Pushing the Online Matrix-vector conjecture off-line and identifying its easy cases

Leszek Gąsieniec Jesper Jansson Christos Levcopoulos Andrzej Lingas
Mia Persson

Abstract

Henzinger et al. posed the so called Online Boolean Matrix-vector Multiplication (OMv) conjecture and showed that it implies tight hardness results for several basic partially dynamic or dynamic problems [STOC'15].
We show that the OMv conjecture is implied by a simple off-line conjecture. If a not uniform (i.e., it might be different for different matrices) polynomial-time preprocessing of the matrix in the OMv conjecture is allowed then we can show such a variant of the OMv conjecture to be equivalent to our off-line conjecture. On the other hand, we show that the OMV conjecture does not hold in the restricted cases when the rows of the matrix or the input vectors are clustered.

1 Introduction

Henzinger et al. considered the following *Online Boolean Matrix-vector Multiplication* (OMv) problem in [8]. Initially, there are given an integer n and an $n \times n$ Boolean matrix M. Then, for $i = 1, ..., n$, in the i-th round there is given an n-dimensional Boolean column vector v_i, and the task is to compute the product of M with v_i before the next round. The objective is to design a (possibly randomized) algorithm that solves the OMv problem, i.e., it computes all the n products as quickly as possible. In [8], Henzinger et al. provided efficient reductions of the OMv problem to several basic partially dynamic or dynamic problems including subgraph connectivity, Pagh's problem, d-failure connectivity, decremental single-source shortest paths, and decremental transitive closure.

They also stated the following OMv conjecture in [8].

Conjecture 1. OMv conjecture *For any constant $\epsilon > 0$, there is no randomized algorithm that solves the OMv problem in $O(n^{3-\epsilon})$ time with an error probability of at most $1/3$.*

Their conjecture implies tight hardness results for the aforementioned partially dynamic or dynamic problems [8]. It also implies the following off-line Mv conjecture [8].

Conjecture 2. Mv conjecture *For any constant $\epsilon > 0$ and any polynomial $p()$, there is no randomized preprocessing and randomized algorithm such that any $n \times n$ Boolean matrix M can be preprocessed in $p(n)$ time so the Boolean product of M with an arbitrary Boolean n-dimensional column vector can be computed in $O(n^{2-\epsilon})$ time with an error probability of at most $1/3$.*

The fastest known algorithm for the OMv problem is due to Green Larsen and Williams [7]. Their recent (not combinatorial) randomized algorithm runs in
$O(n^3/2^{\Omega(\sqrt{\log n})})$ time. Williams has also shown in [10] that any $n \times n$ Boolean matrix can be preprocessed in $O(n^{2+\epsilon})$ time so the Boolean product of the matrix with an arbitrary n-dimensional Boolean vector can be computed in $O(n^2/\log^2 n)$ time. This implies that the Mv problem corresponding to the Mv conjecture admits an $O(n^2/\log^2 n)$-time solution. Also recently, Chakrabort et al. have established tight cell probe bounds for succinct Boolean matrix-vector multiplication in [4].

Our contributions. We show that the OMv conjecture is implied by the following simple off-line MvP conjecture: For any constant $\epsilon > 0$ and any polynomial p there is an $n \times n$ Boolean matrix M that cannot be preprocessed in

