.ac.uk, richard.wilson@york.ac.uk)

[‡]Department of Computer Science, TU Dortmund (nils.kriege@tu-dortmund.de)

length does not exceed some maximum value ℓ , usually much smaller than the graph size N . Yet, even with these restrictions, the problem of counting simple cycles or simple paths is known to be difficult:

THEOREM 21 IN FLUM AND GROHE [16]. *Counting simple cycles and simple paths of length ℓ on both directed and undirected graphs, parameterized by ℓ , is $\#W[1]$ -complete.*

The complexity classes $\#W[t]$, $t \geq 1$, introduced by Flum and Grohe, are relevant for parameterized counting problems corresponding to the classes of the W-hierarchy [13] which, in turn, qualify the difficulty of parametrized decision problems according to the type of circuits needed to determine them. Importantly, the class $\#W[1]$ is believed to *strictly* contain the class $\#W[0]$ of all fixed-parameter tractable (FPT) counting problems. We recall that a counting problem P with input x is said to be *fixed-parameter tractable* if there is a computable function f of the parameter k , a constant c and an algorithm solving P in $f(k) \text{poly}(|x|)$ steps. In this expression, $|x|$ designates the size of the input [16, 20]. For the sake of simplicity, an algorithm achieving a $f(k) \text{poly}(|x|)$ time complexity will be said to be FPT.

In this work, we describe a novel general purpose algorithm for the task of counting simple cycles and simple paths of fixed length ℓ and determine its time complexity:

THEOREM 1 (Algorithm for cycle and path counting). *Let $G = (V, E)$ be a graph, possibly directed, on N vertices and M edges. Let $|S_\ell|$ be the number of connected induced subgraphs of G on at most ℓ vertices. Let Δ be the maximum degree of any vertex on G or, if G is directed, let Δ be the maximum degree of any vertex on the undirected version of G . Then all the simple cycles of length up to ℓ on G can be counted in time*

$$O(N + M + (\ell^\omega + \ell\Delta)|S_\ell|),$$

and $O(N + M)$ space. The same complexity is achieved when counting the simple paths of length up to ℓ or the simple cycles/paths with fixed endpoints of length up to ℓ .

The important result of Theorem 1 is that the time complexity of the general purpose algorithm presented here scales as $\text{poly}(\ell)|S_\ell|$. In comparison, we show in Section 4 that the time complexities of all other general purpose algorithms scale either with N^ℓ , which is always larger than $|S_\ell|$ unless the graph is complete, or with the number $|\text{Cycle}_\ell|$ of simple cycles of length at most ℓ on the graph.¹ From these observations, we expect that the algorithm presented here be the best available for graphs with less connected induced subgraphs than simple cycles,² something we both confirm and precise in Section 4. While deciding a-priori if a graph obeys this condition is difficult, we will see in Section 5 that it is true for several real-world networks and most Erdős-Rényi random graphs.

¹There is one exception to this observation: by extending an approach of Merris to count Hamiltonian cycles [32], we show in Section 4 that all simple cycles can be counted with a time complexity scaling as $\ell t_{\text{imm}}(\ell)|S_\ell|$, where $t_{\text{imm}}(\ell)$ is *exponential* in ℓ . Hence, this extension is still not competitive with the algorithm presented here.

²Note in this context, backtracks, that is bidirected edges, count as simple cycles. Furthermore the orientation of the cycles counts as well. Thus, for example, the complete graph on three vertices with no self-loops, K_3 , has two simple cycles of length 3 and three of length 2.

83 REMARK 1.1. *The algorithm presented here is FPT for the problem of counting*
 84 *simple cycles or simple paths of length ℓ , parameterized by ℓ , for the class of graphs*
 85 *where the number of connected induced subgraphs on at most ℓ vertices fulfils $|S_\ell| =$*
 86 *$O(f(\ell) \text{poly}(N))$, with f a computable function. This class is the class of bounded*
 87 *degree graphs, for which the existence of an FPT algorithm is not surprising. Indeed,*
 88 *the number of simple cycles / paths of length ℓ is upper bounded by the number of*
 89 *walks of this length, which is at most $\Delta^\ell N = f(\ell) \text{poly}(N)$. In fact, on bounded degree*
 90 *graphs, even a direct search of the simple cycles achieves the same complexity and*
 91 *constitutes a FPT algorithm.*

92

93 We prove Theorem 1 in Section 3 using the analytical framework outlined in
 94 Section 2 below. Following the proof of Theorem 1, we compare in Section 4 the
 95 performance of the algorithm presented here with the time complexities achieved by
 96 existing algorithms for the same task. These fall in five families: i) combinatorial
 97 sieves; ii) cycle counts by zeon-algebras; iii) cycle counts from combinations of im-
 98 manants; iv) special identities for short length cycles on undirected graphs; and v)
 99 counting via enumeration. In Section 5, we present numerical experiments validating
 100 the results of Theorem 1 and the comparisons of Section 4. We then demonstrate the
 101 performance of the algorithm on real-world networks. The *Matlab* implementation
 102 and data sets used for these experiments is available for download at [https://www-](https://www-users.cs.york.ac.uk/~plg508/)
 103 [users.cs.york.ac.uk/~plg508/](https://www-users.cs.york.ac.uk/~plg508/). We conclude in Section 6 with an extension of the
 104 algorithm for the enumeration of simple cycles on graphs with few labels.

105

106 2. Analytical framework.

107 **2.1. Counting simple cycles.** Our algorithm is based on a recent result from
 108 algebraic combinatorics relating the numbers of walks and of simple cycles on any
 109 (directed) graph. This result provides an explicit formula for the ordinary generating
 110 function of the number $\gamma(\ell)$ of simple cycles of length ℓ multiplied ℓ [17]

$$111 \quad (1) \quad \sum_{\ell} \ell \gamma(\ell) z^\ell = \sum_{H \prec_{\text{conn}} G} \text{Tr} \left((z \mathbf{A}_H)^{|H|} (1 - z \mathbf{A}_H)^{|N(H)|} \right),$$

112 where the sum runs over all weakly connected induced subgraphs H of G . Recall
 113 that a directed graph is said to be weakly connected if and only if its undirected
 114 version is connected. Thus such subgraphs can be found by an algorithm for finding
 115 connected induced subgraphs running over the undirected version $G_{\text{undir.}}$ of G . In
 116 this expression, $|H|$ designates the number of vertices of the subgraph H and $|N(H)|$
 117 is the number of neighbours of H in G . A neighbour of H in G is a vertex v of G
 118 which is not in H and such that there exists at least one edge, possibly directed, from
 119 v to a vertex of H or from a vertex of H to v . Finally, \mathbf{A}_H is the adjacency matrix of
 120 H . From the formula of Eq. (1) for the generating function of $\ell \gamma(\ell)$, we obtain $\gamma(\ell)$
 121 analytically as

$$122 \quad (2) \quad \gamma(\ell) = \frac{(-1)^\ell}{\ell} \sum_{H \prec_{\text{conn}} G} \binom{|N(H)|}{\ell - |H|} (-1)^{|H|} \text{Tr} (\mathbf{A}_H^\ell).$$

123 This explicit result forms the basis of the algorithm proposed here: counting the sim-
 124 ple cycles can be achieved by evaluating Eq. (2).

