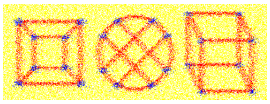
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 25, no. 1, pp. 563–580 (2021)
DOI: 10.7155/jgaa.00571

A family of tree-based generators for bubbles in directed graphs

Vicente Acuña[1] *Leandro Ishi Soares de Lima*[2] *Giuseppe F. Italiano*[3,4]
Luca Pepè Sciarria[5] *Marie-France Sagot*[4,6] *Blerina Sinaimeri*[3,4]

[1]Center for Mathematical Modeling (FB210005),
University of Chile and IRL 2807 CNRS, Santiago, Chile.
[2]European Bioinformatics Institute, Cambridge, UK
[3]LUISS University, Rome, Italy
[4]Erable, INRIA Grenoble Rhône-Alpes, France
[5]University of Rome Tor Vergata, Rome, Italy
[6]Université de Lyon, Université Lyon 1, Laboratoire de Biométrie et Biologie Evolutive, UMR 5558, Villeurbanne France.

Submitted:	February 2021	Reviewed:	September 2021	Revised:	October 2021
Accepted:	October 2021	Final:	October 2021	Published:	October 2021
Article type:	Regular paper			Communicated by:	G. Liotta

Abstract. Bubbles are pairs of internally vertex-disjoint (s,t)-paths in a directed graph. In de Bruijn graphs built from reads of RNA and DNA data, bubbles represent interesting biological events, such as alternative splicing (AS) and allelic differences (SNPs and indels). However, the set of all bubbles in a de Bruijn graph built from real data is usually too large to be efficiently enumerated and analysed in practice. In particular, despite significant research done in this area, listing bubbles still remains the main bottleneck for tools that detect AS events in a reference-free context. Recently, in [1] the concept of a bubble generator was introduced as a way for obtaining a compact representation of the bubble space of a graph. Although this bubble generator was quite effective in finding AS events, preliminary experiments showed that it is about

A preliminary version of this paper was presented at the 31st Intern. Workshop on Combinatorial Algorithms [2].

V. Acuña is supported by CMM ANID PIA FB210005 and ACE210010 - CNRS IRL #2807 and Center for Genome Regulation FONDAP 15090007. G. F. Italiano is partially supported by MIUR, the Italian Ministry for Education, University and Research, under PRIN Project AHeAD (Efficient Algorithms for HArnessing Networked Data). B. Sinaimeri and M.-F. Sagot are partially funded by the French ANR project Aster (2016-2020). Part of this work was done while G. F. Italiano was visiting Université de Lyon and B. Sinaimeri and M.-F. Sagot were visiting LUISS University in Rome.

E-mail addresses: viacuna@dim.uchile.cl (Vicente Acuña) leandro@ebi.ac.uk (Leandro Ishi Soares de Lima) gitaliano@luiss.it (Giuseppe F. Italiano) luca.pepesciarria@gmail.com (Luca Pepè Sciarria) Marie-France.Sagot@inria.fr (Marie-France Sagot) bsinaimeri@luiss.it (Blerina Sinaimeri)

5 times slower than state-of-art methods. In this paper we propose a new family of bubble generators which improve substantially on previous work: bubble generators in this new family are about two orders of magnitude faster and are still able to achieve similar precision in identifying AS events. To highlight the practical value of our new bubble generators, we also report some experimental results on real datasets.

Keywords: bubble generator, directed graphs, alternative splicing

1 Introduction

The advent of sequencing technologies has revolutionised the study of DNA and RNA data. The information contained in the reads coming from genome or transcriptome sequencing is usually represented by a de Bruijn graph (see *e.g.,* [20, 22]). A de Bruijn graph is a graph whose vertices correspond to sequences of length k on the DNA alphabet $\{A, C, T, G\}$, and there is an edge (u, v) if the $k-1$ suffix of u coincides with the $k-1$ prefix of v. An example of a de Bruijn graph is given in Fig. 1.

In this graph *bubbles*, *i.e.,* pairs of internally vertex-disjoint (s, t)-paths, play an important role in the study of genetic variations, which include Alternative Splicing (AS) in RNA-data [18, 23, 22, 24] and SNPs (Single Nucleotide Polymorphism), and indels in DNA-data [12, 26, 27]. Since bubbles can be associated to such biologically relevant events, in recent years there have been several theoretical studies on bubbles (see *e.g.,* [5, 6, 21, 23, 25]), and in particular there has been a growing interest in algorithms for listing all bubbles in a directed graph. However, in real data graphs the number of bubbles can be exponential in the size of the graph. As a consequence, in practice current algorithms are able to list only a subset of the bubble space, thus losing the information related to the bubbles that are left unexplored. Furthermore, not every bubble corresponds to a biological event. Indeed, a significant number of these bubbles can be false positives (*i.e.,* they are not biologically relevant events), and are produced as artifacts of the underlying construction of the de Bruijn graph. In this framework, the main question is how to find a subset of bubbles that can be efficiently computed in practice and that correspond to relevant biological events.

To tackle this question, the notion of bubble generator was first introduced in [1]. Intuitively, a bubble generator is a subset of bubbles of polynomial size, from which all the other bubbles in the graph can be obtained through a suitable application of a specific symmetric difference operator. In particular, the bubble generator proposed in [1] contains at most $m \cdot n$ bubbles, where m and n denote respectively the number of edges and vertices in the input graph. Furthermore, the authors of [1] provided an algorithm that, given any bubble B in the graph, is able to find in $O(n^3)$ time the bubbles of the generator that can be combined to produce B through a symmetric difference operator. To test its practical value, the bubble generator was used to find AS events in a real dataset. As reported in [1], this bubble generator was able to achieve about the same precision in identifying AS events as the state-of-art-algorithm KisSplice [18, 22], but unfortunately building the bubble generator was about 5 times slower than finding AS events with KisSplice. Despite its great theoretical value, this poses a serious limitation on the practical application of this bubble generator to large-scale datasets, which are typical of biological applications.

To address this issue, in this paper we present a new family of bubble generators which improves substantially on the one proposed in [1]. In particular, in the same RNA dataset used in [1], bubble generators in our family are about two orders of magnitude faster in practice than the bubble generator in [1], and improve the precision in identifying AS events from 77.3% to 90%.

Furthermore, as the generators are truly fast in practice, we are now able to analyse a larger dataset that was not possible with the bubble generator proposed in [1]. When compared to the state-of-the-art algorithm for identifying AS events, our bubble generators are also much faster than KisSplice [18, 22], have similar precision, and find AS events that KisSplice cannot find. In the experiments, we observed that our new generators also contain many bubbles that correspond to a particular type of AS event, namely *intron retention* (IR), which is usually considered a hard-to-find event. We believe that our experimental findings make the new bubble generators the method of choice for finding AS events in a reference-free context, especially in large-scale data sets.

From the theoretical viewpoint, our new generators are of minimum size (*i.e.* size $m - n + 1$) for flow graphs, *i.e.,* graphs in which there exists a vertex that can reach all other vertices. In case of general graphs, their size is bounded by $|S|(m - n + 1)$, where S is the source set, *i.e.,* a minimum set of vertices that can reach every other vertex in the graph. Although in the worst case this is asymptotically equivalent to the size of the generator in [1], in our experiments the new generators had a much smaller size in practice. Furthermore, the new generators have a much faster decomposition algorithm: given a bubble B it is possible to compute in $O(n)$ time the set of bubbles in the new generators from which B can be composed, while the bubble decomposition algorithm of [1] required as much as $O(n^3)$ time for this task.