$p(n)$ time such that the Boolean product of M with an arbitrary n-dimensional column vector v can be computed in $O(n^{2-\epsilon})$ time with an error probability of at most $1/3$. There is a subtle but a substantial difference between our MvP conjecture and the Mv conjecture, the latter stated and shown to be implied by the OMv conjecture in [8]. In our conjecture the preprocessing is not uniform with respect to the matrices while in the Mv conjecture in [8] one considers a uniform, universal preprocessing. It follows that the difficulty of proving the OMv conjecture lies between the two aforementioned off-line conjectures: OMv is not more difficult than MvP and it is not easier than Mv.
We also show that if we relax the OMv problem by allowing for a not uniform polynomial-time preprocessing of the matrix M then the corresponding online conjecture will be equivalent to our MvP conjecture.
Basically, the Combinatorial Boolean Matrix Multiplication conjecture (CBMM conjecture) states that there is no combinatorial (randomized) algorithm for the Boolean product of two $n \times n$ Boolean matrices that runs in substantially subcubic time [2, 8]. Marginally, we also observe that if we strengthen the CBMM conjecture by allowing for a polynomial-time uniform preprocessing of one of the matrices, the resulting conjecture will be equivalent to the original CBMM conjecture.
On the other hand, by adapting known algorithms for Boolean matrix product of matrices with clustered data [3, 5, 6], we obtain a combinatorial randomized algorithm for the product of a Boolean $n \times n$ matrix M and an arbitrary Boolean n-dimensional column vector v running in $\tilde{O}(n + ST_M)$ time after an $\tilde{O}(n^2)$-time preprocessing of M, where ST_M stands for the cost of a minimum spanning tree of the rows of M under the extended Hamming distance (never exceeding the Hamming distance). Consequently, we obtain a combinatorial randomized algorithm for the OMv problem running in $\tilde{O}(n(n + ST_M))$ time. We also show that OMv admits a combinatorial randomized algorithm running in $\tilde{O}(n \max\{ST(V), n^{1+o(1)}))$ time, where $ST(V)$ stands for the cost of a minimum spanning tree of the column vectors $v_1, ..., v_n$ under the extended Hamming distance. The time analysis of the latter algorithm relies in part on our analysis of an approximate nearest-neighbour online heuristic for the aforementioned minimum spanning tree.
The overwhelming majority of the reductions of the OMv problem to other partially dynamic or dynamic problems in [8] are unfortunately one-way reductions that do not yield applications of our algorithms for the OMv and Mv problems. Following the applications of the Mv problem given in[2, 10], we provide analogous applications of our algorithms to vertex subset queries (e.g., for a given graph, such a query asks if a given subset of vertices is independent), triangle membership queries and 2-CNF formula evaluation queries.

Organization of the Paper. Section 2 introduces three new conjectures and it shows implications and equivalences between them and the OMv conjecture. Section 3 presents our algorithms for the OMv and Mv problems whose time complexity is expressed in terms of the minimum cost of a spanning tree of the rows of the matrix or the input column vectors under the extended Hamming distance. Section 4 presents applications of our algorithms. Section 5 concludes with some final remarks. Because of space considerations, the proof of Lemma 3 as well as a marginal subsection of Section 2 on Combinatorial Boolean Matrix Product, and an additional application are omitted in this extended abstract.

2 An off-line conjecture

By the auxiliary Boolean Matrix-vector multiplication problem (AMv) we shall mean the problem of efficiently computing the product of a fixed $n \times n$ Boolean matrix M, that can be (not uniformly) preprocessed in $O(n^{3-\epsilon})$ time for some fixed $\epsilon > 0$, with an arbitrary n-dimensional Boolean column vector v. We state the following conjecture related to the AMv problem.

Conjecture 3. AMv conjecture *For any constant $\epsilon > 0$ and constants c_1, c_2, there is an $n \times n$ Boolean matrix M that cannot be preprocessed in $c_1 n^{3-\epsilon}$ time such that the Boolean product of M with an arbitrary n-dimensional Boolean column vector v can be computed in $c_2 n^{2-\epsilon}$ time with an error probability of at most $1/3$.*

We shall show the AMv conjecture to imply the OMv one.

Lemma 1. *Let ϵ be a positive constant and let M be an $n \times n$ Boolean matrix. If the OMv problem for M can be solved in $O(n^{3-\epsilon})$ time with an error probability of at most $1/3$ then the matrix M can be (not uniformly) preprocessed in $O(n^{3-\epsilon})$ time such that the Boolean product of M with an arbitrary input n-dimensional Boolean column vector v*

can be computed in $O(n^{2-\epsilon})$ time with an error probability of at most $1/3$. Consequently, the AMv conjecture implies the OMv conjecture.

