Efficient Bubble Enumeration in Directed Graphs

Etienne E. Birmelé, Pierluigi Crescenzi, Rui Ferreira, Roberto Grossi, Vincent Lacroix, Andrea Marino, Nadia Pisanti, Gustavo Sacomoto, Marie-France Sagot

Efficient bubble enumeration in directed graphs*

E. Birmelé[1,3], P. Crescenzi[4], R. Ferreira[5], R. Grossi[5], V. Lacroix[1,2], A. Marino[1,4], N. Pisanti[5], G. Sacomoto[1,2], M.-F. Sagot[1,2]

[1] INRIA Grenoble Rhône-Alpes, France
[2] Université de Lyon 1, Villeurbanne, France
[3] Université d'Évry, France
[4] Dipartimento di Sistemi e Informatica, Università di Firenze, Firenze, Italy
[5] Dipartimento di Informatica, Università di Pisa, Pisa, Italy

Abstract. Polymorphisms in DNA- or RNA-seq data lead to recognisable patterns in a de Bruijn graph representation of the reads obtained by sequencing. Such patterns have been called mouths, or bubbles in the literature. They correspond to two vertex-disjoint directed paths between a source s and a target t. Due to the high number of such bubbles that may be present in real data, their enumeration is a major issue concerning the efficiency of dedicated algorithms. We propose in this paper the first linear delay algorithm to enumerate all bubbles with a given source.

1 Introduction

In recent papers [2, 4], algorithms for identifying two types of polymorphim, respectively SNPs (Single Nucleotide Polymorphisms) in DNA, and alternative splicing in RNA-seq data were introduced. Both correspond to recognisable patterns in a de Bruijn graph (DBG) built from the reads provided by a sequencing project. In both cases, the pattern corresponds to two vertex-disjoint paths between a pair of source and target vertices s and t. Properties on the lengths or sequence similarity of the paths then enable to differentiate between different types of polymorphism.

Such patterns have been studied before in the context of genome assembly where they have been called bulges [8] or bubbles [1, 3, 12]. However, the purpose in these works was not to enumerate all these patterns, but "only" to remove them from the graph, in order to provide longer contigs for the genome assembly. More recently, ad-hoc enumeration methods have been proposed but are restricted to non-branching bubbles [6], *i.e.*, each vertex from the bubble has in-degree and out-degree 1, except for s and t. Furthermore, in all these applications [1, 3, 6, 8, 12] since the patterns correspond to SNPs or sequencing errors, the authors only considered paths of length smaller than a constant. On the other hand, bubbles of arbitrary length have been considered in the context

* This work was supported by the french ANR MIRI BLAN08-1335497 Project and the ERC Advanced Grant Sisyphe held by Marie-France Sagot. Partially supported by Italian project PRIN AlgoDEEP (2008TFBWL4) of MIUR. The second author received additional support from the Italian PRIN project 'DISCO'.

of splicing graphs [9]. However, in this context, a notable difference is that the graph is a DAG. Additionally, vertices are coloured and only unicolour paths are then considered for forming bubbles. Finally, the concept of bubble also applies to the area of phylogenetic networks [5], where it corresponds to the notion of a recombination cycle. Again for this application, the graph is a DAG.

In this paper, we adopt the term bubble, which is being most used in the community, and this will denote two vertex-disjoint paths between a pair of source and target vertices with no condition on the path length or the degrees of the internal nodes. We then consider the more general problem of enumerating all bubbles in a arbitrary directed graph. That is, our solution is not restricted to acyclic or de Bruijn graphs. This problem is quite general but it was still an open question whether a polynomial-delay algorithm could be proposed for solving it. The algorithm presented in [2] was an adaptation of Tiernan's algorithm for cycle enumeration [11] which does not have a polynomial delay, in the worst case the time elapsed between the output of two solutions is proportional to the number of paths in the graph, i.e. exponential in the size of the graph. It was not clear at the time if more efficient cycle enumeration methods in directed graphs such as Tarjan's [10] or Johnson's [7] could be adapted to efficiently enumerate bubbles in directed graphs.

The aim of this paper is to show a non trivial adaptation of Johnson's cycle (what he called elementary circuit) enumeration algorithm to identify all bubbles in a directed graph in the same theoretical complexity. Notably, the method we propose enumerates all bubbles with a given source with $O(|V| + |E|)$ delay, where V, resp. E, is the set of vertices, resp. arcs, of the graph. The algorithm requires an initial transformation, described in Section 3, of the graph for each source s that takes $O(|V|+|E|)$ time and space. Moreover, we briefly describe, in Section 6, a slightly more complex version of the algorithm (but with the same overall complexity) that is more space and time efficient in practice.