To design our new family of generators, we find a way to exploit some connections with cycle bases. We observe that the techniques developed for cycle bases (both in undirected and in directed graphs) cannot be applied directly to bubble generators. Indeed, as reported in [1], the main difference with cycle bases is that in our problem, in order to have biological relevance the following two properties are needed:

($\mathcal{P}_1$) A bubble generator for a directed graph G *must contain only bubbles*;

($\mathcal{P}_2$) For each bubble of G there exists a decomposition into bubbles of the generator, so that *only bubbles are generated at each step of this decomposition.*

We remark that ensuring properties ($\mathcal{P}_1$) and ($\mathcal{P}_2$) for cycles (in place of bubbles) is already non-trivial. Indeed, Gleiss *et al.* [10] have shown that it is possible to find a basis composed of directed cycles if the graph is strongly connected. However, this is not known in the case of general directed graphs. On the other side, Property ($\mathcal{P}_2$) is somewhat reminiscent of the notion of *cyclically robust cycle bases* which allows one to generate all cycles of a given graph by iteratively adding cycles of the basis [13, 17]. Unfortunately, not all graphs have a cyclically robust cycle basis [11] and understanding for which graph classes such a basis can be found is still an important open problem (see *e.g.,* [17]). Despite all these difficulties, we prove that a bubble generator based on spanning trees of the input graph satisfies properties ($\mathcal{P}_1$) and ($\mathcal{P}_2$). Since our bubble generators are identified from a chosen spanning tree, we also investigate the influence of the choice of spanning tree on the resulting generator.

The remainder of this paper is organised as follows. Section 2 presents some definitions that will be used throughout the paper. Section 4 introduces our family of bubble generators for flow graphs and for arbitrary graphs and we prove that it satisfies properties ($\mathcal{P}_1$) and ($\mathcal{P}_2$). Section 5 presents our experimental results: we first provide an empirical analysis of the characteristics of our new bubble generators based on the choice of the spanning tree (Subsection 5.1) and then we show an application of our new bubble generators in processing and analysing RNA data (Subsection 5.2). Finally, we conclude with some open problems in Section 6.

2 Preliminaries

Throughout the paper, we assume that the reader is familiar with the standard graph terminology, as contained for instance in [8]. A graph is a pair $G = (V, E)$, where V is the set of vertices, and $E \subseteq V \times V$ is the set of edges. For convenience, we may also denote the set of vertices V of G by $V(G)$ and its set of edges E by $E(G)$. We further set $n = |V(G)|$ and $m = |E(G)|$. A graph may be *directed* or *undirected*, depending on whether its edges are directed or undirected. In this paper, we deal with graphs that are directed, unweighted and finite. An edge $e = (u, v)$ is said to be *incident* to the vertices u and v, and u and v are said to be the endpoints of $e = (u, v)$. For a directed graph, edge $e = (u, v)$ is said to be leaving vertex u and entering vertex v. Alternatively, $e = (u, v)$ is an outgoing edge for u and an incoming edge for v. The *in-degree* of a vertex v is given by the number of edges entering v, while the *out-degree* of v is the number of edges leaving v. The *degree* of v is the sum of its in-degree and out-degree.

We say that a graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Given a subset of vertices $V' \subseteq V$, the subgraph of G *induced* by V', denoted by $G_{V'}$, has V' as vertex set and contains all edges of G that have both endpoints in V'. Given a subset of edges $E' \subseteq E$, the subgraph of G *induced* by E', denoted by $G_{E'}$, has E' as edge set and contains all vertices of G that are endpoints of edges in E'. Given a subset of edges $E' \subseteq E$, we denote by by $G \setminus E'$ the graph induced by $E \setminus E'$. Given two subgraphs G and H, their union $G \cup H$ is the graph F for which $V(F) = V(G) \cup V(H)$ and $E(F) = E(G) \cup E(H)$. Their intersection $G \cap H$ is the graph F for which $V(F) = V(G) \cap V(H)$ and $E(F) = E(G) \cap E(H)$.

Let s, t be any two vertices in G. A (*directed*) *path* from s to t in G, denoted as $s \rightsquigarrow t$, is a sequence of vertices and edges $s = v_1$, e_1, v_2, e_2, …, v_{k-1}, e_{k-1}, $v_k = t$, such that $e_i = (v_i, v_{i+1})$ for $i = 1, 2, \ldots, k-1$. Since there is no danger of ambiguity, in the remainder of the paper we will also denote a path simply as $s = v_1$, v_2, …, v_{k-1}, $v_k = t$ (*i.e.,* as a sequence of vertices). A path is *simple* if it does not contain repeated vertices, except possibly for the first and the last vertex. Throughout this paper, all the paths considered will be simple and referred to as paths. A path from s to t is also referred to as an (s, t)-path. The *length* of a path is given by the number of edges that belong to it.

Definition 1 *Given a directed graph G and two (not necessarily distinct) vertices $s, t \in V(G)$, an* (s, t)-bubble *consists of two directed (s, t)-paths referred to as* legs*, that are internally vertex disjoint. Vertex s is the source and t is the target of the bubble. If $s = t$ then exactly one of the paths of the bubble has length 0, and therefore B corresponds to a directed cycle. In this case, we say that B is a* degenerate bubble*.*

We say that a vertex u is *reachable* from the vertex v if there exists a path from v to u in the graph. An undirected graph is *connected* if every vertex is reachable from any other vertex in the graph. A directed graph G is *(weakly) connected* if the underlying undirected graph is connected.

Any maximal connected subgraph of G is called a *connected component*. We denote by c the number of connected components of G. A directed graph G is a *tree* if it is connected and has $n - 1$ edges. A subgraph T of G is called a *spanning tree* if it constitutes a tree on all vertices in G. A directed graph G is connected if and only if there is a spanning tree of G.

A directed graph G is a *flow graph* if there is at least one vertex s (referred to as a *start vertex*) which can reach all other vertices. Given a directed and connected graph G, any spanning tree T containing directed paths from a vertex r to each leaf is a called a *directed spanning tree rooted at r*. Notice that a directed graph G is a flow graph with start s if and only if there is a directed spanning tree rooted at s.

3 Bubble generators in the space of even subgraphs

We recall here the operation used to combine two bubbles [1]. Two subgraphs G_1, G_2 of G can be combined by the operator $\triangle$ that simply consists in the graph induced by the symmetric difference of the set of edges. More formally, $G_1 \triangle G_2 = (G_1 \cup G_2) \setminus (E(G_1) \cap E(G_2))$ where $E(G_i)$ is the set of edges of G_i. If $G_3 = G_1 \triangle G_2$, we say that G_3 is the sum of G_1 and G_2. Notice that by definition G_3 is a graph induced by a set of edges and thus it has no isolated vertices.

Given this operator, there are examples (see Figure 1) of two biological events (one nested inside the other) that generate three bubbles in their graph representation. However, those three bubbles are not independent since any one of them can be generated as the sum of the other two. This motivates defining a subset of bubbles that can generate all the bubbles in the graph.

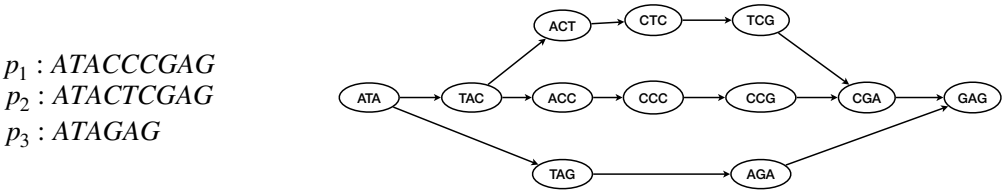

Figure 1: A toy example illustrating a SNP inside an AS event. We show three different sequences and the corresponding de Bruijn graph constructed for k-mer size 3. The bubble between the paths p_1 and p_2 is generated by a SNP while the bubble between p_1 and p_3 corresponds to an AS event generated by the skipping of the subsequence CCC.