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

Any algorithmic implementation of Eq. (2) can be easily compared with the best existing combinatorial sieve for counting simple cycles, that of Bax and Franklin [5, 6], by observing that Eq. (2) involves a sum over the weakly *connected* induced subgraphs of the graph. In contrast, Bax and Franklin’s algorithm evaluates a formula involving a sum over *all* the induced subgraphs, including the non-connected ones. Remarkably, in the worst case scenario—the complete graph—every induced subgraph is connected, making it look like both algorithms should have a comparable complexity. On any other graph however, there are far more induced subgraphs than connected induced subgraphs. This crucial difference means that evaluating Eq. (2) *must* yield a significant speed-up as compared to Bax and Franklin’s algorithm. This argument is made rigorous by the proof of Theorem 1, which we present in Section 3, and see also Section 4.1. Before we proceed to this proof however, we present extensions of Eq. (2) for counting simple paths and simple cycles with fixed end points.

140

2.2. Counting simple paths and simple cycles visiting a fixed vertex.

141

142

143

To find all the simple paths, we rely once more on a recent result from algebraic combinatorics according to which the ordinary generating function of the number $\pi_{i \rightarrow j}(\ell)$ of simple paths of length ℓ from vertex i to j is the ij -entry of [17]

144

$$(3) \quad \sum_{\ell} \pi_{i \rightarrow j}(\ell) z^{\ell} = \left(\sum_{H \prec_{\text{conn}} G} (zA|_H)^{|H|-1} (1 - zA|_H)^{|N(H)|} \right)_{ij}.$$

This expression employs the same notation as that presented in Section 2.1 with the exception of $A|_H$, which represents the adjacency matrix of G restricted to the connected induced subgraph H . That is,

$$(A|_H)_{ij} = \begin{cases} A_{ij}, & \text{if } i, j \in H, \\ 0, & \text{otherwise,} \end{cases} \quad i, j \in G.$$

145

146

147

148

This construction allows one to formally write the sum of the various terms on the right hand side of Eq. (3). Most importantly, from a computational point of view, multiplying by A_H or $A|_H$ has the same time complexity $O(|H|^{\omega})$. The number $\pi_{i \rightarrow j}(\ell)$ then follows analytically as

149

$$(4) \quad \pi_{i \rightarrow j}(\ell) = (-1)^{\ell+1} \sum_{\substack{H \prec_{\text{conn}} G \\ i, j \in H}} \binom{|N(H)|}{\ell + 1 - |H|} (-1)^{|H|} (A|_H^{\ell})_{ij},$$

150

151

152

where the sum now runs over all weakly connected induced subgraphs of G containing both i and j . With the notation introduced here, we may also extend the result of Eq. (2) to count only those simple cycles passing through any specified vertex i with

153

$$(5) \quad \gamma_i(\ell) = (-1)^{\ell} \sum_{\substack{H \prec_{\text{conn}} G \\ i \in H}} \binom{|N(H)|}{\ell - |H|} (-1)^{|H|} (A|_H^{\ell})_{ii}.$$

154

In particular, we verify immediately that

155

$$(6) \quad \gamma(\ell) = \frac{1}{\ell} \sum_{i=1}^N \gamma_i(\ell),$$

156

as expected.

157 **3. Proof of Theorem 1.**

158 **3.1. The Algorithm: Evaluating Equation (2).** The algorithm consists sim-
 159 ply in evaluating Eq. (2), (4) or (5), depending on what one wants to count. Given
 160 the similar structures of these equations it is readily apparent that of Eqs. (2), (4)
 161 and (5), it is Eq. (2) that necessitates the greatest computational effort to be evalu-
 162 ated. This observation is best encapsulated by Eq. (6). For the sake of simplicity and
 163 to concretely illustrate our arguments, we will thus determine the time complexity
 164 explicitly only in the costliest situation: evaluating Eq. (2).
 165

166 We first remark that the binomial coefficient appearing in Eq. (2) is non-zero if
 167 and only if $|H| \leq \ell \leq |N(H)| + |H|$. Thus, only those weakly connected induced
 168 subgraphs H of G on $|H| \leq \ell$ vertices contribute a term of Eq. (2) when calculating
 169 $\gamma(\ell)$. Equivalently, all the weakly connected induced subgraphs of G such that $|H| \leq \ell$
 170 provide all the terms needed to calculate all $\gamma(k)$, $k \leq \ell$. Therefore, for an algorithm
 171 based on Eq. (2) to count the simple cycles, it is sufficient for it to find weakly
 172 connected induced subgraphs of bounded size. This observation leads to the following
 173 result:

LEMMA 2. *Let $t(|S_\ell|)$ be the time complexity of finding all the weakly connected induced subgraphs of G on at most ℓ vertices. Then the time complexity of determining the number $\gamma(k)$ of simple cycles of length k for all $k \leq \ell$ is*

$$O(t(|S_\ell|) + (\ell^\omega + \ell\Delta)|S_\ell|).$$

174 *Proof.* This is straightforward from Eq. (2). First, all the weakly connected indu-
 175 ced subgraphs on $k \leq \ell$ vertices must be found, costing $O(t(|S_\ell|))$ time, by defini-
 176 tion. Second, the terms of Eq. (2) must be evaluated, of which there are $|S_\ell|$ in total.
 177 Each term involves counting: i) the number of neighbours $|N(H)|$ of a subgraph H
 178 on the graph; and ii) the walks of length ℓ on this subgraph, its size being $|H| \leq \ell$.
 179 The first step thus costs at most $|H|\Delta = O(\ell\Delta)$ time and the second step requires
 180 $|H|^\omega = O(\ell^\omega)$ time. \square

183 **3.2. Time complexity of finding the connected induced subgraphs.** As

184 we have seen it is necessary and sufficient to find all the weakly connected induced
 185 subgraphs of size at most ℓ to count simple cycles of length up to ℓ . This can be done
 186 using the standard reverse search algorithm for finding connected induced subgraphs
 187 introduced by Avis and Fukuda [3], running on G_{undir} , the undirected version of the
 188 graph G . The total running time of this algorithm in our case is [39, 14]

189 (7)
$$t(|S_\ell|) = O\left(N + M + \sum_{k=1}^{\ell-1} |S_{=k}| + \ell^2 |S_{=\ell}|\right) = O\left(N + M + \ell^2 |S_\ell|\right),$$

190 where $M = |E|$ is the number of edges in G_{undir} and $|S_{=k}|$ designates the number
 191 of connected induced subgraphs on *exactly* k vertices in G_{undir} . Furthermore this
 192 algorithm uses $O(N + M)$ space. We also remark that thanks to reverse search, the
 193 algorithm for counting simple cycles can be parallelised: indeed, the contribution of
 194 each connected induced subgraph to Eq. (2) can be calculated independently of the
 195 other subgraphs. Now combining Eq. (7) with Lemma 2 and noting that $\omega \geq 2$ con-
 196 cludes the proof of Theorem 1.
 197

198 REMARK 3.1. *The time complexity of the reverse search algorithm for finding the*
 199 *connected induced subgraphs was recently improved upon by Karakashian et al. [26] and*
 200 *then further by K. Elbassioni [14]. Elbassioni describes a polynomial delay algorithm*
 201 *for this task yielding the following time complexity:*

COROLLARY 1 IN ELBASSIONI [14]. *Finding all the connected induced subgraphs*
of size $k \leq \ell$ in a graph with maximum degree Δ can be done in

$$O\left(\ell^2 \min\{(N - \ell), \ell\Delta\}(\log N + \Delta + \log \ell) |S_\ell|\right),$$

202 *total time.*

203 *Elbassioni also presents an algorithm with a slightly worse time complexity, but en-*
 204 *sureing a $O(N + M)$ space complexity, see Theorem 1 in [14]. Unfortunately, imple-*
 205 *mentations of these recent algorithms have not yet been produced.*

206

207 **3.3. Understanding the time complexity.** In the worst case scenario, that
 208 is the complete graphs K_N , Theorem 1 implies that the time complexity for counting
 209 all the simple cycles using the algorithm proposed here is $O(2^N N^\omega)$ since all induced
 210 subgraphs are connected, i.e. $|S_N| = 2^N$. This is marginally better than the com-
 211 plexities reported in [27, 5, 6, 38]. However, it is the performance of our algorithm
 212 on non-complete graphs that we want to highlight. To this end, it is helpful to recast
 213 the time complexity of the algorithm in terms of simple graph parameters.

