Amortized $\tilde{O}(|V|)$-Delay Algorithm for Listing Chordless Cycles in Undirected Graphs*

Rui Ferreira[1], Roberto Grossi[2], Romeo Rizzi[3], Gustavo Sacomoto[4,5], and Marie-France Sagot[4,5]

[1] Microsoft Bing, UK
[2] Università di Pisa, Italy
[3] Università di Verona, Italy
[4] INRIA Grenoble Rhône-Alpes, France
[5] UMR CNRS 5558 - LBBE, Université Lyon 1, France

Abstract. Chordless cycles are very natural structures in undirected graphs, with an important history and distinguished role in graph theory. Motivated also by previous work on the classical problem of listing cycles, we study how to list chordless cycles. The best known solution to list all the C chordless cycles contained in an undirected graph $G = (V, E)$ takes $O(|E|^2 + |E| \cdot C)$ time. In this paper we provide an algorithm taking $\tilde{O}(|E| + |V| \cdot C)$ time. We also show how to obtain the same complexity for listing all the P chordless st-paths in G (where C is replaced by P).

1 Introduction

A *chordless (induced) cycle* c in an undirected graph G is a cycle such that the subgraph induced by its vertices contains exactly the edges of c. A chordless cycle is called a *hole* when its length is at least 4. Similarly, a *chordless (induced) path* π in G is such that the subgraph of G induced by π contains exactly the edges of π. Both chordless cycles and paths are very natural structures in undirected graphs with an important history, appearing in many papers in graph theory related to chordal graphs, perfect graphs and co-graphs (e.g. [11,6,3]), as well as many NP-complete problems involving them (e.g. [2,7,9]).

In this paper we consider algorithms for listing chordless cycles and st-paths in an undirected graph $G = (V, E)$, with $n = |V|$ vertices and $m = |E|$ edges, motivated by the algorithms for listing cycles and st-paths that have been produced by an active area of research since the early 70s [10,13,1].

In this paper we present an algorithm for listing all the C chordless cycles in an undirected graph $G = (V, E)$ in $\tilde{O}(m+n \cdot C)$ time, hence with an amortized $\tilde{O}(n)$ time delay, where $\tilde{O}(f(n, m))$ is used as a shorthand for $O(f(n, m) \operatorname{polylog} n)$. We also show that the same algorithm may be used to list all the P chordless st-paths in $\tilde{O}(m + n \cdot P)$ time, hence amortized $\tilde{O}(n)$ time delay.

* GS and MFS were partially supported by the ERC programme FP7/2007-2013 / ERC grant agreement no. [247073]10, and the French project ANR-12-BS02-0008 (Colib'read). RG was partially supported by Italian project PRIN 2012C4E3KT (AMANDA).

A. Schulz and D. Wagner (Eds.): ESA 2014, LNCS 8737, pp. 418–429, 2014.

There are very few algorithms in the literature for listing chordless cycles and/or paths, where some of them have no guaranteed performance [12,16]. The most notable and elegant listing algorithm is by Uno [15], with a cost of $O(m^2 + m \cdot C)$ time for chordless cycles and $O(m^2 + m \cdot P)$ time for chordless st-paths, hence amortized $O(m)$ time delay.

2 Preliminaries

Our graphs are finite, undirected, and *simple*, *i.e.* without self-loops or parallel edges. Given a graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges, our task is to list out fast all its chordless cycles. We hence assume that G is connected. Given $V' \subseteq V$, we denote by $E\langle V'\rangle := \{uv \in E \mid u, v \in V'\}$ the set of those edges which are contained in V'. A graph $G' = (V', E')$ is called a *subgraph* of G if $V' \subseteq V$ and $E' \subseteq E$. The subgraph G' is called *induced* (or *chordless*) if $E' = E\langle V'\rangle$. For any $V' \subseteq V$, we denote by $G[V'] := (V', E\langle V'\rangle)$ the *subgraph of G induced by V'*. Given $e \in E$, we denote by $G \setminus e := (V, E \setminus \{e\})$ the subgraph obtained from G by *deleting* the edge e. Given $v \in V$, we denote by $G \setminus v := G[V \setminus \{v\}]$ the subgraph obtained from G by first deleting all the edges incident to v, and then removing the isolated vertex v. Given a vertex $u \in V$, we denote by $N_G(u) := \{v \in V \mid uv \in E\}$ the *neighbourhood* of u, the subscript is omitted whenever the graph is clear from the context.

A *cycle* is a connected graph in which every vertex has degree 2. A *path* is a connected graph in which every vertex has degree 2 except for two degree-1 vertices, s and t, called the *endvertices* of the path. This is also called an *st-path* and denoted by π_{st}. Indeed, when building a path from s to t edge after edge, it will be most natural, and more precise, to think like we are orienting the traversed edges. For this reason, we will also write (u, v) for an edge that, when building a path, has been traversed from u to v.