Although the previous example shows a sum of two bubbles that yields a bubble, this is not always the case. To tackle this issue, we identify a bigger space that properly contains bubbles by introducing the notion of *even subgraphs*. An *even subgraph* is a directed subgraph where every vertex has total even degree. Note that bubbles (and any sum of two bubbles) are even subgraphs, and it can be shown that the sum of two even subgraphs (under operator $\triangle$) is an even subgraph. Moreover, the space of all even subgraphs of G, under the defined operation, is a vector space that we call the *even space* of G (also called the *cycle space* in undirected graphs [10, 14, 15, 19]). The dimension of the even space, and therefore the size of any basis, is known to be $m - n + c$ (where m, n and c are respectively the number of edges, vertices and connected components of G).

Bubbles are particular cases of even subgraphs, but in general it is not true that a basis of the even space can be composed only by bubbles. The set of even subgraphs that can be obtained by the sum of the bubbles is a vector subspace $\mathbb{B}$ of the even space. Therefore, the dimension of $\mathbb{B}$ is bounded by $m - n + c$. If we are not taking into account Property $\mathcal{P}_2$, then the problem of finding a small set $\mathcal{B}$ of bubbles that can generate the set of all bubbles is reduced to the problem of finding a basis of $\mathbb{B}$. Hence, the size of $\mathcal{B}$ is bounded by $m - n + c$. However, if we require $\mathcal{B}$ to satisfy also Property $\mathcal{P}_2$, then the problem becomes more complicated and the size of $\mathcal{B}$ can be greater than $m - n + c$ (see Figure 2).

Let $\mathcal{B}$ be a set of bubbles in G. We say that a bubble B has *a tree decomposition* in $\mathcal{B}$, if B can be decomposed in a binary-tree-like-fashion where the leaves correspond to bubbles in $\mathcal{B}$ and the internal nodes are bubbles. We say that $\mathcal{B}$ is a *bubble generator* if each bubble in G has a tree decomposition in $\mathcal{B}$. Hence, by definition, a bubble generator satisfies properties $\mathcal{P}_1$ and $\mathcal{P}_2$.

A bubble generator $\mathcal{B}$ is *minimal* if no proper subset of $\mathcal{B}$ is a bubble generator. A bubble generator is *minimum* if it has the minimum cardinality.

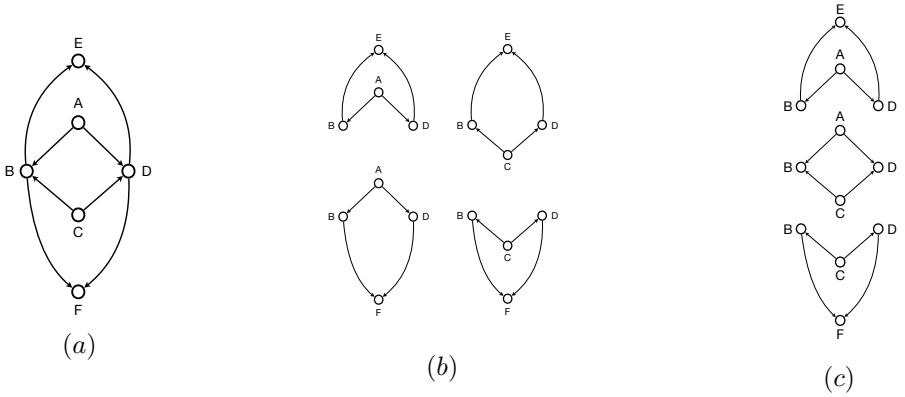

Figure 2: (*a*) A graph G, with a minimal bubble generator having size larger than the size of a basis of its even space. (*b*) A minimal bubble generator of G. (*c*) A basis for the even space.

4 Defining a bubble generator from a spanning tree

In this section, we define a bubble generator (that is, a generator of all bubbles that satisfies properties $\mathcal{P}_1$ and $\mathcal{P}_2$). We consider first flow graphs and then we extend our results to general graphs. Given a flow graph G with start vertex s, we find a directed spanning tree T of G, by performing any graph visit starting from s. In the experimental results in Section 5 we consider different types of visits, such as Depth-First Search, Breadth-First Search and Scan-First Search [7].

Let T be a directed spanning tree rooted at s and let $e = (u, v)$ be a non-tree edge. The insertion of e to the tree T produces a unique bubble B_e. The source of this bubble is the least common ancestor w of u and v, and its target is v. The two legs of this bubble are the tree path from w to v and the tree path from w to u followed by the edge (u, v). We denote by $B_T(G)$ the set of bubbles obtained in this way for all non-tree edges of the flow graph G. Notice that the set of bubbles with only one non-tree edge is exactly the set $B_T(G)$ as every non-tree edge generates exactly one bubble in $B_T(G)$. Since T has $n-1$ edges, the set $B_T(G)$ contains $m-n+1$ bubbles.

Before proving that $B_T(G)$ is a bubble generator (satisfying Property $\mathcal{P}_2$), we first show that $B_T(G)$ can generate any bubble. The definition of $B_T(G)$ is inspired on how Kirchhoff bases are built in the cycle space of undirected graphs [16]. Indeed, our proof shows that $B_T(G)$ is a basis; this implies that it can generate any even subgraph, and thus any bubble.

Theorem 1 *Let G be a flow graph with start vertex s. Let T be a directed spanning tree rooted at s and let $B_T(G)$ be the set of $m-n+1$ bubbles identified by the non-tree edges of G. Then any bubble B of G can be expressed as the sum of bubbles in $B_T(G)$.*

Proof: Let B be any even subgraph of G. We show that $B_T(G)$ can generate B. Since bubbles are even subgraphs, then the theorem follows.

We proceed by induction on the number of non-tree edges in B. If B has exactly one non-tree edge, then B belongs to $B_T(G)$ and the theorem follows trivially. Assume now that the theorem holds for even subgraphs with $k-1$ non-tree edges. Let B be an even subgraph with k non-tree edges and let e_1 be one of its non-tree edges. If B_1 is the bubble in $B_T(G)$ identified by the non-tree edge e_1, then the even subgraph $B \triangle B_1$ has $k-1$ non-tree edges. By the induction hypothesis, there are $k-1$ bubbles $B_2, \ldots, B_k$ such that $B \triangle B_1 = B_2 \triangle \ldots \triangle B_k$ and therefore $B = B_1 \triangle B_2 \triangle \ldots \triangle B_k$. □

Notice that $|B_T(G)| = m - n + 1$ and that any bubble in $B_T(G)$ has an exclusive edge (*i.e.* not contained in any other bubble of the set) and therefore no bubble in $B_T(G)$ can be generated from the remaining bubbles in $B_T(G)$. Thus, $B_T(G)$ is an independent set of bubbles and any bubble generator of a flow graph must have at least $m - n + 1$ bubbles.

Theorem 1 gives also a direct way to obtain the decomposition of a bubble B into bubbles of $B_T(G)$ for flow graphs. We show in Theorem 3 that these bubbles allow for a decomposition in a tree-like fashion and thus Properties $\mathcal{P}_1$ and $\mathcal{P}_2$ are satisfied.

Since each non-tree edge (u, v) is contained exactly in one bubble of $B_T(G)$, one needs to consider all and only the bubbles of $B_T(G)$ identified by the non-tree edges of B (with respect to T). Moreover, the set $B_T(G)$ can be found efficiently by simply performing a visit from the start vertex s and by returning the non-tree edges.

