Efficient Enumeration of Subgraphs and Induced Subgraphs with Bounded Girth

Kazuhiro Kurita[1(✉)], Kunihiro Wasa[2], Alessio Conte[2], Takeaki Uno[2], and Hiroki Arimura[1]

[1] IST, Hokkaido University, Sapporo, Japan
{k-kurita,arim}@ist.hokudai.ac.jp
[2] National Institute of Informatics, Tokyo, Japan
{wasa,conte,uno}@nii.ac.jp

Abstract. The girth of a graph is the length of its shortest cycle. Due to its relevance in graph theory, network analysis and practical fields such as distributed computing, girth-related problems have been object of attention in both past and recent literature. In this paper, we consider the problem of listing connected subgraphs with bounded girth. As a large girth is index of sparsity, this allows to extract sparse structures from the input graph. We propose two algorithms, for enumerating respectively vertex induced subgraphs and edge induced subgraphs with bounded girth, both running in $O(n)$ amortized time per solution and using $O(n^3)$ space. Furthermore, the algorithms can be easily adapted to relax the connectivity requirement and to deal with weighted graphs. As a byproduct, the second algorithm can be used to answer the well known question of finding the densest n-vertex graph(s) of girth k.

1 Introduction

We consider the problem of finding all subgraphs and induced subgraphs with girth at least k of a graph. The girth is a measure of sparsity, as graphs with large girth are inherently sparse. This corresponds to finding *sparse* substructures of the given graph, a problem that was considered under several forms [5,9] and has applications in network analysis. In particular, this problem generalizes two well studied problems, i.e., listing all subtrees and induced subtrees [7,13–15]. Indeed, any graph with girth larger than n may not contain a cycle, i.e., it is a tree, or a forest.

A *subgraph enumeration problem*, given a graph G and some constraint $\mathcal{R}$, consists in outputting all the subgraphs satisfying $\mathcal{R}$ without duplicates. The efficiency of enumeration algorithms is often measured with respect to both the size of the input and that of the output, i.e., the number of solutions: an enumeration algorithm is called an *amortized polynomial time algorithm* if it runs in $O(M \cdot poly(N))$ time, where N is the input size and M is the number of solutions. Furthermore, the algorithm is said to have polynomial *delay* if the maximum time elapsed between two consecutive outputs is polynomial.

C. Iliopoulos et al. (Eds.): IWOCA 2018, LNCS 10979, pp. 201–213, 2018.
https://doi.org/10.1007/978-3-319-94667-2_17

In this paper, we present two amortized polynomial time algorithms for enumerating subgraphs of girth at least k. The first, `EBG-IS`, enumerates *induced* subgraphs, while the second, `EBG-S`, enumerates *edge* subgraphs (also simply called subgraphs). Both `EBG-IS` and `EBG-S` run in $O(n\,|\mathcal{S}|)$ time using $O(n^3)$ space, where n is the number of nodes in G and $\mathcal{S}$ is the set of all solutions. The proposed algorithms will consider the enumeration of *connected* subgraphs in simple graphs. However, both algorithms can easily be applied to the enumeration of non-connected subgraphs, and to weighted graphs by trivial changes, with the same time and space complexity. In these problems, the upper bound of the number of solutions are $O(2^n)$ and $O(2^m)$, respectively, where m is the number of edges. Hence, the brute force algorithms are optimal if we evaluate the efficiency of algorithms only the input size. When we describe a more efficient algorithm, reducing amortized complexity is important [10]. Indeed, our implementation of `EBG-S`[1] is almost 560 times faster than the brute force algorithm when the input graph is a complete graph K_8 and girth is four.

While the problem of efficiently enumerating subgraphs with bounded girth has been considered for *directed* graphs [6], to the best of our knowledge, there is no known efficient algorithm for the *undirected* version of the problem.[2]

An early result on girth computation is the algorithm by Itai and Rodeh [8], that finds the girth of a graph in $O(nm)$ time. In more recent work, the problem was also solved in linear time for planar graphs [4]. However, the problem we consider involves computing the girth of many subgraphs, so relying on these algorithms is not efficient.

A prominent question related to the girth is finding exactly how dense a graph of given girth can be: the maximum number of edges in a d-regular graph with girth k is bounded by the well known *Moore bound* [2], which Alon later proved to be tight on general graphs as well [1]. Erdős conjectured that there exists a graph with $\Omega(n^{1+1/k})$ edges and girth $2k+1$ [12]. On the other hand, some have focused on giving practical lower bounds, i.e., finding ways to generate graphs of given girth as dense as possible [3,11]. We remark that our proposed algorithm `EBG-S` can match theory and practice: the densest n-vertex graph of girth k can be found as a subgraph of the complete graph K_n. While this may not be practical for large values of n, it significantly improves upon the brute force approach by avoiding the generation of subgraphs with girth <k.

2 Preliminaries

Let $G = (V(G), E(G))$ be a simple undirected graph with no self-loops, with vertex set $V(G)$ and edge set $E(G) \subseteq V(G) \times V(G)$. Two vertices u and v are *adjacent* (or neighbors) if there is an edge $e = \{u, v\} \in E(G)$ joining them. We

[1] The implementation of `EBG-S` in the github repository: https://github.com/ikn-lab/EnumerationAlgorithms/tree/master/BoundedGirth/.

