# Identifying Clusters in Graph Representations of Genomes

Eva Herencsárová[1], Broňa Brejová[1]

[1]Department of Computer Science, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, Slovakia

**Abstract**

In many bioinformatics applications the task is to identify biologically significant locations in an individual genome. In our work, we are interested in finding high-density clusters of such biologically meaningful locations in a graph representation of a pangenome, which is a collection of related genomes. Different formulations of finding such clusters were previously studied for sequences. In this work, we study an extension of this problem for graphs, which we formalize as finding a set of vertex-disjoint paths with a maximum score in a weighted directed graph. We provide a linear-time algorithm for a special class of graphs corresponding to elastic-degenerate strings, one of pangenome representations. We also provide a fixed-parameter tractable algorithm for directed acyclic graphs with a special path decomposition of a limited width.

**Keywords**

pangenome, elastic degenerate string, maximum-sum segment problem, path decomposition, pathwidth

## 1. Introduction

The rapid decreases in the cost of genome sequencing led to a shift in genomics and bioinformatics from analyzing a single representative genome per species to analyzing genomes of many individuals. A collection of related genomes analyzed jointly is called a *pangenome* [1]. Pangenomes are often represented as graphs, in which nodes correspond to parts of the sequences and edges to adjacencies between these sequences observed in at least one of the studied genomes [2, 3].

Introduction of pangenome graphs gave rise to a need to extend many bioinformatics algorithms from working with single sequences (strings) to graphs representing a family of related sequences. In this work, we introduce algorithms that identify clusters of biologically meaningful positions in such pangenome graphs. In many areas of bioinformatics, one can identify genome positions having some biological function or property and then search for dense clusters of such positions. The simplest examples are based on sequence content, such as looking for GC-rich regions (regions with high density of bases C and G) [4] or CpG islands (regions with high density of C followed by G) [5]. Such areas are often associated with functional elements such as genes or regulatory regions [6, 7]. A more complex example is looking for clusters of motifs representing transcription factor binding sites [8]. We can also identify positions of mutations within or between species and look for conserved regions lacking such mutations [9] or regions with a high density of mutations arising for example from horizontal gene transfer [10]. All of these examples involve identifying individual bases with some biological property and then looking for groups of such bases located close together.

One possible formalization of locating such clusters in a single DNA sequence is to assign a score to each base which is positive for bases with the property of interest and negative for other bases, and then look of high-scoring intervals in the resulting sequence of scores. Miklós Csűrös [4] formulated this approach as looking for a set of disjoint intervals with maximum sum of scores, where the user either restricts the number of intervals to $k$ or assigns some penalty $x$ to each interval in the output set. The latter problem can be solved in linear time by a simple dynamic programming algorithm, and will form the basis of the approach outlined in this article.

Namely, we generalize the maximum-scoring segment set problem [4] from sequences of scores to weighted directed graphs representing pangenomes. The weights of individual nodes represent scores of bases in a pangenome. In a sequence, a cluster is typically defined as a contiguous segment (interval). In the graph extension, one can consider various definitions of the concept of a segment, such as a connected induced subgraph, or subgraphs with special properties, such as superbubbles with a single source and sink [11]. However, we have decided to look for clusters defined as paths in the graph. The advantages of considering paths include a simple problem definition and tractability in some classes of graphs. A path also has an intuitive meaning in a pangenome, as it corresponds to a single sequence (either to a segment of one of the constituent genomes of the pangenome or a combination of multiple such genomes).

Our choice gives rise to the maximum-score disjoint paths problem defined in the next section. In section 3, we provide a linear-time algorithm for a special class of graphs corresponding to elastic-degenerate strings [12]. In section 4, we give an algorithm for general directed acyclic graphs. The complexity of this algorithm is exponential in a parameter of a special path decomposition of the graph, but linear in the overall size of the graph.

*ITAT'23: Information technologies – Applications and Theory, September 22–26, 2023, Tatranské Matliare, Slovakia*

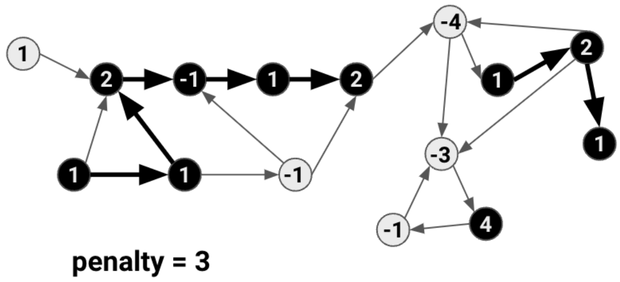


**Figure 1:** An example of a weighted directed graph and the set of paths forming the solution to the maximum-score disjoint paths problem for penalty $x = 3$. The score of this solution is $(1 + 1 + 2 - 1 + 1 + 2 - x) + (1 + 2 + 1 - x) + (4 - x) = 5$.

## 2. Notation and problem definition

In this work, we will consider a weighted directed graph $G$ with vertex set $V$, edge set $E \subseteq V^2$ and weight function $w : V \to \mathbb{R}$. We will first introduce graph terminology and notation used in this work. For each edge $(u, v) \in E$ we call $u$ a predecessor of $v$ and $v$ a successor of $u$. The set of all predecessors of $v$ is denoted $N^-(v)$. The subgraph of $G$ induced by set $X \subseteq V$ is the graph $G' = (X, E \cap X^2)$. A *path* is a sequence of distinct vertices $(v_1, v_2, \ldots, v_n)$ such that $(v_i, v_{i+1}) \in E$ for $i = 1, 2, \ldots, n - 1$. A cycle is a path such that $(v_n, v_1) \in E$. If $G$ does not contain a cycle, we call it a directed acyclic graph (DAG). Vertices of each DAG can be ordered topologically as $v_1, \ldots, v_n$ so that for each edge $(v_i, v_j) \in E$ we have $i < j$.

We are now ready to state our problem. The goal of the *maximum-score disjoint paths* problem is for a given graph $G$ and penalty $x \in \mathbb{R}^+$ to find a set of vertex-disjoint paths with the maximum sum of scores. The score of a single path $P = (v_1, v_2, \ldots, v_n)$ is defined as $\sum_{i=1}^{n} w(v_i) - x$. Figure 1 shows an example of the input and output for this problem.

