A Family of Tree-Based Generators for Bubbles in Directed Graphs

Vicente Acuña, Leandro Ishi Soares de Lima, Giuseppe F Italiano, Luca Pepè Sciarria, Marie-France Sagot, Blerina Sinaimeri

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

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

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

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 generator was quite effective in finding AS events, preliminary experiments showed that it is about 5 times slower than state-of-art methods. In this paper we propose a new family of bubble generators which improve substantially on the previous generator: 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 generators, we also report some experimental results on a real dataset.

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.,* [18, 20]). 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 [16, 21, 20, 22] and SNPs (Single Nucleotide Polymorphism), and indels in DNA-data [10, 24, 25]. Since bubbles can be associated to such biologically relevant events, in recent years there have been several theoretical studies on bubbles (see *e.g.,* [3, 4, 19, 21, 23]), 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 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 generator was used to find AS events in a real dataset. As reported in [1], this generator was able to achieve about the same precision in identifying AS events as the state-of-art-algorithm KisSplice [16, 20], but unfortunately building the 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 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 generator of [1]. In particular, in the same RNA dataset used in [1], generators in our family are about two orders of magnitude faster in practice than the generator in [1], and improve the precision in identifying AS events from 77.3% to 90%. When compared to the state-of-the-art algorithm for identifying AS events, our generators are also much faster than KisSplice [16, 20], 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 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 (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*;
($\mathcal{P}_2$) Each bubble of G should be decomposed 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.* [8] 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 [11, 15]. Unfortunately, not all graphs have a cyclically robust cycle basis [9] and understanding for which graph classes such a basis can be found is still an important open problem (see *e.g.,* [15]). 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 3 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 4 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 4.1) and then we show an application of our new bubble generators in processing and analysing RNA data (Subsection 4.2). Finally, we conclude with some open problems in Section 5.

2 Preliminaries

Throughout the paper, we assume that the reader is familiar with the standard graph terminology, as contained for instance in [6]. 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, finite and without parallel edges. 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 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 $s = v_1, e_1, v_2, e_2, \ldots, v_{k-1}, e_{k-1}, v_k = t$, 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 $s = v_1, v_2, \ldots, v_{k-1}, v_k = t$ (*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.

A directed graph G is a *flow graph* if there is one vertex s (referred to as the *start vertex*) which can reach all other vertices. Given a graph G, a rooted spanning tree T of G is a tree where each leaf is reachable from the root by a directed path. Notice that any flow graph has a spanning tree rooted at the start vertex through a graph visit.

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 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*.*

Let G be an undirected graph. Two subgraphs G_1, G_2 of G can be combined by the operator Δ that simply consists in the symmetric difference of the set of edges. More formally, $G_1 \Delta G_2 = (G_1 \cup G_2) \setminus (E(G_1) \cap E(G_2))$ where $E(G_i)$ is the set of edges of G_i. If $G_3 = G_1 \Delta G_2$ we say that G_3 is *generated* by G_1 and G_2. With this operation, it can be shown that the space of all Eulerian subgraphs of G (called the *cycle space* of G) is a vector space [8, 12, 13, 17].

It is known that a cycle basis for a connected undirected graph G, denoted by $\mathcal{C}(G)$, has dimension $m - n + 1$. If the graph G is not connected this is generalised to $m - n + c$, where c is the number of connected components (see, *e.g.,* [8, 12, 13, 17]). For a given graph G and a spanning tree T on it, the insertion of one further edge e of the graph to this tree produces a unique cycle $C(T, e)$. Given a spanning tree T of G, the set $\mathcal{C}(G) = \{C(T, e) | e \in E(G) \setminus E(T)\}$ is called *Kirchhoff* cycle basis [14].

Let $\mathcal{B}$ be a set of bubbles in G. $\mathcal{B}$ is a *bubble generator* if each bubble in G can be generated by a subset of bubbles in $\mathcal{B}$. A generator is *minimal* if it does not contain a proper subset that is also a generator; and a generator is *minimum* if it has the minimum cardinality. We say that 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. Notice that a bubble generator satisfies Property $\mathcal{P}_2$ if every bubble of the graph has a tree-decomposition in $\mathcal{B}$.

3 Defining a bubble generator from a spanning tree

In this section, we define a bubble generator that satisfies properties ($\mathcal{P}_1$) and ($\mathcal{P}_2$) starting from a spanning tree of the input graph. 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 rooted spanning tree T of G, by performing any graph visit starting from s. In the experimental results in Section 4 we consider different types of visits, such as Depth-First Search, Breadth-First Search and Scan-First Search [5].