A (chordless) path (or cycle) of G is a (chordless) subgraph of G which is a path (or cycle). We denote by $\mathcal{C}(G)$ the set of all chordless cycles in G. We denote by $\mathcal{P}(G)$ (by $\mathcal{P}_{st}(G)$) the set of all chordless paths (st-paths) in G. When $s = t$, we get those cycles visiting s. We refer to a path $\pi \in \mathcal{P}(G)$ by its natural sequence of vertices or edges. A *hole* is a chordless cycle of size at least 4. Thus $\mathcal{C}(G)$ comprises holes and triangles. Since there are at most mn triangles, our algorithm can be used to list the holes of G in $\tilde{O}(n)$ time each, with an overall $\tilde{O}(mn^2)$ additive time cost.

Uno [15] proposed an algorithm that lists each chordless cycle in an undirected graph $G = (V, E)$ in $O(m)$ time while using $O(m)$ space. The first step is the following reduction to the problem of enumerating the chordless st-path in a graph G. Based on the fact that for any vertex $s \in V$ the chordless cycles in $G \setminus s$ are also chordless cycles in G, the algorithm proceeds by listing all chordless cycles passing through s; and repeating the process in $G \setminus s$, until the graph is empty. Then, to list all chordless cycles passing through s in $G' = G$, the algorithm follows the approach of listing the chordless paths $s \rightsquigarrow t$ in $G' \setminus (s, t)$, for each $t \in N_{G'}(s)$; and to avoid duplications, at the end of each iteration the graph is updated to $G' = G' \setminus t$.

Given a previously computed chordless st-path $\pi = v_0 v_1 \dots v_l$, Uno's algorithm identifies the set of vertices $U \subseteq V$ such that each $u \in U$ is adjacent to some $v_j \in \pi$, and the edge (v_j, u) is contained in a chordless st-path $\pi' \neq \pi$ extending the prefix $\pi_j = v_0 v_1 \dots v_j$. The algorithm is kick-started by taking a shortest st-path (as a shortest path has the property of also being a chordless path) and employs a recursive strategy of vertex removal to avoid listing the same chordless path multiple times. This ensures that each chordless path is listed once. Uno's algorithm takes $O(m)$ time to compute U and prepare the recursive calls before it either outputs a new path or stops. The total time is therefore $O(m^2 + m \cdot |\mathcal{C}(G)|)$.

3 Our Approach and Key Ideas

We outline the main ideas which allow us to reduce the amortized cost for a chordless cycle from $O(m)$ to $\tilde{O}(n)$, giving a total $\tilde{O}(m + n \cdot |\mathcal{C}(G)|)$ time to list all the chordless cycles. Our approach relies on a variant of the *cleaning* operation introduced in [6] to recognize linear balanced matrices and even holes in graphs [4,5].

3.1 Certificates for Chordless *st*-path

A listing algorithm usually takes the form of a recursive procedure exploring the space of all solutions. A key idea employed since the first listing papers [10] is to check for the existence of at least one solution before branching, *i.e.* before partitioning the solution space in subspaces to be assigned to the children. This avoids unproductive recursive calls, *i.e.* calls that do not list any solution and whose overhead cost could completely dominate the cost of reporting the solutions (*e.g.* see [14]). In a previous work [1], we stressed the notion of certificate since, in a more refined recursive scheme, passing a certificate of existence as an extra parameter may facilitate the work of the children which may avoid running the existence check: if they have a single child, they could be done by just passing the certificate received as an input or a small adaptation of it. We also saw that more structural facts around the certificate could be useful. For the case of st-paths [1], the certificate is a DFS tree rooted in s and reaching t, which contained an st-path and also helped in other ways. Until now, the certificate was itself a solution or explicitely contained one.

Here we try out something new: what if our certificate guarantees the existence of a solution but is *not* itself a solution? The following fact suggests that the certificate for the existence of a chordless st-path might be just any st-path.

Fact 1. *Given two vertices s, t in G, there is a* chordless *st-path in G iff there is an st-path in G.*

Thus we allow for certificates which are somewhat less refined than actual solutions, in the same spirit that a binary heap demands a less strict and lazy notion of order. This is a new asset of the notion of certificate and opens up new possibilities.

3.2 From Chordless Cycles to Chordless *st*-paths

Uno [15] shows how to reduce listing chordless cycles in a graph to listing chordless *st*-paths for all edges (s, t) chosen in a specific order (see Section 2), which is necessary to avoid duplications in the output. The initialization step for each edge (s, t) takes $O(m)$ time as it requires to find one chordless *st*-path. This gives the m^2 term in the total cost of $O(m^2 + m \cdot |\mathcal{C}(G)|)$ for chordless cycles.