214 We can do so by using an upper bound on the number $|S_k|$ of connected induced
 215 subgraphs on k vertices that involves the maximum degree of any vertex. This result
 216 is due to Uehara:

LEMMA 3 (Uehara [39]). *Let Δ be the maximum degree of the undirected version*
 $G_{undir.}$ of G . Then the number of connected induced subgraphs on exactly k vertices
in $G_{undir.}$ is bounded by

$$|S_{=k}| \leq N \frac{(e\Delta)^k}{(\Delta - 1)k^2},$$

with e the base of the natural logarithm. It follows that

$$|S_\ell| = \sum_{k \leq \ell} |S_{=k}| = O\left(N \frac{\Delta^\ell}{(\Delta - 1)\ell^2}\right).$$

218 Furthermore, on a graph with maximum degree Δ , there are at most $M \leq N\Delta$ edges,
 219 so that, by Theorem 1, the time-complexity of counting all the simple cycles of length
 220 $k \leq \ell$ is upper bounded by

$$221 \quad (8a) \quad O\left(N(\Delta + 1) + (\ell^\omega + \ell\Delta)N \frac{\Delta^\ell}{(\Delta - 1)\ell^2}\right) = O\left(N\Delta + N(\ell^{-1}\Delta + \ell^{\omega-2})\Delta^{\ell-1}\right),$$

$$222 \quad (8b) \quad \sim O(N\ell^{-1}\Delta^\ell),$$

223 where we used that $\Delta/(\Delta - 1) \leq 2$ as soon as the graph has a connected component
 224 with at least 3 vertices.

225 The bound on $|S_\ell|$ obtained from Uehara's work is typically *very far from tight*,
 226 especially on graphs that are far from regular, such as scale-free networks. Conse-
 227 quently, the time complexity predicted by Eq. (8b) is typically much larger than that
 228 observed in numerical experiments. However, Eq. (8b) simplifies the analysis of the
 229
 230

231 time complexity of the algorithm, which will help us compare it with other algorithms
 232 for the same task. Observe also that we now easily verify the claim of Remark 1.1 that
 233 the algorithm is FPT on bounded degree graphs. In fact, on such graphs $\Delta = O(1)$,
 234 consequently the time complexity scales as N , that is the algorithm is fixed parameter
 235 *linear*.

236

237 4. Detailed comparisons with existing algorithms.

238 **4.1. Sieve methods.** Bax and Bax and Franklin authored two articles detailing
 239 the use of combinatorial sieves to count simple cycles [5, 6], which extend previous re-
 240 sults by Karp [27] for counting Hamiltonian cycles. Similar techniques had previously
 241 been expounded by Khomenko and Golovko [28, 29] and more recently by Perepechko
 242 and Voropaev [36, 37].

244 All these combinatorial sieves produce the simple cycles via sums over all the
 245 induced subgraphs of a graph, i.e. including the non-connected ones. There are $\binom{N}{\ell}$
 246 such subgraphs of size ℓ on a graph on N vertices. Assuming ℓ is fixed and much
 247 smaller than N , the number of subgraphs is $\Omega(N^\ell/\ell!)$. Consequently, counting all
 248 simple cycles of length up to ℓ using these sieves takes at least $\Omega(N^\ell/\ell!)$ time. If Δ is
 249 sub-linear in N , this time complexity is much larger than that achieved by the algo-
 250 rithm presented here, which takes at most $O(N\Delta^\ell/\ell)$ time. In other terms, Eq. (2),
 251 which only involves the *connected* induced subgraphs, yields a significant speed-up.
 252 If instead $\Delta = \alpha N$, $0 < \alpha \leq 1$, the algorithm presented here is still $1/\alpha^\ell$ faster than
 253 other combinatorial sieves.³

254

255 **4.2. Zeons algebras.** An algorithm for counting simple cycles based on zeon
 256 algebras has been proposed by Schott and Staples in [38]. The algorithm relies on the
 257 observation that if one attaches a formal variable ξ_e to each edge e of the graph, such
 258 that any two such variable commute and $\xi_e^2 = 0$, then the corresponding labeled ad-
 259 jacency matrix $(A_\xi)_{ij} := \xi_{ij}A_{ij}$ generates only simple cycles. In other terms, $\text{Tr}(A_\xi^\ell)$
 260 is the number of simple cycles of length ℓ on G . Unfortunately, this method requires
 261 formal matrix multiplications and cannot be implemented fully numerically.

263 Schott and Staples proved that the average time taken by this algorithm to count
 264 simple cycles of length ℓ is $O(N^4(1+q)^N)$ where $q \geq \ell N\Delta/(N^2 - \ell)$ [38]. In the
 265 typical situation where $\Delta, \ell \ll N$, this cost is therefore at least $O(N^4e^{\ell\Delta})$. This is
 266 exponential in both ℓ and Δ and scales as the fourth power of N , hence always much
 267 larger than the $O(N\Delta^\ell/\ell)$ bound obtained earlier.

268

269 **4.3. Counting using immanants.** In 1983, R. Merris discovered an exact for-
 270 mula for counting the Hamiltonian cycles of a graph from a sum over at most N
 271 of its immanants [32, 33]. On noting that any simple cycle is Hamiltonian on an
 272 unique connected induced subgraph of the graph, Merris' formula is easily extended
 273 to count all simple cycles of length up to ℓ via a sum over the $|S_\ell|$ connected induced
 274 subgraphs of size at most ℓ . In this sum, each term is itself a sum over at most ℓ
 275 immanants. Therefore, evaluating the formula takes $O(t(|S_\ell|) + t_{\text{imm}}(\ell)\ell|S_\ell|)$ time,

³In addition, we have empirically observed that the time complexity of the algorithm proposed here scales with an effective parameter $\Delta_{\text{eff}} \ll \Delta$. What determines Δ_{eff} remains unclear.

276 with $t(|S_\ell|)$ and $t_{\text{imm}}(\ell)$ the times taken to find the connected induced subgraphs on at
 277 most ℓ vertices and to calculate the required immanants of $\ell \times \ell$ matrices, respectively.
 278

279 In the same spirit, G. Cash described in 2007 an approach for counting simple
 280 cycles by solving a system of equations involving selected immanantal polynomials of
 281 the graph [9]. For length ℓ simple cycles, Cash's approach stems from the solution
 282 of a system involving $p(\ell) - p(\ell - 1)$ equations, where $p(\ell)$ is the number of integer
 283 partitions of ℓ . This number grows as $O(e^{x\sqrt{\ell}}\ell^{-3/2})$ with $x = \pi\sqrt{2/3} \sim 2.6$ and
 284 consequently solving the system takes $O(e^{7.7\sqrt{\ell}}\ell^{-9/2})$ time. Since the immanantal
 285 polynomials of the graph take $O(t_{\text{imm}}(N))$ time to calculate, the cost of Cash's ap-
 286 proach is $O(t_{\text{imm}}(N)e^{7.7\sqrt{\ell}}\ell^{-9/2})$.
 287