Note that the maximum-score disjoint paths problem is NP-hard for arbitrary weighted directed graphs. The NP-hardness can be easily proved by a reduction from the Hamiltonian path problem. If we set the weight of each vertex to 1 and penalty also to 1, the graph has a Hamiltonian path if and only if the maximum-score disjoint paths problem has a solution with score $|V| - 1$. We will concentrate on DAGs. Our algorithms are an extension of the dynamic programming algorithm by Csűrös [4] for sequences of scores. The related problems of finding a single segment with maximum score or $k$ segments in a sequence was studied by multiple authors [4, 13, 14]. A single path can also be found on a weighted tree [15, 16]. There are also algorithms for the related maximum density segment problem [17].

## 3. An algorithm for $n$-layered bubble graphs

In this section, we present a linear-time algorithm based on dynamic programming for a special class of directed acyclic graphs, which we call $n$-layered bubble graphs.

**Definition 3.1** ($b$*-layered bubble*)**.** *A* $b$-layered bubble *is a directed acyclic graph with a start vertex* $s$*, an end vertex* $t$ *and* $b$ *non-empty vertex-disjoint directed paths, referred to as layers, connecting* $s$ *and* $t$*.*

**Definition 3.2** ($n$*-layered bubble graph*)**.** *An* $n$-layered bubble graph *is a graph that can be constructed by taking a sequence of vertices* $u_1, \ldots, u_k$ *and connecting each pair of* $u_i$ *and* $u_{i+1}$ *by an edge or by a* $b$*-layered bubble with the start vertex* $u_i$ *and the end vertex* $u_{i+1}$ *and with* $2 \leq b \leq n$*.*

An example of a 3-layered bubble graph can be seen in the bottom part of Figure 2.

**Connection to elastic degenerate strings.** Although the structure of the $n$-layered bubble graphs is very simple, they correspond to a well-studied representation of pangenomes called elastic-degenerate strings (EDSs) [12]. An EDS is a string containing *elastic-degenerate symbols.* An elastic degenerate symbol is defined as a set of strings, potentially of different lengths. Thus the EDS represents a set of strings, each obtained by choosing one of the strings from each elastic degenerate symbol and concatenating them.

An EDS with each set containing at most $n$ strings can be easily converted to an $n$-layered bubble graph by replacing each elastic-degenerate symbol with a bubble, each path spelling one string one character per node. We add start and end vertices with zero weight for each

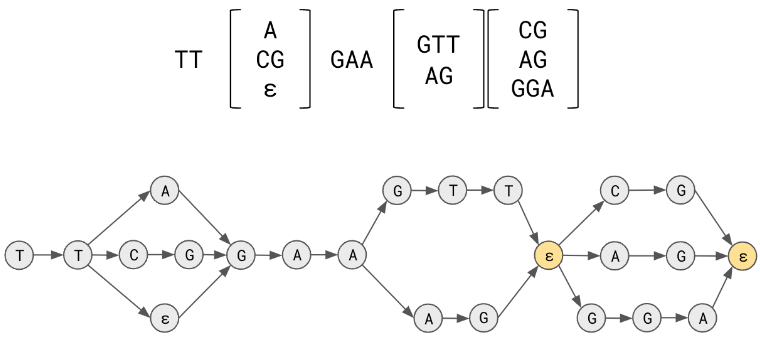


**Figure 2:** An example of an elastic-degenerate string and the corresponding 3-layered bubble graph. The bases are represented as the vertices, their adjacencies as edges. Some vertices contain an empty string denoted as $\varepsilon$.

bubble. Due to zero weight, they do not influence the score of the solution. If an elastic-degenerate symbol contains an empty string in its set, the path for this string will also contain an auxiliary node with zero weight. An example of a conversion of an EDS to a graph is shown in Figure 2.

**An algorithm for a simple path.** Before giving the full algorithm for $n$-layered bubble graphs, we will consider the algorithm for a simple path $(v_1, \dots, v_n)$. This algorithm is very similar to the dynamic programming algorithm given by Csűrös [4] except for a slightly different meaning of the selection value $s$ defined below. The algorithm fills a two-dimensional matrix $W$. For $1 \leq i \leq n$ and $s \in \{0, 1\}$, value $W(i, s)$ is the score of the optimal solution using only vertices $v_1, \dots, v_i$. If $s = 1$, we further constrain the solution to include the last vertex $v_i$ in one of the selected paths. If $s = 0$, we place no further constraints on the solution. Values $W(v_i, s)$ are computed for increasing values of $i$ using the following equations:

$$W(1,1) = w(v_1) - x$$
$$W(1,0) = \max\{0, W(1,1)\}$$
$$W(i,1) = w(v_i) + \max\{W(i-1,0) - x, W(i-1,1)\}$$
$$W(i,0) = \max\{W(i-1,0), W(i,1)\}$$

For $s = 1$, we always use vertex $v_i$ with score $w(v_i)$. One option is that it starts a new path, incurring penalty of $x$. The rest of the solution will use only nodes $v_1, \dots, v_{i-1}$, thus having score $W(i-1, 0)$. If vertex $v_i$ continues an existing path, we instead use subproblem $W(i-1, 1)$ ensuring that such a path exists. For $s = 0$, we consider the case when $v_i$ was used in the path, which has score $W(i, 1)$ and the case when it was not used, which has score $W(i-1, 0)$.

**An algorithm for bubble graphs.** Let $G = (V, E)$ be an $n$-layered bubble graph. We will partition its vertices into sets $N, J, L_1, \dots, L_n$ as follows (see also Figure 3). Each $b$-layered bubble in the graph consists of a start vertex, an end vertex and $b$ disjoint paths $q_1, \dots, q_b$ for $2 \leq b \leq n$. We will place internal vertices of each path $q_i$ to set $L_i$ (the ordering of the paths within the bubble is arbitrary but fixed). The end vertex of the bubble will be placed to set $J$. All remaining vertices will be placed to set $N$. Using the notation from Definition 3.2 for vertices $u_i$ forming the starts and ends of the bubbles, set $N$ includes vertex $u_1$ and any vertex $u_i$ which has a single predecessor. We further split each $L_i$ into sets $L_{i,first}$ and $L_{i,later}$, where $L_{i,first}$ contains vertices from $L_i$ that do not have a predecessor in $L_i$, and $L_{i,later} = L_i \setminus L_{i,first}$.