We observed in Section 3.1 that any *st*-path will suffice as a starter, as they are our certificates of choice. This makes a difference for the above reduction, since using dynamic graph connectivity algorithms [8], it is possible to maintain a spanning tree in $O(\text{polylog}\, n)$ time per edge deletion, perform connectivity queries in $O(\text{polylog}\, n)$, and more importantly obtain an *st*-path in $\tilde{O}(n)$. It is worth noting that it is not known how to obtain a chordless *st*-path faster than $O(m)$. Hence, we first build the dynamic connectivity structure as preprocessing step. Then, for each edge (s, t), in the same order as Uno's reduction, we list the chordless *st*-paths. Before calling our path listing algorithm for edge (s, t), we test if s and t are connected (Fact 1): if so, we call our path listing algorithm, paying $\tilde{O}(n)$ to find one initial *st*-path; otherwise, we skip the edge (s, t) and take the next in order. As a result, the total initialization cost is $\tilde{O}(m + kn)$ for all edges instead of $O(m^2)$, where k is the number of edges for which we find one initial *st*-path. Note that $k \leq |\mathcal{C}(G)|$ as each of them surely gives rise to a chordless *st*-path whence to a distinct chordless cycle. We obtain in this way an $\tilde{O}(m + n \cdot |\mathcal{C}(G)|)$ time algorithm to list chordless cycles, if we can list *st*-paths in amortized $\tilde{O}(n)$ time each.

3.3 Difficulty of Cleaning *st*-paths

Given any *st*-path, as stated in Fact 1 we can *clean* it to obtain a chordless *st*-path in a greedy fashion: start from $u = s$ and iteratively take a neighbour of u that is closest to t along the path. The process stops when $u = t$. The vertices taken in this way form a chordless path. The problem is that the cost of such a greedy traversal of the path is upper bounded by the sum of the degrees of the vertices along it. Unfortunately, this sum could be $\Theta(m)$ in the worst case.

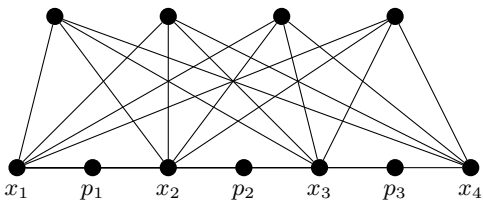

Fig. 1. Sum of degrees on chordless path $x_1, p_1, x_2, p_2, \ldots, x_{r-1}, p_{r-1}, x_r$ is $\Theta(m)$

Even worse, this is still true when the initial path is already chordless, as shown in Fig. 1. Consider the complete bipartite clique $K_{r,r} = (V_1 \cup V_2, E_{12})$,

where $V_1 = \{x_1, x_2, \ldots, x_r\}$. Build a new graph $G = (V, E)$ where the vertex set is $V = V_1 \cup V_2 \cup \{p_1, \ldots, p_{r-1}\}$ for some new vertices $p_1, \ldots, p_{r-1}$, and the edge set is $E = E_{12} \cup \{(x_1, p_1), (p_1, x_2), (x_2, p_2), \ldots, (x_{r-1}, p_{r-1}), (p_{r-1}, x_r)\}$. Now, the path $x_1, p_1, x_2, p_2, \ldots, x_{r-1}, p_{r-1}, x_r$ is chordless but each edge is incident to at least one vertex in that path, so the sum of the degrees is $m = |E| = \Theta(r^2) = \Theta(|V|^2) = \Theta(n^2)$.

What we would like to do: recursively extend a given chordless path π_{su} into a chordless st-path, while maintaining as a certificate an st-path. The recursive extension can be seen as an implicit cleaning of our st-path certificate. Consider a vertex u along a given st-path (our certificate), where initially $u = s$. Our certificate guarantees that there is at least one chordless st-path going through a neighbour of u, say a. However, exploring all of u's neighbours would cost too much so we need to proceed more carefully: consider any neighbour $b \neq a$, the following two situations may occur. *(1)* a and b are both good, meaning that (u, a) and (u, b) are on two distinct chordless st-paths. In this case, the chordless st-paths traversing (u, a) cannot go through b too, as otherwise it would not be chordless (see Remark 1 below), so b should be removed. *(2)* b is not on any chordless st-path, so it is either disconnected from t or every st-path going through b passes through a. In this case, as it will be clear later, we need neither to explore nor to remove b.

In other words, we can treat the neighbours of u as described above, and they will not interfere when cleaning the st-path in the next recursive calls since their are either removed (as in case 1) or implicitly cut out (as in case 2). We make this statement more precise below.