Every non-tree edge $e = (u, v)$ encountered during this visit defines a bubble. The source of this bubble is the least common ancestor w of u and v, and its target is v. The two paths 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 the flow graph G.

Theorem 1. *Let G be a flow graph with start vertex s, and let $B_T(G)$ be the set of bubbles identified by a tree T obtained through a visit starting from s. Then each bubble in G can be generated starting from the bubbles in $B_T(G)$ (with a symmetric difference operator), and $|B_T(G)| = m - n + 1$.*

Proof. Let T be a rooted spanning tree of G obtained by a visit starting from s and let $B_T(G)$ be the set of bubbles identified by the non-tree edges of T. Consider the undirected graph G' obtained by ignoring the direction of edges in G. We now consider two cases, depending on whether there are parallel edges in G' or not.

Assume first that there are no parallel edges in G'. Note that there is a one-to-one mapping between (undirected) cycles in G' and bubbles in G, and that the spanning tree T found in G is trivially a spanning tree for G'. It is well-known (see for example [13]) that, given an undirected graph G' without parallel edges, taking the cycles formed by the combination of a path in the spanning tree and a single edge outside the tree yields a cycle basis in G' (with a symmetric difference operator). Consider any bubble B in G and let $B_1, \ldots, B_k$ be the bubbles in $B_T(G)$ identified by the non-tree edges of B. If we ignore the directions of the edges, the above property implies that $B \Delta B_1 \Delta \ldots \Delta B_k$ is empty. Consider now the directed graph G notice that $B \Delta B_1 \Delta \ldots \Delta B_k$ is again empty as each edge in G appears in exactly one direction. Hence, each bubble in G can be generated starting from the bubbles in $B_T(G)$. Since there are $m - (n - 1)$ non-tree edges, $|B_T(G)| = m - n + 1$.

If G' has parallel edges, the previous argument cannot be applied directly. However, in this case a simple reduction will work. Note that in G' there can be at most two parallel edges between any two vertices u and v, corresponding to the two edges (u, v) and (v, u) in the original directed graph G. To deal with this, we transform G into another directed graph G_o as follows: if there are two edges (u, v) and (v, u) in G, we subdivide one of them, say (u, v), by adding a new vertex x_{uv}, by removing the edge (u, v) and by adding two new edges $(u, x_{uv}), (x_{uv}, v)$. Note that there is a one-to-one mapping between bubbles in G and bubbles in G_o: for any vertex x_{uv} in G_o, $(u, x_{uv}), (x_{uv}, v)$ belong to a bubble B_o in G_o if and only if (u, v) belongs to a corresponding bubble B in G. Furthermore, let G'_o be the undirected graph obtained by ignoring the direction of edges in G_o. Since G'_o has no parallel edges, each bubble of G_o can be generated starting from the

bubbles in $B_T(G_o)$. Due to the one-to-one mapping between bubbles of G and bubbles of G_o, this implies that each bubble of G can be generated starting from the bubbles in $B_T(G_o)$. Let k be the number of new vertices x_{uv} added to G_o: note that for each new vertex added to G_o, the number of edges of G_o increases by one. This implies that $B_T(G) = B_T(G_o) = (m + k) - (n + k) + 1 = m - n + 1$ and yields the theorem. ■

Let G be a flow graph with start vertex s and let T be a spanning tree from s. Since each non-tree edge (u, v) is contained exactly in one bubble of $B_T(G)$, Theorem 1 implies that, in order to decompose a generic bubble B into the bubbles 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 in 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 4 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. To prove this we need first two propositions.

Given a bubble B and two *distinct* vertices u, v in B (not necessarily distinct from s, t), an (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 an (u, v)-chord of B such that either there is no directed path $v \rightsquigarrow u$ in B or $\{u, v\} \cap \{s, t\} \neq \emptyset$, then the chord defines two bubbles B_1 and B_2 such that $B = B_1 \Delta B_2$.*

Proof. If u and v are on 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 be considered to be in different legs as s and t belong to both legs of B. It is easy to see that $B = B_1 \Delta B_2$. These cases are depicted in Fig. $1(a) - (d)$. If u and v are on 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. However, if there exists a path from $v \rightsquigarrow u$ in B (see Fig. $1(e_2)$) 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 $B = B_1 \Delta B_2$. ■