2 De Bruijn graphs and bubbles

A de Bruijn graph (DBG) is a directed graph $G = (V, E)$ whose set of vertices V are labelled by k-mers, i.e. words of length k. An arc in E links a vertex u to a vertex v if the suffix of length $k-1$ of u is a prefix of v. Given two vertices s and t in G, an (s,t)-path is a path from s to t. By an (s,t)-bubble, we mean two vertex-disjoint (s,t)-paths that only shares s and t.

In the case of next generation sequencing (NGS) data, the k-mers correspond to all words of length k present in the reads (strings) of the input dataset, and only those. In relation to the classical de Bruijn graph for all possible words of size k, the DBG for NGS data may then not be complete. Vertices may also be labelled by the number of times each k-mer is present in the reads. In general a vertex will be labelled by both a k-mer and its reverse complement, and the DBG used in practice will thus be a bi-directed multigraph. Figure 1 gives an example of a portion of a DBG that corresponds to a bubble generated by a SNP or a sequencing error.

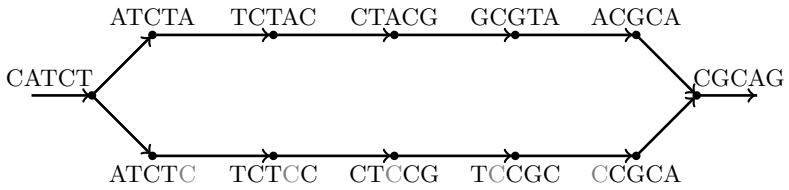


Fig. 1: Bubble due to a substitution (gray letter).

In this paper, we ignore all details related to the treatment of NGS data using De Bruijn graphs that are not essential for the algorithm described, and consider instead the more general case of finding all (s,t)-bubbles in an arbitrary directed graph.

3 Turning bubbles into cycles

Let $G = (V, E)$ be a directed graph, and let $s \in V$. We want to find all (s,t)-bubbles for all possible target vertices t. We transform G into a new graph $G'_s = (V'_s, E'_s)$ where $|V'_s| = 2|V|$ and $|E'_s| = O(|V| + |E|)$. Namely,

$$V'_s = \{v, \overline{v} \mid v \in V\}$$

$$E'_s = \{(u,v), (\overline{v}, \overline{u}) \mid (u,v) \in E \text{ and } v \neq s\} \cup \{(v, \overline{v}) \mid v \in V \text{ and } v \neq s\} \cup \{(\overline{s}, s)\}$$

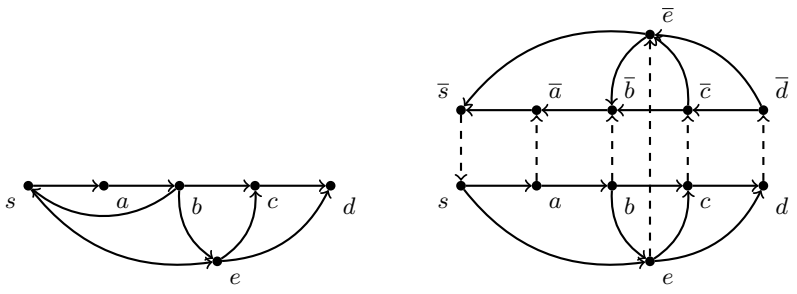
Let us denote by $\overline{V}$ the set of vertices of G'_s that were not already in G, that is $\overline{V} = V'_s \setminus V$. The two vertices $x \in V$ and $\overline{x} \in \overline{V}$ are said to be *twin vertices*. Observe that the graph G'_s is thus built by adding to G a reversed copy of itself, where the copy of each vertex is referred to as its *twin*. The arcs incoming to s (and outgoing from $\overline{s}$) are not included so that the only cycles in G'_s that contain s also contain $\overline{s}$. New arcs are also created between each pair of twins: the new arcs are the ones leading from a vertex u to its twin $\overline{u}$ for all u except for s where the arc goes from $\overline{s}$ to s. An example of a transformation is given in Figure 2.