In our algorithm we will process the vertices in order $O = v_1, \dots, v_{|V|}$, which is a topological order of the graph, and in which for each bubble we first list its vertices from $L_1$, then from $L_2$ and so on.

Our dynamic programming algorithm fills a three-dimensional matrix of scores $W(i, s, \ell)$, where $v_i \in V$, $s \in \{0, 1\}$ is a selection value, and $\ell \in \{I, E\}$ is a path continuation value. Value $W(i, s, \ell)$ is the best score among all sets of disjoint paths within some induced subgraph of $G$ satisfying some additional properties specified below.

The induced subgraph considered in $W(i, s, \ell)$ is defined as follows:

- for $v_i \in N \cup J \cup L_1$:
  the subgraph induced by $\{v_1, \dots, v_i\}$,
- for $v_i \in L_k, 2 \leq k \leq n$ in a bubble $B$:
  the subgraph induced by $\{v_1, \dots, v_i\} \cap L_k \cap B$.

Thus the score in the first layer of the bubble contains information about all previous vertices of the graph, whereas in the remaining layers, we compute only local scores along one path of the bubble.

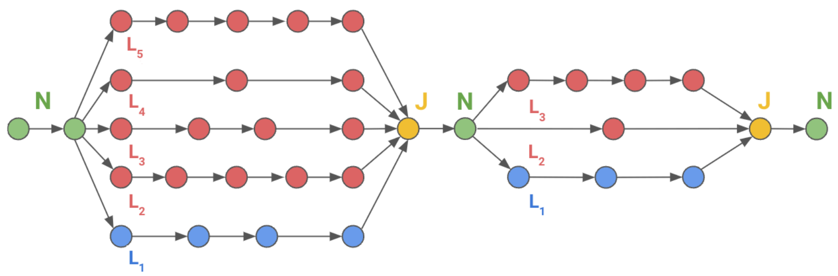


**Figure 3:** An example of a 5-layered bubble graph and its split into sets $N$, $J$, $L_1 \dots L_5$.

The selection value $s$ constrains the set of paths in the same way as in the simpler algorithm for a single path:

- $s = 1$ means $v_i$ is selected in a path,
- $s = 0$ means $v_i$ may or may not be selected in a path (no constraint),

The constraint imposed by the path continuation value $\ell$ depends on the type of the vertex and allows us to ensure that a path entering a bubble from its start vertex will continue in at most one layer of the bubble. For $v_i \in L_1$:

- $\ell = I$: there is no selected path which contains both the bubble's start vertex and the subsequent vertex from $L_{k,first}$ where $k > 1$ (no constraint on $L_{1,first}$).
- $\ell = E$: the bubble's start vertex is selected on a path which continues with a vertex from $L_{k,first}$ where $k > 1$.

For $v_i \in L_k, k > 1$:

- $\ell = I$: the path from the bubble's start vertex continues on layer $L_k$, i.e. both the bubble's start vertex and the $L_{k,first}$ vertex of the current bubble are selected;
- $\ell = E$: there is no path containing both the current bubble's start vertex and the $L_{k,first}$ vertex of the current bubble.

In both cases, value $\ell = I$ means that the path from the start of the bubble continues along the path to which the current vertex $v_i$ belongs. The possibility that the path does not continue to any of the paths of the bubble is included in case $\ell = I$ for the first layer $L_1$. Value $\ell = E$ always includes all cases not considered for $\ell = I$.

For $v_i \in N \cup J$, we will use only $\ell = I$, and we will not impose any additional constraint. Value $W(v_i, s, E)$ is not defined and can be considered as being $-\infty$.

To initialize the algorithm for the first node $v_1$ in ordering $O$, we use similar formulas, as for the simpler case of a single path:

$$W(1, 1, I) = w(v_1) - x$$

$$W(1, 0, I) = \max\{0, W(1, 1, I)\}$$

Since $v_1 \in N$, we have $W(1, 0, E) = W(1, 1, E) = -\infty$.

Let us now consider some vertex $v_a$ for $a \geq 2$. We will distinguish several cases. If $v_a \notin J$, it has a single predecessor, which we denote $v_p$. The simplest case is analogous to the algorithm operating on a single path, and applies to three cases: (1) $v_a \in N$ and $\ell = I$, (2) $v_a \in L_{k,later}$ for any $k$ and any $\ell \in \{I, E\}$, and (3) $v_a \in L_{1,first}, \ell = I$.

$$W(a, 1, \ell) = w(v_a) + \max\{W(p, 0, \ell) - x, W(p, 1, \ell)\}$$
$$W(a, 0, \ell) = \max\{W(p, 0, \ell), W(a, 1, \ell)\}$$

Note that the value of $\ell$ is propagated along the layers in the bubble.

The next case is $v_a \in L_{1,first}$ and $\ell = E$. Value $\ell = E$ means that the path from the predecessor (start of the bubble) continues to some other layer of the bubble, and thus we always apply the penalty if $v_a$ is included in a path. Also, the predecessor $v_p$ is constrained to be on a path, and thus we use $W(p, 1, I)$ instead of $W(p, 0, I)$.

$$W(a, 1, E) = w(v_a) + W(p, 1, I) - x$$
$$W(a, 0, E) = \max\{W(p, 1, I), W(a, 1, E)\}$$

In case of $v_a \in L_{k,first}$ for $k > 1$, we will not use the scores computed for the predecessor, because those are propagated along the first layer. For $\ell = E$ we use similar formulas as for $v_1$. For $\ell = I$ the path has to continue from $v_p$ to $v_a$, leading to more constrained formulas.

$$W(a, 1, I) = w(v_a)$$
$$W(a, 0, I) = W(a, 1, I)$$
$$W(a, 1, E) = w(v_a) - x$$
$$W(a, 0, E) = \max\{0, W(a, 1, E)\}$$

Finally, we will consider the most complex case $v_a \in J$, that is, the end vertex of a processed bubble with $b$ layers. Vertex $v_a$ has in this case $b$ predecessors denoted here as $v_{p_1}, \ldots, v_{p_b}$, where $v_{p_k}$ is in layer $L_k$. The values stored in score matrix $W$ for $p_1, \ldots, p_b$ were calculated for $b$ disjoint subgraphs, and thus to get the score for $a$, the algorithm has to sum up the scores for $p_1, \ldots, p_b$, while ensuring that both at the start and end of the bubble the penalties for new paths are applied properly.