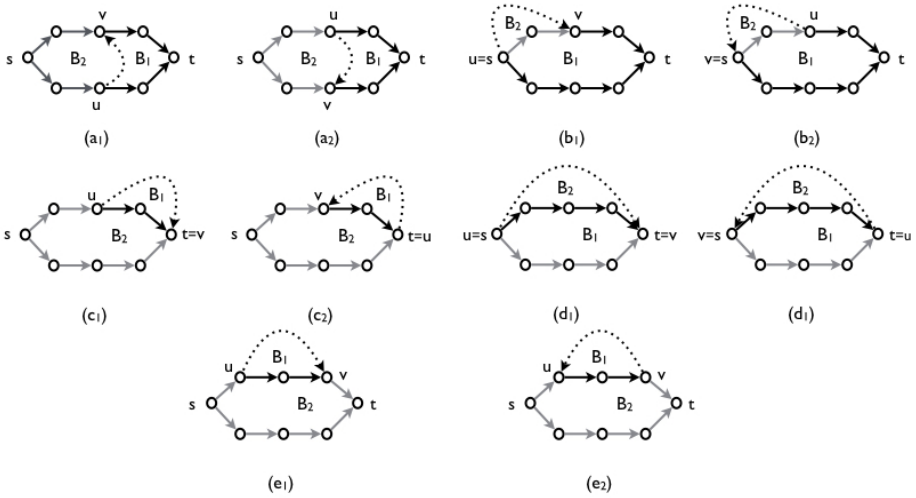

Fig. 1: All the possible cases considered in Proposition 1. In dotted line we have the edges of the (u, v)-chord, in black the bubble B_1 and in grey the bubble B_2.

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

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 can be used to prove the following theorem. For lack of space, its proof is deferred to the Appendix. Moreover, using the same arguments as for Theorem 2, we can extend it to general graphs.

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 in r. Then any bubble B in G can be decomposed in $O(n)$ time in bubbles in $B_T(G)$ in a tree-like fashion.*

4 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 [16, 20] and to the generator defined in [1], we used in our experiments exactly the same dataset as in [1]. This dataset is constructed by selecting the reads corresponding to chromosome 10 from the set of 58 million RNA-seq Illumina paired-end reads extracted from the mouse brain tissue (available in the ENA repository under the following study: PRJEB25574). This leads to a set of 4,932,572 reads. We built the de Bruijn graph from these reads and applied standard sequencing-error-removal procedures by using KisSplice [16, 20]. 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 [5] 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.

4.1 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 [16, 20]).

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.

4.2 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 [16] 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 map the putative bubbles to the *Mus musculus* reference genome and annotations (Ensembl release 94) using STAR [7], which are then analysed by KisSplice2RefGenome [2]. Following [16], 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 $TP/(TP + FP)$.

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

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.

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: (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 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?

Acknowledgments

V. Acuña is supported by Fondecyt 1140631, PIA Fellowship AFB170001 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.

References

1. V. Acuña, R. Grossi, G. F. Italiano, L. Lima, R. Rizzi, G. Sacomoto, M.-F. Sagot, and B. Sinaimeri. On bubble generators in directed graphs. *Algorithmica*. Announced at WG2017.
2. 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.
3. 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.
4. 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.
5. 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.
6. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
7. 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.
8. P. M. Gleiss, J. Leydold, and P. F. Stadler. Circuit bases of strongly connected digraphs. *Discussiones Mathematicae Graph Theory*, 23(2):241–260, 2003.
9. 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.
10. 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.
11. 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.
12. 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.
13. T. Kavitha and K. Mehlhorn. Algorithms to compute minimum cycle bases in directed graphs. *Theory of Computing Systems*, 40(4):485 – 505, 2007.
14. 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.
15. 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.

16. 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.
17. S. MacLane. A combinatorial condition for planar graphs. *Fundamenta Mathematicae*, 28:22–32, 1937.
18. J. R. Miller, S. Koren, and G. Sutton. Assembly algorithms for next-generation sequencing data. *Genomics*, 95(6):315–327, 2010.
19. 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.
20. 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.
21. 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.
22. M. Sammeth. Complete alternative splicing events are bubbles in splicing graphs. *Journal of Computational Biology*, 16(8):1117–1140, 2009.
23. 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.
24. 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.
25. 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.

Appendix

5.1 Tree-decomposition for flow graphs

Let G be a flow graph with source r. Let T be directed rooted tree, with root r. Let $\mathcal{B}_T$ be a bubble generator of G with respect to T. We show that each bubble B in G has a tree decomposition using only $O(n)$ number of bubbles in $\mathcal{B}_T$ and such that in each step, we combine only bubbles.