We define a cycle of G'_s as being *bipolar* if it contains vertices of both V and $\overline{V}$. As the only arc from $\overline{V}$ to V is $(\overline{s}, s)$, then every bipolar cycle C contains also only one arc from V to $\overline{V}$. This arc, which is the arc $(t, \overline{t})$ for some $t \in V$, is called the *swap arc* of C. Moreover, since $(\overline{s}, s)$ is the only incoming arc of s, all the cycles containing s are bipolar. We say that C is *twin-free* if it contains no pair of twins except for $(s, \overline{s})$ and $(t, \overline{t})$.

Definition 1 (Bubble-cycle). *A* bubble-cycle *in G'_s is a twin-free cycle of size greater than four*[1].

Proposition 1. *Given a vertex s in G, there is a one-to-two correspondence between the set of (s,t)-bubbles in G for all $t \in V$, and the set of bubble-cycles of G'_s.*

[1] The only twin-free cycles in of size four in G'_s are generated by the outgoing edges of s. There are $O(|V|)$ of such cycles.



(a) Graph G (b) Graph G'_s

Fig. 2: Graph G and its transformation G'_s. We have that $\langle s, e, \overline{e}, \overline{b}, \overline{a}, \overline{s}, s\rangle$ is a bubble-cycle with swap arc $(e, \overline{e})$ that has a correspondence to the (s, e)-bubble composed by the two vertex-disjoint paths $\langle s, e\rangle$ and $\langle s, a, b, e\rangle$.

Proof. Let us consider an (s, t)-bubble in G formed by two vertex-disjoint (s, t)-paths P and Q. Consider the cycle of G'_s obtained by concatenating P (resp. Q), the arc $(t, \overline{t})$, the inverted copy of Q (resp. P), and the arc $(\overline{s}, s)$. Both cycles are bipolar, twin-free, and have $(t, \overline{t})$ as swap arc. Therefore both are bubble-cycles.

Conversely, consider any bubble-cycle C and let $(t, \overline{t})$ be its swap arc. C is composed by a first subpath P from s to t that traverses vertices of V and a second subpath $\overline{Q}$ from $\overline{t}$ to $\overline{s}$ composed of vertices of $\overline{V}$ only. By definition of G'_s, the arcs of the subpath P form a path from s to t in the original graph G; given that the vertices in the subpath $\overline{Q}$ from $\overline{t}$ to $\overline{s}$ are in $\overline{V}$ and use arcs that are those of E inverted, then Q corresponds to another path from s to t of the original graph G. As no internal vertex of $\overline{Q}$ is a twin of a vertex in P, these two paths from s to t are vertex-disjoint, and hence they form an (s, t)-bubble.

Notice that there is a cycle $s, v, \overline{v}, \overline{s}$ for each v in the out-neighborhood of s. Such cycles do not correspond to any bubble in G, and the condition on the size of C allows us to rule them out. □

4 The algorithm

Johnson [7] introduced a polynomial delay algorithm for the cycle enumeration problem in directed graphs. We propose to adapt the principle of this algorithm, the prunned backtracking, to enumerate bubble-cycles in G'_s. Indeed, we use a similiar prunning strategy, modified to take into account the twin nodes. Proposition 1 then ensures that running our algorithm on G'_s for every $s \in V$ is equivalent to the enumeration of (twice) all the bubbles of G. To do so, we explore G'_s by recursively traversing it while maintaining the following three variables. We denote by $N^+(v)$ the set of out-neighbors and $N^-(v)$ as the set of in-neighbors of v.

1. A variable *stack* which contains the vertices of a path (with no repeated vertices) from s to the current vertex. Each time it is possible to reach $\overline{s}$

from the current vertex by satisfying all the conditions to have a bubble-cycle, this stack is completed into a bubble-cycle and its content output.

2. A variable *status(v)* for each vertex v which can take three possible values:
   free: v should be explored during the traversal of G'_s;
   blocked: v should not be explored because it is already in the stack or because it is not possible to complete the current stack into a cycle by going through v – notice that the key idea of the algorithm is that a vertex may be blocked without being on the stack, avoiding thus useless explorations;
   twinned: $v \in \overline{V}$ and its twin is already in the stack, so that v should not be explored.
3. A set $B(v)$ of in-neighbors of v where vertex v is blocked and for each vertex $w \in B(v)$ there exists an arc (w, v) in G'_s (that is, $w \in N^-(v)$). If a modification in the stack causes that v is unblocked and it is possible to go from v to $\overline{s}$ using free vertices, then w should be unblocked if it is currently blocked.

