On the Complexity of Positional Sequencing by Hybridization [*]

A. Ben-Dor [†] I. Pe'er [‡] R. Shamir [‡] R. Sharan [‡]

April 2001

Abstract

In sequencing by hybridization (SBH), one has to reconstruct a sequence from its l-long substrings. SBH was proposed as an alternative to gel-based DNA sequencing approaches, but in its original form the method is not competitive. Positional SBH (PSBH) is a recently proposed enhancement of SBH in which one has additional information about the possible positions of each substring along the target sequence. We give a linear time algorithm for solving PSBH when each substring has at most two possible positions. On the other hand, we prove that the problem is NP-complete if each substring has at most three possible positions. We also show that PSBH is NP-complete if the set of allowed positions for each substring is an interval of length k, and provide a fast algorithm for the latter problem when k is bounded.

Keywords: Positional Sequencing by Hybridization, Complexity, Eulerian Graphs, NP-Hardness, Parameterized Algorithms.

1 Introduction

Sequencing by hybridization (SBH) was proposed in the late eighties as an alternative approach to gel-based DNA sequencing [Bains and Smith, 1988, Lysov et al., 1988, Southern, 1988, Drmanac and Crkvenjakov, 1987, Macevics, 1989]. Using DNA chips, cf. [Southern, 1996], one can in principle determine exactly which l-mers

[*] Portions of this paper appeared in the Proceedings of the Tenth Annual Symposium on Combinatorial Pattern Matching [Ben-Dor et al., 1999].

[†] Department of Computer Science and Engineering, University of Washington, Seattle, Washington. amirbd@cs.washington.edu.

[‡] School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel. {izik,shamir,roded}@tau.ac.il.

ISSN 1433-8092

(l-tuples) appear as substrings in a target that needs sequencing, and try to infer its sequence. Practical values of l are 8 to 10.

The fundamental computational problem in SBH is the reconstruction of a sequence from its *spectrum* - the list of all l-mers that are included in the sequence along with their multiplicities. In [Pevzner, 1989] it was shown that the reconstruction problem can be solved efficiently by a reduction to finding an Eulerian path in the following graph: Vertices correspond to $(l-1)$-tuples, and for each l-tuple in the spectrum, an edge connects the vertices corresponding to its $(l-1)$-long prefix and suffix.

The main handicap of SBH is ambiguity of the solution. Alternative solutions are manifested as *branches* in the graph (i.e., two or more edges leaving the same vertex), and unless the number of branches is very small, there is no good way to determine the correct sequence. Theoretical analysis and simulations [Southern et al., 1992, Pevzner and Lipshutz, 1994, Arratia et al., 1997, Dyer et al., 1994] have shown that the average length of a uniquely reconstructible sequence using an 8-mer chip is only about two hundred, way below a single read length on a commercial gel-lane machine.

Due to the centrality of the sequencing problem in biotechnology and in the Human Genome Project, and due to its mathematical elegance, SBH continues to draw a lot of attention. Many authors have suggested ways to improve the basic method. Alternative chip designs [Bains and Smith, 1988, Khrapko et al., 1989, Pevzner et al., 1991, Preparata et al., 1999] as well as interactive protocols [Skiena and Sundaram, 1995] were suggested. An effective and competitive sequencing solution using SBH has yet to be demonstrated.

Recently, several authors have suggested enhancements of SBH based on adding location information to the spectrum [Adleman, 1998, Broude et al., 1994, Hannenhalli et al., 1996, Gusfield et al., 1998]. In *positional sequencing by hybridization* (PSBH), additional information is gathered concerning the position of the l-mers in the target sequence. More precisely, for each l-mer in the spectrum its allowed positions along the target are registered. The reduction to the Eulerian path problem still applies, but for each edge in Pevzner's graph we now have constraints restricting its position in the Eulerian path. Mathematically, this gives rise to the *positional Eulerian path* problem (PEP): Given a directed graph with a list of allowed positions on each edge, decide if there exists an Eulerian path in which each edge appears in one of its allowed positions. In [Hannenhalli et al., 1996] it was shown that PEP is NP-complete, even if all the lists of allowed positions are intervals of equal length. Note that this leaves open the complexity of PSBH in this case. They also gave a polynomial algorithm for the problem when the length of the intervals is bounded.

In this paper we address the positional sequencing by hybridization problem in the case that the number of allowed positions per l-mer is bounded, and the positions need not be consecutive. We give a linear time algorithm for solving the positional Eulerian path problem, and hence, the PSBH problem, in the case that each edge is allowed at most two positions. On the negative side, we show that the PSBH problem is NP-complete, even if each l-mer has at most three allowed positions and multiplicity one. We use in our hardness proof a reduction from the PEP problem restricted to the case where each edge is allowed at most three positions. The latter problem is shown to be NP-complete as well. We also study the complexity of PSBH in the case that the set of allowed positions for each substring is an interval of bounded length. We strengthen the results of [Hannenhalli et al., 1996] with respect to this problem in two ways. First, we show that PSBH is NP-complete, even if all sets of allowed positions are k-long intervals. Second, we give a faster parameterized algorithm for the problem, where the parameter k is an upper bound on the size of the intervals. Our algorithm requires $O(mk^{1.5}4^k)$ time, compared to the $O(mk^3 \log k4^k)$ bound of [Hannenhalli et al., 1996].

The paper is organized as follows: In Section 2 we define the PSBH and the PEP problems. In Section 3 we describe a linear time algorithm for the PEP problem when each edge has at most two allowed positions. In Section 4 we prove that the PEP problem is NP-complete if each edge has at most three allowed positions. In Section 5 we show that the PSBH problem is NP-complete when each l-mer is allowed at most three positions. Furthermore, in Section 6 we prove that the PSBH problem is NP-complete when all sets of allowed positions are intervals of equal length. Finally, in Section 7 we give a parameterized algorithm for the latter problem.

2 Preliminaries

All graphs in this paper are finite and directed. Let $D = (V, E)$ be a graph. We denote $m = |E|$ throughout. For a vertex $v \in V$, we define its *in-neighbors* to be the set of all vertices from which there is an edge directed into v. We denote this set by $N_{in}(v) = \{u : (u, v) \in E\}$. We define the *in-degree* of v to be $|N_{in}(v)|$. The *out-neighbors* $N_{out}(v)$ and *out-degree* are similarly defined.

Let $E = \{e_1, \ldots, e_m\}$ and let P be a function mapping each edge of D to a non-empty set of integer labels from $\{1, \ldots, m\}$. The set $P(e)$ is called the set of *allowed positions* of edge e. The pair (D, P) is called a *positional graph*. If for all e, $|P(e)| \leq k$, then (D, P) is called a *k-positional graph*. Let $\pi = \pi(1), \ldots, \pi(m)$ be a permutation of the edges in E. If π defines a (directed) path, i.e., for each $1 \leq i < m$, $\pi(i) = (u, v)$ and $\pi(i + 1) = (v, w)$, for some $u, v, w \in V$, then we say that π is an *Eulerian path* in D.

An Eulerian path π in D is said to be *compliant* with the positional graph (D, P), if $\pi^{-1}(e) \in P(e)$ for every $e \in E$, that is, each edge in π occupies an allowed position. The k-positional Eulerian path problem is defined as follows:

Problem 1 (k-PEP)

Instance: *A k-positional graph (D, P).*

Question: *Is there an Eulerian path compliant with (D, P)?*

Let $\Sigma = \{A, C, G, T\}$. The *p-spectrum* of a string $X \in \Sigma^*$ is the multi-set of all p-long substrings of X. The problem of sequencing by hybridization is defined as follows:

Problem 2 (SBH)

Instance: *A multi-set S of p-long strings.*

Question: *Is S the p-spectrum of some string X?*

For simplicity, we shall call the input multi-set a spectrum, even if it does not correspond to a sequence. The SBH problem is solvable in polynomial time by a reduction to finding an Eulerian path in Pevzner's graph [Pevzner and Lipshutz, 1994]. More specifically, construct a graph D whose vertices correspond to ($p-1$)-long substrings of strings in S, and in which edges are directed from $\sigma_1 \cdots \sigma_{p-1}$ to $\sigma_2 \cdots \sigma_p$ for each $\sigma_1 \cdots \sigma_p \in S$. Then every solution $\sigma_1 \cdots \sigma_{m+p-1}$ to the SBH instance naturally corresponds to an Eulerian path $\sigma_1 \cdots \sigma_{p-1}, \ldots, \sigma_{m+1} \cdots \sigma_{m+p-1}$ in D.

