Hardness and Approximation of The Asynchronous Border Minimization Problem*

Cindy Y. Li[1] Alexandru Popa[2,3] Prudence W.H. Wong[4] Fencol C.C. Yung[4]

[1] NHS Blood and Transplantation, National Health Service (NHS), Bristol, UK
Ying.Li@nhsbt.nhs.uk
[2] Department of Computer Science, University of Bucharest, Bucharest, Romania
[3] National Institute for Research and Development in Informatics, Bucharest, Romania
alexandru.popa@fmi.unibuc.ro
[4] Department of Computer Science, University of Liverpool, Liverpool, UK
pwong@liverpool.ac.uk, ccyung@graduate.hku.hk

May 3, 2017

Abstract

We study a combinatorial problem arising from the microarray synthesis. The objective of the Border Minimization Problem (BMP) is to place a set of sequences in the array and to find an embedding of these sequences into a common supersequence such that the sum of the "border length" is minimized. A variant of the problem, called P-BMP, is that the placement is given and the concern is simply to find the embedding.

An exponential time algorithm has been proposed for the problem but it is unknown whether the problem is NP-hard or not. In this paper, we give a comprehensive study of different variations of BMP by presenting NP-hardness proofs and approximation algorithms. We show that BMP, P-BMP, and 1D-BMP are all NP-hard and 1D-P-BMP is polynomial time solvable. The interesting implications include (i) the BMP is NP-hard regardless of the dimension (1D or 2D) of the array; (ii) the array dimension differentiates the complexity of the P-BMP; and (iii) for 1D array, whether placement is given differentiates the complexity of the BMP. Another contribution of the paper is devising approximation algorithms, and in particular, we present a randomized approximation algorithm for BMP with approximation ratio $O(n^{1/4} \log^2 n)$, where n is the total number of sequences.

1 Introduction

In this paper, we study an optimization problem called (asynchronous) border minimization problem (BMP), arising from a biological problem of microarray synthesis. We first describe the

*Preliminary versions of the paper appeared in "Approximating Border Length for DNA Microarray Synthesis" in Proceedings of the 5th Annual Conference on Theory and Applications of Models of Computation, 2008, pp. 410–422 and "Hardness and Approximation of The Asynchronous Border Minimization Problem" in Proceedings of the 9th Annual Conference on Theory and Applications of Models of Computation, 2012, pp. 164–176.

BMP (formal definition is given in Section 2) and then explain its relation with the biological problem. The input is a set of sequences $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$. We want to find a common supersequence $\mathcal{D}$ of $\mathcal{S}$ and an embedding ε_i for each sequence s_i into $\mathcal{D}$, where ε_i is obtained by inserting spaces into s_i up to length $|\mathcal{D}|$ with the constraint that the j-th position of ε_i is either the character at the j-th position of $\mathcal{D}$ or a space. The border length of s_i with respect to s_j is the number of non-space positions of ε_i that are different from ε_j. We then have to "place" the sequences into a $\sqrt{n} \times \sqrt{n}$ array such that the total border length is minimized (the total border length is the sum of the border length between every two sequences that are neighbors in the array). We study the complexity of BMP and give approximation algorithms.

Motivation. DNA and peptide microarrays [8,13] are important research tools used in gene discovery, multi-virus discovery, disease and cancer diagnosis. Apart from measuring the amount of gene expression [29], microarrays are an efficient tool for making a qualitative statement about the presence or absence of biological target sequences in a sample, e.g., peptide microarrays are used for detecting tumor biomarkers [6,25,31]. Microarray design raises a number of challenging combinatorial problems, such as probe selection [17,23,30], deposition sequence design [20,26] and probe placement and synthesis [3–5,15,18,19].

A microarray is a plastic or glass slide consisting of thousands of sequences called *probes*. The synthesis process [12] consists of two components: *probe placement* and *probe embedding*. In the probe placement the goal is to place each probe to a unique array cell. In the probe embedding we want to find a common supersequence of all sequences, called the *deposition sequence*, and a sequence of 2D arrays, called *masks*. The cells of a mask can be either opaque or transparent allowing the deposition of the character associated with the mask. For any cell, concatenating the characters for which the cell is transparent has to be the same as the probe in that cell of the microarray. See Figure 1(a) for an example. The embedding of a probe placed in a cell c is a sequence in which the ith character is "$-$" if cell c is opaque in the ith mask, or the ith character of the deposition sequence if transparent (see Figure 1(b)).

Due to diffraction, the cells on the *border* between the masked and the unmasked regions are often subject to unintended illumination [12], and can compromise experimental results. As the microarray chip is expensive to synthesize, unintended illumination should be minimized. The magnitude of unintended illumination can be measured by the *border length* of the masks used, which is the number of borders shared between masked and unmasked regions, e.g., in Figure 1(a), the border length of $\mathcal{M}_1, \mathcal{M}_3, \mathcal{M}_4$ is 2 and $\mathcal{M}_2$ is 4. Note that the sum of the border length of all the masks is the same as the sum of border length as defined by the corresponding embedding (c.f. the first paragraph).

In this paper we study the asynchronous synthesis where a mask may deposit a character to different positions of different probes. For example, in Figure 1(a), we want to synthesize the microarray with the four sequences AC, TA, CT, CA in the respective cells as shown in the left hand side. The right is four masks $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$ and $\mathcal{M}_4$, where $\mathcal{M}_1$ deposits the character C and there are two transparent cells at the bottom row and two opaque cells at the top of $\mathcal{M}_1$, and so on. This sequence of masks shows an asychronous synthesis because $\mathcal{M}_2$ deposits the character T to the second position of the sequence CT and the first position of TA (different positions of different probes). On the other hand, in synchronous synthesis, each deposition character can only be deposited to the i-th position of the probes for a particular i. The synchronous variant of the problem was first studied [15]. For this problem, if the placement is fixed, the border length is unique and is proportional to the Hamming distance of neighboring

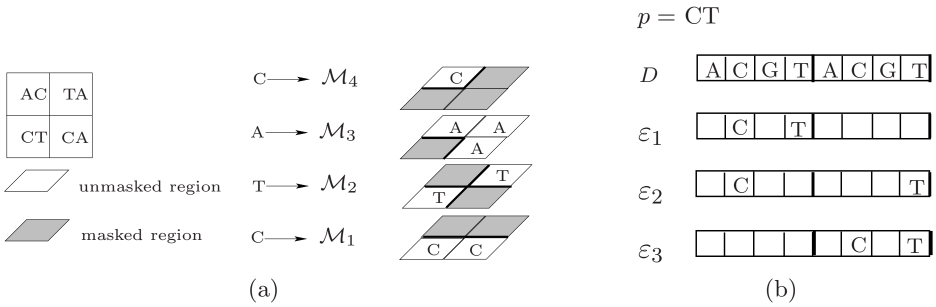


Figure 1: (a) Asynchronous synthesis of a 2×2 microarray with four input sequences AC, TA, CT, CA in the four respective cells (left). The deposition sequence $\mathcal{D}$ = CTAC corresponds to the sequence of four masks $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, and $\mathcal{M}_4$ (right). The corresponding embeddings are $--$AC, $-$TA$-$, CT$--$, and C$-$A$-$. The masked regions are shaded. The borders between the masked and unmasked regions are represented by bold lines. (b) Different embeddings of the sequence s = CT into deposition sequence $\mathcal{D} = (\text{ACGT})^2$.

probes. Thus the only problem is the placement of the probes. The synchronous version is NP-hard [21], $O(\sqrt{n})$-approximable [22] and there are also some experimental results [4,18,19]. Notice that the NP-hardness of the synchronous BMP [21] does not imply that asynchronous BMP—the problem that we study—is NP-hard.

Previous work on asynchronous BMP. The Asynchronous Border Minimization Problem (BMP) was introduced by Kahng et al. [18]. The problem appears to be difficult as they studied a special case in which the deposition sequence is given and the embeddings of all but one probes are known. A polynomial time dynamic programming algorithm was proposed to compute the optimal embedding of this single probe. This algorithm is used as the basis for several heuristics [3–5,18,19] that are shown experimentally to reduce unintended illumination. The dynamic programming [18] computes the optimal embedding of a single probe in time $O(\ell|\mathcal{D}|)$, where ℓ is the length of a probe and $\mathcal{D}$ is the deposition sequence. The algorithm can be extended to an exponential time algorithm to find the optimal embedding of all n probes in $O(2^n \ell^n |\mathcal{D}|)$ time. It is however unknown whether the general problem is NP-hard or not. This naturally raises a number of questions. Let us denote by P-BMP the problem with placement already given.

- Is the BMP NP-hard?
- If BMP is NP-hard, can we derive approximation algorithms for the problem?
- Does the complexity of the problem change for some restricted cases? e.g., when the placement is given or when the array is one-dimensional.

Our contributions. We give a comprehensive study of different variations of the asynchronous border minimization problem. We answer the above questions by giving several NP-hardness proofs and approximation algorithms. Our contributions are listed below (see Table 1 also):

1. We show that P-BMP is NP-hard via a reduction from the Shortest Common Supersequence problem [27]. On the other hand, we show that 1D-P-BMP is polynomial time solvable. This means that the dimension differentiates the complexity of P-BMP.

2. We show that 1D-BMP (placement not given) is NP-hard, via a reduction from the Hamming Traveling Salesman Problem [9]. This result implies that when the array is one dimensional, whether placement is given differentiates the complexity of BMP as 1D-P-BMP is polynomial time solvable.

3. We also show that 1D-BMP can be reduced to BMP, i.e. BMP is NP-hard. This means that BMP is NP-hard regardless of the dimension of the array.

4. For the variants that are NP-hard, we devise approximation algorithms:
   - a randomized $(\log n)$-approximate algorithm for P-BMP;
   - a randomized $(n^{\frac{1}{4}} \log^2 n)$-approximate algorithm for BMP; and
   - a deterministic 3/2-approximate algorithm for 1D-BMP.

5. Furthermore, we show that BMP can be defined as a maximization problem, namely, the agreement maximization problem (AMP), with the objective to maximize the "agreement" among sequences. We mention that a polynomial time exact algorithm for one problem implies a polynomial time exact algorithm for the other problem. However, from the approximation point of view, the problems are totally different. In particular, we are able to devise $O(1)$-approximation algorithms for AMP regardless of whether the placement is given in advance or not.

   This is not uncommon in the design of approximation algorithms. Consider, for example, the minimum vertex cover and the maximum independent set problems. The minimum vertex cover in a graph $G = (V, E)$ is equal to $|V|$ minus the size of the maximum independent set. Thus, a polynomial time exact algorithm for the minimum vertex cover implies a polynomial time exact algorithm for the maximum independent set. However, the approximability of the two problems is different.

We note that the reductions for (1) and (2) work for constant alphabet size. An interesting implication of (1) is that with placement already given, the synchronous problem [15] is trivial as the border length equals the Hamming distance. Nevertheless, the asynchronous problem is NP-hard. This indicates that the difficulty of the asynchronous problem is due to both the asynchronicity and the need to find a placement. Furthermore, our approximation algorithm also gives an $O(n^{\frac{1}{4}})$ approximation for the synchronous problem.