It is worth mentioning that Theorem 1 can be extended to general graphs as follows. Let G be an arbitrary directed graph G. Let S be a minimum set of vertices from which every vertex of G can be reached. We denoted by S a *source set* of G. Note that in the worst case, $|S| = O(n)$. For each $s \in S$, let $B_T(G, s)$ be the set of bubbles identified by a visit starting from the vertex s of G. Consider the set $B(G, S) = \cup_{s \in S} B_T(G, s)$. Observe that the source of any bubble B in G can be reached by at least one vertex s in S. Thus B belongs to a subgraph of G, which is a flow graph rooted at s, and hence can be expressed as a composition of bubbles in $B_T(G, s)$. This can be summarised by the following theorem.

Theorem 2 *Let G be a directed graph and let S be its source set. Then there is a set of bubbles $\mathcal{B}$, such that each bubble in G can be generated starting from the bubbles in $\mathcal{B}$ (with a symmetric difference operator), and $|\mathcal{B}| \leq |S|(m - n + 1)$.*

Notice that for general graphs, our generator can reach the size of the generator proposed in [1]. However, it will be shown in Section 5 that in practice the size of our generator is much smaller. Finally, we show that our generator ensures a tree-like decomposition and thus satisfies Property $\mathcal{P}_2$. In other words, we show that each bubble B in G has a tree decomposition using a subset of bubbles in $\mathcal{B}_T$ and such that in each step we combine only bubbles. This result is quite technical and is the main theoretical contribution of our work. To prove this we first need two propositions.

Given a bubble B and two *distinct* vertices u, v in B (not necessarily distinct from s, t), a (u, v)-*chord* of B is a directed path from u to v that is internally vertex disjoint with B (*i.e.* except for u and v, the path $u \rightsquigarrow v$ has no other vertex in common with B).

Proposition 1 *Given a non-degenerate (s, t)-bubble B and a (u, v)-chord of B such that at least one of the followings hold: (i) $\{u, v\} \cap \{s, t\} \neq \emptyset$, (ii) there is no directed path $v \rightsquigarrow u$ in B. Then the (u, v)-chord defines two bubbles B_1 and B_2 such that $B = B_1 \triangle B_2$.*

Proof: We consider two cases depending whether u and v are in different legs of B or not. If u and v are in different legs of B, then we define B_1 to be the bubble with source u and target t and B_2 to be the bubble with source s and target v. Notice that if at least one of u and v coincides with s or t, they can still be considered to be in different legs as s and t belong to both legs of B. It is easy to see that $B = B_1 \triangle B_2$. These cases are depicted in Fig. $3(a) - (d)$. If u and v are in the same leg of B then we define B_1 to be the bubble with source u and target v and B_2 to be the bubble with source s and target t (see Fig. $3(e_1)$). However, if there exists a path from $v \rightsquigarrow u$ in B (see Fig. $3(e_2)$) then it is not possible to define the two bubbles B_1 and B_2. Notice that this is the only case where the (u, v)-chord does not allow to define the two bubbles for which $B = B_1 \triangle B_2$.□

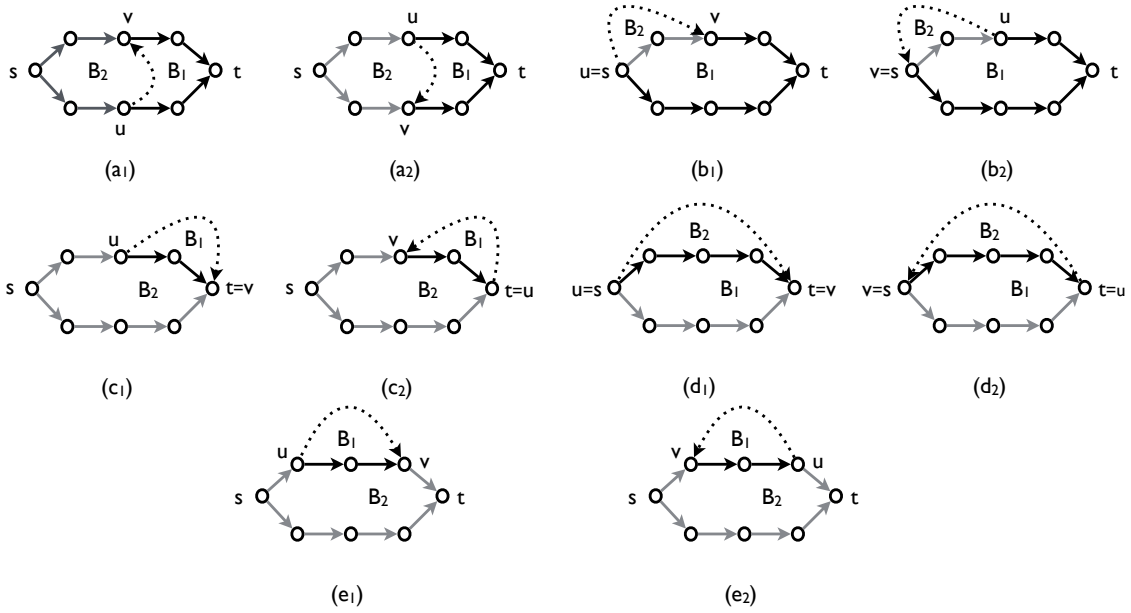

Figure 3: All the possible cases considered in Proposition 1. In dotted line we have the edges of the (u, v)-chord, the bubble B is composed by the black and grey edges, the bubble B_1 is composed by the black and the dotted line edges and the bubble B_2 by the grey and the dotted line edges.

Proposition 2 *Given a degenerate bubble B then any (u, v)-chord of B defines two bubbles B_1 and B_2 such that $B = B_1 \triangle B_2$.*

Proof: The proof follows straightforwardly by observing that every vertex in a directed cycle C has in-degree and out-degree equal to one. After adding the edges of the (u, v)-chord, u has out-degree equal to 2 and v has in-degree 2. Thus the directed cycle C can be written as the sum of B_1 that is the non-degenerate bubble with source u and target v and B_2 that is the degenerate bubble with source and target u (or v). □

Propositions 1 and 2 are used to prove the following theorem.

Theorem 3 *Let G be a flow graph with start vertex r, and let $B_T(G)$ be the set of bubbles identified by a spanning tree T rooted at r. Then any bubble B in G can be decomposed in $O(n^2)$ time in bubbles in $B_T(G)$ in a tree-like fashion.*

Proof: Let G be a flow graph with start vertex r, and let $B_T(G)$ be the set of bubbles identified by a spanning tree T starting from r. Let B be a bubble in G. We prove the theorem by induction on the number of non-tree edges k in B. We show that B can be decomposed in at most $k - 1$ steps in bubbles from $B_T(G)$ in a tree-like fashion.

The base case $k = 1$ is trivial as the bubble is already in the generator and thus no decomposition is needed. For the induction step suppose that the claim is true for bubbles with up to $k - 1$ non-tree edges. Let B be a bubble with source s and target t with k non-tree edges. If B is a cycle, then $s = t$ can be chosen arbitrarily. Let E be the set of non-tree edges of B ($|E| = k$).