[2] We remark that the techniques in [6] do not extend to undirected graphs, thus motivating a separate study. In directed graphs, a u-v path and a v-u path are distinct. However, a u-v path and a v-u path may be same in undirected graphs.

call e incident to v and we denote the set of incident edges to v $E(v)$. The set of neighbors of u in G is called its *neighborhood* and denoted by $N_G(u)$ and the size of $N_G(u)$ is called the *degree* of u in G. Let $N_G[u] = N_G(u) \cup \{u\}$ be the closed neighborhood of u. The *set of neighbors* of $U \subseteq V$ is defined as $N_G(U) = \bigcup_{u \in U} N_G(u) \setminus U$. Similarly, $N_G[U]$ denotes $N_G(U) \cup U$. For any vertex subset $S \subseteq V$, we call $G[S] = (S, E[S])$ an *induced subgraph*, where $E[S] = E(G) \cap (S \times S)$. Since $G[S]$ is uniquely determined by S, we sometimes identify $G[S]$ with S. For any edge subset $E' \subseteq E$, we call $G[E'] = (V'(E'), E')$ *edge induced subgraph*, where $V'(E') = \bigcup_{\{u,v\} \in E'} u$. We define $G \setminus \{e\} = (V, E \setminus \{e\})$ and $G \setminus \{v\} = G[V \setminus \{v\}]$. For simplicity, we use $v \in G$ and $e \in G$ to refer to $v \in V(G)$ and $e \in E(G)$, respectively. If G is clear from the context, we will also use simplified notation such as V, E, $N(u)$ instead of $V(G)$, $E(G)$, $N_G(u)$.

A sequence $P = (v_1, \ldots, v_{k+1})$ of distinct vertices is a *path* from v_1 to v_{k+1} (v_1-v_{k+1} path for short) in $G = (V, E)$ if for any $i \in [1, k]$, $\{v_i, v_{i+1}\} \in E$. P is a *shortest path* between two vertices if there is no shorter path between them. Let us denote by $V(P)$ and $E(P)$ the set of vertices and edges in P, respectively. We say that G is *connected* if for any two vertices $u, v \in V$, there is a u-v path. We say that a sequence $C = (v_1, \ldots, v_{k+1})$ of vertices is a *cycle* if $(v_1, \ldots, v_k)$ is a v_1-v_k path, $v_{k+1} = v_1$, and $\{v_k, v_{k+1}\} \in E$. The *length* of a path or cycle is defined by its number of edges. The *distance* between two vertices is the length of a shortest path between them. The *girth* of G, denoted by $g(G)$, is the length of a shortest cycle in G. For simplicity, we say that G has girth k if $g(G) \geq k$. The girth of acyclic graphs is usually assumed to be ∞.

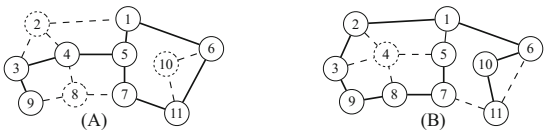

Fig. 1. Dashed edges and vertices are not included by an induced subgraph and a subgraph. An induced subgraph of girth five (A) and a subgraph of girth six (B).

We define our problems as follows and Fig. 1 shows examples of solutions Problem 1 and Problem 2. If we store all outputs, then it is easy to avoid duplicates. Our algorithms achieve without duplicates in polynomial space.

Problem 1 (k-girth connected induced subgraph enumeration). Enumerate all connected induced subgraphs S of a graph G with $g(S) \geq k$, without duplicates.

Problem 2 (k-girth connected subgraph enumeration). Enumerate all connected subgraphs S of a graph G with $g(S) \geq k$, without duplicates.

3 Enumeration by Binary Partition

The *binary partition method* is one of the fundamental frameworks for designing enumeration algorithms. Typically, a binary partition algorithm $\mathcal{A}$ has the fol-

Algorithm 1. Enumerate all connected induced subgraphs with girth k.

```
1 Procedure EBG(G, k)                    // G: an input graph, k: positive integer
2 |  RecEBG(∅, G);
3 Procedure RecEBG(S, G)                                  // S: the current solution
4 |  Output S;
5 |  DONE ← ∅;
6 |  for v ∈ C(S) do
7 |  |  RecEBG(S ∪ {v}, G \ DONE);
8 |  |  DONE ← DONE ∪ {v};
9 |  return;
```

lowing structure: first $\mathcal{A}$ picks an element x of the input, then divides the search space into two disjoint spaces, one containing the solutions that include x, and one those that do not. $\mathcal{A}$ recursively executes the above step until all elements are picked. Whenever the search space contains exactly one solution, $\mathcal{A}$ outputs it. We call each dividing step an *iteration*.

Algorithm EBG, detailed in Algorithm 1, represents a basic strategy for Problem 1. Algorithm 1 is based on binary partition, although each iteration divides the search space in more than two subspaces. While EBG enumerates solutions by picking vertices on each iteration, we can obtain an enumeration algorithm for Problem 2 by modifying EBG so that it picks edges instead.

Let G, X, and $S(X)$ be respectively an input graph, an iteration, and the solution received by the iteration X. A vertex $v \notin S(X)$ is a *candidate vertex* for $S(X)$ if $g(S(X) \cup \{v\}) \geq k$ and $S(X) \cup \{v\}$ is connected, that is, the addition of a candidate vertex generates a new solution. Let $C(S(X))$ be a set of candidate vertices for $S(X)$. We call $C(S(X))$ the *candidate set* of $S(X)$. Now, suppose that X generates new iterations $Y_1, \ldots, Y_d$ by adding vertices in $C(S(X)) = \{v_1, \ldots, v_d\}$ on line 7. For each i, we say that X is the *parent* of Y_i, and Y_i is a *child* of X. Note that, on iteration Y_i and its descendant iterations, EBG outputs solutions that do not include $v_1, \ldots, v_{i-1}$ but do include v_i. This implies that the solution space of Y_i is disjoint from those of each $Y_{j<i}$ created so far, i.e., EBG divides the solution space of X in d disjoint subspaces. The only iteration without a parent is the one generated on line 2, which we call the *initial iteration* and denote by I. We remark that $S(I) = \emptyset$ and that $\emptyset$ is a solution.