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 in r. Then any bubble B in G can be decomposed in $O(n)$ 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 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.

Base case: $k = 0$ and $k = 1$ trivial.

Induction step: Suppose that the induction hypothesis is true up to $k-1$. Let B be a bubble with source s and target t. If B is a cycle, then $s = t$ is arbitrarily chosen. Let E be the set of non-tree edges of B. We have $|E| = k$.

It is sufficient to show that we can decompose B into 2 bubbles B_1, B_2, each of of them having E_1 and E_2 non-tree edges such that $E_1 \neq \emptyset, E_2 \neq \emptyset$ and E_1, E_2 form a partition of E. We will use the following straightforward observation:

Remark 1. If $E(B_1) \cup E(B_2) \subseteq E(B) \cup E(T)$ then B_1 and B_2 will have each one strictly less than k non-tree edges. This is due to the fact that we did not add non-tree edges to the ones already present in B and any bubble **must** have at least 1 non-tree edge (as it cannot be entirely in the tree T).

We consider two main cases:

Case I: B is a non-degenerate bubble In this case, necessarily at least one of the two edges entering in t is a non-tree edge. Let $(x, t) \in E$. Consider the path $p = r \rightsquigarrow t$ in T. We follow this path "backwards" starting from t. Notice that we do not require $t \neq r$. The following cases can happen:

(a) The path p leaves the bubble and re-enters in it. Formally, there exists two vertices v and u in $V(B)$ (with v not necessarily distinct from t and u not neccessarly 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 of the legs of B. This case is depicted in Fig. 2(a). Notice that clearly as p is in T, the path cannot touch the same vertex twice, so u and v are distinct. We then have a (u, v)-chord that satisfies the conditions of Proposition 1 and thus we can define B_1, B_2 such that $B = B_1 \Delta B_2$. Notice that from Remark 1, B_1 and B_2 will have each one strictly less than k non-tree edges.

(b) The path p never leaves the bubble, in other words p is entirely included in a leg of B. This means that r belongs to this leg. Notice that this case includes the case $r = t$. There are two cases to consider:

  (b_1) $s \neq r$: then there is a path $q : r \rightsquigarrow s$ in T.This case is depicted in Fig. 2(b_1). Then again starting from s, we follow this path backwards. Notice that as s has no incoming edges in B, the first edge encountered (w, s) is not in B. Clearly q touches again B. Let u be the first vertex we encountered following backwards q, that belongs to B. Notice that u exists as it can be r. Then the (u, s)-chord satisfies the conditions of Proposition 1 and we can define B_1, B_2 such that $B = B_1 \Delta B_2$. From Remark 1 each one of B_1, B_2 has less than k non-tree edges.

  (b_2) We consider now the case $s = r$. In this case the path p coincides with one of the legs of B which then is entirely in T. This case is depicted in Fig. 2(b_2) Now, there must be another edge (w_1, w_2) not in T, otherwise B contains only one non-tree edge and is in the bubble generator. Notice that there must be a path q from r to w_2 in T that is not entirely contained in B. Consider the path q starting from $r = s$ and let u be the first vertex in B such that $(u, u_1) \notin E(B)$ (hence $u \in V(B)$). Starting from u_1, let v be the first vertex in q that belongs to $V(B)$. The (u, v)-chord satisfies the conditions of Proposition 1 and thus we can define B_1, B_2 such that $B = B_1 \Delta B_2$. From Remark 1 each one of B_1, B_2 has less than k non-tree edges.

(c) The path p leaves the bubble but 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 exists as it can be t. In this case, clearly $r \notin V(B)$. There must then be a path $q = r \rightsquigarrow s$ that is not entirely contained in B. Starting from s, we consider the path q backwards. The following two cases can happen:

(c_1) q touches the bubble B. There exists a vertex in q, different from s that belongs to B. This case is depicted in Fig. 2(c_1). Let u be the last vertex that touches B. Again then, the (u, s)-chord satisfies the conditions of Proposition 1 and thus we can define B_1, B_2 such that $B = B_1 \Delta B_2$. From Remark 1 each one of B_1, B_2 has less than k non-tree edges.

(c_2) q does not touch the bubble B. This case is depicted in Fig. 2(c_2). Notice that the paths $r \rightsquigarrow s$ and $r \rightsquigarrow v$ may share some edges, thus we consider r' the least common ancestor of s and v in T. Notice that 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 Remark 1 each one of B_1, B_2 has less than k non-tree edges.