Technically speaking, the results for (1) and (4) are more challenging. The reduction for the NP-hardness proof of P-BMP proves that the Shortest Common Supersequence problem on binary alphabets can be solved with polynomially many calls to P-BMP. As for the approximation algorithm for BMP, it is more tricky to find a good placement. Our idea is to define a metric and use the randomized algorithm in [10] for "embedding" the metric into a tree distribution. This is a crucial step, since in this way we can control both the border length on the rows and the border length on the columns. Another idea is to use an embedding in other metrics (e.g. Euclidean), but it is not at all clear how this can yield a better approximation algorithm.

Setting	2D	1D
BMP	NP-hard $O(n^{1/4}\log^2 n)$-approximate	NP-hard $\frac{3}{2}$-approximate
P-BMP	NP-hard $O(\log n)$-approximate	polynomial time solvable

Table 1: Results on BMP and P-BMP.

Organization of the paper. The rest of the paper is organized as follows. Section 2 gives formal problem definitions and preliminaries. Section 3 discusses the P-BMP in both one- and two-dimension. Sections 4 and 5 give the NP-hardness proofs and approximations for the BMP, respectively. In Section 6, we present approximations for the AMP. We conclude in Section 7.

2 Notation and preliminaries

In this section we define some notations (Section 2.1), give the formal definitions of the problems BMP and P-BMP (Section 2.2) and discuss several problems that are closely related to our problems (Section 2.3).

2.1 Notations

The input to the problems is a set of n sequences $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$ and a $\sqrt{n} \times \sqrt{n}$ array, where $\sqrt{n}$ is an integer. For any sequence s_i, we write ℓ_i for the length of s_i and $s_i[t]$ for the t-th character of s_i.

We use the notion of *subsequence* and *supersequence*, which is defined formally below.

Definition 1. *A sequence $a = a_1a_2\ldots a_n$ is a* subsequence *of a sequence $b = b_1b_2\ldots b_m$ if there exist n integers $1 \le i_1 < i_2 < \cdots < i_n \le m$ such that $a_1 = b_{i_1}, a_2 = b_{i_2}, \ldots, a_n = b_{i_n}$. In this case, we say that b is a supersequence of a.*

We give definitions for *deposition sequence* and *mask*, which we use to introduce the embedding of a set of sequences and finally to define the problems BMP and P-BMP.

Definition 2. *A* deposition sequence *$\mathcal{D}$ is a supersequence of all the input sequences (the characters of $\mathcal{D}$ are named deposition characters).*

Definition 3. *A* mask *$\mathcal{M}$ is a 2D-array associated to each character of the deposition sequence such that $\mathcal{M}(i,j)$ is either the character associated with $\mathcal{M}$ (i.e. the cell is transparent) or a space "−" (i.e. the cell is opaque).*

Figure 1(a) shows a deposition sequence CTAC and the four corresponding masks.

Placement and embedding. A placement is to map each sequence to a unique cell in the array and an embedding of a sequence describes the relationship between the sequence and the deposition sequence. We give the formal definitions as follows.

Definition 4. *A* placement *of a set of sequences $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ to a $\sqrt{n} \times \sqrt{n}$ array is a bijective function $\phi : \mathcal{S} \to \{1, 2, \ldots, \sqrt{n}\} \times \{1, 2, \ldots, \sqrt{n}\}$.*

For example, in Figure 1(a) the sequence AC is placed on the position $(1, 1)$, TA on $(1, 2)$, CT on $(2, 1)$ and CA on $(2, 2)$.

Definition 5. *An* embedding $\varepsilon = \{\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n\}$ *of a set of sequences* $\mathcal{S} = \{s_1, s_2, \dots, s_n\}$ *into a deposition sequence* $\mathcal{D}$ *is a set of* n *length-*$|\mathcal{D}|$ *sequences such that for all* $1 \leq i \leq n$*:*

1. $\varepsilon_i[t]$ *is either* $\mathcal{D}[t]$ *or a space "* $-$ *"; and*
2. *Removing all spaces from* ε_i *gives* s_i*.*

For example, referring to Figure 1(a), the embeddings of the four sequences AC, TA, CT, and CA into the deposition sequence CTAC are respectively $--$AC, $-$TA$-$, CT$--$ and C$-$A$-$.

Border length. Before we define the border length we formally state what the neighbor of a cell in an array is (informally, the neighbors of a cell are the four cells on the top, right, bottom and left).

Definition 6. *Given an* $\sqrt{n} \times \sqrt{n}$ *array, two cells* $v_1 = (x_1, y_1)$ *and* $v_2 = (x_2, y_2)$ *are said to be* neighbors *if* $|x_1 - x_2| + |y_1 - y_2| = 1$*. For each cell* v*, we denote the set of neighbors of* v *by* $\mathcal{N}(v)$*.*

We use two alternative definitions for border length, one in terms of masks and the other in terms of embeddings. The latter definition is useful when we want to analyze the contribution to the total border length of two adjacent strings in the microarray. We use the first definition, which is more intuitive and perhaps easier to understand for the reader, when we analyze the contribution of the total border length of a certain character—this is often useful in the hardness reduction.

In the first definition the border length is the sum over all masks of the border between the transparent and opaque regions.

Definition 7. *For any mask* $\mathcal{M}$ *of deposition character* X*, the* border length *of* $\mathcal{M}$*, denoted by* $\mathrm{BL}(\mathcal{M})$*, is defined as the number of neighboring cells* (i_1, j_1) *and* (i_2, j_2) *such that* $\mathcal{M}(i_1, j_1) = X$ *and* $\mathcal{M}(i_1, j_1) \neq \mathcal{M}(i_2, j_2)$*.*

For example, referring to Figure 1(a), $\mathrm{BL}(\mathcal{M}_1) = \mathrm{BL}(\mathcal{M}_3) = \mathrm{BL}(\mathcal{M}_4) = 2$ and $\mathrm{BL}(\mathcal{M}_2) = 4$.

Definition 8. *For a placement* ϕ *and embedding* ε *that corresponds to a sequence of masks* $\mathcal{M}_1$*,* $\mathcal{M}_2$*,* $\cdots$*,* $\mathcal{M}_d$*, we define the* border length $\mathrm{BL}(\phi, \varepsilon)$ *to be:*

$$\mathrm{BL}(\phi, \varepsilon) = \sum_{h=1}^{d} \mathrm{BL}(\mathcal{M}_h)$$

Alternatively, we can define border length in terms of the embeddings of the sequences into the deposition sequence.

Definition 9. *Consider an embedding* ε*. For two sequences* s_i *and* s_j*, the border length of* s_i *with respect to* s_j*, denoted by* $\mathrm{border}_\varepsilon(s_i, s_j)$*, is the number of positions* p*'s of the embedding* ε_i *such that* $\varepsilon_i[p] \neq$ *"$-$" and* $\varepsilon_j[p] =$ *"$-$" and the* share *of* s_i *and* s_j*, denoted by* $\mathrm{share}_\varepsilon(s_i, s_j)$*, is the number of positions* p*'s of the embedding* ε_i *such that* $\varepsilon_i[p] \neq$ *"$-$" and* $\varepsilon_j[p] \neq$ *"$-$".*

By definition, $\text{border}_\varepsilon(s_i, s_j) + \text{share}_\varepsilon(s_i, s_j) = \ell_i$. Note that border() is an asymmetric measure, i.e., $\text{border}_\varepsilon(s_i, s_j) \neq \text{border}_\varepsilon(s_j, s_i)$, because the length of s_i and s_j may differ. On the other hand, share() is symmetric, i.e., $\text{share}_\varepsilon(s_i, s_j) = \text{share}_\varepsilon(s_j, s_i)$, Furthermore, the Hamming distance between ε_i and ε_j equals to $\text{border}_\varepsilon(s_i, s_j) + \text{border}_\varepsilon(s_j, s_i)$. Suppose s_1 = ATT and s_2 = CT, $\mathcal{D}$ = ATCT, $\varepsilon_1 = \text{AT} - \text{T}$ and $\varepsilon_2 = - - \text{CT}$. Then $\text{border}(s_1, s_2) = 2$, $\text{border}(s_2, s_1) = 1$, $\text{share}(s_1, s_2) = \text{share}(s_2, s_1) = 1$, and the Hamming distance between ε_1 and ε_2 is 3.

Definition 10. *The* border length *of a placement ϕ and an embedding ε is defined as the sum of borders over all pairs of neighboring sequences*

$$\text{BL}(\phi, \varepsilon) = \sum_{\substack{s_i, s_j :\\ \phi(s_j) \in \mathcal{N}(\phi(s_i))}} \text{border}_\varepsilon(s_i, s_j) \,.$$

Agreement. In Definition 9, we define the counterpart of border length, namely, the share. The sum of share is defined as the agreement.

Definition 11. *The* agreement *of a placement ϕ and an embedding ε is defined as the sum of shares over all neighboring sequences*

$$\text{A}(\phi, \varepsilon) = \sum_{\substack{s_i, s_j :\\ \phi(s_j) \in \mathcal{N}(\phi(s_i))}} \text{share}_\varepsilon(s_i, s_j)$$

In the example in Figure 1(a), $\text{border}_\varepsilon(\text{AC}, \text{CT}) = \text{border}_\varepsilon(\text{CT}, \text{AC}) = 2$ and the border length of all other pairs of neighbors is 1, while $\text{share}_\varepsilon(\text{AC}, \text{CT}) = \text{share}_\varepsilon(\text{CT}, \text{AC}) = 1$ and the share of all the rest is 2. Furthermore, $\text{BL}(\phi, \varepsilon) = 10$, and $\text{A}(\phi, \varepsilon) = 6$,

We say that an algorithm is a c-approximation for a minimization (maximization) problem if the value of the solution returned by the algorithm is less than c (more than $1/c$) times the value of the optimal solution.

2.2 Problem definitions

Using the previous definitions we present the *border minimization problem.*

Problem 1 (BMP)**.** *Given a set of sequences over an alphabet Σ, $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ and a $\sqrt{n} \times \sqrt{n}$ array, the objective of* BMP *is to find a placement ϕ and an embedding ε so that $\text{BL}(\phi, \varepsilon)$ is minimized.*

P-BMP is the variant of BMP when the placement is given.

Problem 2 (P-BMP)**.** *Given a set of sequences over an alphabet Σ, $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, a $\sqrt{n} \times \sqrt{n}$ array and a placement ϕ, the objective of* P-BMP *is to find an embedding ε so that $\text{BL}(\phi, \varepsilon)$ is minimized.*

1D-BMP and 1D-P-BMP are the variants of BMP and P-BMP, respectively, when the array is one dimensional.

Problem 3 (1D-BMP and 1D-P-BMP)**.** *Given a set of sequences over an alphabet* Σ*,* $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ *and a* $1 \times n$ *array, the objective of* 1D-BMP *is to find a placement* ϕ *and an embedding* ε *so that* $\mathrm{BL}(\phi, \varepsilon)$ *is minimized. Given a set of sequences over an alphabet* Σ*,* $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$*, a* $1 \times n$ *array and a placement* ϕ*, the objective of* 1D-P-BMP *is to find an embedding* ε *so that* $\mathrm{BL}(\phi, \varepsilon)$ *is minimized.*

We further define the problem AMP, which is the counterpart of BMP.