To prove the theorem, it suffices to show that B can be decomposed into two bubbles B_1, B_2, with non-tree edges E_1 and E_2 respectively, such that $E_1 \neq \emptyset, E_2 \neq \emptyset$ and E_1, E_2 form a partition of E. Moreover, by the induction hypothesis B_1 and B_2 can be decomposed in at most $|E_1|$ and $|E_2|$ steps with $|E_1| + |E_2| = k$. We will use the following observation:

Proposition 3 *Let B be a bubble with k non-tree edges, and let B_1 and B_2 be two bubbles such that $B = B_1 \triangle B_2$. If $E(B_1) \cap E(B_2) \subseteq E(T)$ then both B_1 and B_2 will have strictly less than k non-tree edges.*

Indeed, Proposition 3 follows from three simple observations: (i) if $E(B_1) \cup E(B_2) \subseteq E(B) \cup E(T)$ then B_1 and B_2 can only contain non-tree edges already in B; (ii) no non-tree edge can be in both B_1 and B_2, and (iii) any bubble must have at least one non-tree edge (as it cannot be entirely in the tree T). The main idea to prove the induction step is to identify a chord consisting only of tree edges and use it to decompose B into two bubbles according to Proposition 1. As the chord contains only tree edges, the result will then follow by Proposition 3. We consider two cases:

Case I: B is a non-degenerate bubble Note that there must necessarily be a path $p = r \rightsquigarrow t$ in T (as a special case, p can be empty whenever $r = t$). To identify a chord consisting only of tree edges, we follow this path "backwards" starting from t. The following cases are possible:

(a) The path $p = r \rightsquigarrow t$ (traced backwards) leaves and re-enters the bubble B. More formally, there exist two vertices v and u in $V(B)$ (with v not necessarily distinct from t and u not necessarily distinct from s) such that the path $v \rightsquigarrow t$ is in B and the path $u \rightsquigarrow v$ is internally vertex-disjoint from both legs of B. This case is depicted in Fig. 4(a_1)-(a_3). As the path p is in T, it cannot touch the same vertex twice, and so u and v must be distinct. We then have a (u, v)-chord that satisfies Proposition 1 and thus we can define B_1, B_2 such that $B = B_1 \triangle B_2$. By Proposition 3, both B_1 and B_2 will have strictly less than k non-tree edges.

(b) The path $p = r \rightsquigarrow t$ (traced backwards) never leaves the bubble B. In other words, p is entirely included in one leg of B. This implies that r belongs to this leg. In this case we need to identify a new path in T as a chord of B. We distinguish the following two cases:

(b_1) Assume first $s \neq r$. Then, there must be a path $q : r \rightsquigarrow s$ in T. This case is depicted in Fig. 4(b_1). Starting from s, we trace this path backwards. Notice that as s has no incoming edges in B, the first edge encountered (w, s) cannot be in B. Let u be the first vertex in B that we encountered tracing backwards this path q. Notice that must u exist, since we eventually end up in r, and r is in B. Then, the (u, s)-chord satisfies Proposition 1 and we can define B_1, B_2 such that $B = B_1 \triangle B_2$. From Proposition 3 both B_1 and B_2 must have less than k non-tree edges.

(b_2) Assume now $s = r$. In this case, the path p coincides with one of the legs of B, which then is entirely contained in the tree T. This case is depicted in Fig. 4(b_2). Note that B must have at least two non-tree edges, otherwise it would be in the bubble generator. As one leg of B is entirely in the tree T, both those non-tree edges must be in the other leg of B. As a consequence, there must be a non-tree edge (w_1, w_2) in B, which is not incident to t. Consider now the path $q = r \rightsquigarrow w_2$ in T: note that the tree path q cannot be entirely in B. Follow the path q starting from $r = s$, and let u be the first vertex where q departs from B: more formally, let (u, u_1) be the first edge of q such that $(u, u_1) \notin E(B)$ (hence $u \in V(B)$). Vertex u can belong to any of the two legs of

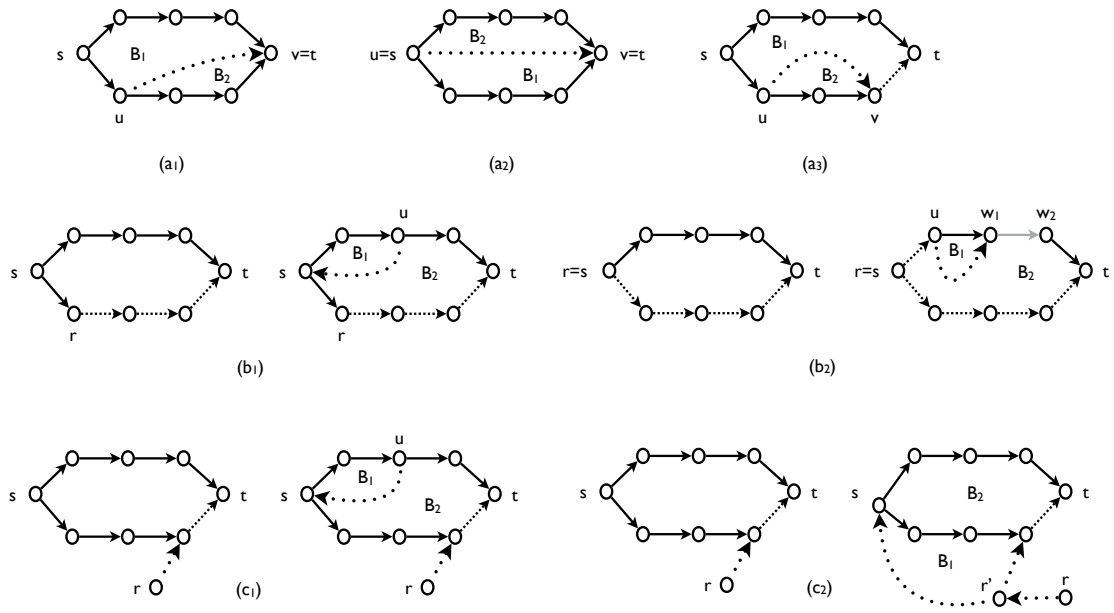

Figure 4: All the possible cases considered in Case I of Theorem 3. In dashed line we indicate edges of the bubble B that belong to the tree T and in a dotted line a path that belongs to the tree but not to B. In grey the edges of B that do not belong to the tree T.

B. Note that the path q must enter B again at some point, since q ends in w_2 and $w_2 \in V(B)$. Following q forward from u, let v be the first vertex in q that belongs to $V(B)$. Notice that v must necessarily belong to the the leg of B that contains w_2 as the other leg belongs entirely to T and none of its vertices can have in-degree greater than one in T. The (u, v)-chord satisfies the conditions of Proposition 1 and thus we can define B_1, B_2 such that $B = B_1 \triangle B_2$. From Proposition 3 both B_1 and B_2 must have less than k non-tree edges.

(c) The path $p = r \rightsquigarrow t$ (traced backwards) leaves the bubble B and never re-touches it again. Let $v \in V(B)$ be the first vertex such that the edge $(v', v) \notin E(B)$. Notice that v must exist as it can be t. In this case, clearly $r \notin V(B)$. Then, there must be a path $q = r \rightsquigarrow s$ which is not entirely contained in B. Starting from s, we trace the path q backwards. The following two cases can happen:

(c_1) q touches the bubble B. As s has no incoming edges in B the first edge of q is not in B. Hence, there exists a vertex in q, different from s that belongs to B. This case is depicted in Fig. 4(c_1). Let u be the first vertex we encounter that touches B. The (u, s)-chord satisfies the conditions of Proposition 1 and thus we can define B_1, B_2 such that $B = B_1 \triangle B_2$. From Proposition 3 both B_1 and B_2 must have less than k non-tree edges.

(c_2) q does not touch the bubble B. This case is depicted in Fig. 4(c_2). Notice that the paths $r \rightsquigarrow s$ and $r \rightsquigarrow v$ may share some edges, and so we consider r' as the least common

ancestor of s and v in T. In this case, B can be written as the sum of two bubbles: B_1 with source r' and target v (v is not necessarily distinct from t), and B_2 with source r' and target t. From Proposition 3 both B_1 and B_2 must have less than k non-tree edges.