Proof. Construct a sequence of n-dimensional Boolean vectors v_1,....v_n iteratively by picking as v_i a vector that jointly with the preceding vectors maximizes the total time of the assumed OMv solution for $v_1, ..., v_i$. Since the assumed OMv solution for the whole sequence takes $O(n^{3-\epsilon})$ time, there must be $i \in \{1, ..., n\}$ such that the product of M with v_i is computed in $O(n^{2-\epsilon})$ time after computing the products of M with the preceding vectors in the sequence. The computation of all the products clearly takes $O(n^{3-\epsilon})$ time and it has an error probability of at most $1/3$. By the definition of v_i, if we compute instead of the product of M with v_i, the product of M with an arbitrary n-dimensional input vector v, the computation will take only $O(n^{2-\epsilon})$ time after the products with the preceding vectors have been computed. Again, the computation of all the products, and hence in particular that of M with v, will have an error probability of at most $1/3$. Since the vectors $v_1...v_{i-1}$ are fixed, the computation of the products of M with the preceding vectors can be regarded as an $O(n^{3-\epsilon})$-time preprocessing. □ □

Unfortunately, we cannot show the reverse implication, i.e., that the OMv conjecture implies the AMv one like it implies the Mv conjecture [8]. The reason is that in the definition of the AMv problem, we do not require a universal preprocessing that could work for any matrix M of size $n \times n$, we only require the existence of an individual preprocessing for a given M.

In the next lemma, we demonstrate that if we allow for an arbitrary (not uniform) polynomial-time preprocessing instead of the substantially subcubic one, we will obtain a problem equivalent to the AMv one. This lemma and its proof idea of dividing the matrix and the vector into appropriate submatrices and subvectors are similar to Lemma 2.3 and its proof idea in [8], respectively.

Lemma 2. *Let δ and ϵ be positive constants. If for any $n \times n$ Boolean matrix M there is an $O(n^{3+\delta})$-time preprocessing such that the product of M with an arbitrary n-dimensional Boolean column vector v can be computed in $O(n^{2-\epsilon})$ time with an error probability of at most $1/3$ then there is a positive constant ϵ such that after an $O(n^{3-\epsilon'})$-time preprocessing the product of M with such a vector v can be computed in $O(n^{2-\epsilon'})$ time with an error probability of at most $1/3$.*

Proof. Divide M into $n^{2\alpha}$ quadratic submatrices $M_{i,j}$ of size $n^{1-\alpha} \times n^{1-\alpha}$, where $i, j \in \{1, ..., n^{\alpha}\}$. Preprocess all the submatrices in $O(n^{2\alpha} \times (n^{1-\alpha})^{3+\delta})$ time. Then, the product of M with the vector v can be computed in $O(n^{2\alpha} \times (n^{1-\alpha})^{2-\epsilon} + n^{1+\alpha})$ time. The last term in the expression represents the cost of summing the results of the products of the submatrices with respective subvectors of v of length $n^{1-\alpha}$. In order to obtain an exponent of the total preprocessing time in the form $3 - \epsilon'$ and the exponent of computing the product in the form $2 - \epsilon'$, it is sufficient to solve the inequalities $2\alpha + (1-\alpha)(3+\delta) < 3$, $2\alpha + (1-\alpha)(2-\epsilon) < 2$ and $1 + \alpha < 2$ with respect to α. Any α in the open interval $(\frac{\delta}{1+\delta}, 1)$ satisfies these inequalities.

Following the proof of Lemma 2.3 in [8], we can keep the error probability below $1/3$ by repeating the computation of each of the products of a submatrix of M with a respective vector $O(\log n)$ times, and picking the most frequent answer. In order to tackle the additional logarithmic factor in the time complexity, we can slightly decrease our ϵ'. □ □

We shall call the problem and the conjecture resulting from the AMv problem and the AMv conjecture by replacing an $O(n^{3-\epsilon})$ (not uniform) preprocessing time with a polynomial (not uniform) preprocessing time, a *Boolean Matrix-vector multiplication with (polynomial-time not uniform) preprocessing problem* and a *Boolean Matrix-vector multiplication with (polynomial-time not uniform) preprocessing conjecture* (MvP for short), respectively. Since the MvP conjecture trivially implies the AMv conjecture, by Lemmata 1, 2, we obtain the following theorem.

Theorem 1. *The AMv and MvP conjectures are equivalent and they imply the OMv conjecture.*

Relaxing the OMv problem. In order to obtain a version of OMv equivalent to AMv and MvP, we shall consider generalized versions of the OMv problem and the OMv conjecture allowing for a not uniform polynomial-time preprocessing of the matrix. We shall term them, the OMvP problem and the OMvP conjecture, respectively.

The proof of the following lemma is analogous to that of Lemma 1..