To ensure that the selected path from the bubble's start vertex continues with at most one vertex $v \in L_{k,first}, 1 \leq k \leq b$, we have to use $\ell = I$ for exactly one predecessor and $\ell = E$ for all the others. Recall that the score for $p_1$ when $\ell = I$ also includes the possibility that the selected path does not continue to any of the layers from the bubble's start vertex.

To calculate the score for $W(a, 1, I)$ efficiently, three groups of sums are created and their maximum is used as $W(a, 1, I)$.

The first group corresponds to the situation where $v_a$ starts a new path and incurs a penalty. Therefore it is not important which of the predecessors, if any, were included in some paths.

$$group_1 = \max_k \left( W(p_k, 0, I) + \sum_{i \neq k} W(p_i, 0, E) \right) - x$$

The maximum in $group_1$ goes over $b$ sums, each having path continuation value $I$ for a different predecessor $v_{p_k}$. The value $group_1$ can be calculated in $O(b)$ time. First, the sum $W(p_1, 0, E) + W(p_2, 0, E) + \cdots + W(p_b, 0, E) - x$ is calculated. Then the algorithm changes exactly one addend at a time from $W(p_k, 0, E)$ to $W(p_k, 0, I)$ and chooses the maximum sum.

The second group corresponds to the situation when the path containing $v_a$ continues from some predecessor $v_{p_k}$ without incurring a penalty, and $\ell = I$ for the same predecessor $v_{p_k}$:

$$group_2 = \max_k \left( W(p_k, 1, I) + \sum_{i \neq k} W(p_i, 0, E) \right)$$

Similarly as for $group_1$, the value $group_2$ can be calculated in $O(b)$ time by first calculating the sum $W(p_1, 0, E) + W(p_2, 0, E) + \cdots + W(p_b, 0, E)$, and then always changing exactly one addend at a time from $W(p_k, 0, E)$ to $W(p_k, 1, I)$ and choosing the maximum sum at the end.

The third group corresponds to the situation when the path through $v_a$ continues from some predecessor $v_{p_k}$ without incurring a penalty, and $\ell = I$ for another predecessor $v_{p_j}$ where $k \neq j$. This means that the path from the start of the bubble continues through a different path than the path leading to the end of the bubble.

$$group_3 = \max_{k \neq j} \Bigg( W(p_k, 1, E) + W(p_j, 0, I) + \sum_{i \neq k, i \neq J} W(p_i, 0, E) \Bigg)$$

In this case, $\Theta(b^2)$ sums of length $b$ need to be calculated and compared. This could be done in $O(b^2)$ time similarly as above, only considering all pairs of predecessors $v_{p_k}$ and $v_{p_j}$. However, with some care, the maximum sum can be calculated in $O(b)$ time as follows. The algorithm first calculates the sum $W(p_1, 0, E) + W(p_2, 0, E) + \cdots + W(p_b, 0, E)$ as in group 2. Then it finds addends $W(p_y, 0, E)$ and $W(p_z, 0, E)$ which when replaced with $W(p_y, 0, I)$ and $W(p_z, 1, E)$ maximize the sum.

To do this, the algorithm first finds $p_c$ and $p_d$ ($c \neq d$) for which the difference $W(p_y, 0, I) - W(p_y, 0, E)$ is the largest and second largest, respectively. Next, it finds $p_e$ and $p_f$ ($e \neq f$) for which the difference $W(p_z, 1, E) - W(p_z, 0, E)$ is the largest and second largest, respectively. Both these computations can be done in $O(b)$ time. Finally we use these values to assemble the final value for $group_3$. If $p_c \neq p_e$, then the addend $W(p_c, 0, E)$ is replaced with $W(p_c, 0, I)$, and $W(p_e, 0, E)$ with $W(p_e, 1, E)$. If $p_c = p_e$, then one of the addends is replaced by $W(p_d, 0, I)$ or $W(p_f, 1, E)$ instead, whichever results in a larger sum.

Finally, the value of $W(a, 1, I)$ is derived in the following way:

$$W(a, 1, I) = w(v_a) + \max \begin{cases} group_1 \\ group_2 \\ group_3 \end{cases}$$

To compute $W(a, 0, I)$, we take the maximum of $W(a, 1, I)$ representing the case that $v_a$ is selected and the value $group_4$ representing the case that $v_a$ is not selected. The value of $group_4$ is computed similarly as $group_1$, except that the penalty term $-x$ is not applied. For $v_a \in J$, value $W(a, 0, E)$ and $W(a, 1, E)$ are not defined and can be considered as being $-\infty$.

Once the algorithm fills in the entire matrix $W$, the overall score can be found in $W(|V|, 0, I)$. Note the the last vertex $v_{|V|}$ belongs to $N \cup J$, and thus the value for $\ell = I$ does not pose any constraint on the selected paths. To reconstruct the set of paths leading to the optimal score, we can store for each value of matrix $W$ which case was used to obtain it and then follow these values from $W(|V|, 0, I)$ all the way to $W(1, ?, I)$.

Regarding the time complexity of the algorithm, calculating the scores for each vertex outside of $J$ is done in $O(1)$ time. Calculating the scores for a vertex $v_a \in J$

with indegree $b$ takes $O(b)$ time, but this can be amortized among the $b$ predecessors of $v_a$, each of which has indegree 1. Therefore both the time and space complexity of the algorithm is $O(|V|)$.

## 4. An algorithm for general DAGs

In the previous section, we described an algorithm for the maximum-score disjoint paths problem on $n$-layered bubble graphs. Although such graphs can provide a representation of a pangenome, their power is limited. In this section, we provide a fixed-parameter tractable algorithm for a general DAG, which can solve the problem in time $O(2^w \cdot w \cdot |V|)$ if it is provided with a special directed path decomposition with the width bounded by parameter $w$. In the rest of the section, we first define this decomposition and then describe the algorithm.