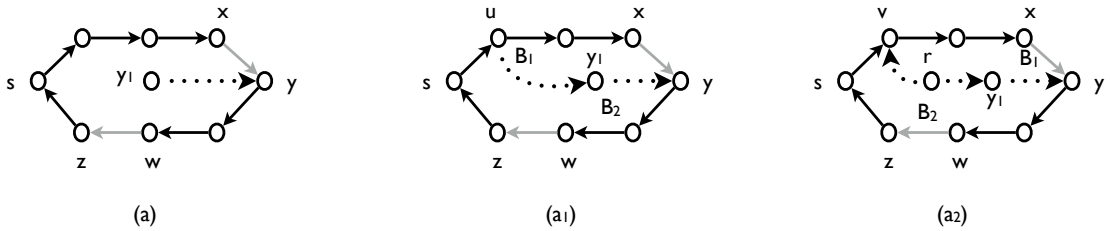

Figure 5: All the possible cases considered in Case II of Theorem 3. In a dotted line we indicate a path that belongs to the tree but not to B. In grey the edges of B that do not belong to the tree T.

Case II: B is a cycle As B is not in the generator, then there must be at least two non-tree edges in B, say (x, y) and (w, z). This case is depicted in Fig. 5(a) (notice that y is not necessarily distinct from w). At least one among y and z must be different from the tree root r: without loss of generality, assume $y \neq r$. Consider the path $p = r \rightsquigarrow y$ in T and follow this path backwards starting from y. Since (x, y) is a non-tree edge, the first edge (y_1, y) traced back in the path p is not in B. The following two cases can happen:

(a_1) The path $r \rightsquigarrow y_1$ touches the bubble B. Starting from y_1 and following the edges backwards, let u be the first vertex that $u \in V(B)$. This case is depicted in Fig. 5(a_1) Then we have found a (u, y) -chord. From Proposition 2, we can define B_1, B_2 such that $B = B_1 \triangle B_2$. From Proposition 3 both B_1 and B_2 must have less than k non-tree edges.

(a_2) The path $r \rightsquigarrow y_1$ does not touch the bubble B. This implies that the only vertex in common between the path $p = r \rightsquigarrow y$ and B is y. Then clearly $r \notin V(B)$. As a consequence, there must exist a path $r \rightsquigarrow z$ that is not entirely contained in B (as r is not in B). This case is depicted in Fig. 5(a_2). Starting from r, let v be the first vertex in the path $r \rightsquigarrow z$ that belongs to $V(B)$. Clearly v exists as it can be z. Notice that B can be written as the sum of two bubbles: B_1 with source r and target y and B_2 with source r and target v. By Proposition 3, both B_1 and B_2 must have less than k non-tree edges.

Finally, notice that each step of the decomposition takes $O(n)$ time (as we are traversing a bubble and a path in a tree) and each bubble can have at most $O(n)$ non-tree edges. Therefore the decomposition can be done in $O(n^2)$ time. □

Using the same arguments used for Theorem 2, we can extend Theorem 3 to general graphs as follows.

Corollary 1 *Let G be a directed graph, then there is a set of bubbles $\mathcal{B}$, such that each bubble in G can be decomposed in $O(n^2)$ time in bubbles in $\mathcal{B}$ in a tree-like fashion.*

5 Experimental results

To test the usefulness of our family of generators in practice, we applied it to the identification of AS events in RNA data in a reference-free context. In order to compare our generators to both the state-of-art algorithm KisSplice [18, 22] and to the generator defined in [1], we used two datasets: *Dataset 1* is the same dataset as in [1] which consists of a sample (corresponding to chromosome 10) of 4,932,572 RNA-seq Illumina paired-end reads extracted from the mouse brain tissue (available in the ENA repository under the following study: PRJEB25574). *Dataset 2* corresponds to a sample of 10 million reads from the dataset presented in [3] of human lung cells infected by Influenza A viruses (IAVs).

We built the de Bruijn graph from these reads and applied standard sequencing-error-removal procedures by using KisSplice [18, 22]. We recall that KisSplice is a method to find AS events in a reference-free context by enumerating bubbles in a de Bruijn Graph.

For our family, we considered generators coming from three different types of underlying spanning trees, namely Depth-First Search (DFS), Breadth-First Search (BFS) and Scan-First Search (SFS). We recall here that Scan-First Search is the graph search procedure introduced in [7] and which works as follows. As with DFS and BFS, we start from a specified source vertex s and we mark it. At each step, we perform what we call a *scan*. This selects a marked vertex v and marks all previously unmarked neighbours of v. In other terms, SFS proceeds by scanning a marked and unscanned vertex until all vertices are scanned. Notice that both BFS and DFS can be seen as special cases of SFS. Similarly to BFS and DFS, also SFS can produce a tree as follows. Initially, the tree is empty. Whenever a vertex v is scanned, all the edges between v and its previously unmarked neighbours are added to the tree. In our experiments, we implemented SFS with a random choice of the next vertex to be scanned, and averaged on 1,000 runs with different random seeds.

To compute the source set of the de Bruijn graph, we computed in linear time the DAG of its strongly connected components and chose a vertex from each source. The de Bruijn graph corresponding to our dataset had a total of 83,400 vertices, 99,038 edges and 18,385 source vertices.

Finally, we recall that for general graphs, our new generators are not necessarily minimal. In order to avoid producing duplicates of the same bubble, we discarded a bubble whenever its source was already contained in a tree previously computed from another start vertex. Notice that this does not guarantee the minimality of the generator as there can still be bubbles that can be composed from bubbles that were already present in the generator. For this reason, in general graphs we expect that the size of the generator may vary substantially, depending on the underlying tree chosen.

All our experiments were carried out on a 64-bit machine running Ubuntu 16.04 LTS, equipped with a 2.30 GHz processor Intel(R) Xeon(R) Gold 511, 192 GB of RAM, 16MB of L3 cache and 1 MB of L2 cache.

5.1 An empirical analysis of the characteristics of the bubble generator based on the choice of the spanning tree

We first explore experimentally some characteristics of bubble generators in our family, depending on the choice of the underlying spanning tree. The parameters we consider are: (i) the size of the generator, (ii) the number of degenerate bubbles (cycles), (iii) the average length of the longest leg, (iv) the average length of the shortest leg, (v) the number of branching bubbles (a branching bubble is a bubble containing more than 5 vertices of in-degree or out-degree greater than 1 [18, 22]).

Table 1 shows the main characteristics of generators in our family. We also include the time

required to compute each generator. We do not include in this running time the pre-processing time spent in creating the de Bruijn graph, which is exactly the same for all generators. We refer to a generator in our family simply by the graph search used to generate it and we denote by SP-Gen the generator defined in [1].

Generator		Size	$\#ND_{Bubbles}$	$\#D_{Bubbles}$	AvgLong	AvgShort	time(s)
DFS		12175	11792	383	90.53	40.5	3
BFS		42324	41959	365	33.57	21.23	3
SFS	Mean	41388	41187	201	56.58	41.47	3
	STD	1102.8	1096	6.8	0.3	0.32	0.09
SP-Gen [1]		91486	80108	11378	70.12	31.31	380

Table 1: Characteristics of the generators in our family. The columns represent: the size of the generator, $\#ND_{Bubbles}$ the number of non degenerate bubbles found, $\#D_{Bubbles}$ the number of degenerate bubbles (*i.e.* cycles), AvgLong and AvgShort the average length of the longest and shortest leg, respectively, and the time the algorithm spent in seconds. Notice that for Scan-First search trees (SFS) we report the mean and the standard deviation of 1000 different runs.