Lemma 3. *Let ϵ be a positive constant, and let M be an $n \times n$ Boolean matrix. If the OMvP problem for M and any positive natural number n, after a polynomial-time (not uniform) preprocessing of M can be solved in $O(n^{3-\epsilon})$ time with an error probability of at most $1/3$ then the matrix M can be (not uniformly) preprocessed in polynomial time such that the Boolean product of M with an arbitrary input n-dimensional Boolean column vector v can be computed in $O(n^{2-\epsilon})$ time with an error probability of at most $1/3$. Consequently, the MvP conjecture implies the OMvP conjecture.*

Lemma 4. *Let ϵ be a positive constant, and let M be an $n \times n$ Boolean matrix. If the AMv problem for M can be solved in $O(n^{2-\epsilon})$ time with an error probability of at most $1/3$ after an $O(n^{3-\epsilon})$ not uniform preprocessing of M then the OMvP problem for the matrix M and n Boolean column vectors can be solved in $O(n^{3-\epsilon})$ time with an error probability of at most $1/3$. Consequently, the OMvP conjecture implies the AMv conjecture.*

Proof. Before computing the product of M with the first vector, perform the appropriate not uniform $O(n^{3-\epsilon})$ time preprocessing of M. After that the product of M with each consecutive vector can be computed in $O(n^{2-\epsilon})$ time, so the total time for n vectors becomes $O(n^{3-\epsilon})$. We can keep the error probability below $1/3$ for the whole sequence of input vectors similarly as in the proof of Lemma 2. □ □

Since the conjectures AMv and MvP are equivalent (see Theorem 1), by Lemmata 3 and 4, we obtain the following extension of Theorem 1.

Theorem 2. *The MvP, AMv and OMvP conjectures are equivalent.*

3 Easy cases of matrices and vectors for the conjectures

Björklund et al. [3] proposed a method of multiplying two Boolean matrices by using a close approximation of the minimum spanning tree of the rows or columns of one of the matrices under the Hamming distance. Subsequently, the method has been generalized to include the so called extended Hamming distance [6] and integer matrix multiplication [5]. In the first warming-up subsection, we present an explicit adaptation of the aforementioned generalizations to the case of the product of an $n \times n$ Boolean (or $0-1$) matrix M and an n-dimensional Boolean (or $0-1$) column vector v in the context of the OMv conjecture. Several results presented in the first subsection can be regarded as implicit in [5, 6]. This is not the case in the second subsection handling the online scenario where the input column vectors are clustered. Here, we have to develop a novel online approach involving among other things an analysis of an approximate nearest-neighbour online heuristic for minimum spanning tree of the vectors under the extended Hamming distance. We shall use the following concepts in both subsections.

Definition 1. *For two $0-1$ strings $s = s_1 s_2 ....s_m$ and $u = u_1 u_2 ...u_m$, their Hamming distance, i.e., the number of $k \in \{1, ..., m\}$, s.t., $s_k \neq u_k$, is denoted by $H(s, u)$. An extended Hamming distance, $EH(s, u)$, between the strings, is defined by a recursive equation $EH(s, u) = EH(s_{l+1}...s_m, u_{l+1}...u_m) + (s_1 + u_1 \mod 2)$, where l is the maximum number, s.t., $s_j = s_1$ and $u_j = u_1$ for $j = 1, ..., l$.*
A differentiating block *for the strings s, u is a maximal consecutive subsequence w of $1, 2, ...n$, s.t., either for each $i \in w$ $s_i = 1$ and $u_i = 0$ or for each $i \in w$ $s_i = 0$ and $u_i = 1$. In the first case, we set $h(s) = -1$ while in the second case $h(s) = 1$.*

3.1 Small spanning tree of the rows of the matrix (warming up)

For $c \geq 1$ and a finite set S of points in a metric space, a *c-approximate minimum spanning tree for S* is a spanning tree in the complete weighted graph on S, with edge weights equal to the distances between the endpoints, whose total weight is at most c times the minimum.

Fact 1. *(Lemma 3 in [6]) For $\epsilon > 0$, a $2 + \epsilon$-approximate minimum spanning tree for a set of n $0-1$ strings of length d under the extended Hamming metric can be computed by a Monte Carlo algorithm in time $O(dn^{1+1/(1+\epsilon/2)})$.*

By selecting $\epsilon = 2 \log n$, we obtain the following lemma.