3.4 Reduced Degree Property

We introduce a notion of reduced degree with a stronger property in mind. Consider a chordless st-path $\pi_{st} = v_0 v_1 \ldots v_\ell$ in the graph G, for some integer $\ell > 1$, where $v_0 = s$ and $v_\ell = t$. For a vertex v_i, a neighbour $v \in N(v_i)$ is *good* if there exists a chordless st-path in G with prefix $v_0 v_1 \ldots v_i v$ (*i.e.* it extends $v_0 v_1 \ldots v_i$ by adding the edge (v_i, v) as illustrated in Fig. 2). We denote by $N^{good}(v_i) \subseteq N(v_i)$ the set of *good* neighbours of v_i, noting that $v_{i+1} \in N^{good}(v_i)$. For each v_i, its *reduced degree* d_i is given by the number of non-good neighbours, namely, $d_i = |(N(v_i) \setminus \bigcup_{j \leq i} N^{good}(v_j)) \cup \{v_{i+1}\}|$.

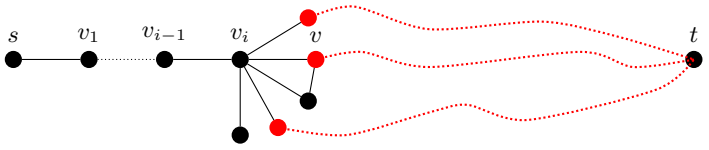

Fig. 2. Good neighbours (in red) of vertex v_i in G_i

The rationale is that exploring the good neighbours of v_i will list further chordless paths while examining its neighbours that are not good is a waste of computation. The reduced degree of v_i is actually an upper bound on the number of not-good vertices examined when exploring v_i to produce the chordless st-path π_{st} and gives an upper bound on the waste. Lemma 1 below shows that while examining the neighbours of the vertices along a chordless path still takes $O(m)$ time, only $O(n)$ neighbours are a waste while the remaining ones lead to further chordless paths (which is a good argument for amortization).

Lemma 1. *For a chordless path π_{st}, we have $\sum_{v_i \in \pi_{st}} d_i \leq 2n$, where d_i is the reduced degree of $v_i \in \pi_{st}$.*

Proof. We will show that each vertex x of G is a non-good neighbour of at most two vertices in π_{st}. To this purpose, we prove that if x is a non-good neighbour of both v_i and v_j then $|i - j| \leq 1$. We choose such three vertices v_i, v_j and x where the difference $j - i$ is the largest possible and assume by contradiction that $i < j - 1$. Thus v_i and v_j are not adjacent in π_{st}, whence (v_i, v_j) is not an edge of G since π_{st} is chordless. Also, being non-good, $x \notin \pi_{st}$. Consider the st-path $\pi^* = v_0 \dots v_i x v_j \dots v_l$. Clearly, π^* contains no repeated vertices and we will prove that π^* is a chordless st-path, contradicting the fact that x is not a good neighbour of v_i. The fact that π^* is chordless follows from the fact that there is no $v_k \in \pi^*$, $k \neq i$ and $k \neq j$, such that (v_k, x) is an edge of G, otherwise $j - k$ or $k - i$ would be strictly larger than $j - i$, contradicting our choice of v_i, v_j and x. □

3.5 Cleanup of Current Vertex

Suppose we are extending the chordless path π_{su}, while cleaning the st-path certificate. We identify a good vertex $v \in N(u)$, which closest to t along the st-path. Ideally, we would clean the vertex u by throwing away all its other neighbours but this could cost $\Omega(m)$ per chordless path as illustrated in Fig. 1 and discussed in Section 3.3. We thus perform a partial cleaning, called *cleanup*, which consists in identifying and removing, among all neighbours of u (*i.e.* $|N(u)|$ elements) only its set $N^{good}(u)$ of good ones.

For a given u in a chordless st-path $\pi_{st} = v_0 \dots v_i u \dots v_l$, we let emerge the good neighbours in $N^{good}(u)$ one by one as follows. Consider the graph G' where the vertices $v_0 \dots v_i$ and its good neighbours were removed. If u and t are not connected, then there cannot be further chordless paths from u and so there cannot be further good neighbours. Otherwise, if u and t are connected, we take any path from u to t, and select its neighbour v that appears along the path and is closest to t, as illustrated in Fig. 3. After that, we remove v and its incident edges, and iterate what described above until u is disconnected from t. The vertices v thus selected form the set $N^{good}(u)$ of good neighbours.