As illustrated in Table 1, the size of all our new generators, independently of the underlying spanning tree, is much smaller than the size of SP-Gen [1]. Furthermore, all our new generators can be computed two orders of magnitude faster than SP-Gen. Furthermore, compared to BFS and SFS, the DFS generator usually has smaller size and its bubbles have longer legs. We also observe that, compared to SP-Gen, the percentage of cycles significantly drops in our new generators: from 12.4% for SP-Gen to 3.1% for DFS, 0.8% for BFS and 0.5% for SFS. This is desirable as cycles are degenerate bubbles that do not correspond to AS events, and thus generators that avoid cycles are preferable.

5.2 Application of the bubble generator to the identification of AS events in RNA-seq data

As already mentioned in the introduction, identifying AS events in the absence of a reference genome remains a challenging problem. Local assemblers such as KisSplice [18] are faced with a dramatically large (and often practically unfeasible) running time due to the exponentially large number of bubbles present, most of which are false positives, *i.e.* they are artificial bubbles not associated with biological events. Indeed, a significantly large number of such artificial bubbles comes from complex subgraphs created by the presence of approximate repeats in the transcriptomic sequence. Thus, tools such as KisSplice use heuristics in order to avoid dealing with large portions of a de Bruijn graph containing such complex subgraphs. Here we show how the set of bubbles belonging to generators in our family can be used to predict AS events. Notice that our method is reference-free; however, in order to evaluate it, we make use of annotated reference genomes to assess if our predictions are correct.

To estimate the precision of our new generators in predicting AS events we proceed as follows. We consider the whole set of bubbles belonging to the generator. We then apply the same filter (based on the length of the legs) as in KisSplice to extract the bubbles that can be considered as putative AS events. To determine the true AS events, we used STAR [9] to map the putative bubbles of Dataset 1 to the *Mus musculus* reference genome and annotations (Ensembl release 94)

and those of Dataset 2 to the human genome (hg38, Gencode v36). We then analysed the results using KisSplice2RefGenome [4]. Following [18], a bubble corresponds to a true AS event (or a true positive (TP)) if one leg matches the inclusion isoform and the other the exclusion isoform. Otherwise, the bubble is classified as a false positive. The precision of the method is defined as $TP/(TP + FP)$.

Below we detail the analysis of the results for the two datasets considered. We compared our results to both SP-Gen and KisSplice. We report that Dataset 2 was already too large to be handled by SP-Gen, which did not finish even after 1 day, whereas KisSplice and the tree generators from our family produced the results in approximately 10 minutes. As a result, we will not report data from SP-Gen on Dataset 2.

5.3 Analysis of Dataset 1: comparison with the SP-Gen and KisSplice

The results for DFS/BFS/SFS and SP-Gen are reported in Table 2. The results show that the number of true AS events found by our generators is comparable to the number of true AS events found by SP-Gen whereas the number of false positives is significantly smaller. Indeed, our generators have a precision between 87.7% and 91.6%, compared to 77.3% for the SP-Gen. An interesting aspect of SP-Gen was that it contained many bubbles that were classified as Intron Retention (IR), which is a type of AS event that is generally particularly hard to identify. As shown in Table 2, the number of IR for our generators remains similar to the one found by SP-Gen.

Algorithm	#putatitve AS events	#true AS events	precision	#IR
BFS	1046	959	(91.6%)	319
DFS	1178	1034	(87.7%)	392
SFS	1163	1053	(90.5%)	391
SP-Gen [1]	1403	1085	(77.3%)	377

Table 2: Precision of the generators in our family. The columns represent: number of putative AS events, number of true AS events, precision and number of intron retation events.

Since the computation of generators in our family is truly fast in practice, we combined them by taking the union of bubbles coming from different generators and tested whether this would increase the number of AS events found. Notice that the same bubble could be found in two different generators in our family, and thus we eliminated duplicate bubbles in this process. In Table 3 we report the results of different unions of generators in our family (DFS, BFS and 10 randomly chosen runs of SFS), together with the results of SP-Gen and KisSplice. As can be seen, the union of different generators in our family allows us to find more true AS events than both SP-Gen and KisSplice.

Finally, in [1] it was shown that SP-Gen was able to identify some AS events that will certainly be lost by KisSplice. Indeed, the heuristic used by KisSplice does not generate bubbles containing a number of branching vertices (*i.e.,* vertices with in-degree or out-degree at least 2) higher than some threshold. In KisSplice, the default value for this branching threshold is 5. Increasing the value of this threshold will increase exponentially the running time of the algorithm and thus a large branching threshold is unfeasible in practice. As reported in [1], around 27 true AS events in SP-Gen have a branching number higher than 5, and are lost by KisSplice. For the family of our generators, we have that the number of true AS events that are certainly lost by KisSplice is:

Algorithm	#putatitve AS events	#true AS events	precision
BFS + DFS	1245	1099	88.3%
10-SFS	1622	1179	72.7%
BFS + DFS + 10-SFS	1677	1196	71%
SP-Gen [1]	1403	1085	77.3%
KisSplice	1293	1159	89.63%

Table 3: Combining different generators in our family. The columns represent: number of putative AS events, number of true AS events and precision.

(a) 16 for the BFS, (b) 77 for the DFS, and (c) an average of 80 for SFS (averaged over different choices of the random seed).

5.4 Analysis of Dataset 2: comparison with KisSplice

As already mentioned, this dataset was too big to analyse by SP-Gen. Indeed, SP-Gen did not finish even after 1 day, while the generators of our family and KisSplice took approximately 10 minutes to produce the results. As a result, we report here only the comparison of our new tree generators with KisSplice. As reported in Table 4, our new generators produce results that are comparable to KisSplice. As a matter of fact, our generators produce a number of true AS events that is similar to the one generated by KisSplice and the same holds for IR events. By combining the results of our generators, we get a slightly higher number than KisSplice for both true AS events and IR events.

Notice also that both the generator and KisSplice have a low precision on this dataset, meaning that most of the bubbles found in the sample do not correspond to AS events in human. This is not surprising as this dataset contains many reads coming from the virus [3].

Finally, the generators produce up to 155 true AS events that are certainly missed by KisSplice as they correspond to bubbles with more than 5 branching vertices.

Algorithm	#putatitive AS events	#true AS events	# IR	# high branching
BFS	133 780	2172	159	76
DFS	49 884	2118	165	76
BFS + DFS	160652	2231	166	101
3-SFS	218 104	2162	162	83
BFS + DFS + 3-SFS	498048	2302	170	155
KisSplice	11 512	2257	165	-

Table 4: Combining different generators in our family. The columns represent: number of putative AS events, number of true AS events, number of Intron Retention events and true AS events whose corresponding bubbles have more than 5 branching vertices.

In summary, our new generators allowed us to analyse this dataset, which was not possible with SP-Gen. Although the new generators show a similar performance as KisSplice in terms of both

the number of true AS events and computational time, they offer a nice complementary view as they are able to include true AS events that were missed by KisSplice.

6 Conclusions and open problems

In this paper, we have proposed a new family of bubble generators which improves substantially on the previous generator (SP-Gen [1]): generators in the new family are much faster, *i.e.,* about two orders of magnitude faster than SP-Gen, and they are still able to achieve similar (and sometimes higher) precision in identifying AS events.