Lemma 5. *Let M be an $n \times n$ Boolean matrix. An $O(\log n)$-approximation minimum spanning tree for the set of rows of M under the extended Hamming distance can be constructed by a Monte Carlo algorithm in $\tilde{O}(n^2)$ time.*

We shall also use the following data structure, easily obtained by computing all prefix sums:

Fact 2. *(e.g., see [5]) For a sequence of integers a_1, a_2,...,a_n, one can construct a data structure that supports a query asking for reporting the sum $\sum_{k=i}^{j} a_k$ for $1 \leq i \leq j \leq n$ in $O(1)$ time. The construction takes $O(n)$ time.*

By using Lemma 5 and Fact 2, we obtain the following algorithm which in fact computes the arithmetic product of the input Boolean matrix M and Boolean vector v interpreted as $0-1$ ones. Observe that the aforementioned arithmetic product immediately yields the corresponding Boolean one.

Algorithm 1

Input: An $n \times n$ Boolean matrix M and an n-dimensional Boolean column vector v.

Output: The arithmetic product $c = (c_1, ..., c_n)$ of M and v interpreted as a $0-1$ matrix and a $0-1$ vector, respectively.

1. Find an $O(\log n)$-approximate spanning tree T for the rows $row_i(M)$, $i = 1, \ldots, n$, of M under the extended Hamming distance and a traversal (i.e., a not necessarily simple path visiting all vertices) of T.
2. For any pair $(row_i(M), row_l(M))$, where the latter row follows the former in the traversal, find the differentiating blocks s for $row_i(M)$ and $row_l(M)$ and as well as the differences $h(s)$ (1 or -1) between the common value of each entry in $M_{l,\min s}, \ldots, M_{l,\max s}$ and the common value of each entry in $M_{i,\min s}, \ldots, M_{i,\max s}$.
3. Initialize a data structure D for counting partial sums of the values of coordinates on continuous fragments of the vector v.
4. Iterate the following steps:
   (a) Compute c_q where q is the index of the row from which the traversal of T starts.
   (b) While following the traversal of T, iterate the following steps:
      i. Set $i,\ l$ to the indices of the previously traversed row and the currently traversed row, respectively.
      ii. Set c_l to c_i.
      iii. For each differentiating block s for $row_i(M)$ and $row_l(M)$, compute $\sum_{k \in s} v_k$ using D and set c_l to $c_l + h(s) \sum_{k \in s} v_k$.
5. Output the vector $(c_1, c_2, ..., c_n)$

Definition 2. *For an $n \times n$ Boolean matrix A, let ST_A stand for the minimum cost of a spanning tree of $row_i(A)$, $i \in \{1, ..., n\}$, under the extended Hamming distance.*

Lemma 6. *Algorithm 1 runs in $\tilde{O}(n^2 + ST_M)$ with high probability. If Steps 1, 2, 3 are treated as a preprocessing of the matrix M then it runs in $\tilde{O}(n + ST_M)$ time after an $\tilde{O}(n^2)$-time preprocessing.*

Proof. The approximate minimum spanning tree T in Step 1 can be constructed by a Monte Carlo algorithm in $\tilde{O}(n^2)$ time by Lemma 5. Its traversal can be easily found in $O(n)$ time. Since the length of the traversal is linear in n, Step 2 can be easily implemented in $O(n^2)$ time. Step 3 takes $O(n)$ time by Fact 2. Finally, based on Step 2, Step 4 (b)-ii takes $\tilde{O}(1 + EH(row_i(M), row_l(M)))$ time. Let U stand for the set of directed edges forming the traversal of the spanning tree T. It follows that Step 4 (b) can be implemented in $\tilde{O}(n + \sum_{(i,l) \in U} EH(row_i(M), row_l(M)))$ time, i.e., in $\tilde{O}(n + ST_M)$ time by Lemma 5. Consequently, Step 4 takes $\tilde{O}(n + ST_M)$ time. □ □

By Lemma 6, we obtain:

Theorem 3. *The Boolean product c of an $n \times n$ Boolean matrix M and an n-dimensional Boolean column vector v can be computed in $\tilde{O}(n + ST_M)$ time with high probability after $\tilde{O}(n^2)$–time preprocessing.*

Proof. The correctness of Algorithm 1 follows from the observation that a differentiating block s for $row_i(M)$ and $row_l(M)$ yields the difference $h(s)\sum_{k\in s} v_k$ between c_l and c_i just on the fragment corresponding to $M_{i,\min s},\ldots,M_{i,\max s}$ and
$M_{l,\min s},\ldots,M_{l,\max}$, respectively. Lemma 6 yields the upper bounds in terms of ST_M. □ □

Corollary 1. *The OMv problem for an $n\times n$ Boolean matrix can be solved in*
$\tilde{O}(n(n+ST_M))$ time while the Mv problems can be solved in $\tilde{O}(n+ST_M)$ time after $\tilde{O}(n^2)$-time preprocessing.