By using the above parent-child relation, we introduce the *enumeration tree* $\mathcal{T}(G) = \mathcal{T} = (\mathcal{V}, \mathcal{E})$. Here, $\mathcal{V}$ is the set of iterations of EBG for G and $\mathcal{E}$ is a subset of $\mathcal{V} \times \mathcal{V}$. For any pair of iterations X and Y, $(X, Y) \in \mathcal{E}$ if and only if X is the parent of Y. We can observe that $\mathcal{T}$ has no cycles since every child iteration of X receives a solution whose size is larger than $S(X)$. In addition, each iteration other than the initial iteration has exactly one parent. This implies that the initial iteration is an ancestor of all iterations and thus $\mathcal{T}$ is connected. Thus, $\mathcal{T}$ forms a tree. Next three lemmas show the correctness of EBG. Due to the space limitation, we omit some proofs (which can be found in Appendix).

Lemma 1. *Let G be a simple undirected graph and k a positive integer. Then, every output of* EBG *induces a connected subgraph of girth k.*

Lemma 2. *If X and Y are two distinct iterations on* EBG*, then $S(X) \neq S(Y)$.*

Lemma 3. *Let G be a simple undirected graph and k a positive integer.* EBG(G, k) *outputs all connected induced subgraphs with girth k in G exactly once.*

Proof. By Lemma 1, EBG outputs only solutions, and by Lemma 2 it does not output each solution more than once. We show that EBG outputs all solutions by induction. Let S be a solution. If $|S| = 0$, EBG outputs the empty set.

Otherwise, there is an iteration X_0 such that $S(X_0) \subseteq S$ and $S \subseteq V(G)$ (that is, no vertex of S has been removed from G). This is trivially true, e.g. for $X_0 = I$, since $S(I) = \emptyset$ and nothing has been removed from G. Note that every subgraph of a graph with girth at least k must also have girth at least k, thus every $v \in S \setminus S(X_0)$ such that $G[S(X_0) \cup \{v\}]$ is connected must be in $C\,(S(X_0))$. As S is connected there is at least one such v in $C\,(S(X_0))$.

Consider the first execution of Line 7 in X for which a vertex $v \in S \setminus S(X_0)$ is considered to generate a child iteration X_1. As no vertex of S was added to DONE in X_0, we still have that $S(X_1) \subseteq S$ and $S \subseteq V(G)$ in iteration X_1, but $|S(X_1)| = |S(X_0)| + 1$. Hence, by induction, EBG will eventually find S. □

Using Itai's algorithm [8] to compute the girth of a graph in $O(mn)$, we can obtain a first trivial complexity bound for Algorithm 1.

Theorem 1. EBG *solves Problem 1 with delay $O(n^2 m)$.*

Non-induced, weighted, and non-connected case. Let us briefly show how EBG also applies to some variants of the problem. Firstly, we can solve Problem 2, i.e., enumerate *edge* subgraphs, by modifying EBG as follows: Each solution is a set of edges $S \subseteq E$, and the candidate set $C\,(S(X))$ becomes $C\,(S(X)) = \{e \in E(X) \mid G[S(X) \cup \{e\}]$ is connected and $g(G[S(X) \cup \{v\}]) \geq k\}$. It is straightforward to see that Lemma 3 still holds (replacing the word *induced* with *edge* in the statement), and that the modified algorithm will solve Problem 2 in polynomial delay and polynomial space.

Furthermore, we can consider the *weighted* version of the problem, where the length of a cycle is the sum of the weights of its edges: we can find the girth in this case by adapting the Floyd-Warshall algorithm, and thus still enumerate all solutions for both the induced and edge subgraph version of the problem, in polynomial delay and polynomial space.

Finally, we consider non-connected case, i.e., where the solutions are all induced or edge subgraphs of girth k, and not just the connected ones: this is trivially done by redefining the candidate set as $C\,(S(X)) = \{v \in V(G) \mid g(G[S(X) \cup \{v\}]) \geq k\}$ for Problem 1, and similarly for Problem 2. If $G[S]$ is not connected, its girth is the minimum among that of its connected components, thus we can still use Itai's algorithm (or Floyd-Warshall if weighted edges are considered as well), and again obtain polynomial delay and polynomial space.

Algorithm 2. Updating data structures in `EBG-IS`.