Our work raises several new and perhaps intriguing questions. First, we notice that while for flow graphs our family produces minimum generators, for general graphs it is still open to find a minimum bubble generator. Second, the fast computation of our new generators opens the way to the design of algorithms that efficiently combine the bubbles of a generator in order to find more AS events. Third, we believe that the number of false positives could be reduced by adding more biologically motivated constraints. An example of constraint that can be introduced toward this aim is to give a weight to each edge of the de Bruijn graph based on the reads coverage. A true AS event would then correspond to bubbles in which the edges inside a leg must have similar weights (but different legs may have different coverage). Fourth, when constructing a de Bruijn graph from RNA-seq reads, some filters are applied that are meant to eliminate sequencing errors. These filters remove vertices and edges whose coverage by the set of reads is below some given thresholds. Changing those thresholds has a significant impact on the resulting de Bruijn graph, and hence on the set of solutions. Is it possible to compute in a dynamic fashion a bubble generator when this coverage threshold is changing, without having to recompute everything from scratch?

References

[1] V. Acuña, R. Grossi, G. F. Italiano, L. Lima, R. Rizzi, G. Sacomoto, M. Sagot, and B. Sinaimeri. On bubble generators in directed graphs. *Algorithmica*, 82(4):898–914, 2020. doi:10.1007/s00453-019-00619-z.

[2] V. Acuña, L. Lima, G. F. Italiano, L. Pepè Sciarria, M. Sagot, and B. Sinaimeri. A family of tree-based generators for bubbles in directed graphs. In *Combinatorial Algorithms - 31st International Workshop, IWOCA 2020, Bordeaux, France, June 8-10, 2020, Proceedings*, pages 17–29, 2020.

[3] U. Ashraf, C. Benoit-Pilven, V. Navratil, C. Ligneau, G. Fournier, S. Munier, O. Sismeiro, J.-Y. Coppée, V. Lacroix, and N. Naffakh. Influenza virus infection induces widespread alterations of host cell splicing. *NAR Genomics and Bioinformatics*, 2(4), 11 2020. doi:10.1093/nargab/lqaa095.

[4] C. Benoit-Pilven, C. Marchet, E. Chautard, L. Lima, M.-P. Lambert, G. Sacomoto, A. Rey, A. Cologne, S. Terrone, L. Dulaurier, J.-B. Claude, C. Bourgeois, D. Auboeuf, and V. Lacroix. Complementarity of assembly-first and mapping-first approaches for alternative splicing annotation and differential analysis from RNAseq data. *Scientific Reports*, 8(1), 2018. doi:10.1038/s41598-018-21770-7.

[5] E. Birmelé, P. Crescenzi, R. Ferreira, R. Grossi, V. Lacroix, A. Marino, N. Pisanti, G. Sacomoto, and M.-F. Sagot. Efficient Bubble Enumeration in Directed Graphs. In *SPIRE*, pages 118–129, 2012. doi:10.1007/978-3-642-34109-0_13.

[6] L. Brankovic, C. S. Iliopoulos, R. Kundu, M. Mohamed, S. P. Pissis, and F. Vayani. Linear-time superbubble identification algorithm for genome assembly. *Theoretical Computer Science*, 609:374–383, 2016. doi:https://doi.org/10.1016/j.tcs.2015.10.021.

[7] J. Cheriyan, M.-Y. Kao, and R. Thurimella. Scan-first search and sparse certificates: An improved parallel algorithm for k-vertex connectivity. *SIAM Journal on Computing*, 22(1):157–174, 1993. doi:10.1137/0222013.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[9] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras. STAR: ultrafast universal RNA-seq aligner. *Bioinformatics*, 29(1):15–21, 2013. doi:10.1093/bioinformatics/bts635.

[10] P. M. Gleiss, J. Leydold, and P. F. Stadler. Circuit bases of strongly connected digraphs. *Discussiones Mathematicae Graph Theory*, 23(2):241–260, 2003.

[11] R. H. Hammack and P. C. Kainen. Robust cycle bases do not exist for $K_{n,n}$ if $n \geq 8$. *Discrete Applied Mathematics*, 235:206 – 211, 2018. doi:10.1016/j.dam.2017.10.001.

[12] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean. De novo assembly and genotyping of variants using colored de bruijn graphs. *Nat Genet*, 44(2):226–232, 2012. doi:10.1038/ng.1028.

[13] P. C. Kainen. On robust cycle bases. *Electronic Notes in Discrete Mathematics*, 11:430 – 437, 2002. The Ninth Quadrennial International Conference on Graph Theory, Combinatorics, Algorithms and Applications.

[14] T. Kavitha, C. Liebchen, K. Mehlhorn, D. Michail, R. Rizzi, T. Ueckerdt, and K. A. Zweig. Cycle bases in graphs characterization, algorithms, complexity, and applications. *Computer Science Review*, 3(4):199 – 243, 2009. doi:https://doi.org/10.1016/j.cosrev.2009.08.001.

[15] T. Kavitha and K. Mehlhorn. Algorithms to compute minimum cycle bases in directed graphs. *Theory of Computing Systems*, 40(4):485 – 505, 2007. doi:10.1007/s00224-006-1319-6.

[16] G. Kirchhoff. Ueber die auflösung der gleichungen, auf welche man bei der untersuchung der linearen vertheilung galvanischer ströme geführt wird. *Annalen der Physik*, 148(12):497–508, 1847.

[17] K. Klemm and P. F. Stadler. A note on fundamental, non-fundamental, and robust cycle bases. *Discrete Applied Mathematics*, 157(10):2432 – 2438, 2009. Networks in Computational Biology. doi:10.1016/j.dam.2008.06.047.

[18] L. Lima, B. Sinaimeri, G. Sacomoto, H. Lopez-Maestre, C. Marchet, V. Miele, M.-F. Sagot, and V. Lacroix. Playing hide and seek with repeats in local and global de novo transcriptome assembly of short RNA-seq reads. *Algorithms Mol Biol*, 12, 2017. doi:10.1186/s13015-017-0091-2.

[19] S. MacLane. A combinatorial condition for planar graphs. *Fundamenta Mathematicae*, 28:22–32, 1937.

[20] J. R. Miller, S. Koren, and G. Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010. doi:https://doi.org/10.1016/j.ygeno.2010.03.001.

[21] T. Onodera, K. Sadakane, and T. Shibuya. Detecting Superbubbles in Assembly Graphs. In *Algorithms in Bioinformatics*, volume 8126 of *LNCS*, pages 338–348. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40453-5_26.

[22] G. Sacomoto, J. Kielbassa, R. Chikhi, R. Uricaru, P. Antoniou, M.-F. Sagot, P. Peterlongo, and V. Lacroix. Kissplice: de-novo calling alternative splicing events from rna-seq data. *BMC Bioinformatics*, 13(S-6):S5, 2012. doi:10.1186/1471-2105-13-S6-S5.

[23] G. Sacomoto, V. Lacroix, and M.-F. Sagot. A polynomial delay algorithm for the enumeration of bubbles with length constraints in directed graphs and its application to the detection of alternative splicing in RNA-seq data. In *WABI*, pages 99–111, 2013.

[24] M. Sammeth. Complete alternative splicing events are bubbles in splicing graphs. *Journal of Computational Biology*, 16(8):1117–1140, 2009. doi:10.1089/cmb.2009.0108.

[25] W.-K. Sung, K. Sadakane, T. Shibuya, A. Belorkar, and I. Pyrogova. An $O(m \log m)$-time algorithm for detecting superbubbles. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 12(4):770–777, 2015. doi:10.1109/TCBB.2014.2385696.

[26] R. Uricaru, G. Rizk, V. Lacroix, E. Quillery, O. Plantard, R. Chikhi, C. Lemaitre, and P. Peterlongo. Reference-free detection of isolated SNPs. *Nucleic Acids Research*, 43(2):e11, 2015. doi:10.1093/nar/gku1187.

[27] R. Younsi and D. MacLean. Using $2k + 2$ bubble searches to find single nucleotide polymorphisms in k-mer graphs. *Bioinformatics*, 31(5):642–646, 2015. doi:10.1093/bioinformatics/btu706.