Algorithm 1 enumerates all the bubble-cycles in G by fixing the source s of the (s, t)-bubble, computing the transformed graph G'_s and then listing all bubble-cycles with source s in G'_s. This procedure is repeated for each vertex $s \in V$. To list the bubble-cycles with source s, procedure CYCLE(s) is called. As a general approach, Algorithm 3 uses classical backtracking with a pruned search tree. The root of the recursion corresponds to the enumeration of all bubble-cycles in G'_s with starting point s. The algorithm then proceeds recursively: for each free out-neighbor w of v the algorithm enumerates all bubble-cycles that have the vertices in the current stack plus w as a prefix. If $v \in V$ and $\overline{v}$ is twinned, the recursion is also applied to the current stack plus $\overline{v}$, $(v, \overline{v})$ becoming the current swap arc. A base case of the recursion happens when $\overline{s}$ is reached and the call to CYCLE($\overline{s}$) completed. In this case, the path in *stack* is a twin-free cycle and, if this cycle has more than 4 vertices, it is a bubble-cycle to output.

The key idea that enables to make this pruned backtracking efficient is the block-unblock strategy. Observe that when CYCLE(v) is called, v is pushed in the stack and to ensure twin-free extensions, v is blocked and $\overline{v}$ is twinned if $v \in V$. Later, when backtracking, v is popped from the stack but it is *not necessarily* marked as free. If there were no twin-free cycles with the vertices in the current stack as a prefix, the vertex v would remain blocked and its status would be set to free only at a later stage. The intuition is that either v is a dead-end or there remain vertices in the stack that block all twin-free paths from v to $\overline{s}$. In order to manage the status of the vertices, the sets $B(w)$ are used. When a vertex v remains blocked while backtracking, it implies that every out-neighbor w of v has been previously blocked or twinned. To indicate that each out-neighbor $w \in N^+(v)$ (also, $v \in N^-(w)$ is an *in-neighbor* of w) blocks vertex v, we add v to each $B(w)$. When, at a later point in the recursion, a vertex $w \in N^+(v)$ becomes unblocked, v must also be unblocked as possibly there are now bubble-cycles that include v. Algorithm 2 implements this recursive unblocking strategy.

An important difference between the algorithm introduced here and Johnson's is that we now have three possible states for any vertex, *i.e.* free, blocked

Algorithm 1: Main algorithm

```
1 for s ∈ V do
2 |   stack:=∅;
3 |   for v ∈ G'_s do
4 |   |   status:=free;
5 |   |   B(v) = ∅;
6 |   end
7 |   CYCLE(s);
8 end
```

Algorithm 2: Procedure $UNBLOCK(v)$

```
1 /* recursive unblocking of vertices for which popping v creates a
     path to s̄                                                    */
2 status(v):=free;
3 for w ∈ B(v) do
4 |   delete w from B(v);
5 |   if status(w)==blocked then
6 |   |   UNBLOCK(w);
7 |   end
8 end
```

and twinned, instead of only the first two. The twinned state is necessary to ensure that the two paths of the bubble share no internal vertex. Whenever $\overline{v}$ is twinned, it can only be explored from v. On the other hand, a blocked vertex should never be explored. A twin vertex $\overline{v}$ can be already blocked when the algorithm is exploring v, since it could have been unsuccessfully explored by some other call. In this case, it is necessary to verify the status of $\overline{v}$, as it is shown in the graph of Figure 3a. Indeed, consider the algorithm starting from s with (s, a) and (a, b) being the first two arcs visited in the lower part. Later, when the calls CYCLE($\bar{c}$) and CYCLE($\bar{b}$) are made, since $\bar{a}$ is twinned, both $\bar{b}$ and $\bar{c}$ remain blocked. When the algorithm backtracks to a and explores (a, c), the call CYCLE(c) is made and $\bar{c}$ is already blocked.

Another important difference with respect to Johnson's algorithm is that there is a specific order in which the out-neighborhood of a vertex should be explored. In particular, notice that the order in which Algorithm 3 explores the neighbors of a vertex v is: first the vertices in $N^+(v) \setminus \{\overline{v}\}$ and then $\overline{v}$. A variant of the algorithm where this order would be reversed, visiting first $\overline{v}$ and then the vertices in $N^+(v) \setminus \{\overline{v}\}$, would fail to enumerate all the bubbles. Indeed, intuitively a vertex can be blocked because the only way to reach $\bar{s}$ is through a twinned vertex and when that vertex is untwinned the first one is not unblocked. Indeed, consider the graph in Figure 3b and the twin-first variant starting in s with (s, a) and (a, b) being the first two arcs explored in the lower part of the graph. When the algorithm starts exploring b the stack contains $\langle s, a, b\rangle$. After,