3.2 Small spanning tree of the input vectors

In this subsection, we assume an online scenario where besides the Boolean matrix there is given a sequence of n-dimensional Boolean vectors received one at time. In order to specify and analyze our algorithm, we need the following concepts and facts on them.

Definition 3. *For a metric space P and a point $q\in P$, an* c-approximate nearest neighbour *of q in P is a point $p\in P$ different from q such that for all $p'\in P, p'\neq q$, $dist(p,q)\leq c\times dist(p',q)$. The ϵ-approximate nearest neighbour search problem*
(ϵ-NNS) in P is to find for a query point $q\in P$ a $(1+\epsilon)$-approximate nearest neighbour of q in P.

Fact 3. *(See 3rd row in Table 4.3.1.1 in [1]) For $\epsilon>0$, there is a Monte Carlo algorithm for the dynamic ϵ-NNS in $\{0,1\}^d$ under the Hamming metric which requires $O(d\ell^{\frac{1}{1+2\epsilon}+o(1)})$ query time and $O(d\ell^{\frac{1}{1+2\epsilon}+o(1)})$ update time, where ℓ is the maximum number of stored vectors in $\{0,1\}^d$.*

Fact 4. *[6] There is a simple, linear-time, transformation of any $0-1$ string w into the string $t(w)$ such that for any two $0-1$ strings s and u, $EH(s,u)=\lceil\frac{H(t(s),t(u))}{2}\rceil$.*

By combining Facts 3, 4. we obtain the following corollary.

Corollary 2. *There is a randomized Monte Carlo algorithm for a dynamic $O(\log\ell)$-NNS in $\{0,1\}^d$ under the extended Hamming metric which requires $O(d\ell^{o(1)})$ query time and $O(d\ell^{o(1)})$ update time.*

Our online algorithm is as follows.

Algorithm 2
Input: Given a priori an $n\times n$ Boolean matrix M and an online sequence of n-dimensional Boolean vectors $v_1,v_2,...,v_\ell$ received one at time.
Output: For $i=1,...,\ell$, the arithmetic product $c^i=(c^i_1,...,c^i_n)=Mv_i$ of M and v_i, treated as a 0-1 matrix and a 0-1 column vector, is output before receiving v_{i+1}.

1. For $j=2,\ldots,n$, initialize a data structure D_j that for any interval $s\subseteq\{1,...,n\}$ reports $\sum_{k\in s}M[j,k]$ using Fact 2.
2. Receive the first vector v_1 and compute the arithmetic product $c^1=(c^1_1,...,c^1_n)$ of M with v_1 by the definition.
3. For $i=2,\ldots,\ell$, receive the i-th vector $v_i=(v^i_i,...,v^i_n)$ and iterate the following steps:
   (a) Find an $O(\log\ell)$-approximate nearest neighbour v_m of v_i in the set $\{v_1,...,v_{i-1}\}$.
   (b) Determine the differentiating blocks s and the differences $h(s)$ for v_m and v_i.
   (c) For $j=1,\ldots,n$ iterate the following steps.
      i. Set c^i_j to c^m_j.
      ii. For each differentiating block s of v_m and v_i iterate the following steps.
         A. Compute $\sum_{k\in s}M[j,k]$ using D_j.
         B. Set c^i_j to $c^i_j+h(s)\sum_{k\in s}M[j,k]$.
   (d) Output $c^i=(c^i_1,...,c^i_n)$

In the following lemmata, we analyze the time complexity of Algorithm 2. The first lemma is an immediate consequence of Corollary 2.