The *positional SBH* problem is defined as follows:

Problem 3 (PSBH)

Instance: *A multi-set S of p-long strings. For each $s \in S$, a set $P(s) \subseteq \{0, \ldots, |S| - 1\}$.*

Question: *Is S the p-spectrum of some string X, such that for each $s \in S$ its position along X is in $P(s)$?*

If the set of allowed positions for each string is of size at most k, then the corresponding problem is called k-positional SBH, or k-PSBH. k-PSBH is linearly reducible to k-PEP in an obvious manner.

The Interval PEP and Interval PSBH problems are defined as follows:

Problem 4 (Interval PEP)

Instance: *A positional graph (D, P), such that for each edge e, the set $P(e)$ is a sub-interval of $[1, m]$ (i.e., an interval in the linear order $1, \ldots, m$).*

Question: *Is there an Eulerian path compliant with (D, P)?*

Problem 5 (Interval PSBH)

Instance: *A multi-set S of p-long strings. For each $s \in S$, a set $P(s)$ which is a sub-interval of $[0, |S| - 1]$.*

Question: *Is S the p-spectrum of some string X, such that for each $s \in S$ its position along X is in $P(s)$?*

3 A Linear Algorithm for 2-Positional Eulerian Path

In this section we provide a linear time algorithm for solving the 2-positional Eulerian path problem. A key element in our algorithm is a reduction to 2-SAT. Before the reduction can be applied, the input must be preprocessed, discarding unrealizable edge labels (positions).

Let $(D = (V, E), P)$ be the input 2-positional graph. For every $1 \leq t \leq m$ define $\Delta(t)$ to be the set of edges allowed at position t, i.e., $\Delta(t) \equiv \{e \in E : t \in P(e)\}$. Let us call a position t for which $\Delta(t) = \{e\}$ and $|P(e)| > 1$, a *resolvable* position.

The first phase of the algorithm applies the following preprocessing step:

```
While there exists a resolvable position $t$, do:
    Suppose $\Delta(t) = \{e\}$ and $P(e) = \{t, t'\}$.
    $\Delta(t') \leftarrow \Delta(t') \setminus \{e\}$.
    $P(e) \leftarrow \{t\}$.
```

If at any stage we discover that some set $\Delta(t)$ is empty, then we output *False* and halt, since no edge can be labeled t.

Lemma 1 *The preprocessing step does not change the set of Eulerian paths compliant with* (D, P).

Proof: Let S be the set of Eulerian paths compliant with (D, P) before the preprocessing step. It is obvious that the preprocessing step can only reduce S. It thus suffices to prove that if π is compliant with (D, P) then π remains compliant with the new (D, P) obtained by a single iteration of the while loop.

Suppose there exists a resolvable position $1 \leq t \leq m$ such that $\Delta(t) = \{e\}$ and $P(e) = \{t, t'\}$. Then by definition of $\Delta(t)$, any Eulerian path $\pi \in S$ satisfies $\pi(t) = e$. Hence, $\pi(t') \neq e$ and upon updating $P(e) \leftarrow \{t\}$, π remains compliant with the new (D, P). ■

Lemma 2 *The preprocessing step can be implemented in linear time.*

Proof: We maintain current $P(e)$ for each e, and $\Delta(t)$ for each t. Initialization of $\Delta(t)$ for all t can be done in linear time. We further maintain a set Z of resolvable positions, which can be initialized in linear time. The while loop can be implemented by repeatedly handling the positions in Z. Let t be an arbitrary position in Z, where $\Delta(t) = \{e\}$ and $P(e) = \{t, t'\}$. Upon handling t, we delete it from Z, updating $P(e)$ and $\Delta(t')$ in constant time. If t' becomes resolvable, we add it to Z. ■

In the following, (D, P) and Δ refer to the positional graph obtained after the preprocessing phase.

Lemma 3 *In (D, P) each position is allowed for at most two edges.*

Proof: The preprocessing ensures that if for some position t, $|\Delta(t)| = 1$, then $e \in \Delta(t)$ satisfies $|P(e)| = 1$. Let R be the set of positions t with $|\Delta(t)| = 1$, and let $r = |R|$. Then there are $m - r$ positions t for which $|\Delta(t)| \geq 2$, and $r' \geq r$ edges e with $|P(e)| = 1$. Thus,

$$2(m-r) \leq \sum_{t \notin R} |\Delta(t)| = \sum_{t} |\Delta(t)| - r = \sum_{e} |P(e)| - r = 2m - r' - r \leq 2(m-r) \; .$$

Hence, $r = r'$ and each label $t \notin R$ occurs exactly twice, implying that $|\Delta(t)| \in \{1, 2\}$ for all t. ■

For every vertex $v \in V$ define $In(v, t)$ as the set of t-labeled edges entering v, i.e.,

$$In(v,t) \equiv \{(u,v) : (u,v) \in \Delta(t)\} \; .$$

Similarly define

$$Out(v,t) \equiv \{(v,u) : (v,u) \in \Delta(t)\} \; .$$

We say that a vertex v is *fixed to position* t in (D, P) if $In(v, t) = \Delta(t)$ or $Out(v, t+1) = \Delta(t+1)$. In other words, any Eulerian path compliant with (D, P) must have v as the $(t+1)$-st vertex in the path. Define Boolean variables X_e^t for every e, t such that $t \in P(e)$ ($2m - r$ variables in total). Examine the following sets of Boolean clauses:

$$X_e^t \qquad \text{for every } e, t \text{ such that } P(e) = \{t\} \; . \tag{1}$$

$$X_{e_1}^t \oplus X_{e_2}^t \qquad \text{for every } e_1, e_2, t \text{ such that } \Delta(t) = \{e_1, e_2\} \; . \tag{2}$$

$$X_e^{t_1} \oplus X_e^{t_2} \qquad \text{for every } e, t_1, t_2 \text{ such that } P(e) = \{t_1, t_2\} \; . \tag{3}$$

$$X_{(a,b)}^t \Leftrightarrow X_{(b,c)}^{t+1} \qquad \text{for every } t \in P((a,b)), t+1 \in P((b,c)) \text{ such that } b \text{ is not fixed to position } t \; . \tag{4}$$

$$\overline{X}_{(u,v)}^t \qquad \text{for every } t \in P((u,v)), t < m \text{ such that } Out(v, t+1) = \emptyset \; . \tag{5}$$

$$\overline{X}_{(u,v)}^t \qquad \text{for every } t \in P((u,v)), t > 1 \text{ such that } In(u, t-1) = \emptyset \; . \tag{6}$$

Lemma 4 *There is a positional Eulerian path compliant with (D, P) if and only if the set of clauses (1)-(6) is satisfiable.*

Proof:

$\Leftarrow$ Suppose that a satisfying truth assignment Φ exists. Let us assign an edge e to position t if and only if $\Phi(X_e^t)$ =*True*. Clauses (1) and (2) guarantee that exactly one edge is assigned to each position.

Clauses (1) and (3) guarantee that each edge is assigned to exactly one position, and that this position is allowed for it.

It remains to show that the above assignment of edges to positions yields a path in D. Suppose to the contrary that both $X^t_{(a,b)}$ and $X^{t+1}_{(b',c')}$ are assigned *True*, with $b \neq b'$. Then clauses (5) guarantee the existence of an edge $(b,c) \in \Delta(t+1)$, while clauses (6) guarantee the existence of an edge $(a',b') \in \Delta(t)$. Therefore, b is not fixed to t, and a contradiction follows from clauses (4). Hence, Φ defines an Eulerian path compliant with (D,P).

$\Rightarrow$ Suppose that π is an Eulerian path compliant with (D,P). A truth assignment Φ satisfying clauses (1)-(6) can be defined as follows: Assign $\Phi(X^t_e)$ =*True* if and only if $\pi(t) = e$. As π is a permutation, Φ satisfies clauses (1)-(3). Since π is a path, Φ satisfies clauses (5) and (6). It remains to prove that Φ satisfies clauses (4). Consider the clause $X^t_{(a,b)} \Leftrightarrow X^{t+1}_{(b,c)}$ in (4). Since b is not fixed to t, $\Delta(t) = \{(a,b),(a',b')\}$ and $\Delta(t+1) = \{(b,c),(b'',c'')\}$ where $b', b'' \neq b$. There are two possible cases. Either $\pi(t) = (a,b)$ and then $\pi(t+1) = (b,c)$, or $\pi(t) \neq (a,b)$ and then $\pi(t+1) \neq (b,c)$. In both cases the clause is satisfied.