**Definitions.** We define the decomposition and its width in the next definition, see also example in Figure 4.

**Definition 4.1** (*Directed path decomposition*). *Let $G = (V, E)$ be a directed graph. A* directed path decomposition *of $G$ is a sequence of subsets $(X_1, \dots, X_n)$ of $V$ (we refer to them as bags of vertices), with three properties:*

(i) *For each edge $(u, v) \in E$, there exists an $i \in \{1, \dots, n\}$ such that both $u$ and $v$ belong to bag $X_i$.*
(ii) *For every three bags $X_i$, $X_j$ and $X_k$ such that $1 \leq i \leq j \leq k \leq n$ we have $X_i \cap X_k \subseteq X_j$.*
(iii) *For each edge $(u, v) \in E$ if $v \in X_j$ then there exists a bag $X_i$ containing $u$ where $i \leq j$.*

*The* width *of the path decomposition is $w = \max_{i \in \{1,\dots,n\}} |X_i| - 1$.*

One can also define the directed pathwidth of graph $G$ as the minimum value $w$ such that $G$ has a path decomposition with width $w$.

Our definitions of a directed path decomposition and a directed pathwidth are extensions of the well-studied path decomposition for undirected graphs [18]. The path decomposition of graph $G$ can be interpreted as a *thickened* path graph. The path width is a value describing how much this path is thickened to get $G$. To adapt the undirected path decomposition for our purposes, we added the third condition. It allows the algorithm to process bags in order and ensure that predecessors of each node are already processed when we process the first bag with this node.

A different path decomposition for directed graphs was previously studied [19, 20], which omits the first condition and uses a less strict version of the third condition as follows: "For each edge $(u, v) \in E$ there exists $i \leq j$ such $u \in X_i$ and $v \in X_j$". However, such a relaxed definition does not seem to lead to an efficient algorithm for our problem.

The following lemma shows a useful property of a directed path decomposition.

**Lemma 4.1.** *Let $G = (V, E)$ be a directed graph and $P = (X_1, \dots, X_n)$ its directed path decomposition. Assume $X_i$ is the bag where vertex $v$ appears for the first time in $P$, i.e. $v \in X_i$ and $v \notin X_j$ where $j < i$. Then bag $X_i$ contains all predecessors of $v$.*

*Proof.* From $(iii)$ in Definition 4.1, we know that each predecessor $p$ of $v$ has to be in some bag $X_h$ for $h \leq i$. Based on $(i)$ in Definition 4.1, there exists a bag $X_k$ containing vertices $p$ and $v$. Since $X_i$ is the bag where $v$ appears for the first time in $P$, $i \leq k$. Based on $(ii)$ in Definition 4.1, $X_i$ contains $p \in X_h \cap X_k$. □

**Corollary 4.1.** *The pathwidth of a directed graph $G$ is at least the maximum indegree of $G$, where the indegree of vertex $v$ is the number of $v$'s predecessors $|N^-(v)|$.*

In our algorithm, we will use a special form of the directed path decomposition, in which a single new node is added in each bag. Below we define it formally and show that any directed path decomposition can be efficiently converted into this form without increasing the width.

**Definition 4.2** (*Incremental path decomposition*). *Let $G = (V, E)$ be a DAG and $P = (X_1, \dots, X_n)$ its directed path decomposition. We consider $X_0 = \emptyset$. We call $P$ an* incremental path decomposition *if $|X_i \setminus X_{i-1}| = 1$ for $1 \leq i \leq n$. The vertex in $X_i \setminus X_{i-1}$ is called the* incremental vertex.

Note that an incremental path decomposition of a DAG $G = (V, E)$ consists of exactly $|V|$ bags, as exactly one vertex is added in each bag and each vertex needs to be added exactly once.

**Lemma 4.2.** *Let $G = (V, E)$ be a DAG and $P = (X_1, \dots, X_n)$ its directed path decomposition of width $w$. It can be converted to an incremental path decomposition for $G$ with a width at most $w$ in $O(w \cdot |V|)$ time.*

*Proof.* Let us assume that $|X_i \setminus X_{i-1}| = k$. If $k = 0$ then $X_i \subseteq X_{i-1}$, and therefore, $X_i$ can be left out of the path decomposition without breaking properties $(i)$, $(ii)$ and $(iii)$ from Definition 4.1. If $k > 1$, we create a path decomposition $P' = X_1, \dots X_{i-1}, Y, X_i, \dots X_n$ where $|Y \setminus X_{i-1}| = 1$ and $|X_i \setminus Y| = k - 1$. By repeating these steps, we get an incremental path decomposition.

To construct $Y$, we consider a topological order of vertices in $G$ and select the vertex $v$ which is the first in this topological order among vertices in $X_i \setminus X_{i-1}$. This means that $v$ has no predecessor in $X_i \setminus X_{i-1}$. Bag $Y$ is constructed as $Y = (X_{i-1} \cap X_i) \cup \{v\}$. Clearly,

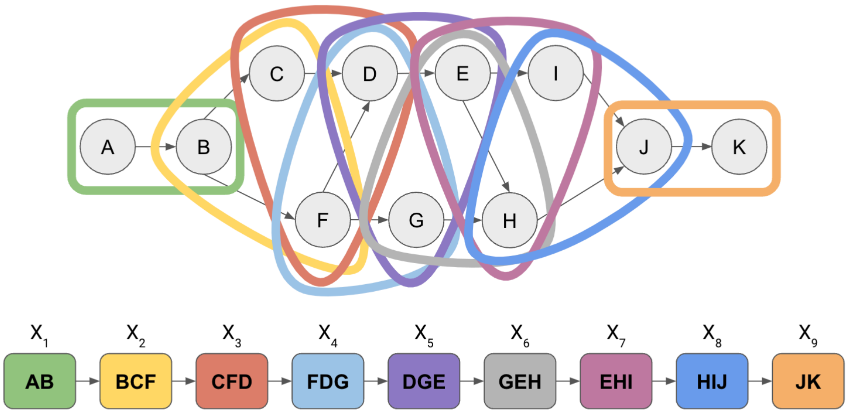


**Figure 4:** An example of a directed path decomposition with width 2 according to Definition 4.1.

decomposition $P'$ satisfies all properties from Definition 4.1. Also notice that $|Y| \leq |X_i|$, i.e. the width of the path decomposition was not increased.