Procedure `NextC`$(v, C(S), D^{(1)}(S), D^{(2)}(S), S, k, G)$
  $C(S \cup \{v\}) \leftarrow$ `UpdateCand`(v, S);
  $D^{(1)}(S \cup \{v\}) \leftarrow$ `Update1`$(v, C(S \cup \{v\}))$;
  $D^{(2)}(S \cup \{v\}) \leftarrow$ `Update2`$(v, C(S \cup \{v\}))$;
Function `UpdateCand`(v, S)
  $C(S \cup \{v\}) \leftarrow N(v) \cup C(S)$;
  foreach $u \in C(S)$ **do**
    if $D^{(1)}_{uv}(S) + D^{(2)}_{uv}(S) \geq k$ **then** $C(S \cup \{v\}) \leftarrow C(S \cup \{v\}) \cup \{u\}$;
  return $C(S \cup \{v\})$;
Function `Update1`$(v, C(S \cup \{v\}))$
  foreach $u \in C(S \cup \{v\}) \cup S, w \in C(S \cup \{v\})$ **do**
    $D^{(1)}_{uw}(S) \leftarrow \min\{D^{(1)}_{uw}(S), D^{(1)}_{uvw}(S)\}$
  return $D^{(1)}(S \cup \{v\})$
Function `Update2`$(v, C(S \cup \{v\}))$
  foreach $u, w \in C(S \cup \{v\})$ **do**
    $p_1 \leftarrow \min\{D^{(1)}_{uw}(S), D^{(1)}_{uvw}(S \cup \{v\}), D^{(2)}_{uw}(S)\}$;
    $p_2 \leftarrow$ the second smallest length in $\{D^{(1)}_{uw}(S), D^{(1)}_{uvw}(S \cup \{v\}), D^{(2)}_{uw}(S)\}$;
    if $p_1 + p_2 \geq k$ **then** // $x \in N(u) \cap S \cup \{v\}$
      $p_2 \leftarrow$ the second smallest length in $\{D^{(1)}_{xw}(S \cup \{v\}) + 1\}$;
    $D^{(2)}_{uw}(S \cup \{v\}) \leftarrow p_2$;
  return $D^{(2)}(S \cup \{v\})$;

4 Induced Subgraph Enumeration

The bottleneck of `EBG` is the computation of the candidate set. In this section, we present a more efficient algorithm `EBG-IS` for Problem 1. `EBG-IS` is based on `EBG`, but each iteration exploits information from the parent iteration, and maintains distances in order to improve the computation of the candidate set. The procedure is shown in Algorithm 2.

`EBG-IS` uses the second distance between vertices defined as follows. Let v be a vertex in $C(S) \cup S$, and u and u' be vertices in $C(S)$. We denote by $D^{(1)}_{uv}(S)$ the distance between v and u in $G[S \cup \{v, u\}]$, and by $D^{(2)}_{uu'}(S)$ the distance between u and u' in $G[S \cup \{u, u'\}] \setminus \{e_0\}$, where $e_0 = (u, \cdot)$ is the first edge on a shortest path between u and u'. Note that for any vertices $x \in G \setminus \{C(S) \cup S\}$, $y \in G \setminus C(S)$, and $y' \in G \setminus C(S)$, $D^{(1)}_{xy}(S) = \infty$ and $D^{(2)}_{yy'}(S) = \infty$. Especially, we call $D^{(2)}_{uu'}(S)$ the *second distance* between u and u' in $G[S \cup \{u, u'\}]$. In addition, we call a path whose length is the second distance a *second shortest path*. Moreover, we write $D^{(1)}_{uwv}(S)$ and $D^{(2)}_{uwv}(S)$ for the distance and the second distance from u to v via a vertex w, respectively. Let P and P' be respectively a v-u shortest path and a v-u second shortest path. Since P and P' do not share e_0 but do share their ends, H must have a cycle including v and u, where H is a subgraph of G such that $V(H) = V(P) \cup V(P')$ and $E(H) = E(P) \cup E(P')$. Figure 2(C) shows an

example of a cycle made by P and P'. To compute the candidate set efficiently, we will use the following lemmas. In the following lemmas, let X and Y be two iterations such that X is the parent of Y, and v be a vertex in $C(S(X))$ such that $S(Y) = S(X) \cup \{v\}$.

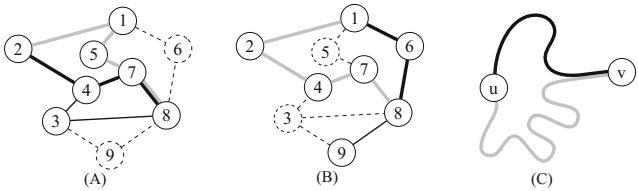

Fig. 2. (A) and (B) show two induced subgraphs. (C) shows a shortest path and a second shortest path. Dashed edges and vertices are not contained by induced subgraphs. Black and gray paths show respectively shortest and second shortest paths.

Lemma 4. *Let u and w be two vertices in $C(S(X))$ and $k = g(G[S(X)])$. (A) $g(G[S(X) \cup \{u, w\}]) \geq k$ if and only if (B) $D^{(1)}_{uw}(S(X)) + D^{(2)}_{uw}(S(X)) \geq k$.*

Proof. Clearly, (A) $\rightarrow$ (B) holds by definition of $D^{(1)}(S(X))$ and $D^{(2)}(S(X))$. For the direction (B) $\rightarrow$ (A), consider a shortest cycle C in $G[S(X) \cup \{u, w\}])$ in the following three cases: (I) $u, w \notin C$: $|C| \geq k$ since $g(G[S(X)]) \geq k$. (II) Either u or w in C: $|C| \geq k$ since u and w belong to $C(S(X))$. (III) Both u and w in C: C can be decomposed into two u-w paths P and Q. Without loss of generality, $|P| \leq |Q|$. If P is a u-w shortest path, then $|C| \geq k$ from (B), since Q is at least as long as the *second distance* $D^{(2)}_{uw}(S(X))$. Otherwise, there is a u-w shortest path P' and a cycle C' consisting of a part of P (or Q) and a part of P'. If C' contains w, then $|C'| = |C| \geq k$ since C is a shortest cycle. If C' does not contain w, then $|C'|$ is a cycle in $G[S(X) \cup \{u\}]$, thus $|C'| \geq k$ because $u \in C(S(X))$. □