■

Lemma 5 *Clauses (1)-(6) can be generated in linear time.*

Proof: By definition and Lemma 3, each of the sets $P(e)$ and $\Delta(t)$ is of size at most 2. Clauses (1)-(3) can be trivially generated by scanning the sets $P(e)$ for all e and $\Delta(t)$ for all t. This computation takes $O(m)$ time. To construct clauses (5) we examine each edge $e = (u,v)$ and each $t \in P(e)$ (there are at most $2m$ such pairs (e,t)). If $\Delta(t+1)$ contains no edge of the form (v,w), we include the clause $\bar{X}^t_{(u,v)}$. Construction of these clauses takes $O(m)$ time. Clauses (6) are constructed in a similar manner. Finally, clauses (4) can be constructed in linear time by examining $\Delta(t)$ and $\Delta(t+1)$ for all $t < m$. If we find a pair of edges $(a,b) \in \Delta(t)$ and $(b,c) \in \Delta(t+1)$, such that there exist edges $(u,v) \in \Delta(t)$, $v \neq b$ and $(w,z) \in \Delta(t+1)$, $w \neq b$, we include the appropriate clause. ■

Theorem 6 2-*PEP is solvable in linear time.*

Proof: The preprocessing phase is linear by Lemma 2. By Lemma 5, the number of clauses (1)-(6) is $O(m)$. Each exclusive OR clause in (2)-(3) and each equivalence clause in (4) can be written as two OR clauses, and therefore, by Lemma 5 one can generate all clauses in linear time. By Lemma 4, the problem is reduced to an instance of 2-SAT which is solvable in linear time [Apsvall et al., 1979]. ■

Corollary 1 2-*PSBH is solvable in linear time.*

All clauses (1)-(6) can take the form of $A \Leftrightarrow B$, with A, B being constants or literals. Hence, the 2-SAT instance can in fact be solved by a linear-time algorithm simpler than [Apsvall et al., 1979] as follows: Let S be the set of all clauses, written as equivalence clauses. Construct an undirected graph $G = (U, F)$ such that $U = \{X_e^t, \bar{X}_e^t : e \in E, t \in P(e)\} \cup \{0, 1\}$. Include (A, B) and $(\bar{A}, \bar{B})$ in F whenever $A \Leftrightarrow B$ is a clause in S. It is easily seen that the 2-SAT instance is satisfiable if and only if no connected component of G contains two vertices corresponding to a variable (or a constant) and its negation.

4 3-Positional Eulerian Path is NP-Complete

In this section we prove that the 3-PEP problem is NP-complete by reduction from 3-SAT.

Theorem 7 *The* 3-*PEP problem is NP-complete.*

Proof: Membership in NP is trivial. We prove NP-hardness by reduction from *3-SAT*. We first provide a sketch of the construction. For each occurrence of a literal in the formula, a *special vertex* is introduced. Special vertices corresponding to the same literal are connected serially to form a *literal path*. Two literal paths of a variable and its negation are connected in parallel to form a *variable subgraph*. For each clause in the formula, the corresponding special vertices are connected by three edges to form a *clause triangle*. Finally, for each special vertex we introduce a triangle incident on it, called its *bypass triangle* (see Figure 1).

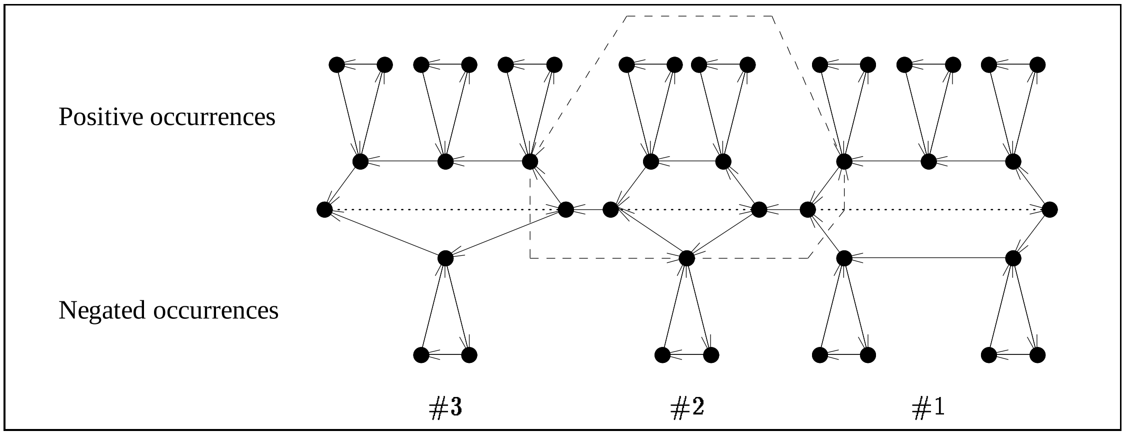

Figure 1: A schematic sketch of the main elements in our construction. The figure includes three variable subgraphs. The first variable, whose subgraph is the rightmost (#1), has three positive occurrences (top) and two negated occurrences (bottom), etc. One of the clause triangles is also drawn, using dashed line.

The sets of allowed positions are chosen so that they force every compliant Eulerian path to visit the literal paths one by one. A compliant Eulerian path corresponds to a satisfying truth assignment. When a special vertex is visited, either its clause triangle, or its bypass triangle are traversed. Traversing the clause triangle while passing through a certain literal's path, corresponds to this literal satisfying the clause. Eventually, we enable visiting all unvisited bypass triangles.

We now give the construction in detail. Let F be a 3-CNF formula with N variables $x_1, \ldots, x_N$, and M clauses $C_1, \ldots, C_M$. We assume, w.l.o.g., that each clause contains three distinct variables, and that all $2N$ literals occur in F. Denote $X_i = \{x_i\} \cup \{\overline{x}_i\}$. For a literal $L \in X_i$, let a_L denote the number of its occurrences in F. For $1 \leq j \leq a_L$ define $L(j) \equiv (L, j)$. Thus, $L(1), \ldots, L(a_L)$ is an enumeration of L's occurrences in F. For a clause $C = L \vee L' \vee L''$ introducing the j-th (j', j'') occurrence of L (L', L'', respectively), we write $C = L(j) \vee L'(j') \vee L''(j'')$. We shall construct a directed graph $D = (V, E)$ and a map P from E to integer sets of size at most 3, such that F is satisfiable if and only if (D, P) has a compliant Eulerian path.

We introduce the following vertices:

- $u_i, \hat{u}_i$ for each variable x_i, $1 \leq i \leq N$.
- $v_{L(j)}, \hat{v}_{L(j)}$ for each occurrence $L(j)$ of the literal L. $v_{L(j)}$ is called *special*. For $L \in X_i$, we shall denote u_i also by $v_{L(0)}$, and $\hat{u}_i$ also by $v_{L(a_L+1)}$.
- $r(C_c)$ for each clause C_c, $1 \leq c \leq M$, identifying $\hat{u}_N$ as $r(C_0)$ and u_1 as $r(C_{M+1})$.

We introduce the following edges:

- For each clause $C = L(j) \vee L'(j') \vee L''(j'')$, a clause triangle consisting of the edges $\{(v_{L(j)}, v_{L'(j')}), (v_{L'(j')}, v_{L''(j'')}), (v_{L''(j'')}, v_{L(j)})\}$.
- For each occurrence $L(j)$ of a literal L in a clause C, a bypass triangle with the edges $\{(v_{L(j)}, \hat{v}_{L(j)}), (\hat{v}_{L(j)}, r(C)), (r(C), v_{L(j)})\}$.
- A literal path $lpath(L)$: $\{(u_i, v_{L(1)}), (v_{L(1)}, v_{L(2)}), (v_{L(2)}, v_{L(3)}), \ldots, (v_{L(a_L)}, \hat{u}_i)\}$, for each literal $L \in X_i$.
- For $i = 1, \ldots, N$, *back edges* $(\hat{u}_i, u_i)$. For $i = 1, \ldots, N-1$, *forward edges* $(\hat{u}_i, u_{i+1})$.
- A *finishing path* $\{(\hat{u}_N, r(C_1)), (r(C_1), r(C_2)), (r(C_2), r(C_3)), \ldots, (r(C_M), u_1)\}$.