Finally, set $Y$ can be constructed in $O(w)$ time, and as we repeat this process at most $|V|$ times, the total running time is $O(w \cdot |V|)$. The topological order can be computed in $O(|V| + |E|)$ time. Note that $|E| \leq w \cdot |V|$ as each vertex has at most $w$ incoming edges. ☐

**An algorithm that uses an incremental decomposition.** We now describe an algorithm for solving the maximum-score disjoint paths problem for a DAG $G = (V, E)$ and penalty $x$. The input to the algorithm is an incremental path decomposition $P = (X_1, \dots, X_n)$ of $G$. The algorithm runs in $O(2^w \cdot w \cdot |V|)$ time where $w$ is the width of $P$. Let $G_i$ be the subgraph of $G$ induced by vertices in $X_1 \cup \dots \cup X_i$.

The algorithm processes individual bags in the decomposition one at a time. When processing bag $X_i$ the algorithm computes maximum scores of solutions in the subgraph $G_i$. In the first algorithm, we have considered for each ending vertex $v_i$ solutions for different settings of binary variables $s$ and $\ell$. Here we will consider $2^{|X_i|}$ different solutions, each corresponding to a different subset $A \subseteq X_i$. We will call these subsets *configurations*. Configuration $A$ determines which vertices from $X_i$ are the last vertices in individual paths contained in a solution of the problem. The algorithm thus computes a score matrix $W(i, A)$ which contains the score of the best set of paths within $G_i$ such that if $B$ is the set of last vertices of these paths, then $A = B \cap X_i$.

Let us assume that the scores for $X_{i-1}$ are already known, and we want to calculate scores for $X_i$. Let $v_i$ be the incremental vertex of $X_i$. Consider a configuration $A \subseteq X_i$. We will consider two cases.

First, if $v_i \notin A$, then the incremental vertex is not used in any path because it is not the last vertex of any path, and it cannot be in the middle of a path, as it does not have any successors in $G_i$. Therefore we copy some score computed for $X_{i-1}$ to $W(i, A)$. However, we need to consider multiple configurations for $X_{i-1}$ as there can be multiple vertices in $X_{i-1}$ which are not part of $X_i$. These vertices can be part of a configuration for $X_{i-1}$ but are no longer relevant for $X_i$. To this end, for each configuration $A$ of $X_i$ we will define set $p(i, A)$ of configurations of $X_{i-1}$ that agree with $A$ on the vertices shared between $X_{i-1}$ and $X_i$. Formally,

$$p(i, A) = \{B \subseteq X_{i-1} \mid X_{i-1} \cap X_i \cap A = X_{i-1} \cap X_i \cap B\}.$$

Score $W(i, A)$ can then computed as follows:

$$W(i, A) = \max_{B \in p(i, A)} W(i - 1, B)$$

In the second case, $v_i \in A$. The path containing $v_i$ can be either a single vertex, in which case we apply penalty $x$, or $v_i$ can follow some vertex $u \in X_{i-1}$. In that case $u$ must be in the configuration for $X_{i-1}$, because it was the last vertex before addition of $v_i$. But it is not in the configuration for $X_i$, because it is now followed by $v_i$. We consider all possibilities for predecessor $u$ of $v_i$ which is not in $A$ and for configuration $B$ for $X_{i-1}$ which contains $u$, but otherwise agrees with $A$ on the vertices shared between $X_{i-1}$ and $X_i$. Note that all predecessors of $v_i$ are in both $X_i$ (according to Lemma 4.1) and $X_{i-1}$ (because only a single vertex is added to $X_i$).

$$W(i, A) = w(v_i) +$$
$$\max \begin{cases} \max_{B \in p(i,A)} W(i-1, B) - x \\ \max_{u \in N^-(v_i) \setminus A} \max_{B \in p(i, A \cup \{u\})} W(i-1, B) \end{cases}$$

To initialize the algorithm, we set $W(0, \emptyset) = 0$. The final score is the maximum of $W(|V|, A)$ among all configurations $A$ of $X_{|V|}$. The paths can be again reconstructed by keeping track of which configuration $B$ was used to compute each score in matrix $W$.

The above formulas are not convenient for implementation because we need to iterate over multiple configurations $B \subseteq X_{i-1}$ for each configuration $A \subseteq X_i$. It is easier to organize computation in a forward fashion, where we first initialize $W(i, A)$ to $-\infty$ for all $A$ and then iterate over all configurations $B$ of $X_{i-1}$ and use $W(i-1, B)$ to update up to $w+2$ relevant values of $W(i, A)$, as shown in Algorithm 1. The algorithm clearly works in $O(2^w \cdot w \cdot |V|)$ time, provided that sets $A$ and $B$ can be manipulated in $O(1)$ time, which is a reasonable assumption since they are used to address the matrix and thus presumably fit into a single computer word.

**Algorithm 1** Computation of matrix $W$ given an incremental path decomposition $X_1, \ldots, X_{|V|}$ and incremental vertices $v_1, \ldots, v_n$.

```
W(0, ∅) = 0
for all i ∈ {1, ..., |V|} do
    for all A ⊆ X_i do
        W(i, A) ← −∞
    end for
    for all B ⊆ X_{i−1} do
        A ← B ∩ X_i
        W(i, A) = max{W(i, A), W(i − 1, B)}
        A' ← A ∪ {v_i}
        W(i, A') = max{W(i, A'), W(i − 1, B) +
        w(v_i) − x}
        for all u ∈ B ∩ N⁻(v_i) do
            A'' ← A' \ {u}
            W(i, A'') = max{W(i, A''), W(i − 1, B) +
            w(v_i)}
        end for
    end for
end for
```

**Creating an incremental path decomposition.** Our algorithm gets the incremental path decomposition as an input. For completeness we describe a heuristic algorithm for creating an incremental path decomposition for a DAG $G$, although, not necessarily the one with the smallest width. Let $v_1, \ldots, v_n$ be a topological ordering of $G$. We put these vertices into subsequent bags, i.e. $X_i = \{v_i\}$. These bags already fulfill property $(iii)$ from Definition 4.1. From Lemma 4.1 we know that the bag where a vertex appears for the first time also contains all its predecessors. To achieve this, we add all predecessors of $v_i$ into bag $X_i$. This does not break property $(iii)$ from Definition 4.1, and it fulfills property $(i)$. To fulfill property $(ii)$ in Definition 4.1, we find the first and last occurrence of each vertex $v$ in the bags, and add vertex $v$ into the bags in between. This does not break property $(i)$ and $(iii)$ from Definition 4.1 and it fulfills property $(ii)$. The complexity of this algorithm is $O(w \cdot |V|)$.