288 Most importantly, we see that the time complexities of both methods are primar-
 289 ily influenced by the time taken to calculate the required immanants. Unfortunately,
 290 these are difficult to obtain. First, as recognized by Cash, they require computing the
 291 matrix of irreducible representations of the symmetric group \mathcal{S}_x , a very costly task
 292 for large x . Second, while the determinant of an $x \times x$ matrix requires only $O(x^3)$
 293 time, the second immanant d_2 already costs $O(x^c)$ with $3 < c \leq 4$ and computing
 294 the last immanant, the permanent, is itself a #P-complete problem [40]. The perma-
 295 nent is required by both Merris' and Cash's approaches, meaning that, assuming the
 296 exponential time hypothesis, $t_{\text{imm}}(x)$ grows exponentially in x . Comparing with The-
 297 orem 1, we observe that neither approach can compete with the algorithm proposed
 298 here.
 299

300 In the same vein, Giscard, Rochet and Wilson showed that simple cycles can
 301 be counted via a combination of permanents and determinants summed over the set
 302 of induced subgraphs of a graph [18], of which there are $\Theta(N^\ell)$. This particular
 303 approach only requires the computation of two immanants per subgraph thereby by-
 304 passing the need for computing the matrices of irreducible representations of large
 305 symmetric groups, yet requires all induced subgraphs to be considered and thus takes
 306 a prohibitive $O(N^\ell t_{\text{imm}}(\ell))$ time.⁴
 307

308 **4.4. Counting short simple cycles on undirected graphs.** When only short
 309 simple cycles on *undirected graphs* are of interest, these may be counted via a set of
 310 special identities involving the adjacency matrix. This approach was pioneered by
 311 Harary and Manvel in the 1970s and has remained popular ever since [21, 1, 10, 34].
 312 In particular, Alon, Yuster and Zwick presented an algorithm for evaluating these
 313 identities up to $\ell = 7$ in $O(N^\omega)$ time and $O(N^2)$ space [1]. This cost grows for longer
 314 cycles, being $O(N^{\omega+1})$ when $\ell = 8$, and then $O(N^{\lfloor \ell/2 \rfloor} \log N)$ when $\ell = 9, 10$. To the
 315 best of our knowledge, no special identity for counting $\ell > 10$ cycles has been found.
 316 There is little doubt that these exist however. Yet, given that the formula for $\ell = 10$
 317 already involves 160 terms [36], any such identity would be extremely cumbersome.
 318

319 From this discussion, we conclude that if the graph is undirected, only short sim-
 320 ple cycles of length $\ell \leq 7$ are desired and $N \leq \Delta^{\frac{\ell}{\omega-1}}$, then we expect Alon, Yuster and
 321 Zwick's approach (AYZ) to be faster than the algorithm presented here. In practice,
 322 we find AYZ to be faster in many cases where this condition is not met, presumably

⁴Even if a reduction to connected induced subgraphs can be devised for this method, which would yield a $O(t(S_\ell) + t_{\text{imm}}(\ell)|S_\ell|)$ time complexity, it would only marginally improve upon Merris' approach and would still be worse than that of the algorithm presented here.

323 because of differing constant factors hidden by the $O(\cdot)$ notation. These, in turn,
 324 might stem from the fact that AYZ is not a general purpose algorithm which relies
 325 on specific, optimised, ways of counting the simple cycles. Since the time complexity
 326 of AYZ scales with N^ω however, for any family of graphs where $\Delta = o(N)$, there is a
 327 graph size above which the algorithm presented here is faster than AYZ.
 328

329 Finally, we note that the space required for running AYZ scales as $O(N^2)$ rather
 330 than $O(N + M)$, the former being much larger than the latter on sparse graphs. We
 331 found this to be AYZ main limitation in practice,⁵ barring us from making compu-
 332 tations on networks with over 12,000 nodes. This memory cost is unavoidable since
 333 AYZ necessitates the computation of powers of the adjacency matrix A of the graph,
 334 which quickly become dense even on large sparse graphs. Recall in particular, that
 335 A^x is full for x larger than the graph diameter.
 336

337 **4.5. Counting simple cycles via enumeration.** Enumerating the simple cy-
 338 cles or simple paths of a graph, that is producing their vertex sequences, is much more
 339 time consuming than simply counting them. The best general purpose algorithm for
 340 this task is still Johnson’s 1975 landmark algorithm [25, 31], which achieves a time
 341 complexity of $O((N + M)(|\text{Cycle}_N| + 1)) \sim O(N\Delta|\text{Cycle}_N|)$. In this expression,
 342 $|\text{Cycle}_N|$ is the total number of simple cycles (or of simple paths) on G , including
 343 backtracks, that is simple cycles of length 2. This result was recently improved on
 344 undirected graphs to $O(N(|\text{Cycle}_N| + 1) + M)$, a scaling which is optimal for this
 345 task [7].
 346

347 In the worst case scenario, i.e. on the complete graph K_N , $|\text{Cycle}_N| = O(N!)$,
 348 that is enumerating all simple cycles takes factorial time. For this reason, counting
 349 simple cycles via enumeration has often been deemed greatly inefficient, in particular
 350 in comparison with the “only” exponential cost $O(2^N \text{poly}(N))$ achieved by the algo-
 351 rithm presented here as well as other approaches [6]. This conclusion follows from a
 352 peculiarity of dense graphs however and for sparse graphs it is not so.
 353

Indeed, evaluating Eq. (2) to count all the simple cycles on a graph costs $O(N^\omega |S_N|)$.
 It follows that if $N^\omega |S_N| \geq N\Delta|\text{Cycle}_N|$, then Johnson’s algorithm and its variants
 can count all the simple cycles of a graph via enumeration faster than any combina-
 torial sieve, including the one presented here. When counting simple cycles of fixed
 maximum length ℓ , Johnson’s algorithm takes $O(N + M + (\ell + \ell\Delta)|\text{Cycle}_\ell|)$ time,
 $|\text{Cycle}_\ell|$ being the total number of simple cycles of length up to ℓ . This means in
 order for the algorithm presented here to be faster than Johnson’s the following must
 hold

$$\left(\frac{\ell^{\omega-1}}{\Delta} + 1\right) |S_\ell| \leq |\text{Cycle}_\ell|.$$

354 Unfortunately, it is very difficult to estimate the ratio $|S_\ell|/|\text{Cycle}_\ell|$ in a preprocessing
 355 stage so as to decide which algorithm to use. Furthermore, the problem of charac-
 356 terising graphs for which the number of connected induced subgraphs is larger than
 357 the number of simple cycles is, to the best of our knowledge, an open mathematical
 358 question beyond the scope of this work. Rigorously, we may only conclude that for
 359 any number N of vertices, there must be a critical density above which the algorithm
 360 presented here will be faster than Johnson’s. We undertake an empirical study of
 361 this density in Section 5 on Erdős-Rényi random graphs and show it to be so small

⁵That is, beyond the fact that AYZ is limited to $\ell = 7$ on undirected graphs.