Figure 2 shows an example of the constructed graph. The motivation for this construction is the following: Using the position sets, we intend to force the literal paths of the different variables to be traversed in the natural order, where the only degree of freedom is switching the order between $lpath(x_i)$ and $lpath(\overline{x}_i)$. This switch will correspond to a truth assignment for the variable x_i, by assigning *True* to the literal in X_i whose $lpath$ was visited first. After visiting a special vertex along this first path, we either visit its clause triangle, or its bypass triangle. Along the other path (the one of the literal assigned *False*) only a bypass triangle can be visited.

Eventually, the finishing path is traversed. Its vertices are visited in the natural order. Upon visiting a vertex $r(C)$, we visit only one bypass triangle - the yet unvisited triangle among those corresponding to the literals of clause C. The truth assignment will satisfy that literal.

We now describe the sets $P(e)$. We use the following notation:

$$\begin{aligned}
b_i &= a_{x_i} + a_{\overline{x}_i} \quad \text{for } i = 1, \ldots, N \; . \\
B_i &= \sum_{j=1}^{i} b_j \quad \text{for } i = 0, \ldots, N \; (B_0 = 0) \; . \\
Base_L = Base_{\overline{L}} = Base_i &= 4B_{i-1} + 4(i-1) \quad \text{for } L \in X_i \; . \\
Alternate_L &= Base_i + 4a_{\overline{L}} + 2 \quad \text{for } L \in X_i \; . \\
ClauseBase_c &= Base_{N+1} + 4c \quad 0 \le c \le M \; .
\end{aligned}$$

- For each forward edge $e = (\hat{u}_{i-1}, u_i)$, $2 \le i \le N$, we set $P(e) = \{Base_i\}$. This is intended to ensure that the literal paths are traversed in a constrained order: $lpath(x_i)$ and $lpath(\overline{x}_i)$ are allocated a time interval $[Base_i + 1, Base_{i+1} - 1]$ of length $4b_i + 3$, during which they must be traversed.

- For each back edge $e = (\hat{u}_i, u_i)$ we set $P(e) = \{Alternate_{\overline{x}_i}, Alternate_{x_i}\}$. This enables either visiting $lpath(x_i)$ first, then e and $lpath(\overline{x}_i)$; or visiting $lpath(\overline{x}_i)$ first, followed by e and $lpath(x_i)$.

- For each literal path edge $e = (v_{L(j)}, v_{L(j+1)})$, with $L \in X_i$, $0 \le j \le a_L$, we set $P(e) = \{Base_i + 4j + 1, Alternate_L + 4j + 1\}$. Consecutive edges in a literal path are thus positioned 4 time units apart (allowing a triangle in-between).

- For each clause $C = L_1(j_1) \vee L_2(j_2) \vee L_3(j_3)$ with the clause triangle $\{e_1 = (v_{L_1(j_1)}, v_{L_2(j_2)}), e_2 = (v_{L_2(j_2)}, v_{L_3(j_3)}), e_3 = (v_{L_3(j_3)}, v_{L_1(j_1)})\}$ such that $L_k \in X_{i_k}$, define $t_k \equiv Base_{i_k} + 4j_k - 2$ and set

$$\begin{aligned}
P(e_1) &= \{t_1, t_3 + 1, t_2 + 2\} \; , \\
P(e_2) &= \{t_2, t_1 + 1, t_3 + 2\} \; , \\
P(e_3) &= \{t_3, t_2 + 1, t_1 + 2\} \; .
\end{aligned}$$

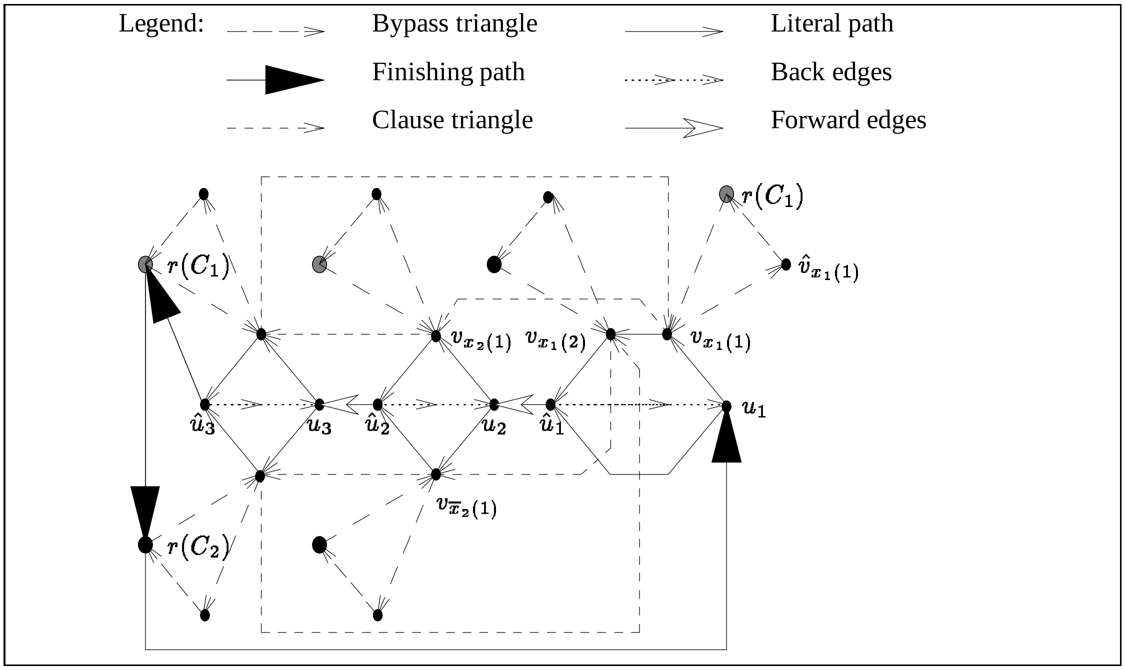

Figure 2: An example of the construction for the formula $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_3)$. All large grey (black) vertices are actually the same vertex $r(C_1)$ ($r(C_2)$).

(See Figure 3.) This means that the edges of a clause triangle must be visited consecutively during the traversal of $lpath(L_k)$, for some k. Furthermore, note that this may happen only if $lpath(L_k)$ is traversed immediately after time $Base_{L_k}$, that is, only if it precedes $lpath(\overline{L}_k)$.

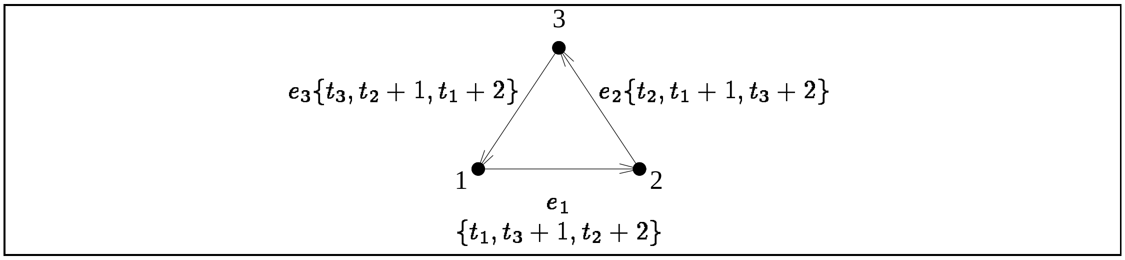

Figure 3: A clause triangle, with vertex $v_{L_k(j_k)}$ denoted by k. The allowed positions for each edge appear in braces.

- For each finishing edge $e = (r(C_c), r(C_{c+1}))$, $0 \leq c \leq M$, we set $P(e) = \{ClauseBase_c\}$, thus determining the order in which the vertices of the finishing path are traversed, allowing a time slot $[ClauseBase_c + 1, ClauseBase_{c+1} - 1]$ of length 3, for the bypass triangle visited while traversing $r(C_c)$.