Case II: B is a cycle As B is not in the generator, then there must be at least two edges (x, y) and (w, z) that belong to E. This case is depicted in Fig. 3(a). Notice that y is not necessarily distinct from w. At least one among y and z does not coincide with r and w.l.o.g. we assume $y \neq r$. Consider the path $p = r \rightsquigarrow y$ in T. We follow this path "backwards" starting from y. Clearly the first edge (y_1, y) in this path is not in B. The following 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. 3(a_1) Then we have defined the (u, y) -chord. From Proposition 2, we can define B_1, B_2 such that $B = B_1 \Delta B_2$. From Remark 1 each one of B_1, B_2 has less than k non-tree edges.

(a_2) The path $r \rightsquigarrow y_1$ does not touch the bubble B. This means that the only vertex in common between q 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 z has no incoming edge). This case is depicted in Fig. 3(a_2). Starting from r, let v be the first vertex in the path $r \rightsquigarrow z$ that belongs to $V(B)$. Clearly u 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. From Remark 1 each one of B_1, B_2 has less than k non-tree edges.

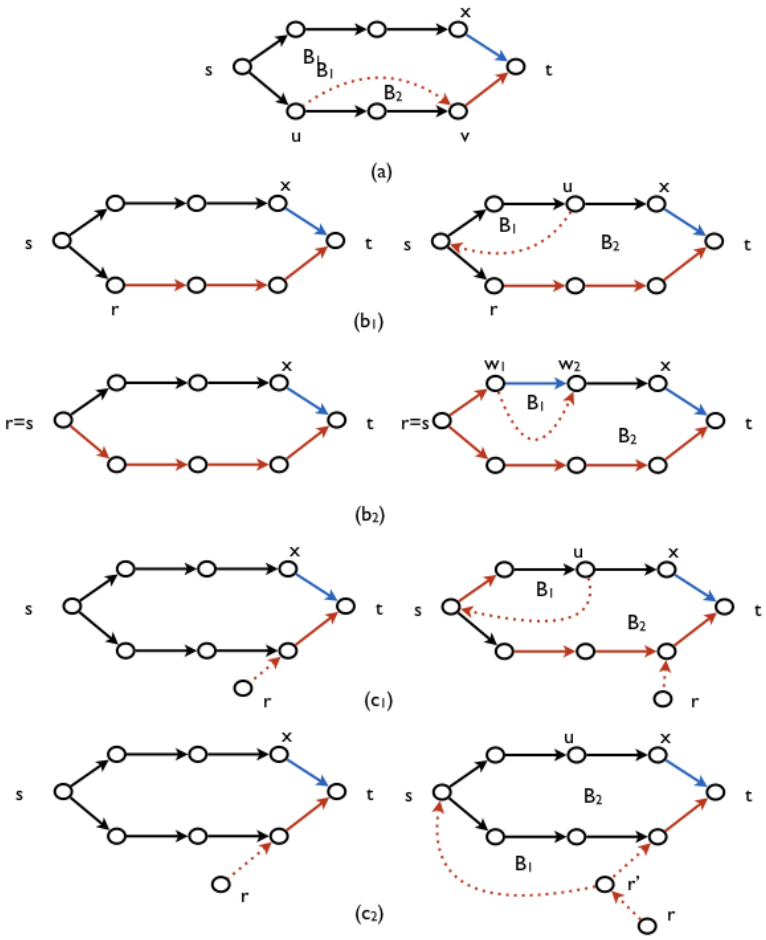

Fig. 2: All the possible cases considered in Case I of Theorem 3. In red we have the edges that belong to the tree T and in a red dotted line a path that belongs to the tree.

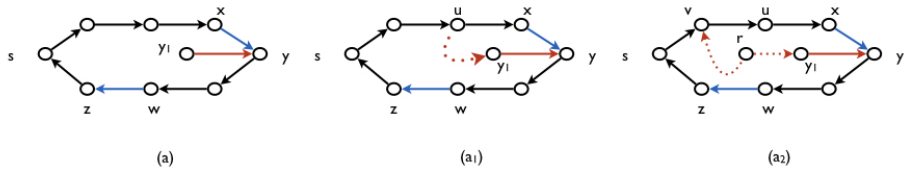

Fig. 3: All the possible cases considered in Case II of Theorem 3. In red we have the edges that belong to the tree T.