362 that the resulting graphs are disconnected with very high probability. In addition,
 363 we also study two families of real-world networks exemplifying the interplay between
 364 connected induced subgraphs and simple cycles.
 365

366 **4.6. Relation to subgraph counting algorithms.** Given a pattern graph H
 367 and a graph G , the *subgraph counting problem* is to determine the number of (induced)
 368 subgraphs of G that are isomorphic to H . This problem is well studied in undirected
 369 graphs and generalizes the problem of counting cycles. Therefore, we briefly summa-
 370 rize results on the subgraph counting problem, in particular, when parameterized by
 371 the size of the pattern graph $k = |V(H)|$.
 372

373 When G is a planar graph on N vertices the problem can be solved in time
 374 $N2^{O(k)}$ [12] improving the seminal FPT algorithm by Eppstein, which achieves $Nk^{O(k)}$
 375 time [15]. Nešetřil and Ossona de Mendez introduced *classes of graphs with bounded*
 376 *expansion*, which include the planar graphs as well as the graphs of bounded degree.
 377 For these classes they have shown that the number of satisfying assignments of a
 378 Boolean query with a fixed number of free variables can be counted in time linear in
 379 N [35, Theorem 18.9], which solves the subgraph counting problem for patterns of a
 380 fixed size as a special case. An improved algorithm tailored to counting subgraphs
 381 was proposed by Demaine et al. [11] and achieves a time complexity of $O(6^{kt^k}k^2N)$,
 382 where t is the height of a tree-depth decomposition of G . Again the approach yields a
 383 linear time FPT algorithm in graph classes of bounded expansion when parametrised
 384 by k . The running times of the above mentioned algorithms typically hide enormous
 385 constants and, to the best of our knowledge, have for this reason not been applied in
 386 practice.
 387

388 Technically related to our work is the method introduced by Amini et al. [2] to
 389 count subgraphs by homomorphisms using a combinatorial sieve. Their approach can
 390 be seen as a generalisation of the standard sieve methods for counting cycles to arbi-
 391 trary graphs, but does not overcome the drawbacks regarding running time discussed
 392 in Section 4.1.

393

394 **5. Experiments.** In this section we present numerical evidence for the perfor-
 395 mance of a *Matlab* implementation of the algorithm presented in this work. Note that
 396 this implementation incorporates a preprocessing stage which removes all sources,
 397 sinks and isolated vertices of the graph. We compare it with both Johnson’s and AYZ
 398 algorithms since, following Section 4, these are the only competitive algorithms for
 399 counting simple cycles. For the former we use Howbert’s freely available *Matlab* im-
 400 plementation [22], while for the latter we wrote a *Matlab* code, available for download.
 401 All the calculations reported here have been made on a MacBook Pro laptop with 3.1
 402 GHz Intel Core i7 processor and 8 GB of RAM running *Matlab* R2016a.
 403

404 **5.1. Erdős-Rényi random graphs.** We begin by considering undirected Erdős-
 405 Rényi random graphs $ER(N, p)$. These random graphs are determined by two param-
 406 eters: the number N of vertices and the probability p that any one undirected edge
 407 in the graph exists (with the exception of self-loops). The expected number of edges
 408 in $ER(N, p)$ is $pN(N - 1)/2$ so that the expected graph sparsity equals p .
 409

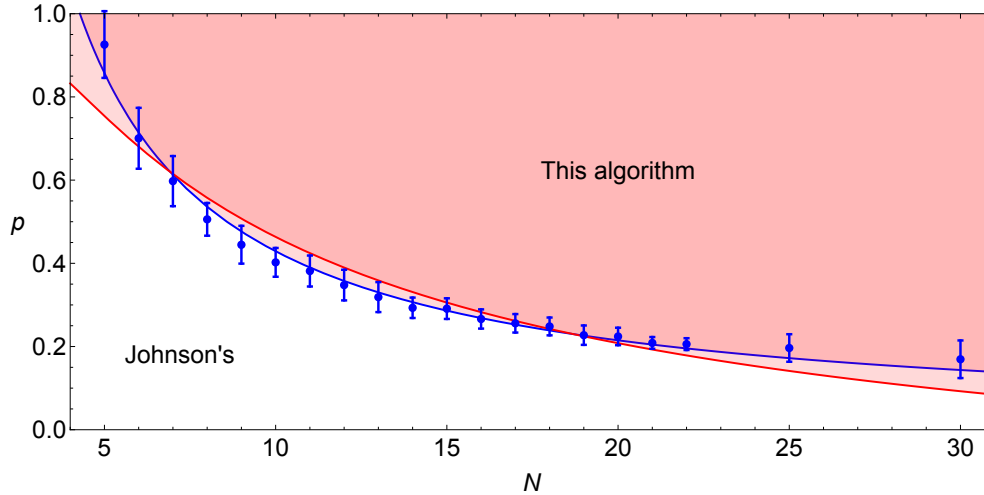


FIGURE 1. *Experimental comparison of Johnson’s algorithm with the algorithm presented here. The blue dots are the observed values of the critical edge probability below which Johnson’s algorithm is the fastest. The red line shows the best fit of the blue data points of the form $p_{\text{critical}}(N) = a + b \log(N)/N$, which is $p_{\text{critical}}(N) = -0.27 + 3.17 \log(N)/N$. The blue line shows the best fit of the blue data points of the form $p_{\text{critical}}(N) = a + b/N$, which is $p_{\text{critical}}(N) = 8.5 \times 10^{-4} + 4.28/N$. Since the latter fit is clearly better than the former, we retain $p_{\text{critical}}(N) \simeq 4.3/N$ as model when discussing p_{critical} in the text.*

410 **5.1.1. Comparison with Johnson’s algorithm.** We undertook the compari-
 411 son on two ranges of parameters: small graphs $5 \leq N \leq 30$, on which we compared
 412 the times taken by both algorithms to count all the simple cycles; and on large graphs
 413 $N \geq 1,000$, for which we counted simple cycles of length up to 5 only.
 414

415 On small graphs, for each value of N from 5 to 22 as well as for $N = 25$ and 30,
 416 we determined the critical value $p_{\text{critical}}(N)$ of p below which Johnson’s algorithm is
 417 the fastest by incrementing p from 0 to 1 by steps of 10^{-2} at N fixed. For each value
 418 of p , we ran both algorithms 20 times and compared the averaged time taken, except
 419 for $N = 25$ and 30 where we ran the algorithms only twice per value of p . The results
 420 are shown on Figure 1. Empirically we observe that for small graphs, $N \leq 30$, John-
 421 son’s algorithm is faster to count all simple cycles whenever $p \leq p_{\text{critical}}(N) \simeq 4.3/N$.
 422 Equivalently, this means that Johnson’s algorithm can be expected to be faster than
 423 the algorithm presented here whenever the average degree is close to 4 or smaller.
 424

425 On large graphs $p_{\text{critical}}(N)$ falls further, being seemingly less than $1/N$ when
 426 counting simple cycles of length up to 5 on Erdős-Rényi graphs with 20,000 vertices.
 427 In any case, we remark that for $N \gg 1$ and $p < \log(N)/N$, $\text{ER}(N, p)$ is known to be
 428 almost surely disconnected. Given that we observed that $p_{\text{critical}}(N) = O(1/N)$, it
 429 seems that for Johnson’s algorithm to be the fastest, an Erdős-Rényi random graph
 430 must be so sparse as to be disconnected in many small components.
 431

432 **5.1.2. Comparison with AYZ.** The algorithm of Alon, Yuster and Zwick is
 433 almost always the fastest to count simple cycles of length only up to 7 on undirected
 434 Erdős-Rényi random graphs. Indeed, we find that as soon as the graph is *denser* than
 435 $p_{\text{critical}} \approx 1.23/N$, for $N \lesssim 10,000$, then AYZ is faster than the algorithm presented
 436 here. The value of p_{critical} slowly increases with larger values of N and is around

437 $p_{\text{critical}} \simeq 1.3/N$ for $N \sim 12,000$. Unfortunately, the memory consumption of AYZ
 438 barres us from directly studying the performances of both algorithms on larger graphs.
 439 Rather, an extrapolation of the increase of p_{critical} suggests that it crosses $\log(N)/N$
 440 when N is well over 10^6 . This is widely beyond what can be reached by AYZ, and
 441 so large as to render the algorithm presented here prohibitively slow. We must thus
 442 conclude that AYZ remains the fastest algorithm on Erdős-Rényi random graphs for
 443 counting simple cycles of length 7 or less.