- For a bypass triangle with the edges $\{e = (v_{L(j)}, \hat{v}_{L(j)}),\ e' = (\hat{v}_{L(j)}, r(C_c)),\ e'' = (r(C_c), v_{L(j)})\}$, we set:

$$\begin{aligned} P(e) &= \{Base_L + 4j - 2, Alternate_L + 4j - 2, ClauseBase_c - 2\} \ , \\ P(e') &= \{Base_L + 4j - 1, Alternate_L + 4j - 1, ClauseBase_c - 1\} \ , \\ P(e'') &= \{Base_L + 4j, Alternate_L + 4j, ClauseBase_c - 3\} \ . \end{aligned}$$

This means that the bypass triangle edges must be visited consecutively, and there are three possible time slots for that:

- While traversing $lpath(L)$, before traversing $lpath(\overline{L})$.
- While traversing $lpath(L)$, after traversing $lpath(\overline{L})$.
- While traversing $r(C_c)$ along the finishing path.

The reduction is obviously polynomial. We now prove validity of the construction.

$\Leftarrow$ Suppose that F is satisfiable. We will show that (D, P) is a ”yes” instance of the 3-positional Eulerian path problem. Let ϕ be a truth assignment satisfying F. For each clause C_c, let $L_c(j_c)$ be a specific literal occurrence satisfying C_c.

We describe an Eulerian path π in D. Set $\pi(ClauseBase_c) = (r(C_c), r(C_{c+1}))$, for $c = 0, \ldots, M$. Set $\pi(Base_i) = (\hat{u}_{i-1}, u_i)$, for $i = 2, \ldots, N$. For all i, if $\phi(x_i)$ =*True*, set $\pi(Alternate_{\overline{x}_i}) = (\hat{u}_i, u_i)$. Otherwise, set $\pi(Alternate_{x_i}) = (\hat{u}_i, u_i)$.

For each literal $L \in X_i$:

- If $\phi(L)$ =*True*: For each $0 \leq j \leq a_L$, set $\pi(Base_i + 4j + 1) = (v_{L(j)}, v_{L(j+1)})$ (see Figure 4, top).

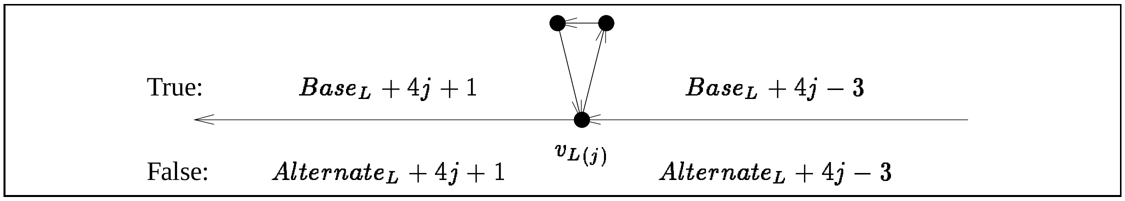

Figure 4: Either a clause triangle or a bypass triangle must be traversed upon visiting a special vertex $v_{L(j)}$, due to time constraints. Edge positions in case L is assigned *True* (*False*) are shown at the top (bottom).

We further distinguish between two cases:

- $*$ If $L(j) = L_c(j_c)$ for the clause $C_c = L(j) \vee L'(j') \vee L''(j'')$ in which $L(j)$ occurs, then set π to visit the edges of the clause triangle of C_c as follows:

$$\begin{aligned} \pi(Base_L + 4j - 2) &= (v_{L(j)}, v_{L'(j')}) \ , \\ \pi(Base_L + 4j - 1) &= (v_{L'(j')}, v_{L''(j'')}) \ , \\ \pi(Base_L + 4j) &= (v_{L''(j'')}, v_{L(j)}) \ . \end{aligned}$$

Furthermore, in this case we set π to visit the edges of the bypass triangle of $L(j)$ as follows:

$$\begin{aligned} \pi(ClauseBase_c - 3) &= (r(C_c), v_{L(j)}) \ , \\ \pi(ClauseBase_c - 2) &= (v_{L(j)}, \hat{v}_{L(j)}) \ , \\ \pi(ClauseBase_c - 1) &= (\hat{v}_{L(j)}, r(C_c)) \ . \end{aligned}$$

- $*$ Otherwise, $L(j) \neq L_c(j_c)$ for the clause C_c in which $L(j)$ occurs. In this case we set π to visit the edges of the bypass triangle of $L(j)$ as follows:

$$\begin{aligned} \pi(Base_L + 4j - 2) &= (v_{L(j)}, \hat{v}_{L(j)}) \ , \\ \pi(Base_L + 4j - 1) &= (\hat{v}_{L(j)}, r(C_c)) \ , \\ \pi(Base_L + 4j) &= (r(C_c), v_{L(j)}) \ . \end{aligned}$$

– If $\phi(L)$ =*False*: For each $0 \leq j \leq a_L$, set $\pi(Alternate_L + 4j + 1) = (v_{L(j)}, v_{L(j+1)})$. Furthermore, in this case we set π to visit the edges of the bypass triangle of $L(j)$ as follows (see Figure 4, bottom):

$$\begin{aligned} \pi(Alternate_L + 4j - 2) &= (v_{L(j)}, \hat{v}_{L(j)}) \ , \\ \pi(Alternate_L + 4j - 1) &= (\hat{v}_{L(j)}, r(C_c)) \ , \\ \pi(Alternate_L + 4j) &= (r(C_c), v_{L(j)}) \ . \end{aligned}$$

It is easy to see that π is a permutation of the edges, and if $\pi(k) = (u, v)$,$\pi(k+1) = (u', v')$ then $v = u'$. Hence, π is an Eulerian path. Furthermore, by our construction π is compliant with (D, P), proving that (D, P) is a ”yes” instance.

$\Rightarrow$ Let π be an Eulerian path compliant with (D, P). We shall construct an assignment ϕ satisfying F. In order to determine $\phi(x_i)$ we consider the edge $\pi(Base_i + 1)$. By construction, $\pi(Base_i + 1) = (u_i, v_{L(1)})$ for $L \in X_i$. We therefore set $\phi(L)$ =*True* (and of course $\phi(\overline{L})$ =*False*). We observe that for any other edge $e' = (v_{L(j)}, v_{L(j+1)})$ along $lpath(L)$, we must have $\pi(Base_i + 4j + 1) = e'$ if and only if $\phi(L)$ =*True*.

We now prove that ϕ satisfies each clause $C_c = L_1(j_1) \vee L_2(j_2) \vee L_3(j_3)$ of F. Consider the clause triangle of C_c: $\{e_1 = (v_{L_1(j_1)}, v_{L_2(j_2)}), e_2 = (v_{L_2(j_2)}, v_{L_3(j_3)}), e_3 = (v_{L_3(j_3)}, v_{L_1(j_1)})\}$. Denote $t_k = Base_{L_k} + 4j_k - 2$. By the positional constraints, there exists some $1 \leq k \leq 3$ for which $\pi(t_k) = e_k$. The edge e preceding e_k in π must have $t_k - 1 = Base_{L_k} + 4(j_k - 1) + 1 \in P(e)$. The only such edge entering $v_{L_k(j_k)}$ is the literal path edge $(v_{L_k(j_k-1)}, v_{L_k(j_k)})$. Therefore, $\phi(L_k)$ =*True*, satisfying C_c.

This proves that F is satisfiable if and only if (D, P) is a ”yes” instance, completing the proof of Theorem 7.

■

Observe that the graph constructed in the proof of Theorem 7 has in-degree and out-degree bounded by 4, giving rise to the following result:

Corollary 2 *3-PEP is NP-complete, even when restricted to graphs with in-degree and out-degree bounded by 4 .*

Henceforth, we call this restricted problem *(3,4)-PEP*. We comment that a slight modification of the construction results in a graph whose in-degree and out-degree are bounded by 2.

5 3-Positional SBH is NP-Complete

We show in this section that the problem of sequencing by hybridization with at most 3 positions per spectrum element is NP-complete, even if each element in the spectrum is unique. The proof is by reduction from (3,4)-PEP.

Theorem 8 *The 3-PSBH problem is NP-complete, even if all spectrum elements are of multiplicity one.*

Proof: It is easy to see that the problem is in NP. We reduce (3,4)-PEP to 3-PSBH. Let $(D = (V, E), P)$ be an instance of (3,4)-PEP. Let $k = \lceil \log_4 |V| \rceil + 2$, $p = 3k + 1$ and $c = p + 1$. In order to construct an instance of 3-PSBH we first encode the edges and vertices of D. In the following, we denote string concatenation by $|$. We let σ_1 =’A’, σ_2 =’C’, σ_3 =’G’ and σ_4 =’T’.