Problem 4 (AMP)**.** *Given a set of sequences over an alphabet* Σ*,* $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ *and a* $\sqrt{n} \times \sqrt{n}$ *array, the objective of the* Agreement Maximization Problem *(*AMP*) is to find a placement* ϕ *and an embedding* ε*, so that* $\mathrm{A}(\phi, \varepsilon)$ *is maximized.*

Note that if all sequences have the same length ℓ, then $\mathrm{A}(\phi, \varepsilon) = 4\ell(n - \sqrt{n}) - \mathrm{BL}(\phi, \varepsilon)$. In this case, minimizing the border length $\mathrm{BL}(\phi, \varepsilon)$ is equivalent to maximizing the agreement $\mathrm{A}(\phi, \varepsilon)$.

2.3 Related problems

In this section we discuss several problems that are related to the BMP. Roughly speaking, the length of common subsequence of two sequences gives us a lower bound on the border length of the two sequences (to be used in Section 5); the approximation algorithm for the weighted multiple sequence alignment problem gives us an approximation algorithm for P-BMP, where the former is obtained via an approximation for the minimum routing cost tree problem (to be used in Section 3.2).

Common subsequence and common supersequence. The border length is closely related to the common subsequence and common supersequence between neighboring sequences in the placement. Consider any two sequences p and q of length ℓ_p and ℓ_q, respectively. We denote the longest common subsequence and shortest common supersequence of two sequences p and q by $LCS(p, q)$ and $SCS(p, q)$, respectively, and the corresponding length as $lcs(p, q)$ and $scs(p, q)$, respectively. $SCS(p, q)$ can be obtained by finding $LCS(p, q)$ and inserting into p the characters in q that are not in $LCS(p, q)$ while preserving the order in q. Therefore, $scs(p, q) = \ell_p + \ell_q - lcs(p, q)$. For any embedding ε, the maximum number of common deposition characters between p and q is $lcs(p, q)$, in other words, $\mathrm{border}_\varepsilon(p, q) + \mathrm{border}_\varepsilon(q, p) \geq \ell_p + \ell_q - 2 \cdot lcs(p, q)$ and $\mathrm{share}_\varepsilon(p, q) + \mathrm{share}_\varepsilon(q, p) \leq 2 \cdot lcs(p, q)$. We define the *LCS distance* to be $\ell_p + \ell_q - 2 \cdot lcs(p, q)$, denoted by $\mathrm{dist}(p, q)$. In other words, $\mathrm{dist}(p, q)$ is a lower bound of $\mathrm{border}_\varepsilon(p, q) + \mathrm{border}_\varepsilon(q, p)$ for any embedding ε.

Note that the embeddings needed to achieve $\mathrm{dist}(p, q)$ may not be compatible with each other in a particular placement. For example, for the placement ϕ in Figure 1, $\mathrm{dist}(p, q) = 2$ for every neighboring pair p and q, and so the sum is 8. Yet the minimum border length is 10 as shown in the figure.

Multiple sequence alignment (MSA) and Weighted MSA (WMSA). As we will see in Section 3.2, the variant of BMP problem, namely P-BMP (BMP in which the placement is given), is polynomial time reducible to WMSA. As a consequence, we can apply the approximation results on WMSA to P-BMP, which we can further use as a building block for the approximation for BMP. We first review the MSA and WMSA problems. MSA and WMSA have been studied extensively [2, 11, 14, 28]. Let Σ be the set of characters and $S = \{S_1, S_2, \ldots, S_k\}$ be a set of k sequences, with maximum length m, over Σ. An alignment of S is a matrix $S' = (S'_1, S'_2, \ldots, S'_k)$

such that $|S'_i| = m'$ and S'_i is formed by inserting spaces into S_i. For a given distance function $\delta(a, b)$ where $a, b \in \Sigma \cup \{-\}$, the *pairwise score* of S'_i and S'_j is defined as $\sum_{1 \le y \le m'} \delta(S'_i[y], S'_j[y])$. Given a weight function $w(i, j)$ for the pair of sequences S_i and S_j, the *weighted sum-of-pair* (SP) score $\mathrm{SP}(S', w) = \frac{1}{2} \sum_{1 \le i,j \le k} w(i, j) \sum_{1 \le y \le m'} \delta(S'_i[y], S'_j[y])$. The WMSA problem is to find an alignment S' such that $\mathrm{SP}(S', w)$ is minimized. WMSA has been proved to be NP-complete and there is a randomized algorithm with approximation ratio $O(\log n)$ [1, 10, 32], via a reduction to the minimum routing cost tree problem (MRCT), see Lemma 1.

Minimum routing cost tree problem (MRCT). In this problem, a graph with weighted edges is given. For a spanning tree of the graph, the *routing cost* between two vertices is the sum of weights of the edges on the unique path between the two vertices in the spanning tree. The *routing cost* of the spanning tree is defined as the sum of routing cost between every pair of two vertices. The MRCT problem is to find a spanning tree whose routing cost is minimum. The results in [1, 32] imply that there is a polynomial time reduction from WMSA to MRCT. Each sequence in the input of WMSA corresponds to a vertex in the input graph of MRCT. The edge weight between two vertices is set to be the weighted edit distance between the two corresponding sequences. The reduction result states that (1) there is a routing spanning tree T whose routing cost is at most $O(\log^2 n)$ times $\sum_{i,j} w(i, j)d(i, j)$, where $d(i, j)$ is the edit distance between the two sequences i and j; and (2) there is an alignment S' whose $\mathrm{SP}(S', w)$ is at most the routing cost of T. Note that $\sum_{i,j} w(i, j)d(i, j)$ is a lower bound on the weighted SP score. The approximation ratio of the algorithm for the WMSA problem is given by the distortion of $O(\log^2 n)$ of the metric embedding algorithm[1] of Bartal [1]. Fakcharoenphol, Rao and Talwar [10] show a similar algorithm with distortion $O(\log n)$. Thus, we have the following lemma.

Lemma 1 ([10, 32]). *There is a randomized algorithm that is $O(\log n)$-approximate for the* WMSA *problem, where n is the number of sequences to be aligned.*

3 P-BMP: Finding embeddings when placement is given

We first prove the NP-hardness of P-BMP in Section 3.1 and then give an approximation algorithm in Section 3.2. In Section 3.3, we show that when the array is one-dimension, 1D-P-BMP is polynomial time solvable.

3.1 Hardness of P-BMP

We show the NP-hardness of P-BMP via a reduction from the shortest common supersequence (SCS) problem over a binary alphabet to the P-BMP. Note that the SCS problem is NP-hard [27]. Suppose that the binary alphabet is $\{0, 1\}$. Consider an instance of the SCS problem with a set S of k binary strings $s_1, \cdots, s_k$. Let ℓ_i be the length of s_i, $\ell = \max_{1 \le i \le k} \ell_i$ be the length of the longest string in S, and $L = \sum_{1 \le i \le k} \ell_i$. For any $1 \le p, q \le \ell$, we define an instance of P-BMP, denoted by $I(p, q)$. As we show later, a shortest common supersequence can be found by computing the optimal solutions for a polynomial number of instances $I(p, q)$.

[1]Notice that we use the term "embedding" in two contexts: probe embedding refers to finding the deposition sequence while embedding a metric into a tree distribution is used in the approximation algorithm. The meaning of the term "embedding" should be clear from the context and should not cause confusion.

\$	\$	\$	\$	\$	\$	\$
000	000	000	000	000	000	000
\$	010	\$	100	\$	00	\$
111	111	111	111	111	111	111
\$	\$	\$	\$	\$	\$	\$
\$	\$	\$	\$	\$	\$	\$
\$	\$	\$	\$	\$	\$	\$

Table 2: s_1 = "010", s_2 = "100", s_3 ="00". The supersequence $\mathcal{D}$ = "010011" is an optimal deposition sequence for $I(3,3)$. Ignoring the border of the mask for the dummy strings "\$", the optimal border length equals $2(p^* + q^*) \times (2k+1) + 2L = 100$, where $p^* = q^* = k = 3$ and $L = 8$.

The input $I(p,q)$. We construct a $(2k+1) \times (2k+1)$ array. The sequences are over the alphabet $\{0, 1, \$\}$, where \$ is a character different from 0 or 1.

- Except for row 2-4, each cell of row 1, 5, 6, 7, 8, $\cdots$ of the array contains the string "\$". We call these rows *dummy-rows*.
- All the cells of row 2 contain the same string "0^p". We call this row *all-0-row*.
- All the cells of row 4 contain the same string "1^q". We call this row *all-1-row*.
- Row 3 contains s_1, s_2, $\cdots$, s_k in alternate cells, and the rest of the cells contain the string "\$", precisely, row 3 contains "\$", s_1, "\$", s_2, "\$", $\cdots$, "\$", s_k, "\$". We call this row *seq-row*.

Tables 2 and 3 show examples of $I(3,3)$ and $I(1,1)$, respectively.

Common supersequence and deposition sequence. Consider an instance $I(p,q)$, we need at least one mask for the dummy strings "\$", and the best is to use exactly one mask, say $M_\$$ for all these strings. For $M_\$$, row 1 (dummy-row) incurs a border length of $2k+1$ on the bottom boundary with all-0-row, and row 5 (dummy-row) incurs $2k+1$ on the top boundary with all-1-row. For seq-row, the border length on top boundary with all-0-row is $k+1$, on bottom boundary with all-1-row is also $k+1$, and within seq-row on left and right boundaries is $2k$. Therefore, the border length $\mathrm{BL}(M_\$) = 4(2k+1)$. The total border length for $I(p,q)$ equals to $\mathrm{BL}(M_\$)$ plus the border length of the remaining deposition sequence, which in turn is related to a common supersequence of the strings in S. We ignore the quantity $\mathrm{BL}(M_\$)$ when we discuss the border length for $I(p,q)$, since $\mathrm{BL}(M_\$)$ is present in all the embeddings. The following lemma states the relationship between a common supersequence and an embedding of the sequences. Table 2 gives an example.

Lemma 2. *If $\mathcal{D}$ is a common supersequence of the strings in S and the number of 0's and 1's in $\mathcal{D}$ is p^* and q^*, respectively, then $\mathcal{D}$ is an optimal deposition sequence for $I(p^*,q^*)$ and the resulting optimal embedding has a border length of $2(p^*+q^*)(2k+1)+2L$.*

Proof. First of all, it is easy to observe that $\mathcal{D}$ is a deposition sequence for $I(p^*,q^*)$ since it is a common supersequence of the strings in S and has the same number of 0's and 1's in all-0-row

$	$	$	$	$	$	$
0	0	0	0	0	0	0
$	010	$	100	$	00	$
1	1	1	1	1	1	1
$	$	$	$	$	$	$
$	$	$	$	$	$	$
$	$	$	$	$	$	$

Table 3: $s_1 =$ "010", $s_2 =$ "100", $s_3 =$"00". The shortest common supersequence is $\mathcal{D} =$ "0100". The optimal deposition sequence for $I(1,1)$ is $\mathcal{D}$. Ignoring the border of the mask for the dummy strings "$", the optimal border length equals to $(2\times 7+2\times 7+2\times 3+2\times 2)+2\times 8 = 54$ (the first four terms refer to border length with top and bottom boundaries and the last term with left and right boundaries). On the other hand, $2(p^*+q^*)\times(2k+1)+2L = 44 < 54$, where $p^* = q^* = 1$, $k = 3$ and $L = 8$.

and all-1-row of the array in $I(p^*, q^*)$, respectively. Notice that p^* is at least the number of 0's in each of s_i and similarly q^* is at least the number of 1's. In the deposition sequence $\mathcal{D}$, when $\mathcal{D}[j] = 0$, all-0-row incurs a border length of $2k+1$ on the top boundary with row 1 (dummy-row); all-0-row and seq-row together incur a border length of $2k+1$ on the bottom boundary; and a border length of $2x$ within seq-row, where x is the number of cells on seq-row that 0 is deposited. A similar calculation can be done for the case when $\mathcal{D}[j] = 1$. As a whole, the total border length equals $2(p^*+q^*)(2k+1)+2L$.