444

445 **5.2. Real-world networks.** We applied the algorithm presented in this work as
 446 well as those of AYZ and Johnson to compare their performances on three real-world
 447 networks, two of which are undirected and one is both weighted and directed:
 448

449 **ACTORS:** This network represents collaborations between movie actors and was gen-
 450 erated and analysed by Barabási and Albert [4]. Each actor constitutes a
 451 vertex and an edge represents that the two actors were cast together in the
 452 same movie.
 453

454 **INFECTIOUS:** A network representation of the face-to-face contacts between visitors
 455 of the exhibition *Infectious: Stay Away* held in Dublin in 2009 [24]. Each edge
 456 corresponds to face-to-face interaction lasting for at least 20 seconds.
 457

458 All data sets were obtained from the KONECT website [30], where further information
 459 on the data sets is available. The networks are undirected and have parallel edges. In
 460 order to systematically study the effect of sparsity, we generated several instances of
 461 each network by deleting edges with multiplicity below a given threshold as follows.
 462 Starting with the graph with all edges present, we successively removed all edges
 463 with multiplicity $1, 2, \dots$. A new instance is created whenever at least 80 edge were
 464 removed from the previous instance. This results in a sequence of graphs with de-
 465 creasing density progressively retaining only the most important edges. The sequence
 466 based on the ACTORS network comprises 31 graphs, while the sequence based on the
 467 INFECTIOUS network comprises 8 graphs.
 468

469 **WIKIELECTIONS:** A weighted directed network representing the votes of Wikipedia
 470 users during elections to adminship [42]. Each user corresponds to a vertex,
 471 and a directed edge from user u_1 to user u_2 exists if and only if u_1 voted
 472 during the election of u_2 . This edge is given a +1 weight if user u_1 supported
 473 u_2 candidacy and -1 otherwise. No pruning of the edges was operated on
 474 this network.
 475

476 The Wikielections network has 8289 vertices and 12915 directed edges. It is a scale-
 477 free graph with maximum out-degree $\Delta^{\text{out}} = 266$ and maximum in-degree $\Delta^{\text{in}} = 191$.
 478 Being directed, the Wikielections network cannot be studied with AYZ, which is lim-
 479 ited to undirected graphs, nor can it be studied with Howbert’s implementation of
 480 Johnson’s algorithm.
 481

482 In all cases, in order to accurately describe the performances of the algorithm
 483 presented here, we provide, for each graph, the time τ_ℓ it takes for counting all simple
 484 cycles of length up to ℓ , as well as the parameter governing the scaling of this time
 485 with ℓ . Indeed, while τ_ℓ is upper bounded by $N\Delta^\ell/\ell$ as per Eq. (8b), empirically,
 486 we find it to scale as $\tau_\ell \propto \Delta_{\text{eff}}^\ell$ with $\Delta_{\text{eff}} < \Delta$. This effective scaling parameter
 487 is determined numerically by fitting τ_ℓ with $a \times \Delta_{\text{eff}}^\ell + b$, where a and b are fitted

488 constants. Surprisingly, we did not find any relation between Δ_{eff} and the maximum,
489 mean, or median of the vertex degrees. What determines its value in practice remains
490 unclear.

INSTANCE #	N	M	Δ	Δ_{eff}	TIME (SEC.)	SIMPLE CYCLES
31	45	96	7	1.4	A: 4×10^{-2} J: 3×10^{-2}	$\ell = 10 : 0, 48, 32, 48, 48, 44, 16, 0, 0, 0$
30	90	260	12	1.6	A: 1.34 J: 4.06	$\ell = 10 : 0, 130, 202, 652, 2044, 5876, 14046, 25700, 33148, 29820$
29	143	428	17	3.3	A: 19 J: 76	$\ell = 10 : 0, 214, 356, 1328, 4946, 18608, 62038, 175710, 398864, 705874$
28	179	588	24	4.6	A: 248 J: 667	$\ell = 10 : 0, 294, 566, 2564, 11830, 56066, 246604, 970674, 3284880, 9284612$
27	125	748	26	5.4	A: 970 J: 2660	$\ell = 10 : 0, 374, 798, 4110, 22332, 125084, 665030, 3246496, 14068582, 52877616$
26	257	914	30	5.8	A: 2829 J: 8349	$\ell = 10 : 0, 457, 1018, 5726, 34724, 218028, 1310046, 7326752, 37074200, 166360444$
25	310	1118	36	7.3	A: 12301 J: 40124	$\ell = 10 : 0, 559, 1294, 7986, 53828, 377298, 2538470, 16045588, 92969672, 485843893$
24	376	1414	47	9	A: 20992 J: $> 10^5$	$\ell = 9 : 0, 707, 1728, 11686, 85300, 650344, 4744026, 32672232, 207557400$
23	423	1610	49	9.6	A: 5766 J: $> 3 \times 10^4$	$\ell = 8 : 0, 805, 2058, 15008, 118748, 980604, 7827540, 59395940$
22	470	1854	51	11.8	A: 1.02×10^4 J: $> 5 \times 10^4$	$\ell = 8 : 0, 927, 2476, 18674, 154346, 1333982, 11215982, 90027620$
17	863	3894	75	11.8	A: 2.8×10^4 J: – AYZ: 0.45	$\ell = 7 : 0, 1947, 6178, 61640, 688510, 8187720, 96547224$
12	1911	10428	119	22	A: 5.7×10^4 J: – AYZ: 0.91	$\ell = 6 : 0, 5214, 22060, 330498, 5625464, 105644852$
7	6085	48916	238	24	A: 7.3×10^4 J: – AYZ: 4.8	$\ell = 5 : 0, 24458, 181724, 5127548, 169365078$
4	19199	235964	609	~ 70	A: 4×10^5 J: – AYZ: OOM	$\ell = 4 : 0, 117982, 1608856, 103794848$
1: Full graph	382219	30076166	3956	–	A: – J: – AYZ: OOM	$\ell = 4 : –$

TABLE 1

Counting simple cycles on some graphs of the ACTORS data set. The time taken by the algorithm presented in this work is labelled by "A", while "J" refers to the time taken by Johnson's algorithm. We report the time taken by AYZ only when simple cycles of length 7 or less are counted. Because of memory limitations, we could not run AYZ on graphs 1 to 4, which we designated by "OOM" for "out of memory".

This manuscript is for review purposes only.

491 We now turn to the INFECTIOUS family of graphs. Contrary to the ACTORS set of
 492 graphs, we found Johnson's algorithm to run faster on this data set than the algorithm
 493 presented here.

INSTANCE #	N	M	Δ	Δ_{eff}	TIME (SEC.)	SIMPLE CYCLES
8	14	7	1	1	A: 1.7×10^{-3} J: 1.2×10^{-3}	$\ell = 10 : 0, 7, 0, 0, 0, 0, 0, 0, 0, 0$
7	29	30	2	1	A: 4.7×10^{-3} J: 4.3×10^{-3}	$\ell = 10 : 0, 15, 0, 0, 0, 0, 0, 0, 0, 0$ 0
6	42	50	2	1	A: 8.3×10^{-3} J: 1.3×10^{-2}	$\ell = 10 : 0, 25, 4, 0, 0, 0, 0, 0, 0, 0$ 0
5	91	116	4	1	A: 2.1×10^{-2} J: 2.2×10^{-2}	$\ell = 10 : 0, 58, 12, 2, 0, 0, 0, 0, 0, 0$ 0
4	236	394	4	1.01	A: 3.3×10^{-2} J: 1.7×10^{-1}	$\ell = 10 : 0, 197, 94, 60, 16, 0, 0,$ 0, 2, 2
3	337	964	15	3.8	A: 220 J: 33.8	$\ell = 10 : 0, 482, 572, 1340, 3552,$ 9490, 23504, 50900, 92630, 143620
2	368	1760	24	6.4	A: 2.3×10^4 J: 2.2×10^4	$\ell = 10 : 0, 880, 2322, 11506,$ 65356, 391646, 2391434, 14585954, 87432978, 509475403
1: Full graph	410	5530	50	16.8	A: 4.39×10^4 J: 1.8×10^4 AYZ: 0.4	$\ell = 7 : 0, 2765, 14228, 162574,$ 2142470, 30356160, 446411676

TABLE 2

Counting simple cycles on the graphs of the INFECTIOUS data set. The time taken by the algorithm presented in this work labelled by "A", while "J" refers to the time taken by Johnson's algorithm. We report the time taken by AYZ only when simple cycles of length 7 or less are counted.