To each $v \in V$ we assign a distinct string in Σ^{k-2}. We add a leading ’T’ symbol and a trailing ’T’ symbol to this string, and call the resulting k-long string the *name* of v. We also assign the string ’A. . . A’ of length k

to encode a *space*. Each vertex is encoded by a $3k$-long string containing two copies of its name separated by a space. We denote the encoding of v by $en(v)$. Each edge $(u, v) \in E$ is encoded by two symbols chosen as follows: Let $N_{out}(u) = \{v_1, \ldots, v_l\}$, where $v = v_i$ for some i, and $l \leq 4$. Let $N_{in}(v) = \{u_1, \ldots, u_r\}$, where $u = u_j$ for some j, and $r \leq 4$. Then (u, v) is encoded by $\sigma_i|\sigma_j$, and we denote its encoding by $en(u, v)$. We call $EN(u, v) \equiv en(u)|en(u, v)|en(v)$ the *representative string* of (u, v) (see Figure 5).

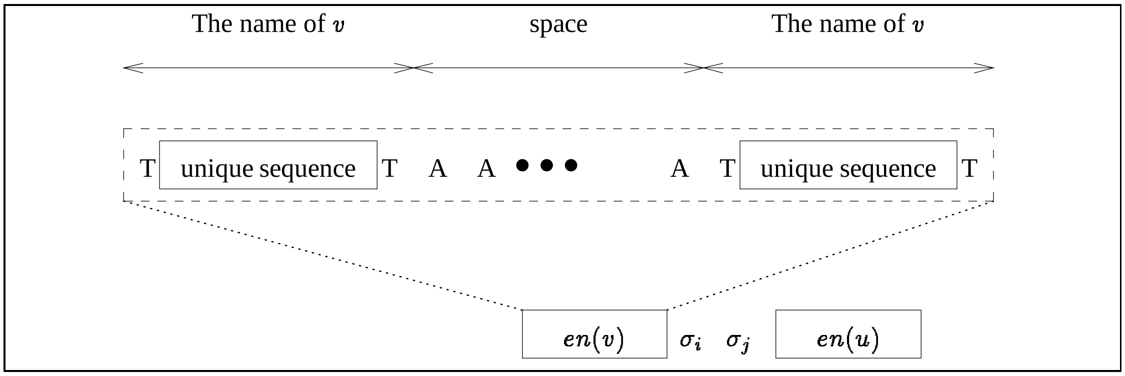

Figure 5: The encoding of vertices and edges into representative strings.

We now construct a 3-PSBH instance, i.e., a spectrum S with position constraints T, as follows: For every edge $(u, v) \in E$ the set S contains all p-long substrings of the $2p$-long string $EN(u, v)$ (c substrings in total). Let $s^i_{(u,v)}$ denote the i-th such substring, $i = 0, \ldots, p$. Let $P((u, v)) = \{t_1, \ldots, t_l\}$, $1 \leq l \leq 3$, be the set of allowed positions for (u, v). Then we set $T(s^i_{(u,v)}) = \{c(t_1 - 1) + i, \ldots, c(t_l - 1) + i\}$ for all i (note that substring positions are numbered starting at zero).

Lemma 9 *Each of the p-long substrings in S is unique.*

Proof: Suppose that $s^i_{(u,v)} = s^j_{(w,z)}$. By construction, either $s^i_{(u,v)}$ contains a complete space, or begins with a (possibly empty) suffix of a space. The position of the space in the first case, or the length of the suffix in the second case, uniquely determines i, which implies that $i = j$. Furthermore, $s^i_{(u,v)}$ contains a name of a vertex plus a unique symbol identifying an edge entering or leaving that vertex, implying that $(u, v) = (w, z)$. ■

We now show validity of the reduction.

$\Leftarrow$ Suppose that $\pi = (v_0, v_1), (v_1, v_2), \ldots, (v_{m-1}, v_m)$ is a solution of the (3,4)-PEP instance. We claim that $X = en(v_0)|en(v_0, v_1)|en(v_1)|en(v_1, v_2)|en(v_2)| \ldots |en(v_{m-1})|en(v_{m-1}, v_m)|en(v_m)$ is a solution of the 3-PSBH instance. By Lemma 9, each p-long substring of X occurs exactly once in X. As π visits all edges in D, we have that S is the p-spectrum of X. The fact that position constraints are satisfied follows directly from the construction.

$\Rightarrow$ Let X be a solution of the 3-PSBH instance. Consider the m substrings of length p, whose starting positions in X are integer multiples of c. By the position constraints, the r-th such substring is an encoding of some vertex v_r, followed by a symbol σ_{i_r}. Denote by w_r the i_r-th out-neighbor of v_r. We prove that $\pi = (v_1, w_1), \ldots, (v_m, w_m)$ is an Eulerian path compliant with (D, P).

Since each string in the p-spectrum of X is unique, π is a permutation of the edges in D. To prove that π is a path in D we have to show that $w_r = v_{r+1}$ for $r = 1, \ldots, m-1$. Let x be the p-long substring of X starting at position $(r-1)c + 2k$. We observe that x must begin with the last k symbols of $en(v_r)$, which compose $name(v_r)$, followed by σ_{i_r}, some symbol, and the first $2k - 1$ symbols of $en(v_{r+1})$, which contain $name(v_{r+1})$. The uniqueness of $name(v_r)$, $name(v_{r+1})$ and the index i_r among the out-neighbors of v_r, implies that $w_r = v_{r+1}$. The claim now follows, since position constraints are trivially satisfied by π.

■

6 Interval PSBH is NP-Complete

In this section we study the complexity of Interval PSBH. In [Hannenhalli et al., 1996] it was proved that the related Interval PEP problem is NP-complete, but the complexity of Interval PSBH was left open. We show below that this problem is NP-complete.

Theorem 10 *The Interval PSBH problem is NP-complete, even if all sets of allowed positions are intervals of equal length.*

Proof: The Interval PEP problem is NP-complete, even if each vertex has in-degree and out-degree at most two, and the sets of allowed positions are intervals of equal length [Hannenhalli et al., 1996]. We reduce this restriction of Interval PEP to Interval PSBH, analogously to the reduction used in the proof of Theorem 8. In the following, we use the same notation as in the proof of Theorem 8.

Let $(D = (V, E), P)$ be an instance of Interval PEP, with each vertex having in-degree and out-degree at most two. Recall, that $EN(u, v)$ denotes a $2p$-long representative string of an edge $(u, v) \in E$ (see Figure 5). We now construct an Interval PSBH instance (S, P'). The spectrum S is the same as in Theorem 8, containing the p-long substrings $s^i_{(u,v)}$ of $EN(u, v)$, for each $(u, v) \in E$. Note, that this construction of S requires that the in-degree and out-degree of each vertex in D will be at most four, a condition which is satisfied by the above restriction of Interval PEP. The set of allowed positions for each substring is defined as follows. Let (u, v) be an edge of E, and suppose that $P((u, v)) = [t_{low}, t_{high}]$. Then, for all i, we set the interval of allowed positions $P'(s^i_{(u,v)})$ to $[c(t_{low} - 1) + i, c(t_{high} - 1) + i]$.

It is enough to show that if X solves the Interval PSBH instance (S, P'), then there is an Eulerian path compliant with (D, P). The proof of the other direction follows immediately from that of Theorem 8.

We first claim that for any $(u, v) \in E$ the string $EN(u, v)$ occurs exactly once along X. Let r be a $(p-1)$-long substring of $EN(u, v)$, which is neither a prefix, nor a suffix of $EN(u, v)$. By construction of $EN(u, v)$, either r contains the name of u and an identification of the outgoing edge to v, or r contains the name of v and an identification of the incoming edge from u. Therefore, similarly to the proof of Lemma 9, we have that r occurs only once in $EN(u, v)$, and does not occur in any other representative string. As $s^0_{(u,v)}, \ldots, s^p_{(u,v)}$ are the p-long substrings of $EN(u, v)$, there exists a unique index $i \in \{0, \ldots, p-1\}$ such that r is a suffix of $s^i_{(u,v)}$ and a prefix of $s^{i+1}_{(u,v)}$. Since X is a valid solution to the Interval PSBH instance, it contains exactly one copy $X_l \ldots X_{l+p-1}$ of $s^i_{(u,v)}$. Since $i < p$, the positional constraints P' assure that X is not terminated at X_{l+p-1}. Hence, $X_{l+1} \ldots X_{l+p}$ is a spectrum element whose prefix is r, and therefore, must be a copy of $s^{i+1}_{(u,v)}$. We conclude, that for each $0 \leq i \leq p-1$, the occurrence of $s^i_{(u,v)}$ is immediately followed by the occurrence of $s^{i+1}_{(u,v)}$. The claim follows.