Algorithm 3: Procedure CYCLE(v)

```
1  f:=false;
2  push v;
3  status(v):=blocked;
4  /* Exploring forward the edges going out from v ∈ V                */
5  if v ∈ V then
6  |  if status(v̄)==free then
7  |  |  status(v̄):=twinned;
8  |  end
9  |  for w ∈ N+(v) ∩ V do
10 |  |  if status(w)==free then
11 |  |  |  if CYCLE(w) then
12 |  |  |  |  f:=true;
13 |  |  |  end
14 |  |  end
15 |  end
16 |  if status(v̄)==twinned then
17 |  |  if CYCLE(v̄) then
18 |  |  |  f:=true;
19 |  |  end
20 |  end
21 end
22 /* Exploring forward the edges going out from v ∈ V̄               */
23 else
24 |  for w ∈ N+(v) do
25 |  |  if w == s̄ then
26 |  |  |  output the cycle composed by the stack followed by s̄ and s;
27 |  |  |  f:=true;
28 |  |  end
29 |  |  else if status(w)==free then
30 |  |  |  if CYCLE(w) then
31 |  |  |  |  f:=true;
32 |  |  |  end
33 |  |  end
34 |  end
35 end
36 if f then
37 |  UNBLOCK(v);
38 end
39 else
40 |  for w ∈ N+(v) do
41 |  |  if v ∉ B(w) then
42 |  |  |  B(w) = B(w) ∪ {v};
43 |  |  end
44 |  end
45 end
46 pop v;
47 return f;
```

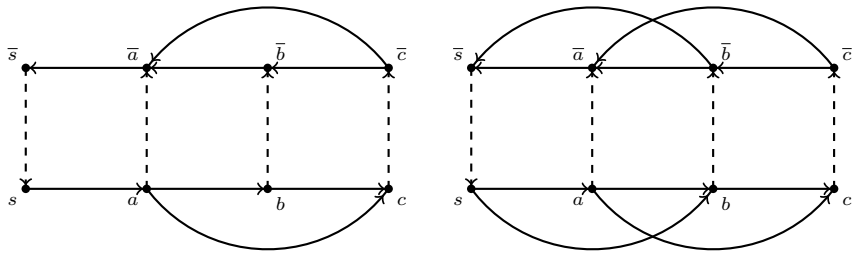


Fig. 3: (a) Example where the twin $\overline{v}$ is already blocked when the algorithm starts exploring v. By starting in s and visiting first (s, a) and (a, b), the vertex $\overline{c}$ is already blocked when the algorithm starts exploring c. (b) Counterexample for the variant of the algorithm visiting first the twin and then the regular neighbors. By starting in s and visiting first (s, a) and (a, b), the algorithm misses the bubble-cycle $\langle s, a, c, \overline{c}, \overline{b}, \overline{s}\rangle$.

the call CYCLE($\bar{b}$) returns *true* and CYCLE(c) returns *false* because $\bar{a}$ and $\bar{b}$ are twinned. After finishing exploring b, the blocked list $B(b)$ is empty. Thus, the only vertex unblocked is b, c (and $\bar{c}$) remaining blocked. Finally, the algorithm backtracks to a and explores the edge (a, c), but c is blocked, and it fails to enumerate $\langle s, a, c, \overline{c}, \overline{b}, \overline{s}\rangle$.

One way to address the problem above would be to modify the algorithm so that every time a vertex $\bar{v}$ is untwinned, a call to UNBLOCK($\bar{v}$) is made. All the bubble-cycles would be correctly enumerated. However, in this case, it is not hard to find an example where the delay would then no longer be linear. Intuitively, visiting first $N^+(v) \setminus \{\bar{v}\}$ and, then $\bar{v}$, works because every vertex u that was blocked (during the exploration of $N^+(v) \setminus \{\bar{v}\}$) should remain blocked when the algorithm explores $\bar{v}$. Indeed, a bubble would be missed only if there existed a path starting from $\overline{v}$, going to $\overline{s}$ through u and avoiding the twinned vertices. This is not possible if no path from $N^+(v) \setminus \{\bar{v}\}$ to u could be completed into a bubble-cycle by avoiding the twinned vertices, as we will show later on.