Lemma 7. *There is a randomized Monte Carlo algorithm for a dynamic $O(\log \ell)$-NNS in $\{0,1\}^d$ under the extended Hamming metric such that:*

- *The insertions of the vectors v_1 through v_ℓ in Algorithm 2 can be implemented in $O(n\ell^{1+o(1)})$ total time.*
- *The $O(\log \ell)$-approximate nearest neighbours of v_i, $i = 2, ..., \ell$, in $\{v_1, ..., v_{i-1}\}$, in Step 3 (a) of Algorithm 2 can be found with high probability in $O(n\ell^{1+o(1)})$ total time.*

Proof. By Corollary 2, the $\ell - 1$ updates and $\ell - 2$ $O(\log \ell)$-approximate nearest neighbour queries take $O(n\ell^{1+o(1)})$ total time. □ □

Lemma 8. *The preprocessing of the matrix M in Step 1 and computing the arithmetic product of M with v_1 in Step 2 takes $O(n^2)$ time. After that, Algorithm 2 for $i = 2, ..., \ell$, computes the arithmetic product c^i of M and v_i before receiving v_{i+1} in $\tilde{O}(n(1 + \min\{dist(v_i, v_j)|j < i\})) + t(i)$ time, where $t(i)$ is the time taken by finding an $O(\log \ell)$-approximate neighbour of v_i in $\{v_1, v_2, ..., v_{i-1}\}$ and inserting v_i in the supporting data structure, with high probability. The total time is $\tilde{O}(n(\ell + ST(V))) + \sum_{i=2}^{\ell} t(i)$, where $ST(V)$ is the minimum cost of the spanning tree of the vectors in $V = \{v_1, v_2, ..., v_\ell\}$ under the extended Hamming distance.*

Proof. Step 1 can be implemented in $O(n^2)$ time by Fact 2 while Step 2 can be easily done in $O(n^2)$ time by the definition. Step 3 (a) takes $t(i)$ time by our assumptions. The differentiating blocks s and the differences $h(s)$ for v_m and v_i can be easily determined in $O(n)$ time in Step 3 (b). Finally, since the number of the aforementioned blocks is within a polylogarithmic factor of $\min\{dist(v_i, v_j)|j < i\}$, the whole update of c^m to c^i in Step 3 (c) takes $\tilde{O}(n(1 + \min\{dist(v_i, v_j)|j < i\}))$ time. □ □

In order to pursue our time analysis of Algorithm 2, we need to specify and analyze the following simple online approximation heuristic for minimum spanning tree (MST).

Approximate Nearest-Neighbour Heuristic for MST

Input: an online sequence V of vectors $v_1, v_2, ...$ received one at time.
Output: a sequence of spanning trees T_i' of the vectors v_1 through v_i constructed before receiving v_{i+1} for all i.
for each new vector v_i **do**
find an $f(i)$-approximate nearest neighbour u of v_i in the set of vectors received so far;
expand the spanning tree built for the vectors received before v by $\{u, v_i\}$.

Theorem 4. *Assume that the function f is not decreasing and the input vectors to the approximate nearest-neighbour heuristic for MST are drawn from a metric space. The spanning tree constructed by the heuristic for the first t vectors has cost not exceeding $\lceil \log_2 t \rceil f(t)$ times the minimum.*

Proof. Assume first that t is a power of two. Let $V = \{v_1, ..., v_t\}$ be the sequence of t vectors received, where v_i is the i-th vector received.

Consider a minimum cost perfect matching P of V. For each edge $\{v_i, v_j\}$ in P, where $i < j$, the cost of connecting v_j to the current spanning tree T_{i-1} of v_1 through v_{j-1} does not exceed $f(t) \times dist(v_i, v_j)$. Thus, for $t/2$ vectors v_l in V, the cost of connecting them to the current spanning tree T_{l-1} does not exceed the total cost of P times $f(t)$. It is well known that the total cost of P is not greater than half the minimum cost $TSP(V)$ of the travelling salesperson tour of V.

In order to estimate from above the cost of connecting the remaining $t/2$ vectors to the current spanning trees, we iterate our argument.