The resulting path decomposition is incremental. Namely, bag $X_i$ is the first bag where vertex $v_i$ appears, and therefore, $|X_i \setminus X_{i-1}| \geq 1$. The difference of the sets cannot be 2 or more, as then the other additional vertex has to be $v_j$ where $j < i$ which means it appeared already in bag $X_j$, and due to condition $(ii)$ from Definition 4.1 it means $v_j \in X_{i-1}$.

## 5. Experiments

We created a prototype implementation of the algorithm for $n$-layered bubble graphs from Section 3; this implementation can be found at https://github.com/evicy/thesis. We tested our implementation on the task of identifying GC-rich regions in a pangenome of *Escherichia coli* bacterium. The GC content of DNA sequences, i.e. the percentage of guanine (G) and cytosine (C) bases, is a frequently used statistic when analyzing genomes. It has been well studied across organisms, revealing connections between the GC content and various genomic characteristics [21]. GC-rich regions were also used in the study of the maximum segment sum problem by Csűrös [4].

To prepare our data set, we used the complete genome of *E. coli K12-MG1655* as the reference genome [22] and sequencing reads from several strains of *E. coli* isolated from supermarket produce [23]. The reads were downloaded from project PRJNA563564 in the European Nucleotide Archive (ENA) database [24]. We mapped the reads to the genome using BWA [25], processed alignments by SAMtools [26] and then discovered sequence variants for individual strains compared to the reference genome using Freebayes [27], The resulting VCF file with sequence variants was used to construct a elastic-degenerate string by the EDSO [28] tool. Our tool then transforms this EDS to an $n$-layered bubble graph where vertices are single bases (as in Figure 2) and runs our algorithm.

We have tested nine inputs listed as $0, \ldots, 8$ in Table 1. The first input with ID 0 contains only the reference genome, where we effectively solve the maximum-scoring segment set problem of Csűrös [4]. Each successive input adds one additional strain of *E. coli* to the growing pangenome. To find paths with a high GC content, we assigned weight 1 to bases $G$ and $C$ and weight -2 to bases $A$ and $T$. We tested several values of penalty $x \in \{5, 6, 7, 8, 9, 10\}$. These weights mean that the GC content of a selected path is at least 66% to achieve posi-

| ID | Used genomes | $\|V\|$ |
|---|---|---|
| 0 | only the reference [22] | 4,641,654 |
| 1 | ID 0 and SRR10058833 | 4,686,570 |
| 2 | ID 1 and SRR10058834 | 4,743,566 |
| 3 | ID 2 and SRR10058835 | 4,768,722 |
| 4 | ID 3 and SRR10058836 | 4,769,070 |
| 5 | ID 4 and SRR10058837 | 4,769,264 |
| 6 | ID 5 and SRR10058838 | 4,883,827 |
| 7 | ID 6 and SRR10058839 | 4,883,922 |
| 8 | ID 7 and SRR10058840 | 4,884,102 |

**Table 1**
Pangenomes used in our experiment.

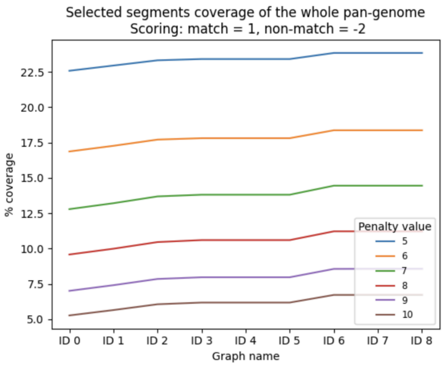


**Figure 5:** Coverage of a pangenome by selected paths representing high GC clusters.

tive score, while the GC content of the *E. coli* genome is 50.8% on average. The penalty ensures that the length of each selected path is at least $x$.

In Figure 5, we can see the coverage, i.e. the percentage of the graph that is covered by the selected paths. As expected, the coverage decreases with increasing penalty. By adding genomic sequences to the pangenome, the coverage is increasing, because some of the new variants will introduce $C$'s and $G$'s that can be used by the selected paths.

## 6. Conclusion

In this work, we have defined the maximum-score disjoint paths problem and provided two algorithms for solving it. The first algorithm runs in linear time on $n$-layered bubble graphs, which can represent pangenomes expressed as elastic-degenerate strings. The second algorithm runs on general DAGs in time $O(2^w \cdot w \cdot |V|)$ where $w$ is the width of a special directed path decomposition defined in this work. We also show the results of a prototype implementation of our first algorithm. In future work, we plan to apply our algorithms to different biological questions stemming from comparative or functional genomics.

Note that our algorithms are purely combinatorial, while many existing approaches for single genomes use statistical methods [29, 10, 30, 31, 32, 33], Csűrös [4] notes that the scores and penalties can be set so that the problem represents finding the maximum likelihood positions of the clusters defined by a two-state hidden Markov model or optimal under complexity penalties, thus providing a link between the combinatorial and statistical versions of the problem for a single genome. Nonetheless, it is an interesting problem to provide an appropriate extensions of statistical models used in sequence analysis for pangenome graphs.

From a more theoretical point of view, it would be interesting to characterize the complexity of our problem on different classes of directed graphs besides the two studied in this work.

## Acknowledgments

This research was supported by the Operational Program Integrated Infrastructure project ITMS:313011ATL7 co-financed by the European Regional Development Fund. The research was also supported by a grant from VEGA 1/0463/20 and the European Union's Horizon 2020 research and innovation program (PANGAIA project #872539 and ALPACA project #956229). We would also like to thank the reviewers for their useful comments.

## References

[1] Computational pan-genomics: status, promises and challenges, Briefings in Bioinformatics 19 (2018) 118–135.