494

495

496 Finally, we turn to the WIKIELECTIONS network which, as indicated earlier, is
 497 both directed and signed. The sign of a simple cycle being the product of the signs
 498 of its edges, we propose to demonstrate the algorithm capabilities by finding the
 499 numbers p_ℓ and n_ℓ of positive and negative simple cycles of length $\ell \leq 6$, respectively.
 500 Indeed, since it is sufficient to run the algorithm twice to obtain both p_ℓ and n_ℓ .
 501 More precisely, running the algorithm once on the signed network yields $p_\ell - n_\ell$,
 502 while running it on its unsigned version provides $p_\ell + n_\ell$.

LENGTH	1	2	3	4	5	6
TIME TAKEN (SEC.)	6×10^{-3}	9×10^{-2}	2.2	84	2981	1.04×10^5
POSITIVE SIMPLE CYCLES	6	337	1683	16369	182657	2170663
NEGATIVE SIMPLE CYCLES	5	12	253	3323	46792	663136
TOTAL	11	349	1936	19692	229449	2833799

TABLE 3

Number of positive and negative simple cycles of length ℓ up to 6 on the Wikielections directed network and the time taken to count them. Here $\Delta_{\text{eff}} \simeq 35$.

503 **6. Finding labelled simple cycles and simple paths.** In some applications,
504 such as cheminformatics, counting the simple cycles or simple paths of a network
505 is not sufficient. Rather, the vertices of the network may be labelled and it is then
506 necessary to find all sequences of labels corresponding to simple cycles/paths on the
507 network. If there are as many different labels as vertices, that is each vertex has its
508 own label, then this task is best addressed by Johnson’s algorithm discussed earlier
509 [25].

510

511 In typical applications however, the number of labels is much smaller than the
512 number of vertices. For example, on a network representing a molecule where vertices
513 are atoms and edges are bonds, vertex labels represent the various atomic species,
514 e.g. carbon, hydrogen, nitrogen etc. There is less than 10 such species in the vast
515 majority of organic molecules in available data sets. In addition, in the standard
516 representation of molecules, hydrogen vertices are omitted altogether and the label
517 of carbon atoms is put to the default value 1 (that is no label), further reducing the
518 number of different labels.

519

520 Finding all simple cycles/paths label sequences in such situations can be done
521 with Eq. (2), (4) or (5) with the following time complexity:

522 **THEOREM 4.** *Let $G = (V, E)$ be a graph, possibly directed, on N vertices. Let Δ*
523 *be the maximum degree of any vertex on G or, if G is directed, let Δ be the maximum*
524 *degree of any vertex on the undirected version of G . Finally, let n be the number of*
525 *different labels attached to graph vertices. Then all the label sequences of simple cycles*
526 *of length up to ℓ on G can be found in time*

$$527 \quad (9) \quad O(N + M + (n^\ell \ell^\omega + \ell \Delta) |S_\ell|)$$

528 and $O(N(\Delta+1))$ space. The same complexities are achieved to find the label sequences
529 of all simple paths up to length ℓ or of simple cycles/paths with fixed endpoints up to
530 length ℓ .

Proof. In the presence of labels, Eqs. (2), (4) and (5) continue to be valid and
provide the label sequences of the simple cycles/paths upon replacing the adjacency
matrix A by the labeled adjacency matrix W , defined by

$$W_{ij} := A_{ij} w_{L(i)L(j)},$$

531 where $L(i)$ is the label of vertex i and w is a formal variable. For example, a nitrogen-
 532 oxygen bond in a molecular graph would appear as w_{NO} in W . The time complexity
 533 of evaluating Eq. (2), (4) or (5) remains unchanged except for the cost of calculating
 534 the traces $\text{Tr}(W_H^\ell)$. These can be obtained through matrix multiplications. Since the
 535 entries of $W_H^{\ell-1}$ are sums of label sequences of walks of length $\ell - 1$ on the subgraph
 536 H and since there are at most $n^{\ell-1}$ such sequences, evaluating the trace costs at
 537 most $|H|^\omega n^{\ell-1} = O(\ell^\omega n^\ell)$ time. Replacing ℓ^ω with this cost in Theorem 1 yields the
 538 result. \square

539

540

541 **7. Conclusion.** We have presented a novel general purpose algorithm for count-
 542 ing simple cycles and simple paths of any length ℓ on any graph, including directed
 543 and weighted ones. The time complexity of this algorithm scales with the number
 544 $|S_\ell|$ of weakly connected induced subgraphs on at most ℓ vertices, making it the best
 545 general purpose algorithm whenever $(\ell^{\omega-1}\Delta^{-1} + 1)|S_\ell| \leq |\text{Cycle}_\ell|$. In this expression
 546 $|\text{Cycle}_\ell|$ is the total number of simple cycles of length up to ℓ , including self-loops and
 547 backtracks. Empirically, we found that this happens on Erdős-Rényi random graphs
 548 when the edge-probability exceeds circa $4/N$, as well as on some real-world networks,
 549 such as those in the ACTORS family of graphs.
 550

551 If the network under study is undirected and if counting simple cycles of length up
 552 to 7 is sufficient, then the algorithm of Alon, Yuster and Zwick is still by far the fastest.
 553 Furthermore, while we can predict that there must a graph size such that AYZ be-
 554 comes slower than the algorithm presented here, on Erdős-Rényi random graphs this
 555 size seems to be much beyond what we can reach. Indeed, we could not run AYZ on
 556 graphs with more than $N \gtrsim 12,000$ vertices owing to its important memory consump-
 557 tion. While this number can likely be increased with an optimised implementation
 558 of AYZ in conjunction with more memory, it is unlikely to get substantially larger as
 559 the memory usage of AYZ scales with N^2 . In contrast, we could run the algorithm
 560 presented here on networks with over 300,000 vertices without running out of memory.
 561

562 Finally, even though the algorithm presented here is the best general purpose al-
 563 gorithm on the class of graphs with $(\ell^{\omega-1}\Delta^{-1} + 1)|S_\ell| \leq |\text{Cycle}_\ell|$, the time necessary
 564 to count simple cycles of length e.g. up to 10 can be prohibitively large on large net-
 565 works. Instead, the algorithm is best used in conjunction with Monte Carlo methods.
 566 We demonstrate this procedure in a separate publication [19], where we use it in a
 567 sociological context to obtain the ratios of negative to positive simple cycles of length
 568 up to 20 on several real-world directed signed networks with up to 130,000+ vertices.