Lemma 2. *For a chordless path π_{st}, the cleanup of vertex $u \in \pi_{st}$ correctly produces the set $N^{good}(u)$ of its good neighbours.*

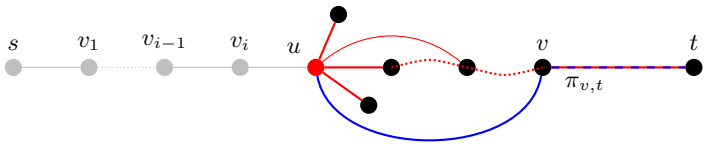

Fig. 3. Cleanup of the neighbours of vertex u

4 Listing Algorithm

We blend the key ideas discussed in Section 3 to get Algorithm 1, which has four parameters as input and lists all the chordless st-paths: the first parameter is the chordless path π_{su} partially built from s to the current vertex u (initially, $u = s$), which is the second parameter; the third parameter is a ut-path π_{ut} that plays the role of certificate by Fact 1; the fourth parameter is the reduced graph G, which changes with the recursive calls.

Algorithm 1. `list_induced_paths`$_{s,t}(\pi_{su}, u, \pi_{ut}, G)$

```
1  if u = t then
2  |  output(π_su)
3  else
4  |  S := ∅
5  |  while true do
6  |  |  v := the vertex in π_ut ∩ N(u) that is closest to t in π_ut
7  |  |  π_vt := the subpath of π_ut from v to t
8  |  |  S := S ∪ {(v, π_vt)}
9  |  |  remove v and its incident edges from G
10 |  |  if u and t are not connected then break
11 |  |  π_ut := any path from u to t
12 |  end
13 |  foreach (v, π_vt) ∈ S do
14 |  |  adds back v and its incident edges to G
15 |  |  list_induced_paths_{s,t}(π_su · (u, v), v, π_vt, G)
16 |  |  remove v and its incident edges from G
17 |  end
18 end
```

The algorithm outputs a chordless st-path if $u = t$ (line 2). Otherwise, it performs a cleanup of u (the loop at lines 5–12). After that, it explores only the good neighbours recursively as they will surely lead to further chordless paths (the other loop at lines 13–17). Observe that S stores the good neighbours v of u and a vt-path for each of them: when performing the recursive call at line 15, only one of the vertices in S appears in the reduced graph G passed as a parameter to

the recursive call (see lines 14 and 16 that guarantee this, and Remark 1 below). Hence, the recursive call now has as parameters the chordless sv-path $\pi_{su} \cdot (u, v)$ ending in v, and a vt-path that guarantees that a chordless st-path exists and has $\pi_{su} \cdot (u, v)$ as a prefix. This recursive call lists all the chordless st-paths that share this prefix.

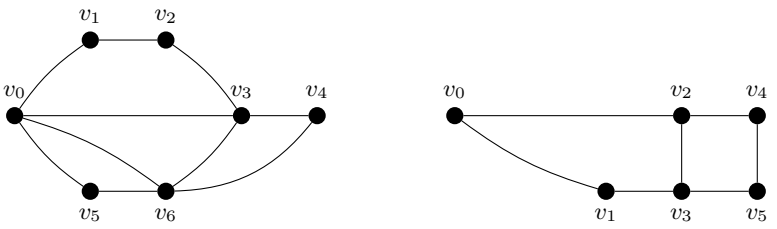

Fig. 4. Two example graphs where $s = v_0$ and $t = v_4$

For example, let us run Algorithm 1 on the input graph shown on the left of Fig. 4, with $u = s = v_0$ and the initial path $\pi_{ut} = v_0v_1v_2v_3v_4$. It computes the pairs (v, π_{vt}) in S as follows. First, (v_3, v_3v_4) is added to S as v_3 is a good neighbour for v_0 (the neighbour closest to t in the path), and the edges incident to v_3 are removed. After this removal, $s = v_0$ is still connected to $t = v_4$ through the path $v_0v_5v_6v_4$, which becomes the input for the next iteration of the while loop. Next, (v_6, v_6v_4) is added to S as v_6 is another good neighbour, and the edges incident to v_6 are removed disconnecting s from t, so the while loop ends. The recursive calls in the foreach loop give the two chordless paths $v_0v_3v_4$ and $v_0v_6v_4$ contained in the graph.