Let $EN(u_1, v_1), \ldots, EN(u_m, v_m)$ be the order of occurrence of representative strings along X. For every $1 \leq j < m$, $EN(u_j, v_j)$ and $EN(u_{j+1}, v_{j+1})$ do not share any p-long substring (see Lemma 9). Consequently, their overlap along X is of length at most $p-1$. But $|X| = m(p+1) + p - 1$, implying that for all $1 \leq j < m$, $EN(u_j, v_j)$ and $EN(u_{j+1}, v_{j+1})$ have a $(p-1)$-long overlap. Hence, $en(v_j) = en(u_{j+1})$ and $v_j = u_{j+1}$. Therefore, denoting $v_0 = u_1$, the string X has the form

$$X = en(v_0)|en(v_0, v_1)|en(v_1)| \ldots |en(v_{m-1})|en(v_{m-1}, v_m)|en(v_m)$$

It is easy to see, that $\pi = (v_0, v_1), \ldots, (v_{m-1}, v_m)$ is an Eulerian path compliant with (D, P).

■

7 A Parameterized Algorithm for Interval PSBH

Hannenhalli et al. have given a linear-time algorithm to solve the parametric version of the Interval PEP (and hence, the Interval PSBH) problem, where the parameter k is an upper bound on the sizes of the intervals of allowed positions for each edge [Hannenhalli et al., 1996]. We provide a faster algorithm for the problem, which uses the same basic idea as in [Hannenhalli et al., 1996], and runs in $O(mk^{1.5}4^k)$ time, improving upon the $O(mk^3 \log k 4^k)$ time bound of [Hannenhalli et al., 1996].

For simplicity, we shall describe our algorithm when all allowed intervals have length exactly k. Let $(D = (V, E), P)$ be the input positional graph, where $P(e) = [l_e, l_e + k - 1]$ for every $e \in E$. For every $1 \leq i \leq m$ define $\underline{i} \equiv \max\{i - k + 1, 1\}$ and $\bar{i} \equiv \min\{i + k - 1, m\}$. For every $1 \leq i \leq m$ define $\delta_i \equiv \{e \in E : l_e = i\}$ and $\Delta_i \equiv \{e \in E : i \in P(e)\}$. That is, Δ_i is the set of edges for which position i is allowed.

The naive approach to the problem would be to try all possible Eulerian paths in D, and test for each one whether it is compliant with P. However, the number of Eulerian paths in D might be exponential (in m). Instead, we iteratively construct for every $i = 1, \ldots, m+1$, a list Φ_i of pairs (v, S), such that v is the last vertex of some path of length $i - 1$, and S is the list of edges that may extend the path from v. The size of each list Φ_i can be at most exponential in k, and this leads to the improved bound. The algorithm is summarized in Figure 6.

```
Compute $\delta_i$ for all $i$.
If for some $i$, $|\delta_i| > k$ then return False.
Initialize $\Phi_1 \leftarrow \{(v, \delta_1) : \exists w, (v, w) \in \delta_1\}$, $\Phi_{m+1} \leftarrow \emptyset$.
Initialize $\Delta_1 \leftarrow \delta_1$, $i \leftarrow 1$.
While $\Phi_i \neq \emptyset$ and $i \leq m$ do:
    $i \leftarrow i + 1$.
    $\Delta_i \leftarrow \Delta_{i-1} \cup \delta_i \setminus \delta_{\underline{i}-1}$.
    If $|\Delta_i| \geq 2k$ then return False.
    $\Phi_i \leftarrow \{(w, S \cup \delta_i \setminus \{(v, w)\}) : (v, S) \in \Phi_{i-1}, (v, w) \in S, S \cap \delta_{\underline{i}-1} \subseteq \{(v, w)\}\}$.
If $\Phi_{m+1} \neq \emptyset$ then return True.
Else return False.
```

Figure 6: An algorithm for Interval PEP.

The following lemma, which is mentioned in [Hannenhalli et al., 1996], is crucial for the analysis of the algorithm. The subsequent theorem proves the correctness of the algorithm.

Lemma 11 *If (D, P) has a compliant Eulerian path, then $|\Delta_i| \leq 2k - 1$ for all i.*

Proof: Let π be an Eulerian path compliant with (D, P). Then $\Delta_i \subseteq \{\pi(\underline{i}), \ldots, \pi(\bar{i})\}$ for all i. ■

Theorem 12 *The algorithm returns* True *if and only if there is an Eulerian path compliant with (D, P).*

Proof: Let us call a pair (w, S) *i-valid* if there exists a path $\{w_1, \ldots, w_i = w\}$ in D such that:

1. $j \in P((w_j, w_{j+1}))$ for all $1 \leq j < i$.

2. $S = \Delta_i \setminus \{(w_1, w_2), \ldots, (w_{i-1}, w_i)\}$.

3. $\bigcup_{j=1}^{i-1} \delta_j \subseteq \{(w_1, w_2), \ldots, (w_{i-1}, w_i)\}$.

Intuitively, (1) ensures that all edges in the path occupy allowed positions, (2) ensures that the next (i-th) edge is both allowed for position i, and was not used already, and finally, (3) ensures that any edge that had to be used before position i, was indeed used. We first prove by induction on i, that Φ_i is the set of all i-valid pairs. The case $i = 1$ is obvious. Suppose the claim holds for $i - 1$. By the induction hypothesis, Φ_{i-1} is the set of all $(i-1)$-valid pairs. By definition, $\Phi_i = \{(w, S \cup \delta_i \setminus \{(v, w)\}) : (v, S) \in \Phi_{i-1}, (v, w) \in S, S \cap \delta_{\underline{i}-1} \subseteq \{(v, w)\}\}$. The claim follows.

If π is an Eulerian path compliant with (D, P), then $\pi(i) \in \Delta_i$ for all $1 \leq i \leq m$, and therefore, $(v, \emptyset)$ is an $(m+1)$-valid pair for the vertex v at which π ends. On the other hand, if $(v, \emptyset)$ is an $(m+1)$-valid pair, then there exists a path in D which traverses all edges and satisfies the positional constraints. ■

We now analyze the complexity of the algorithm. Define $\Psi_i \equiv \{S : \exists w, (w, S) \in \Phi_i\}$ and $V_i \equiv \{v : \exists S, (v, S) \in \Phi_i\}$.

Lemma 13 *For each i there exists a constant s_i, such that all sets $S \in \Psi_i$ satisfy $|S| = s_i$.*

Proof: The proof is by induction on i. Correctness for $i = 1$ is obvious. Suppose the claim holds for $i - 1$. Then by definition, for each $S \in \Psi_i$, there exists a set $S' \in \Psi_{i-1}$ such that $|S| = |S'| + |\delta_i| - 1$. Hence, $s_i = s_{i-1} + |\delta_i| - 1$, which proves the claim for i. ■

Corollary 3 *For all i, $|\Psi_i| = O(\frac{4^k}{\sqrt{k}})$.*

Proof: Every set $S \in \Psi_i$ satisfies $S \subseteq \Delta_i$. Hence, using Lemma 11 $|\Psi_i| \leq \binom{|\Delta_i|}{s_i} \leq \binom{2k-1}{k} = O(\frac{4^k}{\sqrt{k}})$. ■

Theorem 14 *The algorithm can be implemented in $O(mk^{1.5}4^k)$ time and $O(m + k^{1.5}4^k)$ space.*

Proof: The initialization of δ_i for all i takes $O(m)$ time and space, since $\sum_{i=1}^{m} |\delta_i| = m$. We sort all edges in E according to their first allowed position, breaking ties arbitrarily, and renumber the edges $e_1, \ldots, e_m$ in this order. This takes $O(m)$ time using bucket sort. We also compute in $O(m)$ time the numbers $n(e_i) \equiv i \pmod{2k-1}$, for each $e_i \in E$. Hereafter, we assume that $|\Delta_i| < 2k$ for all i, as otherwise the algorithm aborts returning a negative answer. A key property of the numbering $n(\cdot)$ is that for each Δ_i, if $e, e' \in \Delta_i$ then $n(e) \neq n(e')$. This follows since the edges in Δ_i must appear contiguously in the order $e_1, \ldots, e_m$, and $|\Delta_i| \leq 2k - 1$.