Lemma 5. `EBG-IS` *computes $C(S(Y))$ in $O(|C(S(X))| + |N(v)|)$ time.*

Proof. From Lemma 4, vertex u in $C(S(X))$ belongs to $C(S(Y))$ if and only if $D^{(1)}_{uv}(S(X)) + D^{(2)}_{uv}(S(X)) \geq k$. This can be done in constant time. In addition, from the connectivity of $G[S(Y)]$, $C(S(Y)) \setminus C(S(X)) \subseteq N(v)$. Thus, we can find $C(S(Y)) \setminus C(S(X))$ in $O(|C(S(X))| + |N(v)|)$ time. □

Next, we consider how to update the values of $D^{(1)}(S(Y))$ and $D^{(2)}(S(Y))$ when adding v to $S(X)$. We can update the old distances to the ones after adding v as in the Floyd-Warshall algorithm (see Algorithm 2), meaning that we can compute $D^{(1)}(S(Y))$ in $O(|S(X) \cup C(S(X))| \cdot |C(S(X))|)$ time. By the following lemma, the values of $D^{(2)}(S(Y))$ can be updated in $O(|S(Y)|)$ time for each pair of vertices in $C(S(Y))$.

Lemma 6. *Let u and w be two vertices in $C(S(X))$, e_0 be an edge in a u-w shortest path in $G[S(X) \cup \{u, w\}]$, and $H = G[S(X) \cup \{u, w\}] \setminus \{e_0\}$. If $N_H(u) = \emptyset$, then $D^{(2)}_{uw}(S(X)) = \infty$. Otherwise, $D^{(2)}_{uw}(S(X)) = \min_{y \in N_H(u)}\{D^{(1)}_{yw}(S(X)) + 1\}$.*

Proof. From the definition of $D^{(2)}_{uw}(S(X))$, if $N_H(u) = \emptyset$, then $D^{(2)}_{uw}(S(X)) = \infty$. We assume $|N_H(u)| \geq 1$. Since $u \notin S(X)$, every shortest path between u and w in $G[S(X) \cup \{w\}] \cup f$ contains f, where $f = \{u, y\}$. Hence, $D^{(1)}_{yw}(S(X)) + 1$ is equal to the distance between u and w in $G[S(X) \cup \{w\}] \cup \{f\}$. Hence, the statement holds. □

The next lemma implies that if $D^{(1)}_{uw}(S(X)) + D^{(2)}_{uw}(S(X)) < k$, i.e., $G[S(X) \cup \{u, w\}]$ is not a solution, then computing $D^{(2)}_{uw}(S(Y))$ takes constant time.

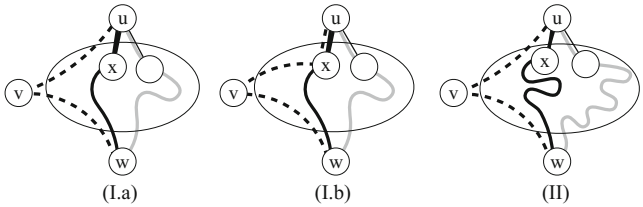

Fig. 3. Examples of each case in Lemma 7. Solid lines are u-v shortest paths in $G[S(X) \cup \{u, w\}]$. Gray solid lines are u-v second shortest paths in $G[S(X) \cup \{u, w\}]$. Dashed lines are u-v-w shortest paths in $G[S(Y) \cup \{u, w\}]$. Let $\{u, x\}$ be the first edge in a shortest path: the sum of lengths of a solid and gray solid line is less than k.

Lemma 7. *Let u and w be two vertices in $C(S(Y))$. If $p_1 + p_3 < k$, then $D^{(2)}_{uw}(S(Y)) = \min\{\max\{p_1, p_2\}, p_3\}$, where $p_1 = D^{(1)}_{uw}(S(X))$, $p_2 = D^{(1)}_{uvw}(S(Y))$, and $p_3 = D^{(2)}_{uw}(S(X))$.*

Proof. Let $G_X = G[S(X) \cup \{u, w\}]$ and $G_Y = G[S(Y) \cup \{u, w\}]$. Note that $p_1 \leq p_3$. We consider the following cases: (I) $p_1 < p_2$: Let $e = \{u, x\}$ be the first edge of a u-w shortest path P in G_Y. Note that P cannot contain v. (I.a) There exists a u-v-w shortest path Q that does not contain e: clearly, $D^{(2)}_{uw}(S(Y)) = \min\{|Q| = p_2, p_3\}$. (I.b) Every u-v-w shortest path Q contains e: there always exists a cycle C in $S(Y) \cup \{w\}$ such that $V(C) \subseteq (V(P) \cup V(Q)) \setminus \{u\}$ and C does not contain u. Note that $|C| < p_1 + p_2$. If $p_2 \leq p_3$, then this contradicts $w \in C(S(Y))$ since $|C| < k$. Thus, $p_2 > p_3$. This implies that $|Q| - 1 \geq p_3$. Hence, $D^{(2)}_{uw}(S(Y)) = p_3$. (II) $p_2 \leq p_1$: this assumption implies that there exists a u-w shortest path P in G_Y that contains v, and $p_1 + p_2 < k$. Let e be the first edge of P in G_Y and Q be a u-v-w shortest path in $G_Y \setminus \{e\}$. Now, we can see $|Q| > p_1$ since if $|Q| \leq p_1$, then $u \notin C(S(Y))$ since P and Q make a cycle C containing u with $|C| < k$. Thus, the length of a u-w shortest path in $G_Y \setminus \{e\}$ is p_1, and $D^{(2)}_{uw}(S(Y)) = p_1$ holds. □

Algorithm 2 shows in detail the update of the candidate set, $D^{(1)}(\cdot)$, and $D^{(2)}(\cdot)$ (done using Lemma 7). We analyze the time complexity of EBG-IS. Let $ch(X)$ be the set of children of X and $\#gch(X)$ be the number of grandchildren of X. The next lemma shows the time complexity for updating $D^{(2)}(S(X))$.