Remark 1. It is important to run the recursive calls with all good neighbours in S removed except one. If we left two or more good neighbours in the recursive call of line 15, they could interfere with each other and we might not obtain the chordless paths correctly. A very simple example is given in the graph shown on the right of Fig. 4. Consider for instance the case where Algorithm 1 would be given as input the path $v_0v_1v_3v_2v_4$. The pair (v_2, v_2v_4) is added to S and v_2 removed. After that, the path $v_0v_1v_3v_5v_4$ is found and the pair $(v_1, v_1v_3v_5v_4)$ is added to S and v_1 removed. Since v_0 and v_4 become disconnected, S contains all the good neighbours of v_0. Algorithm 1 executes the recursive calls with S. Suppose that we keep both good neighbours v_1 and v_2 in G during these calls, in particular for the call with the pair $(v_1, v_1v_3v_5v_4)$ from S. This call will extend in a nested call the chordless path to $\pi_{su} = v_0v_1v_3$ for $u = v_3$, and will claim that the good neighbours of v_3 are v_2 and v_5, which is incorrect since $v_0v_1v_3v_2$ is not chordless. This situation does not arise if v_2 is kept deleted in G when the recursive call on v_1 is performed as done in Algorithm 1.

The correctness of Algorithm 1 follows mostly from Lemma 2. Recall that, it guarantees that for a given path prefix π_{su} the set S contains the good neighbours of u, *i.e.* the neighbours of u that belong to at least one chordless st-path

extending π_{su}. Clearly, we only have to recursively call the algorithm for these neighbours, the others certainly lead to no solution. This implies that Algorithm 1 tries all the possibilities to extend π_{su}, so all chordless st-paths are output. Moreover, since each good neighbour of u leads to a different extension, we have that no st-path is output more than once.

Certainly only st-paths are output by Algorithm 1, but at this point we have no guarantees that the paths are indeed chordless. In fact, after building S, the algorithm proceeds to recursively extend the prefix $\pi_{su} \cdot (u, v)$ for each $v \in S$ in the graph $G' = G \setminus (S \setminus \{v\})$. However, since u was included in current path none of its neighbours can be used later in the recursion. The algorithm removes the good neighbours of u from G, but the other neighbours, $N_G(u) \setminus S$, are still present in G'. They could thus be used to extend the path later in the recursion, resulting in a non-chordless st-path. Lemma 3 shows that this cannot happen.

Lemma 3. *The st-paths output by Algorithm 1 are chordless.*

The previous lemma leads to the following theorem.

Theorem 1. *The algorithm correctly outputs all chordless st-paths of G.*

Theorem 2. *The algorithm takes $O(m + |\mathcal{P}_{st}(G)|(t_p + nt_q + nt_u))$ time, where t_p is the cost of choosing any path from any two given vertices, t_q is the cost of checking if any given two vertices are connected or not, and t_u is the cost of removing/adding back any given edge.*

Proof. See Section 5.

There are several dynamic data structures in the literature [8] that maintain a spanning forest for a dynamic graph, supporting insertions and deletions of edges in polylogarithmic time. Consequently, $t_p = O(n \operatorname{polylog}(n))$, $t_q = O(\operatorname{polylog}(n))$, and $t_u = O(\operatorname{polylog}(n))$, thus giving the following bound.

Corollary 1. *The algorithm takes $\tilde{O}(m + |\mathcal{P}_{st}(G)| \cdot n)$ time to report all the chordless st-paths.*

5 Amortized Analysis

Before starting our analysis, we observe some simple properties of the recursion tree generated by Algorithm 1.

Fact 2. *The recursion tree R of Algorithm 1 has the following properties:*

1. *There is a one-to-one correspondence between paths in $\mathcal{P}_{st}(G)$ and leaves in the recursion tree.*
2. *There is a one-to-one correspondence between proper prefixes of paths in $\mathcal{P}_{st}(G)$ and internal nodes in the recursion tree.*

3. *The number of branching nodes is* $|\mathcal{P}_{st}(G)| - 1$.
4. *The length of a root-to-leaf path is equal to the length of the chordless st-path corresponding to the leaf. In particular, the height of the tree is* $\leq n$.

Fact 2 suggests us to follow the following overall strategy.

1. We analyze the cost of each type (leaf, unary and branching) of node separately.
2. We consider all branching nodes together, and show that their amortized cost is $O(t_p + t_q + nt_u + n) = \tilde{O}(n)$ per solution.
3. We consider all unary nodes together, and show that their amortized cost is $O(|\pi_{st}|t_q + nt_u) = \tilde{O}(n)$ per solution.
4. We deduce that the cost of each solution is $O(t_p + nt_q + nt_u) = \tilde{O}(n)$.

Where the cost of a node is the time spent by the corresponding call without including the time spent by its nested recursive calls.

Lemma 4. *The cost of a leaf is* $O(|\pi_{st}|)$.