We further argue that this is the minimum border length for $I(p^*, q^*)$. In any deposition sequence, the number of 0's is at least p^* and the number of 1's is at least q^*. Therefore, all-0-row and the cells with '0' on seq-row together incur a border length at least $2p^*(2k+1)$, and similarly, all-1-row and the cells with '1' on seq-row incur at least $2q^*(2k+1)$. The cell on seq-row which contains the string s_i incurs $2\ell_i$ on the left and right boundaries, implying all these cells together incur $2L$. Therefore, no matter how we deposit characters to the cell, the total border length is at least $2(p^*+q^*)(2k+1)+2L$. □

Lemma 2 implies that if $p+q$ is large enough, then there is a formula for the optimal border length of the instance $I(p,q)$ in terms of p, q, and L. The following lemma considers the situation when $p+q$ is small. Table 3 gives an example.

Lemma 3. *If $\mathcal{D}$ is a shortest common supersequence of the strings in S and the number of 0's and 1's in $\mathcal{D}$ is p^* and q^*, respectively, then for any p_1, q_1 such that $p_1+q_1 < p^*+q^*$, the optimal embedding for $I(p_1,q_1)$ has a border length greater than $2(p_1+q_1)(2k+1)+2L$.*

Proof. Notice that any deposition sequence must be a common supersequence, and thus must have total length $\ell_{\mathcal{D}} \geq p^*+q^* > p_1+q_1$. With this deposition sequence, the border length equals to $2\ell_{\mathcal{D}}k+2(p_1+q_1)(k+1)+2L > 2(p_1+q_1)k+2(p_1+q_1)(k+1)+2L = 2(p_1+q_1)(2k+1)+2L$. The term $2(p_1+q_1)k$ refers to the top and bottom border length for columns with s_i in the seq-row while the term $2(p_1+q_1)(k+1)$ is for columns with dummy string "$" in the seq-row. □

Using Lemmas 2 and 3, we can find the optimal solution for SCS from optimal solutions for P-BMP as follows. For all pairs of $1 \leq p \leq \ell$ and $1 \leq q \leq \ell$, we find the optimal solution to

$I(p,q)$. If the border length of the optimal solution equals to $2(p+q)(2k+1)+2L$, there is a common supersequence of length $p+q$. Among all such pairs of p and q, those with the minimum $p+q$ correspond to shortest common supersequences. Notice that there are a polynomial number of, precisely ℓ^2, pairs of p and q to be checked. We then have the following theorem.

Theorem 4. *The* P-BMP *is NP-hard.*

3.2 Approximation algorithm for P-BMP

We show that P-BMP is $O(\log n)$-approximable by giving a reduction to the weighted multiple sequence alignment problem (WMSA), for which there is an $O(\log n)$-approximation algorithm (Lemma 1).

Lemma 5. *There is a polynomial time reduction from* P-BMP *to* WMSA*.*

Proof. Let I be an instance of the P-BMP problem with a given placement ϕ. We construct an instance I' for WMSA such that there is a solution for I with border length X if and only if there is a solution for I' with a weighted SP score of X.

Construction of I'. We first show the construction of I'. The input sequence set for WMSA is the same as the input sequence set $\mathcal{S}$. The weight $w(i,j)$ is defined as follows:

$$w(i,j) = \begin{cases} 1 & \text{if } \phi(s_j) \in \mathcal{N}(\phi(s_i)), \\ 0 & \text{otherwise.} \end{cases}$$

The distance function $\delta(a,b)$, for $a,b \in \Sigma \cup \{-\}$, is defined as follows:

$$\delta(a,b) = \begin{cases} 0 & \text{if } a = b, \\ 1 & \text{if } a \neq b \text{ and } (a = \text{“}-\text{”} \text{ or } b = \text{“}-\text{”}), \\ \infty & \text{otherwise.} \end{cases}$$

Solution for I implies solution for I'. Suppose we have an embedding ε for I. Note that $\varepsilon = \{\varepsilon_1 \cdots \varepsilon_n\}$ is an alignment for $\mathcal{S}$ and the pairwise score of ε_i and ε_j equals $\text{border}_\varepsilon(s_i,s_j) + \text{border}_\varepsilon(s_j,s_i)$. So,

$$\begin{aligned} \text{SP}(\mathcal{S}',w) &= \frac{1}{2}\sum_{1\le i,j\le n} w(i,j) \sum_{1\le y\le|\mathcal{D}|} \delta(\varepsilon_i[y],\varepsilon_j[y]) \\ &= \sum_{1\le i,j\le n} w(i,j)\text{border}_\varepsilon(s_i,s_j) \\ &= \sum_{\substack{s_i,s_j\,:\\ \phi(s_j)\,\in\,\mathcal{N}(\phi(s_i))}} \text{border}_\varepsilon(s_i,s_j) \\ &= \text{BL}(\phi,\varepsilon)\ . \end{aligned}$$

The second last equality is due to the definition of $w(i,j)$, which is based on ϕ.

Solution for I' implies solution for I. On the other hand, suppose we have a solution for I', i.e., an alignment $\mathcal{S}' = (s'_1 \cdots s'_n)$ for $\mathcal{S}$ and $|s'_i| = m'$, for some m'. In the alignment $\mathcal{S}'$, each column contains the same character or “ $-$ ” because of the definition of the distance function $\delta(a, b)$. We denote the resulting matrix as $\varepsilon = (\varepsilon_1 \cdots \varepsilon_n)$. It can be seen that ε is an embedding for $\mathcal{S}$ and the Hamming distance between ε_i and ε_j, i.e., $\text{border}_\varepsilon(s_i, s_j) + \text{border}_\varepsilon(s_j, s_i)$, equals the pairwise score of s'_i and s'_j. Then

$$\begin{aligned}
\text{BL}(\phi, \varepsilon) &= \sum_{\substack{s_i, s_j : \\ \phi(s_j) \in \mathcal{N}(\phi(s_i))}} \text{border}_\varepsilon(s_i, s_j) \\
&= \frac{1}{2} \sum_{\substack{s_i, s_j : \\ \phi(s_j) \in \mathcal{N}(\phi(s_i))}} \sum_{1 \le y \le |\mathcal{D}|} \delta(s'_i[y], s'_j[y]) \\
&= \frac{1}{2} \sum_{1 \le i, j \le n} w(i, j) \sum_{1 \le y \le |\mathcal{D}|} \delta(s'_i[y], s'_j[y]) \\
&= \text{SP}(\mathcal{S}', w) \ .
\end{aligned}$$

Note that the second last equality holds for the same reason as above. Therefore, the lemma follows. □

With Lemmas 1 and 5, we have the following corollary.

Corollary 6. *The* P-BMP *is* $O(\log n)$*-approximable.*

3.3 A polynomial time algorithm for 1D-P-BMP

In this section, we study the special case on an 1D array. We mention that, although 1D microarrays are not used in biology, the 1D-P-BMP problem is still interesting from the theoretical perspective. Intuitively, the problem is computationally easier than the 2D case and, indeed, in this section we show that P-BMP on an 1D array can be solved optimally in polynomial time.

The algorithm Embed1D shown in Algorithm 1 makes use of a procedure called Extend. Extend takes two sequences p and q, and a supersequence $\hat{S}$ of p as input and returns a supersequence of $\hat{S}$ and q (recall Section 2.3 for the definitions of LCS and SCS). Let $c = lcs(p, q)$, $x_1, x_2, \ldots, x_c$ be the indices of $\hat{S}$ corresponding to p that belongs to $LCS(p, q)$, and $y_1, y_2, \ldots, y_c$ be the indices of q that belongs to $LCS(p, q)$. Extend then extends $\hat{S}$ by inserting characters in q but not in $LCS(p, q)$: characters between $q[y_{k-1}]$ and $q[y_k]$ are inserted right before $\hat{S}[x_k]$ and characters beyond $q[y_c]$ are appended to the end of $\hat{S}$. Extend keeps track of the indices of the new $\hat{S}$ that correspond to q (see Figure 2).

Algorithm 1 Embed1D: Optimal embedding for P-BMP on 1D array.

Input: Sequence set $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, placed on a 1D array in that order.
Output: An embedding ε with minimum border length.

1: Set $\mathcal{D} = s_1$.
2: For each $i > 1$, call the procedure Extend with s_{i-1}, s_i and $\mathcal{D}$ as the input to obtain a new $\mathcal{D}$.
3: For each s_i, set ε_i such that $\varepsilon[y] = \mathcal{D}[y]$ if $\mathcal{D}[y]$ corresponds to a character in s_i kept track by Extend, and $\varepsilon[y] =$ “ $-$ ” otherwise.

Theorem 7. *Algorithm* Embed1D *finds an optimal embedding for the P-BMP problem on 1D array in polynomial time.*

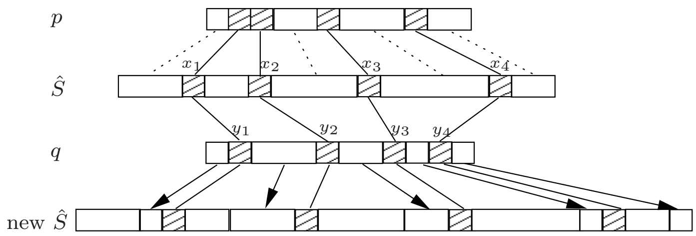


Figure 2: An illustration of EXTEND. Shaded squares refer to characters in $LCS(p, q)$. Characters in q but not in $LCS(p, q)$ are inserted into $\hat{S}$ so that the order preserves as in q (see the arrows).

Proof. We first observe that $\mathcal{D}$ constructed in each iteration by EXTEND is a common supersequence of $s_1, \ldots, s_i$. This is clear from the way EXTEND finds $LCS(s_{i-1}, s_i)$ and inserts characters into $\mathcal{D}$. It also implies that the number of characters shared by s_{i-1} and s_i is maintained as $lcs(s_{i-1}, s_i)$, which is the maximum possible. Note that this property does not change by later steps. Hence, the border length of the final embedding is the minimum. As for time complexity, the bottleneck is finding the longest common subsequences of two sequences, which is known to take polynomial time [16,24]. This is done for $n-1$ times only. Therefore, EMBED1D also takes polynomial time. $\square$

4 Hardness of BMP: Finding placement and embedding

In Section 4.1, we show the NP-hardness of 1D-BMP via a reduction from a variant of the Hamming Traveling Salesman Problem, termed *Hamming a-b Path TSP* in which we want to find a path from vertex a to vertex b (not a tour, as it is the case in the original Hamming TSP), that visits each vertex precisely once. Then in Section 4.2 we give a reduction from 1D-BMP to BMP, thus proving the NP-hardness of BMP regardless of the dimension of the array.