[2] J. M. Eizenga, A. M. Novak, J. A. Sibbesen, et al., Pangenome graphs, Annual Review of Genomics and Human Genetics 21 (2020) 139–162.

[3] J. A. Baaijens, P. Bonizzoni, C. Boucher, G. Della Vedova, Y. Pirola, R. Rizzi, J. Sirén, Computational graph pangenomics: a tutorial on data structures and their applications, Natural Computing 21 (2022) 81–108.

[4] M. Csuros, Maximum-scoring segment sets, IEEE/ACM Transactions on Computational Biology and Bioinformatics 1 (2004) 139–150.

[5] C.-T. Wu, J. C. Dunlap, Homology Effects: Volume 46 - Advances in Genetics, Elsevier Science Publishing Co Inc, 2002.

[6] A. M. Deaton, A. Bird, CpG islands and the regulation of transcription, Genes & Development 25 (2011) 1010–1022.

[7] W. Li, P. Bernaola-Galván, F. Haghighi, I. Grosse, Applications of recursive segmentation to the anal-

ysis of DNA sequences, Computers & Chemistry 26 (2002) 491–510.
[8] X. Wu, S. Liu, G. Liang, Detecting clusters of transcription factors based on a nonhomogeneous poisson process model, BMC Bioinformatics 23 (2022) 535.
[9] N. Stojanovic, L. Florea, C. Riemer, D. Gumucio, J. Slightom, M. Goodman, W. Miller, R. Hardison, Comparison of five methods for finding conserved sequences in multiple alignments of gene regulatory regions, Nucleic Acids Research 27 (1999) 3899–3910.
[10] N. J. Croucher, A. J. Page, T. R. Connor, A. J. Delaney, J. A. Keane, S. D. Bentley, J. Parkhill, S. R. Harris, Rapid phylogenetic analysis of large samples of recombinant bacterial whole genome sequences using Gubbins, Nucleic Acids Research 43 (2015) e15–e15.
[11] L. Brankovic, C. S. Iliopoulos, R. Kundu, M. Mohamed, S. P. Pissis, F. Vayani, Linear-time superbubble identification algorithm for genome assembly, Theoretical Computer Science 609 (2016) 374–383.
[12] C. S. Iliopoulos, R. Kundu, S. P. Pissis, Efficient pattern matching in elastic-degenerate strings, Information and Computation 279 (2021) 104616.
[13] F. Bengtsson, J. Chen, Computing maximum-scoring segments optimally, Luleå tekniska universitet, 2007.
[14] P. Gawrychowski, P. K. Nicholson, Encodings of range maximum-sum segment queries and applications, in: Combinatorial Pattern Matching: 26th Annual Symposium (CPM 2015), Springer, 2015, pp. 196–206.
[15] H.-F. Liu, K.-M. Chao, Algorithms for finding the weight-constrained $k$ longest paths in a tree and the length-constrained $k$ maximum-sum segments of a sequence, Theoretical Computer Science 407 (2008) 349–358.
[16] S. K. Kim, J.-S. Cho, S.-C. Kim, Path Maximum Query and Path Maximum Sum Query in a Tree, IEICE TRANSACTIONS on Information and Systems 92 (2009) 166–171.
[17] K.-M. Chung, H.-I. Lu, An optimal algorithm for the maximum-density segment problem, SIAM Journal on Computing 34 (2005) 373–387.
[18] N. Robertson, P. D. Seymour, Graph minors. i. excluding a forest, Journal of Combinatorial Theory, Series B 35 (1983) 39–61.
[19] J. Barát, Directed path-width and monotonicity in digraph searching, Graphs and Combinatorics 22 (2006) 161–172.
[20] J. Erde, Directed path-decompositions, SIAM Journal on Discrete Mathematics 34 (2020) 415–430.
[21] S. Gelfman, G. Ast, When epigenetics meets alternative splicing: the roles of DNA methylation and GC architecture, Epigenomics 5 (2013) 351–353.
[22] Bethesda (MD): National Library of Medicine (US), National Center for Biotechnology Information, Assembly ASM584v2, Escherichia coli str. K-12 substr. MG1655 (E. coli), https://www.ncbi.nlm.nih.gov/assembly/GCF_000005845.2/, 2013. Accessed: 2023-04-10.
[23] C. J. Reid, K. Blau, S. Jechalke, K. Smalla, S. P. Djordjevic, Whole genome sequencing of Escherichia coli from store-bought produce, Frontiers in Microbiology 10 (2020) 3050.
[24] ENA, Project: PRJNA563564, https://www.ebi.ac.uk/ena/browser/view/PRJNA563564?show=reads, 2019. Accessed: 2023-04-10.
[25] H. Li, Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM, arXiv preprint arXiv:1303.3997 (2013).
[26] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, The sequence alignment/map format and SAMtools, Bioinformatics 25 (2009) 2078–2079.
[27] E. Garrison, G. Marth, Haplotype-based variant detection from short-read sequencing, arXiv preprint arXiv:1207.3907 (2012).
[28] S. P. Pissis, A. Retha, Dictionary matching in elastic-degenerate texts with applications in searching VCF files on-line, in: 17th International Symposium on Experimental Algorithms (SEA 2018), volume 103 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, 2018, pp. 16:1–16:14. Source code is available at https://github.com/webmasterar/edso.
[29] M. Kulldorff, Spatial scan statistics: models, calculations, and applications, in: Scan statistics and applications, Springer, 1999, pp. 303–322.
[30] Z. He, B. Xu, J. Buxbaum, I. Ionita-Laza, A genome-wide scan statistic framework for whole-genome sequence data analysis, Nature Communications 10 (2019) 3018.
[31] F. Ferrari, A. Solari, C. Battaglia, S. Bicciato, Preda: an R-package to identify regional variations in genomic data, Bioinformatics 27 (2011) 2446–2447.
[32] A. Coppe, G. A. Danieli, S. Bortoluzzi, REEF: searching REgionally Enriched Features in genomes, BMC Bioinformatics 7 (2006) 1–7.
[33] E. D. Stavrovskaya, T. Niranjan, E. J. Fertig, S. J. Wheelan, A. V. Favorov, A. A. Mironov, StereoGene: rapid estimation of genome-wide correlation of continuous or interval feature data, Bioinformatics 33 (2017) 3158–3165.