Let us now analyze the cost of the unary nodes. Let $r = \langle \pi_{su}, u, \pi_{ut}, G \rangle$ be a unary node. The vertex $v \in N(u)$ is the only neighbour of u that can extend the prefix π_{su} into a chordless st-path. Thus, removing v from G disconnects u from t, and the algorithm performs a single iteration of the loop in line 5, not executing line 11. In this case, the algorithm performs the following operations: (i) one connectivity query (line 10), (ii) $|N(v)|$ edge update operations on G (lines 9, 14 and 16), and (iii) a scan in the intersection of $N(u)$ and π_{ut} to find v (line 6). The cost of (i) and (ii) is $O(t_q + |N(v)|t_u)$.

A naive implementation of (iii) takes $O(|N(u)|+|\pi_{ut}|)$ time, which is too large to fit in our amortization strategy. In order to reduce this cost to $O(|N(u)|)$ we therefore maintain, as an extra invariant, for each vertex in the current graph its distance to t along the path π_{ut}. In this way, we can find v simply scanning $N(u)$. Thus, assuming the distance information is correctly maintained, we complete the proof of Lemma 5.

Lemma 5. *The cost of a unary node is* $O(t_q + |N(v)|t_u + |N(u)|)$, *where* (u, v) *is the edge added to the chordless path.*

It is not hard to maintain the distance information for $\langle \pi_{su} \cdot (u, v), v, \pi_{vt}, G' \rangle$, the only child of the unary node $\langle \pi_{su}, u, \pi_{ut}, G \rangle$. As the path π_{vt} is a suffix of π_{ut}, the distance of the vertices in π_{ut} does not change. On the other hand, the only vertices that the distances can change are the ones in π_{vt} but not in π_{ut}. These vertices can be identified when scanning $N(v)$ in the child node $\langle \pi_{su} \cdot (u, v), v, \pi_{vt}, G' \rangle$, since their distance is strictly larger than $|\pi_{vt}|$. It remains to show that the distance information can be maintained in the branching nodes.

Lemma 6. *The cost of a branching node* $r \in R$ *is* $O(\beta(r)(t_p + t_q + nt_u))$, *where* $\beta(r)$ *is the number of children of* r.

Let us now show that we can maintain the distance information in branching nodes in the same time bound of Lemma 6. This follows from the fact that in each iteration of the loop (line 5) we are already paying $O(|\pi_{ut}|)$, *i.e.* a full traversal of the path π_{ut}. Before each recursive call in line 15 we can traverse the path π_{ut} adding for each vertex the distance information, *i.e.* their position in the path.

At this point we have bounds for the cost of each node in the recursion tree. However, by directly applying them we cannot achieve our goal of $\tilde{O}(n)$ time per solution. For instance, consider the particular case where all internal nodes of the recursion tree are branching. The cost of each internal node is $O(\beta(r)(t_p + t_q + nt_u)) = \tilde{O}(n^2)$, since $\beta(r) = \Omega(n)$ in the worst case. Then, from item 3 of Fact 2, the number of branching nodes is $|\mathcal{P}_{st}(G)| - 1$. The total cost for the tree is thus $\tilde{O}(|\mathcal{P}_{st}(G)| \cdot n^2)$ or $\tilde{O}(n^2)$ per solution.

In order to get a tighter bound for the total cost of the branching nodes, we use the following amortization strategy. Let $r \in R$ be a branching node. We divide the cost $O(\beta(r)(t_p + t_q + nt_u))$ among the closest descendents that are branching nodes or leaves (no unary nodes), each being charged $O(t_p + t_q + nt_u)$. This can always be done since r has $\beta(r)$ children and the subtree of each child contains at least one leaf, *i.e.* the node r has at least $\beta(r)$ non-unary descendants. In this way, the original cost of node r is completely charged to its non-unary descendants, and the only cost that remains associated to r is the one received from its ancestors. Finally, each branching node can only be charged once, by its lowest non-unary ancestor. Each branching node and each leaf is therefore charged with $O(t_p + t_q + nt_u)$. Thus, the total cost of the branching nodes is $O(|\mathcal{P}_{st}(G)|(t_p + t_q + nt_u))$, completing the proof of Lemma 7.

Lemma 7. $\sum_{r:\text{branching}} T(r) = O(|\mathcal{P}_{st}(G)|(t_p + t_q + nt_u))$.

Let us now bound the total cost of the unary nodes. Similarly to the branching nodes case, a straightforward use of the bound given by Lemma 5 leads to an $\tilde{O}(n^2)$ cost per solution, since in the worst case the recursion tree can have $O(n)$ unary nodes for each leaf. The key idea to obtain a better amortized cost is to consider the bound on the reduced degrees given by Lemma 1.