4.1 1D-BMP: BMP on a 1D array

The Hamming *a-b* Path TSP. The input consists of a set of length-ℓ strings $s_1, s_2, \ldots s_n$ over the alphabet $\{0, 1\}$ and two strings a and b. We denote by $ham(s_i, s_j)$ the Hamming distance between s_i and s_j (i.e., the number of positions at which s_i and s_j differ). The goal is to find a permutation $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ such that $s_{\pi(1)} = a$, $s_{\pi(n)} = b$ and the sum $\sum_{i=1}^{n-1} ham(s_{\pi(i)}, s_{\pi(i+1)})$ is minimized.

In the Hamming TSP problem, the goal is to minimize the sum $\sum_{i=1}^{n-1} ham(s_{\pi(i)}, s_{\pi(i+1)}) + ham(s_{\pi(1)}, s_{\pi(n)})$, i.e., unlike the Hamming *a-b* Path TSP, in the Hamming TSP the goal is to find a closed tour instead of a path. Ernvall et al. [9] proved that the Hamming TSP problem is NP-hard, via a reduction from the Rectilinear TSP, i.e., TSP where the vertices are points in the plane and the distance is the L_1 norm (Manhattan distance). The NP-hardness theorem in [9] implies the NP-hardness of the Hamming a-b Path TSP as stated in Corollary 8.

Corollary 8. *The Hamming a-b Path TSP problem is NP-hard.*

Proof. We show a polynomial time reduction from the Hamming a-b Path TSP to Hamming TSP. For a given graph, compute the minimum a-b TSP path, between any two vertices a and b. The optimum Hamming TSP tour is the one such that the cost of the a-b path, plus the cost of the edge between a and b is minimized. Thus, if we could solve the Hamming a-b Path TSP problem in polynomial time, then we could have solved Hamming TSP problem in polynomial time by checking $O(n^2)$ pairs of a and b and the corresponding a-b-paths. □

Reduction. Consider an Hamming a-b Path TSP instance with n binary strings $s_1, s_2, \dots s_n$. We construct an instance of 1D-BMP with n sequences to be placed on an array of size $1 \times n$. We now define the alphabet Σ and the sequences $\mathcal{S}$.

1. Alphabet: $\Sigma = \{0, 1, \$, \#, *\}$, where \$ is a special character which serves as a delimiter and # and $*$ are characters that appear only in the strings s and t .
2. Sequences: for each string $s = x_1 x_2 \dots x_\ell$ in the Hamming TSP instance, where $x_i \in \{0, 1\}$, we construct the sequence $s' = x_1 \$^{2n\ell} x_2 \$^{2n\ell} \dots \$^{2n\ell} x_\ell$. Moreover, we append $\#^{2n\ell}$ at the end of the string a' and $*^{2n\ell}$ at the end of b'. The role of the characters # and $*$ is to ensure that the strings corresponding to a and b are placed on the first and respectively last position of the microarray.

The hardness of the 1D-BMP is stated in the following theorem.

Theorem 9. *The* 1D-BMP *is NP-hard if the size of the alphabet is at least* 5.

Proof. We claim that the border length of an optimal placement and embedding is precisely $4n\ell$ plus two times the minimum length of a path in the Hamming a-b TSP path instance.

We prove the direct implication. Given the optimal Hamming a-b TSP path π, we define OPT to be $\sum_{i=1}^{n-1} ham(s_{\pi(i)}, s_{\pi(i+1)})$. Then, we place the strings s' in the array exactly in the order given by π. We now argue that the optimal embedding incurs a border length of exactly $2OPT + 4n\ell$. The optimal sequence of masks synthesizes the block of \$ symbols at the same time (and therefore, the border length of this sequence is 0) and the border length obtained in this case is at most $2n\ell$ (in the case where the adjacent strings differ on all the positions). Otherwise, if we skip a block of \$ characters (i.e. we do not select a mask which synthesizes the entire row of \$'s in one step), then the border length is greater than $2n\ell$ (since each of the $2n\ell$ \$ characters increases the border length by one). Thus, the border length of two consecutive strings s'_i and s'_j is two times their Hamming distance (on each position they differ, the two masks that synthesize s'_i and s'_j increase the border length by 1) and the optimal embedding incurs a border length of exactly $2OPT$. The extra $4n\ell$ in the total border length is incurred by the characters # and $*$ appended to a and b, respectively.

To prove the reverse implication we show two facts. First, observe that except for the two characters $*$ and #, the embedding of any placement gives a border length which two times the sum of the Hamming distance between consecutive sequences. This embedding synthesizes all the \$ characters in the same step and thus the cost for these characters is 0. Since the strings have length ℓ, we can obtain an embedding with cost at most $2n\ell$. The embedding of the characters $*$ and # may add either $4n\ell$, $6n\ell$, or $8n\ell$ to the total border length. Assume now, by contradiction that two characters on different positions are synthesized in the same step, and thus, the embedding doesn't correspond to two times the sum of the Hamming distance between consecutive sequences. In this case, the synthesis of the \$ sign, incurs a border length of at least $2n\ell$ which is not optimal (we obtain a border length of at most $2n\ell$ if we synthesize all the \$ signs in a single block).

Secondly, we argue that the strings a' and b' are placed on the first and the last position of the microarray. We showed in the above paragraph that except for the two characters $*$ and $\#$, the embedding of any placement gives a border length which is at most $2n\ell$. Placing the two strings a' and b' on the first, respectively the last, position adds an extra $4n\ell$ to the border length. Otherwise, the border length is at least $6n\ell$, which is not optimal. Thus, finding the optimal placement is equivalent to finding the optimal Hamming a-b TSP path. The theorem follows. □

4.2 BMP on 2D array

In this section, we reduce the BMP on an $1 \times n$ array to BMP on an $n \times n$ array. Together with the hardness result from the previous subsection, this reduction implies that BMP is NP-hard. Consider an instance I_1 for 1D-BMP where there are n sequences $s_1, s_2, \cdots, s_n$ over an alphabet Σ, and the length of s_i is ℓ_i. Let $\ell = \max_{1 \le i \le n} \ell_i$. We construct an instance I_2 for BMP which contains two types of sequences, namely, a given sequence and a dummy sequence. The alphabet Σ' used by the BMP on the 2D array is a superset of Σ, $\Sigma' = \Sigma \cup \{x_1, x_2, \cdots, x_n\} \cup \{\$\}$, where $\$ \notin \Sigma$ and $x_i \notin \Sigma$, for all $1 \le i \le n$. The instance I_2 is constructed as follows. Let $k > \ell$ be a large integer to be determined later.

- Dummy sequences: we create $n^2 - n$ dummy sequences each containing one character \$.
- Given sequences: for each s_i, we create a length k sequence $x_i^{k-\ell_i} \cdot s_i$.

We claim that in an optimal placement the n sequences are on the top row. In that case, the optimal solution for I_1 would give an optimal solution for I_2 and vice versa.

We now prove the claim. For each cell in the array, there are four boundaries, top, bottom, left, and right. A sequence placed in a certain cell contributes to the overall border length an amount of four times its length minus the sharing of characters with its four neighbors. Recall that share(s, s') denotes the number of characters that can be shared between to sequences s and s'. Let gs, ds, and b be a given sequence, a dummy sequence, and the outmost boundary of the array. Notice that the share between of a sequence and the boundary of the array is always the length of that sequence.

Then we have the following relationships.

$$\begin{aligned} &\text{share}(gs, gs) \le \ell, \qquad \text{share}(gs, ds) = 0, \qquad \text{share}(gs, b) = k, \\ &\text{share}(ds, ds) = 1, \qquad \text{share}(ds, b) = 1 \end{aligned}$$

If we arrange all the sequences such that the given sequences are placed on the top row, we would have a sharing of $(n+2) \times \text{share}(gs, b) = (n+2)k$. If any of these given sequences are not placed on the top row, we lose a sharing of at least k. No matter how the sequences are placed, the maximum sharing apart from those with the outmost boundaries of the array is at most $4n^2\ell$. If we set k to be large enough, e.g., $k = 4n^2\ell + 1$, then any possible internal sharing (not with outmost boundaries) is not sufficient to compensate the loss of k.[2] Intuitively, it is

[2]It is possible to set a smaller value of k by more careful analysis. Yet the ultimate conclusion is still the same that we have an instance for BMP that it is the best to have all the given sequences placed on the top row of the array.

ideal to place the given sequences on the top row since they are the longest. If we place them in a different fashion, then the sharing of the characters in the sequences s_i does not compensate the loss of the sharing between the given sequences and the boundary (this motivates our choice of k).

We have argued that all the given sequences should be placed on the top row of the array and the following theorem follows from Theorem 9.

Theorem 10. *The (two-dimensional)* BMP *is NP-hard.*

5 Approximation algorithms for BMP

We give approximation algorithms for the 2D- and 1D-BMP in Section 5.1 and 5.2, respectively.

5.1 An $O(n^{\frac{1}{4}} \log^2 n)$ approximation algorithm for the BMP

In Section 3.2, we showed that there is an $O(\log n)$-approximation for the P-BMP in which the placement of the sequences is given (Corollary 6). Therefore, to obtain an approximation for the BMP, it suffices to find a "good" placement of the sequences. The algorithm named PLACE&EMBED is given in Algorithm 2.

The intuitive ideas of our approximation algorithm are as follows. Recall that in Section 2.3 we define a distance function $\text{dist}(s_i, s_j)$ for any pair of sequences s_i and s_j, which gives a lower bound on $\text{border}(s_i, s_j) + \text{border}(s_j, s_i)$. A placement can be viewed as a permutation π. We define a function $p(\pi)$ based on $\text{dist}(s_i, s_j)$ and show that $p(\pi)$ is a lower bound on the border length of any embedding (including the optimal one) for the permutation π. Therefore, if we can find an embedding such that the border length is at most a certain factor of $p(\pi)$, then we have an approximation for BMP. We then observe that it is difficult to find in polynomial time a permutation optimizing the value $p(\pi)$ on the general metric and turn to embedding the metric into a tree (distribution) such that (in expectation) the distance on the tree $\text{dist}_T(s_i, s_j)$ satisfies the property $\text{dist}(s_i, s_j) \leq \text{dist}_T(s_i, s_j) \leq O(\log n)\,\text{dist}(s_i, s_j)$. Finally, we show that using an Euler tour on the embedded tree as a permutation to place the sequences on the array gives us an $O(n^{\frac{1}{4}})$ approximation on $p_T(\pi)$, which is the counterpart of $p(\pi)$ with $\text{dist}(\cdot)$ replaced by $\text{dist}_T(\cdot)$. Combining all the arguments, we obtain an $O(n^{\frac{1}{4}} \log^2 n)$ approximation for BMP. Details are as follows.

The function $p(\pi)$. In Section 2.3, we define the notion $\text{dist}(s_i, s_j)$ for any two sequences s_i and s_j of length ℓ_i and ℓ_j. Precisely, $\text{dist}(s_i, s_j) = \ell_i + \ell_j - 2 \cdot lcs(s_i, s_j)$ and it is a lower bound on $\text{border}_\varepsilon(s_i, s_j) + \text{border}_\varepsilon(s_j, s_i)$ for any embedding ε. Therefore, the sum over all neighbors of distances $\text{dist}(s_i, s_j)$ is a lower bound on the optimal border length of a given placement. We observe that this distance $\text{dist}(\cdot)$ is a metric.