5 Proof of correctness and complexity analysis

5.1 Proof of correctness: Algorithm 3 enumerates all bubbles with source s

Lemma 1. *Let v be a vertex of G'_s such that status(v) = blocked, S the set of vertices currently in the stack, and T the set of vertices whose status is equal to twinned. Then $S \cup T$ is a $(v, \overline{s})$ separator, that is, each path, if any exists, from v to $\overline{s}$ contains at least one vertex in $S \cup T$.*

Proof. The result is obvious for the vertices in $S \cup T$. Let v be a vertex of G'_s such that $status(v) = blocked$ and $v \notin S \cup T$. This means that when v was popped for the last time, CYCLE(v) was equal to *false* since v remained blocked.

Let us prove by induction on k that each path to $\overline{s}$ of length k from a blocked vertex not in $S \cup T$ contains at least one vertex in $S \cup T$.

We first consider the base case $k = 1$. Suppose that v is a counter-example for $k = 1$. This means that there is an arc from v to $\overline{s}$ ($\overline{s}$ is an out-neighbor of v). However, in that case the output of CYCLE(v) is *true*, a contradiction because v would then be unblocked.

Suppose that the result is true for $k - 1$ and, by contradiction, that there exists a blocked vertex $v \notin S \cup T$ and a path $(v, w, \ldots, \overline{s})$ of length k avoiding $S \cup T$. Since $(w, \ldots, \overline{s})$ is a path of length $k - 1$, we can then assume that w is free. Otherwise, if w were blocked, by induction, the path $(w, \ldots, \overline{s})$ would contain at least one vertex in $S \cup T$, and so would the path $(v, w, \ldots, \overline{s})$.

Since the call to CYCLE(v) returned *false* (v remained blocked), either w was already blocked or twinned, or the call to CYCLE(w) made inside CYCLE(v) gave an output equal to *false*. In any case, after the call to CYCLE(v), w was blocked or twinned and v put in $B(w)$.

The conditional at line 16 of the CYCLE procedure ensures that when untwinned, a vertex immediately becomes blocked. Thus, since w is now free, a call to UNBLOCK(w) was made in any case, yielding a call to UNBLOCK(v). This contradicts the fact that v is blocked. □

Theorem 1. *The algorithm returns only bubble-cycles. Moreover, each of those cycles is returned exactly once.*

Proof. Let us first prove that only bubble-cycles are output. As any call to UNBLOCK (either inside the procedure CYCLE or inside the procedure UNBLOCK itself) is immediately followed by the popping of the considered vertex, no vertex can appear twice in the stack. Thus, the algorithm returns only cycles. They are trivially bipolar as they have to contain s and $\overline{s}$ to be output.

Consider now a cycle C output by the algorithm with swap arc $(t, \overline{t})$. Let (v, w) in C with $v \neq s$ and $v \neq t$. If $\overline{v}$ is free when v is put on the stack, then $\overline{v}$ is twinned before w is put on the stack and cannot be explored until w is popped. If $\overline{v}$ is blocked when v is put on the stack, then by Lemma 1 it remains blocked at least until v is popped. Thus, $\overline{v}$ cannot be in C, and consequently the output cycles are twin-free.

So far we have proven that the output produces bubble-cycles. Let us now show that all cycles $C = \{v_0 = s, v_1, \ldots, v_{l-1}, v_l = \overline{s}, v_0\}$ satisfying those conditions are output by the algorithm, and each is output exactly once.

The fact that C is not returned twice is a direct consequence of the fact that the stack is different in all the leaves of a backtracking procedure. To show that C is output, let us prove by induction that the stack is equal to $\{v_0, \ldots, v_i\}$ at some point of the algorithm, for every $0 \leq i \leq l - 1$. Indeed, it is true for $i = 0$. Moreover, suppose that at some point, the stack is $\{v_0, \ldots, v_{i-1}\}$.

Suppose that v_{i-1} is different from t. As the cycle contains no pair of twins except for those composing the arcs $(s, \overline{s})$ and $(t, \overline{t})$, the path $\{v_i, v_{i+1}, \ldots, v_l\}$

contains no twin of $\{v_0, \ldots, v_{i-1}\}$ and therefore no twinned vertex. Thus, it is a path from v_i to $\overline{s}$ avoiding $S \cup T$. Lemma 1 then ensures that at this point v_i is not blocked. As it is also not twinned, its status is free. Therefore, it will be explored by the backtracking procedure and the stack at some point will be $\{v_0, \ldots, v_i\}$. If $v_{i-1} = t$, $v_i = \overline{t}$ is not blocked using the same arguments. Thus it was twinned by the call to CYCLE(t) and is therefore explored at Line 17 of this procedure. Again, the stack at some point will be $\{v_0, \ldots, v_i\}$. □

5.2 Analysis of complexity: Algorithm 3 has linear delay

As in [7], we show that Algorithm 3 has delay $O(|V| + |E|)$ by proving that a cycle has to be output between two successive unblockings of the same vertex and that with linear delay some vertex has to be unblocked again. To do so, let us first prove the following lemmas.