Lemma 8. *We can compute* $D^{(2)}(S(Y))$ *from* $D^{(2)}(S(X))$ *in* $O(\#gch(Y) \cdot |S(Y)| + |C(S(Y))|^2)$ *time.*

Proof. Let u and w be two vertices in $C(S(Y))$. Two cases are possible: (I) $D^{(1)}_{uw}(S(X)) + D^{(2)}_{uw}(S(X)) \geq k$: By Lemma 6, computing $D^{(2)}_{uw}(S(Y))$ takes $O(|S(Y)|)$ time, checking only vertices in $S(Y)$. As the number of pairs (u, w) that fit this case is bounded by $\#gch(Y)$, EBG-IS needs $O(\#gch(Y) \cdot |S(Y)|)$ time to compute this part. (II) $D^{(1)}_{uw}(S(X)) + D^{(2)}_{uw}(S(X)) < k$: From Lemma 7, computing $D^{(2)}_{uw}(S(Y))$ takes constant time, for a total complexity of $O(|C(S(Y))|^2)$, which proves the statement. □

Theorem 2. EBG-IS *enumerates all solutions in* $O(\sum_{S \in \mathcal{S}} |N[S]|)$ *time using* $O(\max_{S \in \mathcal{S}}\{|N[S]|^3\})$ *space, where* $\mathcal{S}$ *is the set of all solutions.*

Proof. The correctness of EBG-IS follows from Lemma 3. We first consider the space complexity. In an iteration X, EBG-IS uses $O(|C(S(X)) \cup S(X)|^2)$ space for storing values of $D^{(1)}(\cdot)$ and $D^{(2)}(\cdot)$. In addition, the height of $\mathcal{T}$ is at most $\max_{S \in \mathcal{S}}\{|S|\}$. Therefore, EBG-IS uses $O(\max_{S \in \mathcal{S}}\{|N[S]|^3\})$ space.

Let $c(X)$ be $|C(S(X))|$ and $T(X, Y)$ be the time needed to generate Y from X, i.e., an execution of NextC() (Algorithm 2). From Lemma 5, Lemma 6, and the Floyd-Warshall algorithm, $T(X, Y)$ is $O(c(X) + |N(v)| + c(Y) \cdot |S(X)| + \#gch(Y) \cdot |S(Y)| + c(Y)^2)$ time. In addition, $|N[S(X)]| \leq |N[S(Y)]|$, $|N(v)| = O(|N[S(Y)]|)$, and $c(X) = O(N[S(X)])$ since every vertex in the candidate set has a neighbor in $S(X)$. Thus, $T(X, Y) = O(|N[S(Y)]|\,(c(Y) + \#gch(Y)))$ time. Note that the sum of children and grandchildren for all iterations is at most $2|\mathcal{V}|$. Thus, by distributing the $O(|N[S(Y)]|)$ time from X to children and grandchildren of Y, each iteration needs $O(|N[S(Y)]|)$ time since each iteration receives costs only from the parent and the grandparent. In addition, each iteration outputs a solution, and hence the total time is $O(\sum_{S \in \mathcal{S}} |N[S]|)$. □

5 Subgraph Enumeration

We propose an algorithm, EBG-S, for enumerating all subgraphs with girth k in a given graph G, detailed in Algorithm 3. A trivial adaptation of EBG-IS would run in $O(m)$ time per solution, as the candidate sets are sets of edges, whose size is $O(m)$. To improve this running time, EBG-S selects candidates in a certain order, so that the number of candidate edges does not exceed no more than the number of nodes in the previous solution $G[S]$.

Let S be the current solution. Note that S is an edge set. We first define an inner edge and an outer edge as follows: an edge $e = \{u, v\}$ is an *inner edge*

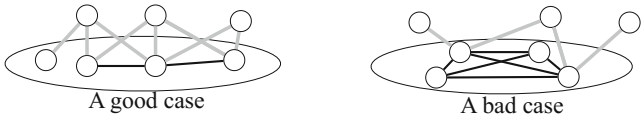

Fig. 4. Black solid lines and gray solid lines represent inner edges and outer edges, respectively. Our main strategy is to reduce the number of inner edges in `EBG-S`.

for S if $u, v \in G[S]$, and an *outer edge* otherwise (see Fig. 4). Let $C_{\mathrm{in}}(S)$ and $C_{\mathrm{out}}(S)$ be a set of inner edges and outer edges in $C(S)$, respectively. We first consider the case when `EBG-S` picks an outer edge. In the following lemmas, let X be an iteration in enumeration tree $\mathcal{T}$, e be an edge not in X, and Y be the child iteration of X satisfying $S(Y) = S(X) \cup \{e\}$.

Lemma 9. *Let $e = \{x, y\}$ be an outer edge such that $x \in V(G[S(X)])$. Then $C(S(Y)) \subseteq (C(S(X)) \cup E(y)) \setminus \{e\}$, where $E(y)$ are the edges incident to y.*

Proof. An edge $g \notin E(y) \cup C(S(X))$ may not be added to $S(Y)$ as the resulting subgraph would be disconnected, and $e \notin C(S(Y))$ since $e \in S(Y)$. □

From Lemma 9, `EBG-S` manages the candidate set $C(S(Y))$ in $O(|C(S(Y))| + |V(G[S(X)])|)$ time when `EBG-S` picks an outer edge e since we can add all edges $e' \notin S(X) \cup C(S(X))$ incident to y and $S(Y) \cup \{e'\}$ is a solution. Moreover, removed edges are at most $|V(G[S(X)])|$ since all removed edges have a vertex in $V(G[S(X)])$. In this case, `EBG-S` can obtain $C_{\mathrm{in}}(S(Y))$ and $C_{\mathrm{out}}(S(Y))$ in $O(S(X))$ time and $O(C(S(Y)))$ time, respectively. Next, we consider that when `EBG-S` picks an inner edge e. When we pick an inner edge, $C(S(Y))$ is monotonically decreasing.