A placement can be viewed as a permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that the sequences $\pi(1), \ldots, \pi(\sqrt{n})$ are placed on the first row of the array in this order, $\pi(\sqrt{n} + 1), \ldots, \pi(2\sqrt{n})$ on the second row and so on. Then any embedding for a placement π has a border length at least $p(\pi)$, which is defined as:

$$
\begin{aligned}
p(\pi) &= \sum_{i=1}^{n-1} \mathrm{dist}(\pi(i), \pi(i+1)) - \sum_{i=1}^{\sqrt{n}-1} \mathrm{dist}(\pi(i\sqrt{n}), \pi(i\sqrt{n}+1)) \\
&\quad + \sum_{i=1}^{\sqrt{n}} \sum_{j=1}^{\sqrt{n}-1} \mathrm{dist}(\pi(i+(j-1)\sqrt{n}), \pi(i+j\sqrt{n})) \ .
\end{aligned}
$$

We name the problem to minimize this "proxy" value $p(\pi)$ the PROXY problem. Note that the border length for a placement π can be much larger than $p(\pi)$ as the embeddings needed to achieve $\mathrm{dist}(s_i, s_j)$ for all s_i and s_j may not be compatible with each other. Nevertheless, as shown in the proof of Proposition 11, the P-BMP approximation algorithm returns an embedding with the border length less than $O(\log n)p(\pi)$. Therefore, if we can place the sequences into the array such that the sum of the distances between any neighbors is within a factor c of $p(\pi)$, then we can apply the $O(\log n)$ approximation algorithm for the P-BMP and obtain a $O(c \log n)$ approximation for the BMP. We summarize this in the following proposition.

Proposition 11. *A c-approximation algorithm for the* PROXY *problem implies an $O(c \log n)$-approximation algorithm for the* BMP.

Proof. First, we show that the lower bound for the P-BMP approximation algorithm for a given placement π is precisely $p(\pi)$. More precisely, the $O(\log n)$-approximation algorithm for the P-BMP on instance π will return a solution $ALG(\pi)$ such that $ALG(\pi) \le O(\log n) \cdot p(\pi)$. As discussed in Section 2.3 the $O(\log n)$ approximation algorithm for the P-BMP uses the approximation algorithm for WMSA, which in turn uses the approximation algorithm for MRCT. The lower bound of the approximation algorithm is the sum of the edit distances of adjacent strings in the microarray. Since the edit distance between two strings x and y is precisely $\mathrm{dist}(x, y)$, it follows that this lower bound is precisely $p(\pi)$.

We turn now to the proof of the proposition. Let π' be a placement such that $p(\pi')$ is minimized. A c-approximation algorithm for the PROXY problem returns a placement π'' such that $p(\pi'') \le c \cdot p(\pi')$. Then, $ALG(\pi'') \le O(\log n) \cdot p(\pi'') \le c \cdot O(\log n) \cdot p(\pi')$. Since $p(\pi')$ is a lower bound for the optimum value of the P-BMP problem on the placement π', we have $p(\pi') \le$ P-BMP(π'). Thus, $ALG(\pi'') \le c \cdot O(\log n) \cdot$ P-BMP(π') and the proposition follows. □

Tree embedding and Euler tour to approximate $p(\pi)$. Since it is difficult to find in polynomial time a permutation which optimizes the function p on this general metric, we first embed the metric into a tree (in fact, into a distribution of trees) with $O(\log n)$ distortion using the algorithm of Fakcharoenphol, Rao and Talwar [10] (the same algorithm used in the MRCT, and implicitly P-BMP, approximation). This randomized embedding algorithm takes the input sequences as tree vertices and returns a tree with a metric $\mathrm{dist}_T(\cdot)$ defined by a tree such that in expectation $\mathrm{dist}(s_i, s_j) \le \mathrm{dist}_T(s_i, s_j) \le O(\log n)\,\mathrm{dist}(s_i, s_j)$. The distance $\mathrm{dist}_T(s_i, s_j)$ on the tree is the sum of distances along the unique path between s_i and s_j. Notice that the resulting tree may have vertices in addition to the n input sequences. Using the metric $\mathrm{dist}_T(s_i, s_j)$, we can define a counterpart of $p_T(\pi)$ by replacing $\mathrm{dist}(s_i, s_j)$ with $\mathrm{dist}_T(s_i, s_j)$. Then a c-approximation to $p_T()$ leads to an $O(c \log n)$ approximation to $p()$. Together with Proposition 11, we have the following proposition.

Proposition 12. *If we can approximate the* Proxy *problem on a tree (i.e., approximate p_T) within a factor of c, then we have an $O(c \log^2 n)$ approximation to the* BMP.

We now present how to approximate p_T. Our approximation algorithm for the Proxy problem on trees is very simple: we consider the ordering of the vertices given by an Euler tour of the tree (we ignore the additional vertices which do not correspond to the input sequences). We then prove that this is an $O(n^{\frac{1}{4}})$ approximation algorithm for p_T. Then, by Proposition 12 we are guaranteed to have an $O(n^{\frac{1}{4}} \log^2 n)$ approximation algorithm for the BMP.

The algorithm for the BMP problem is described formally in Algorithm 2.

Algorithm 2 Place&Embed: The $O(n^{\frac{1}{4}} \log^2 n)$ approximation algorithm for the BMP

1: **Input:** The sequences $s_1, s_2, \ldots, s_n$.
2: Define $\mathrm{dist}(s_i, s_j) = \ell_i + \ell_j - 2 \cdot lcs(s_i, s_j)$
3: Embed the metric given by this distance and the set of input points into a tree T using the algorithm from [10].
4: Let $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ be the ordering of the sequences according to an Euler tour of the tree T from which the additional vertices have been removed.
5: Place the sequences in the array according to π: the sequences $\pi(1), \ldots, \pi(\sqrt{n})$ are placed on the first row of the array in this order, $\pi(\sqrt{n}+1), \ldots, \pi(2\sqrt{n})$ on the second row and so on. (See Figure 3).
6: Apply the P-BMP approximation algorithm to find an embedding for the sequences.
7: **Output:** The placement of the sequences on the array based on the Euler tour and the embedding of the sequences given by the P-BMP approximation algorithm.

Analysis. We denote by T the tree obtained after the tree embedding. Notice that the cost of a solution is given by summing each edge of T several times. We say that an edge $(x, y) \in T$ is *crossed* r times in a solution π if it belongs to exactly r of the $2\sqrt{n}(\sqrt{n}-1)$ paths of the solution.

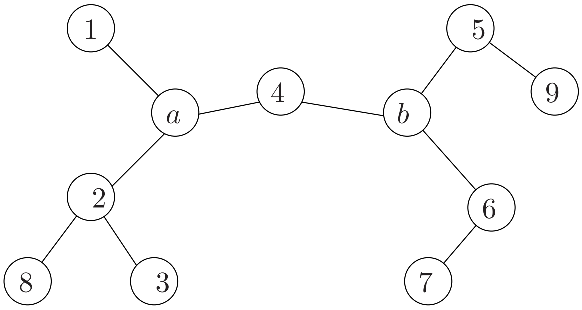
Now, we want to find a lower bound for the optimal solution. We do so, by showing that in any solution, each edge of the tree has to be crossed at least a certain number of times. This is stated formally in the following lemma. Let $(x, y) \in T$ and let A and B be the two connected components resulted from removing (x, y).

Lemma 13. *In any permutation π the edge (x, y) is crossed at least $\sqrt{\min\{|A|, |B|\}}$ times.*

Proof. If we consider an arbitrary placement of the sequences on a grid graph (i.e., an $\sqrt{n} \times \sqrt{n}$ array), then the two sets of sequences A and B determine a cut in the graph. We argue that the size of the cut is exactly the number of times the edge (x, y) is crossed: for each edge $(\pi(i), \pi(j))$ in the cut, we have to add to the solution the corresponding path $\pi(i) \to \pi(j)$. But the path $\pi(i) \to \pi(j)$ has to cross the edge (x, y), since $\pi(i) \in A$ and $\pi(j) \in B$. The minimum cut determined by two sets of size $|A|$ and $|B|$ has size $\sqrt{\min\{|A|, |B|\}}$ and therefore the lemma follows. $\square$

We give an upper bound by considering the ordering of vertices given by an Euler tour of the tree.

Lemma 14. *In an Euler tour ordering, (x, y) is crossed at most $O(\min\{\sqrt{n}, |A|, |B|\})$ times.*



(a)

s_1	s_2	s_8
s_3	s_4	s_5
s_9	s_6	s_7

(b)

Figure 3: (a) Suppose the embedding in [10] returns such a tree for 9 sequences. The vertices that are mapped to input strings are labeled with numbers and the additional vertices introduced by the embedding algorithm are labeled with letters. (b) The placement of these sequences on the array according to an Euler tour of the tree. An Euler tour of the tree is: $1, a, 2, 8, 3, 4, b, 5, 9, 6, 7$. After removing the additional vertices a and b the ordering of n the vertices corresponding to sequences is: $1, 2, 8, 3, 4, 5, 9, 6, 7$.

Proof. Due to the Euler tour, each edge can be crossed by edges from the paths $\pi(i) \to \pi(i+1)$ at most twice. Then we have to count how many edges from the paths $\pi(i) \to \pi(i+\sqrt{n})$ cross the edge (x, y). We argue that (x, y) cannot be crossed more than $4 \min\{|A|, |B|\}$ times. Suppose A is the set with the smaller cardinality. In the worst case for each element in A all its four neighbors are in B and, therefore (x, y) is crossed $4 \cdot \min\{|A|, |B|\}$ (this is actually too pessimistic but this suffices for our analysis since we are not interested in the precise constants).

We also argue that (x, y) cannot be crossed more than $O(\sqrt{n})$ times. Since we follow an Euler tour, for an element $\pi(i)$, we have two cases: either $\pi(i+\sqrt{n}) \in A$, or $\pi(i+\sqrt{n}) \notin A$ and $\pi(j+\sqrt{n}) \notin A, \forall j > i$. Therefore, for only $\sqrt{n}$ elements $\pi(i)$ of A, $\pi(i+\sqrt{n})$ is in B. Then the lemma follows. □

Theorem 15. *The randomized algorithm* PLACE&EMBED *is an $O(n^{\frac{1}{4}} \log^2 n)$-approximation to the* BMP*.*

Proof. By Lemma 14, the cost of the algorithm is at most $O(\min\{\sqrt{n}, |A|, |B|\} \cdot |T|)$. By Lemma 13, the cost of the permutation π is at least $O(\sqrt{\min\{|A|, |B|\}} \cdot |T|)$. Therefore, to analyze the performance of PLACE&EMBED we analyze the ratio of the two multiplicative factors.