Lemma 2. *Let v be a vertex such that CYCLE(v) returns true. Then a cycle is output after that call and before any call to UNBLOCK.*

Proof. Let y be the first vertex such that UNBLOCK(y) is called inside CYCLE(v). Since CYCLE(v) returns true, there is a call to UNBLOCK(v) before it returns, so that y exists. Certainly, UNBLOCK(y) was called before UNBLOCK(v) if $y \neq v$. Moreover, the call UNBLOCK(y) was done inside CYCLE(y), from line 37, otherwise it would contradict the choice of y. So, the call to CYCLE(y) was done within the recursive calls inside the call to CYCLE(v). CYCLE(y) must then return true as y was unblocked from it.

All the recursive calls CYCLE(z) made inside CYCLE(y) must return false, otherwise there would be a call to UNBLOCK(z) before UNBLOCK(y), contradicting the choice of y. Since CYCLE(y) must return true and the calls to all the neighbors returned false, the only possibility is that $\overline{s} \in N^+(y)$. Therefore, a cycle is output before UNBLOCK(y). □

Lemma 3. *Let v be a vertex such that there is a $(v, \overline{s})$-path P avoiding $S \cup T$ at the moment a call to CYCLE(v) is made. Then the return value of CYCLE(v) is true.*

Proof. First notice that if there is such a path P, then v belongs to a cycle in G'_s. This cycle may however not be a bubble-cycle in the sense that it may not be twin-free, that is, it may contain more than two pairs of twin vertices. Indeed, since the only constraint that we have on P is that it avoids all vertices that are in S and T when v is reached, then if $v \in V$, it could be that the path P from v to $\overline{s}$ contains, besides s and $\overline{s}$, at least two more pairs of twin vertices. An example is given in Figure 2b. It is however always possible, by construction of G'_s from G, to find a vertex $y \in V$ such that y is the first vertex in P with $\overline{y}$ also in P. Let P' be the path that is a concatenation of the subpath $s \rightsquigarrow y$ of P, the arc $(y, \overline{y})$, and the subpath $\overline{y} \rightsquigarrow \overline{s}$ in P. This path is twin-free, and a call to CYCLE(v) will, by correctness of the algorithm, return true. □

Theorem 2. *Algorithm 3 has linear delay.*

Proof. Let us first prove that between two successive unblockings of any vertex v, a cycle is output. Let w be the vertex such that a call to UNBLOCK(w) at line 37 of Algorithm 3 unblocks v for the first time. Let S and T be, respectively, the current sets of stack and twinned vertices after popping w. The recursive structure of the unblocking procedure then ensures that there exists a (v, w)-path avoiding $S \cup T$. Moreover, as the call to UNBLOCK(w) was made at line 37, the answer to CYCLE(w) is *true* so there exists also a $(w, \overline{s})$-path avoiding $S \cup T$. The concatenation of both paths is a again a $(v, \overline{s})$-path avoiding $S \cup T$. Let x be the first vertex of this path to be visited again. Note that, if no vertex in this path is visited again there is nothing to prove, since v is free, CYCLE(v) needs to be called before any UNBLOCK(v) call. When CYCLE(x) is called, there is a $(x, \overline{s})$-path avoiding the current $S \cup T$. vertices. Thus, applying Lemma 3 and then Lemma 2, we know that a cycle is output before any call to UNBLOCK. As no call to UNBLOCK(v) can be made before the call to CYCLE(x), a cycle is output before the second call to UNBLOCK(v).