Lemma 10. *If e is an inner edge, then $C_{\mathrm{in}}(S(Y)) \subset C_{\mathrm{in}}(S(X))$ and $C_{\mathrm{out}}(S(Y)) = C_{\mathrm{out}}(S(X))$.*

Proof. Since e is an inner edge $V(G[S(Y)]) = V(G[S(X)])$, thus there is no edge $f \in C_{\mathrm{in}}(S(Y)) \setminus C_{\mathrm{in}}(S(X))$. Since $e \notin C_{\mathrm{in}}(S(Y))$ and no edge in $C_{\mathrm{out}}(S(X))$ is in $C_{\mathrm{in}}(S(Y))$, $C_{\mathrm{in}}(S(Y)) \subset C_{\mathrm{in}}(S(X))$. Moreover, there is no cycle including $f \in C_{\mathrm{out}}(S(X))$ in $G[S(Y) \cup \{f\}]$, hence $C_{\mathrm{out}}(S(Y)) = C_{\mathrm{out}}(S(X))$. □

Next, for any pair of edges e and f not in $G[S(X)]$, we consider the computation of the girth of $G[S(X) \cup \{e, f\}]$ in `EBG-S`. Let $A(X) = \{v \in V(G[S(X)]) \mid E(v) \cap C(S(X)) \neq \emptyset\}$. In a similar fashion as `EBG-IS`, `EBG-S` uses $D^{(3)}(S(X))$ for $A(X)$. The definition of $D^{(3)}(S(X))$ is as follows: For any pair of vertices u and v in $A(X)$, $D^{(3)}_{uv}(S(X))$ is the distance between u and v in $A(X)$. Note that a shortest path between u and v may contain a vertex in $G[S] \setminus A(X)$. The next lemma shows that by using $D^{(3)}(S(X))$, we can compute $C(S(Y))$ in $O(|V(G[S(Y)])|)$ time from $C(S(X))$.

Lemma 11. *For any iteration X, $|C_{\mathrm{in}}(S(X))| \leq |V(G[S(X)])|$.*

Algorithm 3. Updating data structures in `EBG-S`.

Procedure `NextC`$(C(S), D^{(3)}(S), S, k, G)$
  if $C_{\mathrm{in}}(S) \neq \emptyset$ **then** $e \leftarrow C_{\mathrm{in}}(S)$; **else** $e \leftarrow C_{\mathrm{out}}(S)$;
  $C(S \cup \{e\}) \leftarrow$ `UpdateCand`(e, S);
  $D^{(3)}(S \cup \{e\}) \leftarrow$ `Update3`$(e, C(S \cup \{e\}))$;
Function `UpdateCand`$(e = \{u, v\}, S)$
  if $e \in C_{\mathrm{in}}(S)$ **then**
    for $f \in C_{\mathrm{in}}(S) \setminus \{e\}$ **do**
      if $g(G[S \cup \{e, f\}]) \geq k$ **then** $C_{\mathrm{in}}(S) \leftarrow C_{\mathrm{in}}(S) \cup \{f\}$;
  else // We assume $u \in G[S]$ and $v \notin G[S]$
    for $w \in N(v)$ **do** // Let f be an edge $\{v, w\}$
      if $g(G[S \cup \{e, f\}]) < k$ **then** $C_{\mathrm{out}}(S) \leftarrow C_{\mathrm{out}}(S) \setminus f$;
      else if $w \in G[S]$ **then**
        $(C_{\mathrm{in}}(S), C_{\mathrm{out}}(S)) \leftarrow (C_{\mathrm{in}}(S) \cup f, C_{\mathrm{out}}(S) \setminus f)$
      else $C_{\mathrm{out}}(S) \leftarrow C_{\mathrm{out}}(S) \cup f$;
  return $C_{\mathrm{in}}(S) \cup C_{\mathrm{out}}(S)$;
Function `Update3`$(e = \{u, v\}, C(S \cup \{e\}))$
  $A = \{v \in V(G[S]) \mid v$ is incident to $C(S).\}$;
  for $x, y \in A$ **do** // If $e \in C_{\mathrm{out}}(S)$, then $u \in V(G[S]), v \notin V(G[S])$
    if $e \in C_{\mathrm{in}}(S)$ **then**
      $D^{(3)}_{xy}(S) \leftarrow \min\{D^{(3)}_{xy}(S), D^{(3)}_{xu}(S) + D^{(3)}_{vy}(S) + 1, D^{(3)}_{xv}(S) + D^{(3)}_{uy}(S) + 1\}$;
    else $D^{(3)}_{xy}(S) \leftarrow \min\{D^{(3)}_{xy}(S), D^{(3)}_{xu}(S) + 1\}$;
  return $D^{(3)}(S)$;

Proof. The proof follows from these facts: (A) Initially, $C_{\mathrm{in}}(S(X)) = \emptyset$. (B) Choosing $e \in C_{\mathrm{in}}(S(X))$ decreases $|C_{\mathrm{in}}(S(Y))|$. (C) $e = \{x, y\} \in C_{\mathrm{out}}(S(X))$ is chosen iff $|C_{\mathrm{in}}(S(X))| = 0$, and (assuming wlog $y \notin V(G[S(X)])$) it increases $|C_{\mathrm{in}}(S(Y))|$ by at most $|\{\{y, z\} : z \in V(G[S(X)])\}| < |V(G[S(X)])|$. □

Lemma 12. $|C_{\mathrm{out}}(S(X)) \setminus C_{\mathrm{out}}(S(Y))| + |C_{\mathrm{out}}(S(Y)) \setminus C_{\mathrm{out}}(S(X))| \leq V(G[S(Y)])$.