Consider the placement of the sequences in the $\sqrt{n} \times \sqrt{n}$ array in the order given by the Euler tour. For an edge $(x, y) \in T$, there are two cases that we have to consider:

1. If $\min\{|A|, |B|\} \leq \sqrt{n}$, then

$$\frac{\min\{\sqrt{n}, |A|, |B|\}}{\sqrt{\min\{|A|, |B|\}}} = \sqrt{\min\{|A|, |B|\}} \leq n^{\frac{1}{4}}$$

2. If $\min\{|A|, |B|\} > \sqrt{n}$, then

$$\frac{\min\{\sqrt{n}, |A|, |B|\}}{\sqrt{\min\{|A|, |B|\}}} < \frac{\sqrt{n}}{n^{\frac{1}{4}}} = n^{\frac{1}{4}}$$

We then apply Proposition 12 and Lemmas 13 and 14 and the theorem follows. □

5.2 A 3/2 approximation algorithm for 1D-BMP

In Section 3.3, we show that when the array is one dimension, 1D-P-BMP is polynomial time solvable. For 1D-BMP, the algorithm PLACE&EMBED1D (shown in Algorithm 3) finds a placement guided by some traveling salesman path (TSP) on a particular graph (to be defined) and then use Algorithm EMBED1D to find the optimal embedding for that placement.

The algorithm PLACE&EMBED1D first constructs the graph G_c, which is a weighted complete graph vertices representing $\mathcal{S}$ and edge weight representing dist() between the two vertices. A traveling salesman path (TSP) $\tilde{Q}$ is obtained from G_c and we place the sequences on the 1D array in the order of $\tilde{Q}$ from left to right. We then apply the embedding algorithm EMBED1D in Section 3.3. We denote the resulting placement and embedding as $\tilde{\phi}$ and $\tilde{\varepsilon}$, respectively.

Algorithm 3 PLACE&EMBED1D: Approximation algorithm for 1D-BMP.

Input: Sequence set $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ to be placed on a $1 \times n$ array.
1: Construct the weighted complete graph G_c.
2: Find an approximate TSP $\tilde{Q}$ for G_c using algorithm in [7].
3: Place the sequences on the array according to the order of $\tilde{Q}$ to obtain placement $\tilde{\phi}$.
4: Run the algorithm EMBED1D to find an optimal embedding $\tilde{\varepsilon}$ for $\tilde{\phi}$.
Output: A placement $\tilde{\phi}$ and an embedding $\tilde{\varepsilon}$ for $\mathcal{S}$.

Theorem 16. *The algorithm* PLACE&EMBED1D *is a* 3/2*-approximation for the* 1D-BMP*.*

Proof. Consider a one-dimensional array. For any placement ϕ, we denote by $\text{dist}(\phi)$ the sum over all neighboring sequences p and q of the value $\text{dist}(p, q)$. The proof of Theorem 7 implies that for any placement ϕ, $\text{dist}(\phi) = \text{BL}(\phi, \tilde{\varepsilon})$, where $\tilde{\varepsilon}$ is found by Algorithm EMBED1D. Therefore, $\text{dist}(\tilde{\phi}) = \text{BL}(\tilde{\phi}, \tilde{\varepsilon})$ and $\text{dist}(\phi^*) = \text{BL}(\phi^*, \varepsilon^*)$, where ϕ^* and ε^* is the optimal placement and optimal embedding. The optimal TSP gives the placement ϕ^*. Therefore, since dist() satisfies the triangle inequality and since there exists a 1.5-approximation algorithm for metric TSP [7], we have that $\text{dist}(\tilde{\phi}) \leq 3\text{dist}(\phi^*)/2$. Therefore, the theorem follows. □

6 The agreement maximization problem (AMP)

In this section, we study the counterpart of BMP, which we called agreement maximization problem (AMP) (recall definition in Section 2). In contrast to BMP, AMP admits constant approximations, whether the placement is given in advance or not.

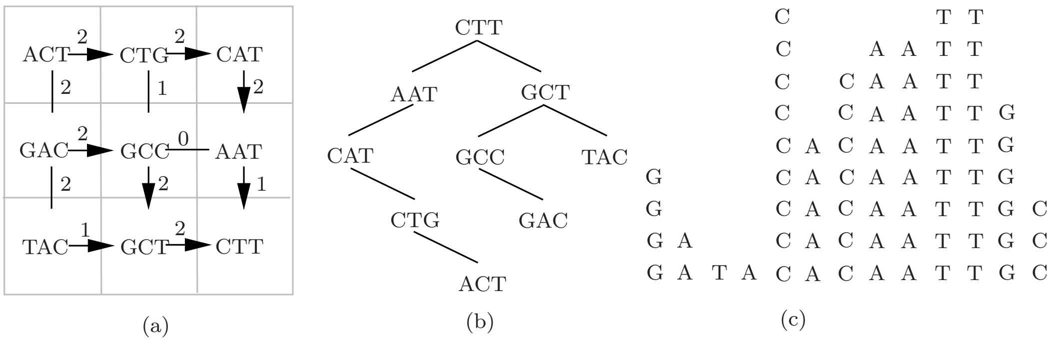


Figure 4: (a) A set of sequences placed on a 3×3 array. The values represent the length of LCS between the two neighboring sequences. An arrow from p to q means $parent(p) = q$. (b) The tree constructed by AEmbed with root CTT. (c) How the deposition sequence $\mathcal{D}$ changes iteratively. The sequences are drawn in a way the characters align with the final $\mathcal{D}$.

6.1 Approximation for P-AMP

We first study the P-AMP, a variant of AMP with a placement already given.

Algorithm AEmbed**.** The algorithm AEmbed (Embed for Agreement) makes use of procedure Extend in Section 3.3. The order of sequences to be considered is determined by a certain tree T with the bottom rightmost sequence in the array being the root. To construct T, for each sequence s, we assign a parent to the sequence, denoted by $parent(s)$. We denote by $r(s)$ and $b(s)$ the right and bottom neighbors of sequence s, respectively. The sequences in the rightmost column has $r(s) = $ NULL and those in the bottommost row has $b(s) = $ NULL. We set $parent(s)$ to $r(s)$ or $b(s)$ depending on whether $lcs(s, r(s))$ or $lcs(s, b(s))$ is larger. A detailed description of AEmbed is shown in Algorithm 4. The embedding found is denoted by $\hat{\varepsilon}$. Figure 4 shows an example.

Algorithm 4 AEmbed: Approximate algorithm for P-AMP.

Input: Sequence set $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ placed on a $\sqrt{n} \times \sqrt{n}$ array according to a placement ϕ.

1: Construct a tree T by assigning parent to each sequence s: if $lcs(s, r(s)) \geq lcs(s, b(s))$ set $parent(s) = r(s)$ else set $parent(s) = b(s)$.
2: Set $\mathcal{D}$ to be the bottom rightmost sequence in the array.
3: Traverse T in a pre-order fashion: for each sequence s traversed, call the procedure Extend with $parent(s)$, s and $\mathcal{D}$ as input.
4: For each s_i, set $\hat{\varepsilon}_i$ such that $\hat{\varepsilon}[y] = \mathcal{D}[y]$ if $\mathcal{D}[y]$ corresponds to a character in s_i kept track by Extend, and $\hat{\varepsilon}[y] =$ " $-$ " otherwise.

Output: The embedding $\hat{\varepsilon}$ for $\mathcal{S}$.

Analysis. To analyze the performance of AEmbed, we first observe that in the final embedding $\hat{\varepsilon}$, the number of characters shared by a sequence and its parent equals to the length of their LCS (by a similar argument as the proof of Theorem 7). We then bound the performance of AEmbed as follows.

Theorem 17. AEmbed *is a polynomial-time* 2*-approximation algorithm for* P-AMP.

Proof. For the given placement ϕ, let ε^* be the optimal embedding. For any embedding ε,

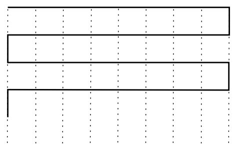


Figure 5: Row-by-row threading of a group of sequences on an array. Solid and dotted edges connect neighbors in the placement that are and are not, respectively, neighbors in the group.

we have $\mathrm{A}(\phi, \varepsilon) = 2 \times \sum_{s \in \mathcal{S}} (\mathrm{share}_\varepsilon(s, r(s)) + \mathrm{share}_\varepsilon(s, b(s)))$. We assume $\mathrm{share}_\varepsilon(s_1, s_2) = 0$ if $s_2 = \mathrm{NULL}$. As mentioned in Section 2.3, for any embedding ε, $\mathrm{share}(s_1, s_2) \le lcs(s_1, s_2)$. Thus, $lcs(s, r(s)) \ge \mathrm{share}_{\varepsilon^*}(s, r(s))$ and $lcs(s, b(s)) \ge \mathrm{share}_{\varepsilon^*}(s, b(s))$. Note that $\mathrm{share}_{\hat{\varepsilon}}(s, parent(s)) = \max\{lcs(s, r(s)), lcs(s, b(s))\} \ge \frac{1}{2}(\mathrm{share}_{\varepsilon^*}(s, r(s)) + \mathrm{share}_{\varepsilon^*}(s, b(s)))$. Therefore,

$$\mathrm{A}(\phi, \hat{\varepsilon}) \ge 2 \times \sum_{s \in \mathcal{S}} \mathrm{share}_{\hat{\varepsilon}}(s, parent(s)) \ge \sum_{s \in \mathcal{S}} (\mathrm{share}_{\varepsilon^*}(s, r(s)) + \mathrm{share}_{\varepsilon^*}(s, b(s))) = \frac{1}{2}\mathrm{A}(\phi, \varepsilon^*) \ .$$

Finally, AEmbed runs in polynomial time as the bottleneck is finding LCS between two sequences. □

6.2 Approximation for AMP

In this section, we study the general AMP problem to find both the placement and the embedding to maximize the agreement. We consider the case when the size of the alphabet is 4, since this is the most relevant in biological applications. We prove that the algorithm APlace&Embed as shown in Algorithm 5 has an asymptotic approximation ratio of 4 when all the sequences have the same length (for general alphabets, the algorithm has approximation ratio $|\Sigma|$).

Algorithm 5 APlace&Embed: Approximation algorithm for AMP.

Input: Sequence set $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ to be placed on a $\sqrt{n} \times \sqrt{n}$ array.

1: Partition $\mathcal{S}$ into four disjoint groups $\mathcal{A}$, $\mathcal{C}$, $\mathcal{G}$ and $\mathcal{T}$: a sequence belongs to $\mathcal{A}$ if the number of A in the sequence is the maximum over the number of other characters (similarly for $\mathcal{C}$, $\mathcal{G}$ and $\mathcal{T}$).

2: "Thread" the sequences in group $\mathcal{A}$ on the array in a row-by-row fashion, followed by threading of sequences in $\mathcal{C}$, $\mathcal{G}$, and $\mathcal{T}$ to form the placement $\check{\phi}$. See Figure 5 for the way threading works.

3: For sequences in $\mathcal{A}$, align them such that the maximum number of A are aligned while different characters are not aligned. This forms a partial embedding $\check{\varepsilon}_a$ with deposition sequence $\mathcal{D}_a$. Similarly, find $\check{\varepsilon}_c$, $\check{\varepsilon}_g$, $\check{\varepsilon}_t$ and $\mathcal{D}_c$, $\mathcal{D}_g$, $\mathcal{D}_t$.

4: Combine $\mathcal{D}_a$, $\mathcal{D}_c$, $\mathcal{D}_g$, and $\mathcal{D}_t$ to form $\mathcal{D}$ (append one after the other).