Let us now consider the delay of the algorithm. In both its exploration and unblocking phases, the algorithm follows the arcs of the graph and transforms the status or the B lists of their endpoints, which overall require constant time. Thus, the delay only depends on the number of arcs which are considered during two successive outputs. An arc (u, v) is considered once by the algorithm in the three following situations: the exploration part of a call to CYCLE(u); an insertion of u in $B(v)$; a call to UNBLOCK(v). As shown before, UNBLOCK(v) is called only once between two successive outputs. CYCLE(u) cannot be called more than twice. Thus the arc (u, v) is considered at most 5 times between two outputs. This ensures that the delay of the algorithm is $\mathcal{O}(|V| + |E|)$. □

6 Practical Speedup

Speeding up preproccessing. In Section 3, the bubble enumeration problem was reduced to the enumeration of some particular cycles in the transformed graph G'_s for each s. It is worth observing that this does not imply building from scratch G'_s for each s. Indeed, notice that for any two vertices s_1 and s_2, we can transform G'_{s_1} into G'_{s_2} by: (a) removing from G'_{s_1} the arcs $(\overline{s}_1, s_1)$, $(s_2, \overline{s}_2)$, (v, s_2), and $(\overline{s}_2, \overline{v})$ for each $v \in N^-(s_2)$ in G; (b) adding to G'_{s_1} the arcs $(s_1, \overline{s}_1)$, $(\overline{s}_2, s_2)$, (v, s_1), and $(\overline{s}_1, \overline{v})$ for each $v \in N^-(s_1)$ in G.

Avoiding duplicate bubbles. The one-to-two correspondence between cycles in G'_s and bubbles starting from s in G, claimed by Proposition 1, can be reduced to a one-to-one correspondence in the following way. Consider an arbitrary order on the vertices of V, and assign to each vertex of $\overline{V}$ the order of its twin. Let C be a cycle of G'_s that passes through s and contains exactly two pairs of twin vertices. Denote again by t the vertex such that $(t, \overline{t})$ is the arc through which C swaps from V to $\overline{V}$. Denote by *swap predecessor* the vertex before t in C and by *swap successor* the vertex after $\overline{t}$ in C.

Proposition 2. *There is a one-to-one correspondence between the set of (s,t)-bubbles in G for all $t \in V$, and the set of cycles of G'_s that pass through s, contain exactly two pairs of twin vertices and such that the swap predecessor is greater than the swap successor.*

Proof. The proof follows the one of Proposition 1. The only difference is that, if we consider a bubble composed of the paths P_1 and P_2, one of these two paths, say P_1, has a next to last vertex greater than the next to last vertex of P_2. Then the cycle of G'_s made of P_1 and $\overline{P_2}$ is still considered by the algorithm whereas the cycle made of P_2 and $\overline{P_1}$ is not. Moreover, the cycles of length four which are of the type $\{s, t, \overline{t}, \overline{s}\}$ are ruled out as $\overline{s}$ is of the same order as s. □

7 Conclusion

We showed in this paper that it is possible (Algorithm 3) to enumerate all bubbles with a given source in a directed graph with linear delay. Moreover, it is possible to enumerate all bubbles, for all possible sources (Algorithm 1), in $O((|E| + |V|)(|C| + |V|))$ total time, where $|C|$ is the number of bubbles. This required a non trivial adaptation of Johnson's algorithm [7].

References

1. G. Robertson et al. De novo assembly and analysis of RNA-seq data. *Nature methods*, 7(11):909–912, 2010.
2. G. Sacomoto et al. KisSplice: de-novo calling alternative splicing events from rna-seq data. In *RECOMB-seq*, BMC Bioinformatics, 2012.
3. J.T. Simpson et al. ABySS: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123, 2009.
4. P. Peterlongo et al. Identifying SNPs without a reference genome by comparing raw reads. In *SPIRE*, Springer LNCS 6393, pages 147–158, 2010.
5. D. Gusfield, S. Eddhu, and C.H. Langley. Optimal, efficient reconstruction of phylogenetic networks with constrained recombination. *J. Bioinf. and Comput. Biol.*, 2(1):173–214, 2004.
6. Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nature genetics*, 2012.
7. D.B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM J. Comput.*, 4(1):77–84, 1975.
8. P.A. Pevzner, H. Tang, and G. Tesler. *De novo* repeat classification and fragment assembly. In *RECOMB*, pages 213–222, 2004.
9. M. Sammeth. Complete alternative splicing events are bubbles in splicing graphs. *J. Comput. Biol.*, 16(8):1117–1140, 2009.
10. R. E. Tarjan. Enumeration of the elementary circuits of a directed graph. *SIAM Journal on Computing*, 2(3):211–216, 1973.
11. J.C. Tiernan. An efficient search algorithm to find the elementary circuits of a graph. *Commun. ACM*, 13(12):722–726, 1970.
12. D.R. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–829, 2008.