Proof. We consider two cases: (I) $C_{\mathrm{in}}(S(X)) \neq \emptyset$: `EBG-S` picks $e \in C_{\mathrm{in}}(S(X))$, and thus, From Lemma 10, $C_{\mathrm{out}}(S(Y)) = C_{\mathrm{out}}(S(X))$. (II) $C_{\mathrm{in}}(S(X)) = \emptyset$: `EBG-S` picks $e = \{u, v\} \in C_{\mathrm{out}}(S(X))$. Without loss of generality, we can assume that $u \in V(G[S(X)])$ and $v \notin V(G[S(X)])$. Let f be an edge $\{v, w\}$ incident to v. Now, $w \in V(G[S(Y)])$. This implies that the number of edges that are added to $C_{\mathrm{out}}(S(Y))$ and removed from $C_{\mathrm{out}}(S(X))$ is at most $|V(G[S(Y)])|$. □

Note that $|V(G[S(X)])| \leq |V(G[S(Y)])|$. Hence, from the above lemmas, we can obtain the following lemma.

Lemma 13. *$C(S(Y))$ can be computed in $O(|V(G[S(Y)])|)$ time from $C(S(X))$.*

Theorem 3. `EBG-S` *enumerates all connected subgraphs with girth k in $O(\sum_{S \in \mathcal{S}} |V(G[S])|)$ total time using $O(\max_{S \in \mathcal{S}}\{|V(G[S])|^3\})$ space.*

Proof. The proof can be obtained by adapting that of Theorem 2. A more detailed proof can be found in the appendix. □

6 Conclusion

In this paper, we addressed the k-girth connected induced/edge subgraph enumeration problems. We proposed two algorithms: `EBG-IS` for induced subgraphs and `EBG-S` for edge subgraphs. Both algorithms have $O(n)$ time delay and require $O(n^3)$ space (exact bounds are reported in Table 1). The algorithms can easily be adapted to relax the connectivity constraint and consider weighted graphs. Other possibilities include applying the algorithms for network analysis and considering the more challenging problem of enumerating maximal subgraphs.

Table 1. Summary of our result. $\mathcal{S}$ is the set of all solutions.

	Total time	Total space
`EBG-IS`	$O(\sum_{S\in\mathcal{S}} \|N[S]\|)$	$O(\max_{S\in\mathcal{S}}\{\|N[S]\|^3\})$
`EBG-S`	$O(\sum_{S\in\mathcal{S}} \|V(G[S])\|)$	$O(\max_{S\in\mathcal{S}}\{\|V(G[S])\|^3\})$

References

1. Alon, N., Hoory, S., Linial, N.: The moore bound for irregular graphs. Gr. Comb. **18**(1), 53–57 (2002)
2. Bollobás, B.: Extremal Graph Theory. Courier Corporation (2004)
3. Chandran, L.S.: A high girth graph construction. SIAM J. Discrete Math. **16**(3), 366–370 (2003)
4. Chang, H.-C., Lu, H.-I.: Computing the girth of a planar graph in linear time. SIAM J. Comput. **42**(3), 1077–1094 (2013)
5. Conte, A., Kanté, M.M., Otachi, Y., Uno, T., Wasa, K.: Efficient enumeration of maximal k-degenerate subgraphs in a chordal graph. In: Cao, Y., Chen, J. (eds.) COCOON 2017. LNCS, vol. 10392, pp. 150–161. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62389-4_13
6. Conte, A., Kurita, K., Wasa, K., Uno, T.: Listing acyclic subgraphs and subgraphs of bounded girth in directed graphs. In: Gao, X., Du, H., Han, M. (eds.) COCOA 2017. LNCS, vol. 10628, pp. 169–181. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-71147-8_12
7. Ferreira, R., Grossi, R., Rizzi, R.: Output-sensitive listing of bounded-size trees in undirected graphs. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 275–286. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23719-5_24
8. Itai, A., Rodeh, M.: Finding a minimum circuit in a graph. SIAM J. Comput. **7**(4), 413–423 (1978)
9. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Inf. Process. Lett. **27**(3), 119–123 (1988)

10. Kurita, K., Wasa, K., Arimura, H., Uno, T.: Efficient enumeration of dominating sets for sparse graphs. arXiv preprint arXiv:1802.07863 (2018)
11. Lazebnik, F., Ustimenko, V.A., Woldar, A.J.: A new series of dense graphs of high girth. Bull. Am. Math. Soc. **32**(1), 73–79 (1995)
12. Parter, M.: Bypassing Erdős' girth conjecture: hybrid stretch and sourcewise spanners. In: Esparza, J., Fraigniaud, P., Husfeldt, T., Koutsoupias, E. (eds.) ICALP 2014. LNCS, vol. 8573, pp. 608–619. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43951-7_49
13. Read, R.C., Tarjan, R.E.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks **3**(5), 237–252 (1975)
14. Shioura, A., Tamura, A., Uno, T.: An optimal algorithm for scanning all spanning trees of undirected graphs. SIAM J. Comput. **26**(3), 678–692 (1997)
15. Wasa, K., Arimura, H., Uno, T.: Efficient enumeration of induced subtrees in a K-degenerate graph. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 94–102. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13075-0_8