5: Extend the embeddings $\check{\varepsilon}_a$, $\check{\varepsilon}_c$, $\check{\varepsilon}_g$, $\check{\varepsilon}_t$ according to $\mathcal{D}$ by inserting "$-$" in the columns corresponding to other groups. The union of the extended embeddings is the resulting embedding $\check{\varepsilon}$.

Output: The placement $\check{\phi}$ and an embedding $\check{\varepsilon}$ for $\mathcal{S}$.

Example 1. *Suppose we are given nine sequences* $\mathcal{S} = \{$AACT, ACGA, ACGG, CAGA, CCTA, GGAA, GGAT, TACT, TTAC$\}$. *Then* $\mathcal{A} = \{$AACT, ACGA, CAGA, GGAA$\}$, $\mathcal{C} = \{$CCTA$\}$, $\mathcal{G} = \{$ACGG, GGAT$\}$, *and* $\mathcal{T} = \{$TACT, TTAC$\}$. *The placement returned by* APlace&Embed *would be*

AACT	ACGA	CAGA
ACGG	CCTA	GGAA
GGAT	TACT	TTAC

As for the embeddings, we have $\mathcal{D}_a$ = GGCACGACT, $\mathcal{D}_c$ = CCTA, $\mathcal{D}_g$ = ACGGAT, *and* $\mathcal{D}_t$ = TACTAC. *Furthermore* $\mathcal{D}$ = GGCACGACT<u>CCTA</u>ACGGAT<u>TACTAC</u>.

Theorem 18. *The asymptotic approximation ratio of* APlace&Embed *is* 4. *when the size of the alphabet is* 4 *and the sequences have the same length* ℓ.

Proof. Consider the optimal placement ϕ^* and embedding ε^*. For every pair of neighboring sequences s_i, s_j, $\mathrm{share}_\varepsilon(s_i, s_j) + \mathrm{share}_\varepsilon(s_j, s_i) \le 2\ell$. There are a total of $2(n - \sqrt{n})$ pairs of neighbors in the array in total. So, the optimal agreement $\mathrm{A}(\phi^*, \varepsilon^*) \le 4\ell(n - \sqrt{n})$. On the other hand, consider $\check{\phi}$ and $\check{\varepsilon}$ returned by APlace&Embed. According to the way we partition the sequences into group, for any two sequences s_i, s_j in a group, the number of characters that can be shared is at least $\ell/4$. Hence, $\mathrm{share}_{\check{\varepsilon}}(s_i, s_j) + \mathrm{share}_{\check{\varepsilon}}(s_j, s_i) \ge 2(\ell/4) = \ell/2$. As we have seen above, there are altogether $2(n - \sqrt{n})$ pairs of neighbors in the array. We may not share any character when the pair belongs to different groups. According to the way we thread the groups, there are at most $3\sqrt{n}+3$ such pairs ($\sqrt{n}$ pairs of vertical neighbors between consecutive groups and 3 pairs of neighbors that are the last one in a group and the first one in the next group). As a result, we have at least $2n - 5\sqrt{n} - 3$ pairs each with sum of share at least $\ell/2$. Therefore, $\mathrm{A}(\check{\phi}, \check{\varepsilon}) \ge \ell(n - 2.5\sqrt{n} - 1.5)$. Then $\mathrm{A}(\check{\phi}, \check{\varepsilon})/\mathrm{A}(\phi^*, \varepsilon^*)$ tends to 4 as $\mathrm{A}(\phi^*, \varepsilon^*)$ tends to infinity. So, the asymptotic approximation ratio of APlace&Embed is 4. □

7 Concluding remarks

We give a comprehensive study of different variations of the *Border Minimization Problem* and present NP-hardness proofs and approximation algorithms. With regard to the complexity of different variants, our results show that (i) the BMP is NP-hard regardless of the dimension of the array; (ii) the array dimension differentiates the complexity of the P-BMP; and (iii) for 1D array, whether placement is given differentiates the complexity of the BMP.

Moreover, our techniques can be used to improve the approximation ratio for the synchronous case from $O(n^{1/2})$ to $O(n^{1/4})$ using the placement method given by Algorithm 2 (where the metric is defined by the Hamming distance between the sequences). Once a placement is found, the synchronous embedding can be computed exactly in polynomial time.

Note that the NP-hardness reduction for the P-BMP works for alphabets of size 3. In contrast, the hardness result for the BMP uses non-constant alphabets. An open problem is to prove that the BMP is hard also on constant alphabets (intuitively the BMP is harder than the P-BMP) but this does not seem to be easy.

Another natural open question is to further improve approximation algorithms for the BMP and the P-BMP and/or to derive inapproximability results. As mentioned in the introduction, there is an exponential time algorithm to compute the optimal BMP solution. Improving the exponential time algorithm could be useful in practice and is of theoretical interest.

References

[1] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *Symposium on Foundations of Computer Science (FOCS)*, pages 184–193, 1996.

[2] P. Bonizzoni and G. D. Vedova. The complexity of multiple sequence alignment with SP-score that is a metric. *Theoretical Computer Science*, 259(1–2):63–79, 2001.

[3] S. A. Carvalho Jr. and S. Rahmann. Improving the layout of oligonucleotide. microarrays: Pivot partitioning. In *Proceedings of the Sixth Workshop on Algorithms in Bioinformatics (WABI)*, pages 321–332, 2006.

[4] S. A. Carvalho Jr. and S. Rahmann. Microarray layout as quadratic assignment problem. In *Proceedings of the German Conference on Bioinformatics (GCB)*, pages 11–20, 2006.

[5] S. A. Carvalho Jr. and S. Rahmann. Improving the design of genechip arrays by combining placement and embedding. In *Proceedings of the Sixth International Conference on Computational Systems Bioinformatics (CSB)*, pages 54–63, 2007.

[6] Madhumita Chatterjee, Saroj Mohapatra, Alexei Ionan, Gagandeep Bawa, Rouba Ali-Fehmi, Xiaoju Wang, James Nowak, Bin Ye, Fatimah A. Nahhas, Karen Lu, Steven S. Witkin, David Fishman, Adnan Munkarah, Robert Morris, Nancy K. Levin, Natalie N. Shirley, Gerard Tromp, Judith Abrams, Sorin Draghici, and Michael A. Tainsky. Diagnostic markers of ovarian cancer by high-throughput antigen cloning and detection on arrays. *Cancer Research*, 66(2):1181–1190, 2006.

[7] N. Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, PA. *(ND33)*, 1976.

[8] Marina Cretich and Marcella Chiari. In *Peptide Microarrays Methods and Protocols*, volume 570 of *Methods in Molecular Biology*. Human Press, 2009.

[9] Jarmo Ernvall, Jyrki Katajainen, and Martti Penttonen. NP-completeness of the hamming salesman problem. *BIT Numerical Mathematics*, 25:289–292, 1985.

[10] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC*, pages 448–455, 2003.

[11] D. F. Feng and R. F. Doolittle. Approximation algorithms for multiple sequence alignment. *Theoretical Computer Science*, 182(1):233–244, 1987.

[12] S. Fodor, J. L. Read, M. C. Pirrung, L. Stryer, A. T. Lu, and D. Solas. Light-directed, spatially addressable parallel chemical synthesis. *Science*, 251(4995):767–773, 1991.

[13] D. Gerhold, T. Rushmore, and C. T. Caskey. DNA chips: promising toys have become powerful tools. *Trends in Biochemical Sciences*, 24(5):168–173, 1999.

[14] D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology*, 55(1):141–154, 1993.

[15] S. Hannenhalli, E. Hubell, R. Lipshutz, and P. A. Pevzner. Combinatorial algorithms for design of DNA arrays. *Advances in Biochemical Engineering/Biotechnology*, 77:1–19, 2002.

[16] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[17] L. Kaderali and A. Schliep. Selecting signature oligonucleotides to identify organisms using DNA arrays. *Bioinformatics*, 18:1340–1349, 2002.

[18] A. B. Kahng, I. I. Mandoiu, P. A. Pevzner, S. Reda, and A. Zelikovsky. Scalable heuristics for design of DNA probe arrays. *JCB*, 11(2/3):429–447, 2004. Preliminary versions in WABI 2002 and RECOMB 2003.

[19] A. B. Kahng, I. I. Mandoiu, S. Reda, X. Xu, and A. Zelikovsky. Computer-aided optimization of DNA array design and manufacturing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(2):305–320, 2006.

[20] S. Kasif, Z. Weng, A. Detri, R. Beigel, and C. DeLisi. A computational framework for optimal masking in the synthesis of oligonucleotide microarrays. *Nucleic Acids Research*, 30(20):e106, 2002.

[21] Vamsi Kundeti and Sanguthevar Rajasekaran. On the hardness of the border length minimization problem. In *BIBE*, pages 248–253, 2009.

[22] Vamsi Kundeti, Sanguthevar Rajasekaran, and Hieu Dinh. Border length minimization problem on a square array. *Journal of Computational Biology*, 21(6):446–455, 2014.

[23] F. Li and G. Stormo. Selection of optimal DNA oligos for gene expression arrays. *Bioinformatics*, 17(11):1067–1076, 2001.

[24] William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[25] Christian Melle, Günther Ernst, Bettina Schimmel, Annett Bleul, Sven Koscielny, Andreas Wiesner, Ralf Bogumil, Ursula Möller, Dirk Osterloh, Karl-Jürgen Halbhuber, and Ferdinand von Eggeling. A technical triade for proteomic identification and characterization of cancer biomarkers. *Cancer Research*, 64(12):4099–4104, 2004.

[26] S. Rahmann. The shortest common supersequence problem in a microarray production setting. *Bioinformatics*, 19(suppl. 2):156–161, 2003.

[27] Kari-Jouko Räihä and Esko Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16(2):187–198, 1981.

[28] K. Reinert, H. P. Lenhof, P. Mutzel, K. Mehlhorn, and J. D. Kececioglu. A branch-and-cut algorithm for multiple sequence alignment. In *Proceedings of the First International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 241–250, 1997.

[29] D. K. Slonim, P. Tamayo, J. P. Mesirov, T. R. Golub, and E. S. Lander. Class prediction and discovery using gene expression data. In *Proceedings of the Fourth International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 263–272, 2000.

[30] W. K. Sung and W. H. Lee. Fast and accurate probe selection algorithm for large genomes. In *Proceedings of the Second International Conference on Computational Systems Bioinformatics (CSB)*, pages 65–74, 2003.

[31] John B. Welsh, Lisa M. Sapinoso, Suzanne G. Kern, David A. Brown, Tao Liu, Asne R. Bauskin, Robyn L. Ward, Nicholas J. Hawkins, David I. Quinn, Pamela J. Russell, Robert L. Sutherland, Samuel N. Breit, Christopher A. Moskaluk, Henry F. Frierson, Jr., and Garret M. Hampton. Large-scale delineation of secreted protein biomarkers overexpressed in cancer tissue and serum. *PNAS*, 100(6):3410–3415, 2003.

[32] B. Y. Wu, G. Lancia, V. Bafna, K. M. Chao, R. Ravi, and C. Y. Tang. A polynomial-time approximation scheme for minimum routing cost spanning trees. *SIAM Journal on Computing*, 29(3):761–778, 1999.