We use the following data structure to keep Δ_i and Φ_i:

- Let $v_1, \ldots, v_n$ be an arbitrary ordering $\prec$ of the vertices in V. Δ_i is kept as a linked list of edges, sorted lexicographically such that $(x_1, x_2) < (y_1, y_2)$ if and only if $x_2 \prec y_2$, or $x_2 = y_2$ and $x_1 \prec y_1$.

- For each $v \in V_i$ we keep a linked list $L_i(v)$ containing all sets $S \in \Psi_i$ such that $(v, S) \in \Phi_i$. We represent a subset $S \subseteq \Delta_i = \{e^i_1, \ldots, e^i_{|\Delta_i|}\}$ by a $(2k-1)$-bit vector $c_i(S)$, whose $n(e^i_j)$-th bit is one if and only if $e^i_j \in S$.

The initialization of Φ_1 requires constructing $O(k)$ linked lists, each containing the $(2k-1)$-bit vector $c_1(\Delta_1)$. This takes $O(k^2)$ time. The computation (and sorting) of Δ_{i+1} from Δ_i takes $O(k \log k)$ time.

The construction of Φ_{i+1} from Φ_i can be done as follows: We consider the edges in Δ_i one at a time according to their lexicographic order. Recall, that these edges are sorted so that edges ending at the same vertex occur consecutively. Thus, for each $w \in V_{i+1}$ we can consecutively traverse all edges in Δ_i that enter w. Note, that we do not know V_{i+1} exactly, but only use the simple fact that $V_{i+1} \subseteq \{w : \exists v, (v, w) \in \Delta_i\}$.

We now construct the linked lists $L_{i+1}(w)$ for every $w \in V_{i+1}$. For each $(v, w) \in \Delta_i$ and for each $S' \in L_i(v)$ we add $S = S' \cup \delta_{i+1} \setminus \{(v, w)\}$ to $L_{i+1}(w)$ if and only if $(v, w) \in S'$ and $S' \cap \delta_{\underline{i}} \subseteq \{(v, w)\}$. In order to add S, we compute the bit vector $c_{i+1}(S)$ in $O(k)$ time. To prevent generating duplicate pairs (w, S) in $L_{i+1}(w)$, we keep an auxiliary 2^{2k-1}-bit vector B which is initialized to zero in the beginning of the algorithm (in $O(4^k)$ time). We add $c_{i+1}(S)$ to $L_{i+1}(w)$ if and only if $B_{c_{i+1}(S)} = 0$, and in this case we also set $B_{c_{i+1}(S)} = 1$. When we finish constructing $L_{i+1}(w)$, we traverse its elements and reset the corresponding bits in B to zero. If no set was added to any linked list during the iteration, we declare that the graph has no compliant Eulerian path and halt. Since $|L_i(v)| = O(|\Psi_i|)$ for each $v \in V_i$, Φ_{i+1} can be computed in $O(|\Delta_i| \cdot |\Psi_i| \cdot k)$ time. By Lemma 11 and Corollary 3, this amounts to $O(k^{1.5}4^k)$ time.

We conclude that the time complexity of the algorithm is $O(m + k^2 + 4^k + mk \log k + mk^{1.5}4^k) = O(mk^{1.5}4^k)$. To save space, we reuse the storage area of Φ_i and Δ_i once Φ_{i+1} and Δ_{i+1} have been computed. Hence, the space complexity of the algorithm is $O(m + k^{1.5}4^k)$. ■

We note, that the algorithm can be modified to generate also one (or all) of the compliant Eulerian paths, e.g., by keeping a record of the last edge(s) leading to each i-valid pair. We also note, that our algorithm can be extended to handle the PEP problem even when the sets of allowed positions for each edge need only be subsets of fixed-length intervals.

Acknowledgments

A. Ben-Dor was supported by the Program for Mathematics and Molecular Biology. I. Pe'er was supported by the Clore Foundation scholarship. R. Shamir was supported in part by a grant from the Ministry of Science, Israel. R. Sharan was supported by an Eshkol fellowship from the Ministry of Science, Israel.

References

[Adleman, 1998] Adleman, L. M. (1998). Location sensitive sequecing of DNA. Technical report, University of Southern California.

[Apsvall et al., 1979] Apsvall, B., Plass, M. F., and Tarjan, R. (1979). A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123.

[Arratia et al., 1997] Arratia, R., Martin, D., Reinert, G., and Waterman, M. S. (1997). Poisson process approximation for sequence repeats, and sequencing by hybridization. *Journal of Computational Biology*, 3(3):425–463.

[Bains and Smith, 1988] Bains, W. and Smith, G. C. (1988). A novel method for nucleic acid sequence determination. *J. Theor. Biology*, 135:303–307.

[Ben-Dor et al., 1999] Ben-Dor, A., Pe'er, I., Shamir, R., and Sharan, R. (1999). On the complexity of positional sequencing by hybridization. In Crochemore, M. and Paterson, M., editors, *Proceedings of the 10th Annual Symposium on Combinatorial Pattern Matching*, volume 1645 of *Lecture Notes in Computer Science*, pages 88–100, Warwick University, UK. Springer-Verlag, Berlin.

[Broude et al., 1994] Broude, S. D., Sano, T., Smith, C. S., and Cantor, C. R. (1994). Enhanced DNA sequencing by hybridization. *Proc. Nat. Acad. Sci. USA*, 91:3072–3076.

[Drmanac and Crkvenjakov, 1987] Drmanac, R. and Crkvenjakov, R. (1987). Yugoslav Patent Application 570.

[Dyer et al., 1994] Dyer, M., Frieze, A., and Suen, S. (1994). The probability of unique solution of sequencing by hybridization. *Journal of Computational Biology*, 1:105–110.

[Gusfield et al., 1998] Gusfield, D., Karp, R., Wang, L., and Stelling, P. (1998). Graph traversals, genes and matroids: An efficient case of the travelling salesman problem. *Discrete Applied Mathematics*, 88:167–180.

[Hannenhalli et al., 1996] Hannenhalli, S., Pevzner, P., Lewis, H., and Skiena, S. (1996). Positional sequencing by hybridization. *Computer Applications in the Biosciences*, 12:19–24.

[Khrapko et al., 1989] Khrapko, K. R., Lysov, Y. P., Khorlyn, A. A., Shick, V. V., Florentiev, V. L., and Mirzabekov, A. D. (1989). An oligonucleotide hybridization approach to DNA sequencing. *FEBS letters*, 256:118–122.

[Lysov et al., 1988] Lysov, Y., Floretiev, V., Khorlyn, A., Khrapko, K., Shick, V., and Mirzabekov, A. (1988). DNA sequencing by hybridization with oligonucleotides. *Dokl. Acad. Sci. USSR*, 303:1508–1511.

[Macevics, 1989] Macevics, S. C. (1989). International Patent Application PS US89 04741.

[Pevzner, 1989] Pevzner, P. A. (1989). l-tuple DNA sequencing: computer analysis. *J. Biomol. Struct. Dyn.*, 7:63–73.

[Pevzner and Lipshutz, 1994] Pevzner, P. A. and Lipshutz, R. J. (1994). Towards DNA sequencing chips. In *Symposium on Mathematical Foundations of Computer Science*, pages 143–158. Springer. LNCS vol. 841.

[Pevzner et al., 1991] Pevzner, P. A., Lysov, Y. P., Khrapko, K. R., Belyavsky, A. V., Florentiev, V. L., and Mirzabekov, A. D. (1991). Improved chips for sequencing by hybridization. *J. Biomol. Struct. Dyn.*, 9:399–410.

[Preparata et al., 1999] Preparata, F., Frieze, A., and E., U. (1999). On the power of universal bases in sequencing by hybridization. In *Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB '99)*, pages 295–301.

[Skiena and Sundaram, 1995] Skiena, S. S. and Sundaram, G. (1995). Reconstructing strings from substrings. *J. Comput. Biol.*, 2:333–353.

[Southern, 1988] Southern, E. (1988). UK Patent Application GB8810400.

[Southern, 1996] Southern, E. M. (1996). DNA chips: analysing sequence by hybridization to oligonucleotides on a large scale. *Trends in Genetics*, 12:110–115.

[Southern et al., 1992] Southern, E. M., Maskos, U., and Elder, J. K. (1992). Analyzing and comparing nucleic acid sequences by hybridization to arrays of oligonucleotides: evaluation using experimental models. *Genomics*, 13:1008–1017.