569

REFERENCES

- 570 [1] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*,
 571 17:209–223, 1997.
- 572 [2] Omid Amini, Fedor V. Fomin, and Saket Saurabh. Counting subgraphs via homomorphisms.
 573 *SIAM Journal on Discrete Mathematics*, 26(2):695–717, 2012.
- 574 [3] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–
 575 46, 1996.
- 576 [4] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*,
 577 286(5439):509–512, 1999.
- 578 [5] E. Bax and J. Franklin. A finite-difference sieve to count paths and cycles by length. *Information*
 579 *Processing Letters*, 60(4):171–176, 1996.
- 580 [6] Eric T Bax. Algorithms to count paths and cycles. *Information Processing Letters*, 52(5):249–

- 581 252, 1994.
- 582 [7] E. Birmelé, R. Ferreira, R. Grossi, A. Marino, N. Pisanti, R. Rizzi, and G. Sacomoto. Optimal
583 listing of cycles and st-paths in undirected paths. In *Proceedings of the Twenty-Fourth*
584 *Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1884–1896, 2013.
- 585 [8] P. Bürgisser, M. Clausen, and M. Amin Shokrollahi. *Algebraic Complexity Theory*, volume 315
586 of *Grundlehren der mathematischen Wissenschaften*. Springer, Berlin; New York, 1997.
- 587 [9] Gordon G Cash. The number of n -cycles in a graph. *Applied mathematics and computation*,
588 184(2):1080–1083, 2007.
- 589 [10] Y. C. Chang and H. L. Fu. The number of 6-cycles in a graph. *Bull. Inst. Combin. Appl.*,
590 39:27–30, 2003.
- 591 [11] Erik D. Demaine, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, Somnath Sikdar,
592 and Blair D. Sullivan. Structural sparsity of complex networks: Random graph models and
593 linear algorithms. *CoRR*, abs/1406.2587, 2014.
- 594 [12] Frederic Dorn. Planar subgraph isomorphism revisited. In Jean-Yves Marion and Thomas
595 Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer*
596 *Science (STACS 2010)*, volume 5 of *Leibniz International Proceedings in Informatics*
597 *(LIPIcs)*, pages 263–274, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum
598 fuer Informatik.
- 599 [13] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science.
600 Springer, New York, 1999.
- 601 [14] K. Elbassioni. A Polynomial Delay Algorithm for Generating Connected Induced Subgraphs
602 of a Given Cardinality. *Journal of Graph Algorithms and Applications*, 19:273–280, 2015.
- 603 [15] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of*
604 *Graph Algorithms & Applications*, 3(3):1–27, 1999.
- 605 [16] Jörg Flum and Martin Grohe. The Parameterized Complexity of Counting Problems. *SIAM*
606 *Journal on Computing*, 33:892–922, 2004.
- 607 [17] P.-L. Giscard and P. Rochet. Enumerating simple paths from connected induced subgraphs.
608 *arXiv:1606.00289*, 2016.
- 609 [18] P.-L. Giscard, P. Rochet, and R. C. Wilson. An Hopf algebra for counting simple cycles.
610 *arXiv:1607.00902*, 2016.
- 611 [19] P.-L. Giscard, P. Rochet, and R. C. Wilson. Evaluating balance on social networks from their
612 simple cycles. *arXiv:1606.03347*, 2016.
- 613 [20] Martin Grohe. Descriptive and parameterized complexity. In Jörg Flum and Mario Rodriguez-
614 Artalejo, editors, *Computer Science Logic*, volume 1683 of *Lecture Notes in Computer*
615 *Science*, chapter 3, pages 14–31. Springer-Verlag, Berlin Heidelberg, 1999.
- 616 [21] F. Harary and B. Manvel. On the number of cycles in a graph. *Matematický časopis*, 21:55–63,
617 1971.
- 618 [22] J. Howbert. Count all cycles in simple undirected graph. File Exchange - MAT-
619 LAB Central - MathWorks, [https://uk.mathworks.com/matlabcentral/fileexchange/
620 29438-count-all-cycles-in-simple-undirected-graph](https://uk.mathworks.com/matlabcentral/fileexchange/29438-count-all-cycles-in-simple-undirected-graph), 2011.
- 621 [23] R. Impagliazzo and R. Paturi. On the Complexity of k -SAT. *Journal of Computer and System*
622 *Sciences*, 62:367–375, 2001.
- 623 [24] Lorenzo Isella, Juliette Stehlé, Alain Barrat, Ciro Cattuto, Jean-François Pinton, and
624 Wouter Van den Broeck. What’s in a crowd? analysis of face-to-face behavioral networks.
625 *Journal of Theoretical Biology*, 271(1):166 – 180, 2011.
- 626 [25] D. B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. Comput.*,
627 4:77–84, 1975.
- 628 [26] S. Karakashian, B. Y. Choueiry, and S. G. Hartke. An Algorithm for Generating All Connected
629 Subgraphs with k Vertices of a Graph. *University of Nebraska, student report*, UNL-CSE-
630 2013-0005:1–38, 2013.
- 631 [27] Richard M Karp. Dynamic programming meets the principle of inclusion and exclusion. *Oper-*
632 *ations Research Letters*, 1(2):49–51, 1982.
- 633 [28] N. P. Khomenko and L. D. Golovko. Identifying certain types of parts of a graph and computing
634 their number. *Ukrainskii Matematicheskii Zhurnal*, 24:385–396, 1972.
- 635 [29] N. P. Khomenko and E. N. Shevchenko. The problem of isolating and counting. *Ukrainian*
636 *Mathematical Journal*, 30(2):152–160, 1978.
- 637 [30] Jérôme Kunegis. KONECT: The Koblenz Network Collection. In *Proceedings of the 22nd*
638 *International Conference on World Wide Web, WWW ’13 Companion*, pages 1343–1350,
639 New York, NY, USA, 2013. ACM. Available at: <http://konect.uni-koblenz.de/>.
- 640 [31] P. Mateti and N. Deo. On Algorithms for Enumerating all Circuits of a Graph. *SIAM J.*
641 *Comput.*, 5:90–99, 1976.
- 642 [32] Russell Merris. Single-hook characters and Hamiltonian circuits. *Linear and Multilinear Alge-*

- 643 *bra*, 14:21–35, 1983.
- 644 [33] Russell Merris. Immanantal invariants of graphs. *Linear Algebra and its Applications*, 401:67–
645 75, 2005.
- 646 [34] N. Movarraei and S. A. Boxwala. On the Number of Cycles in a Graph. *Open Journal of*
647 *Discrete Mathematics*, pages 41–69, 2016.
- 648 [35] Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity — Graphs, Structures, and Algo-*
649 *rithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.
- 650 [36] S. N. Perepechko and A. N. Voropaev. The number of fixed length cycles in an undirected graph.
651 explicit formulae in case of small lengths. *Mathematical Modeling and Computational*
652 *Physics (MMCP2009)*, pages 148–149, 2009.
- 653 [37] S. N. Perepechko and A. N. Voropaev. Количество простых циклов фиксированной длины
654 в неориентированном графе. Явные формулы в случае малых длин. (The Number of
655 Fixed Length Cycles in Undirected Graph. Explicit Formulae in Case of Small Lengths).
656 Вестник РУДН. Серия Математика. Информатика. Физика, 2:5–11, 2012.
- 657 [38] R. Schott and G. S. Staples. Complexity of counting cycles using Zeons. *Computers & Math-*
658 *ematics with Applications*, 62(4):1828–1837, 2011.
- 659 [39] R. Uehara. The number of connected components in graphs and its applications. *IEICE*
660 *Technical Report, COMP99-10*, 1999.
- 661 [40] Leslie G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8:189–
662 201, 1979.
- 663 [41] Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.*,
664 8:410–421, 1979.
- 665 [42] Wikipedia adminship election data. <http://snap.stanford.edu/data/wiki-Elec.html>, 2016.