We first observe that each unary node is contained in some root-to-leaf path $\Pi(l)$, where l is a leaf of the recursion tree. Thus,

$$\sum_{r:\text{unary}} T(r) \leq \sum_{l:\text{leaf}} \sum_{r \in \Pi(l)} T(r). \tag{1}$$

Fact 2 implies that there is a one-to-one correspondence between the prefixes of paths in $\mathcal{P}_{st}(G)$ and nodes in the recursion tree. That is, each leaf corresponds to a solution, and the root-to-leaf path $\Pi(l)$ corresponds to the chordless st-path associated to the leaf l. Moreover, the $O(t_q + |N(v)|t_u + |N(u)|)$ cost of an unary node can be amortized to $O(t_q + |N(v)|t_u)$, since we can always charge $|N(u)| = O(n)$ to its single child. We can thus rewrite the double sum as

$$\sum_{l:\text{leaf}} \sum_{r \in \Pi(l)} T(r) = \sum_{\pi \in \mathcal{P}_{st}(G)} \sum_{v_i \in \pi} (t_q + |N(v_i)|t_u). \tag{2}$$

For each chordless st-path π in the internal sum of Eq. 2, we have that the degrees are actually the reduced degrees of Section 3.4, since the good neighbours (*i.e.* the set S in Algorithm 1) are always removed. Using Lemma 1 we can thus bound the sum of the degrees by $2n$. Therefore,

$$\sum_{r:\text{unary}} T(r) \leq \sum_{\pi \in \mathcal{P}_{st}(G)} (|\pi| t_q + 2nt_u), \tag{3}$$

completing the proof of Lemma 8.

Lemma 8. $\sum_{r:\text{unary}} T(r) = O(\sum_{\pi \in \mathcal{P}_{st}(G)} |\pi| t_q + nt_u)$.

As a corollary of Lemmas 8 and 7, we obtain Theorem 2.

References

1. Birmelé, E., Ferreira, R.A., Grossi, R., Marino, A., Pisanti, N., Rizzi, R., Sacomoto, G.: Optimal listing of cycles and st-paths in undirected graphs. In: SODA 2013, pp. 1884–1896. ACM/SIAM (2013)
2. Chen, Y., Flum, J.: On parameterized path and chordless path problems. In: IEEE Conference on Computational Complexity, pp. 250–263 (2007)
3. Chudnovsky, M., Robertson, N., Seymour, P., Thomas, R.: The strong perfect graph theorem. Annals of Mathematics 164, 51–229 (2006)
4. Conforti, M., Cornuéjols, G., Kapoor, A., Vuskovic, K.: Recognizing balanced 0, +/- matrices. In: SODA 1994, pp. 103–111. ACM/SIAM (1994)
5. Conforti, M., Cornuéjols, G., Kapoor, A., Vuskovic, K.: Finding an even hole in a graph. In: FOCS 1997, pp. 480–485. IEEE Computer Society (1997)
6. Conforti, M., Rao, M.R.: Structural properties and decomposition of linear balanced matrices. Math. Program. 55, 129–168 (1992)
7. Haas, R., Hoffmann, M.: Chordless paths through three vertices. Theoretical Computer Science 351(3), 360–371 (2006)
8. Kapron, B.M., King, V., Mountjoy, B.: Dynamic graph connectivity in polylogarithmic worst case time. In: SODA, pp. 1131–1142 (2013)
9. Kawarabayashi, K.-I., Kobayashi, Y.: The induced disjoint paths problem. In: Lodi, A., Panconesi, A., Rinaldi, G. (eds.) IPCO 2008. LNCS, vol. 5035, pp. 47–61. Springer, Heidelberg (2008)
10. Read, C., Tarjan, R.E.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks 5(3), 237–252 (1975)
11. Seinsche, D.: On a property of the class of n-colorable graphs. Journal of Combinatorial Theory, Series B 16(2), 191–193 (1974)
12. Sokhn, N., Baltensperger, R., Bersier, L.-F., Hennebert, J., Ultes-Nitsche, U.: Identification of chordless cycles in ecological networks. In: Glass, K., Colbaugh, R., Ormerod, P., Tsao, J. (eds.) Complex 2012. LNICST, vol. 126, pp. 316–324. Springer, Heidelberg (2013)
13. Maciej, M.: Syslo. An efficient cycle vector space algorithm for listing all cycles of a planar graph. SIAM J. Comput. 10(4), 797–808 (1981)
14. Uno, T.: Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In: Leong, H.-V., Jain, S., Imai, H. (eds.) ISAAC 1997. LNCS, vol. 1350, pp. 92–101. Springer, Heidelberg (1997)
15. Uno, T.: An output linear time algorithm for enumerating chordless cycles. In: 92nd SIGAL of Information Processing Society Japan, pp. 47–53 (2003) (in Japanese)
16. Wild, M.: Generating all cycles, chordless cycles, and hamiltonian cycles with the principle of exclusion. J. of Discrete Algorithms 6(1), 93–102 (2008)