Thus, let V_1 denote the remaining set of vertices and let P_1 be their minimum-cost perfect matching. We can again estimate the cost of connecting half of the $t/2$ vectors in V_1 to the current spanning trees by the cost of P_1 times $f(t)$. On the other hand, we can estimate the cost of P_1 by $\frac{1}{2}TSP(V_1) \leq \frac{1}{2}TSP(V)$. We handle analogously the remaining $t/4$ vectors and so on. After $\log_2 t$ iterations, we are left with the first vector, and can estimate the total cost of connecting all other vectors to the current spanning trees by $\log_2 t f(t) TSP(V)/2$. On the other hand, by the

doubling MST heuristic, we know that $TSP(V)$ is at most twice the cost $ST(V)$ of minimum-cost spanning tree of V. We conclude that the cost of the spanning tree of V constructed by the approximate nearest-neighbour heuristic does not exceed $\log_2 t f(t) ST(V)$.

If t is not a power of two, we have to consider minimum-cost maximum cardinality matchings instead of minimum-cost perfect matchings. Let $t' = 2^{\lceil \log_2 t \rceil}$. Observe that the number of the remaining vectors after each iteration when we start with a sequence of t vectors will be not greater than that when we start with a sequence of t' vectors having the sequence of t vectors as a prefix. This completes the proof of the $\lceil \log_2 t \rceil f(t) ST(V)$ upper bound. □ □

In the special case when $f(\ ) \equiv 1$, our online heuristic for MST in a way coincides with the greedy one for incremental minimum Steiner tree from [9], which on weighted graphs satisfying triangle inequality could be easily adapted to consider only received vertices. Hence, in this case a logarithmic upper bound on approximation factor could be also deduced from Theorem 3.2 in [9]. Putting together our lemmata and theorem in this subsection, we obtain our main result here.

Theorem 5. *Let M be an $n \times n$ Boolean matrix. For an online sequence V of n-dimensional Boolean vectors $v_1, v_2, ..., v_\ell$ received one at time, the Boolean products Mv_i of M and v_i can be computed before receiving v_{i+1}. in total time*
$\tilde{O}(n(\ell^{1+o(1)} + ST(V)))$ *with high probability by a randomized algorithm, where $ST(V)$ is the minimum cost of the spanning tree of the vectors in V under the extended Hamming distance.*

Proof. The correctness of Algorithm 2 follows from the observation that a differentiating block s for v_m and v_i yields the difference $h(s) \sum_{k \in s} M[j,k]$ between c_j^m and c_j^u just on the fragments $v^m_{\min s}, \ldots, v^m_{i,\max s}$ and $v^i_{\min s}, \ldots, v^i_{\max s}$, respectively. Lemmata 7, 8 and Theorem 4 yield the upper bounds in terms of $ST(V)$. □ □

4 Applications to graph queries

Suppose that we are given a graph $G = (V, E)$ on n vertices and a subset S of V. In [10] Williams observed that the questions if S is a dominating set, an independent set, or a vertex cover in G, can be easily answered by computing the Boolean product of the adjacency matrix of G with appropriate Boolean vectors. Hence, he could conclude (Corollary 3.1 in [10]) that these questions can be answered in $O(n^2/(\epsilon \log n)^2)$ time after an $O(n^{2+\epsilon})$ preprocessing of G by using his method of multiplying $n \times n$ Boolean matrix with an n-dimensional column vector in $O(n^2/(\epsilon \log n)^2)$ time after an $O(n^{2+\epsilon})$-time preprocessing of the matrix. By plugging in our method of Boolean matrix-vector multiplication (Theorem 3) instead, we obtain the following result.

Corollary 3. *A graph G on n vertices can be preprocessed in $\tilde{O}(n^2)$ time such that one can determine if a given subset of vertices in G is a dominating set, an independent set, or a vertex cover of G in $\tilde{O}(n + ST_G)$ time with high probability, where ST_G is the minimum cost of a spanning tree of the rows of the adjacency matrix of G under the extended Hamming distance. Using the same preprocessing, one can determine if a query vertex belongs to a triangle in G in $\tilde{O}(n + ST_G)$ time with high probability.*

To obtain corresponding applications of the results from subsection 3.2, we need to consider the online versions of the graph subset queries. Thus, we are given a graph G on n vertices and an online sequence of subsets S_1,...,S_ℓ of vertices in G. The task is to preprocess G first and then to determine for $i = 1, ..., \ell$, if S_i is a dominating set, an independent set, or a vertex cover of G, respectively, before S_{i+1} has been received.

Corollary 4. *A graph G on n vertices can be preprocessed in $O(n^2)$ time such that for an online sequence S of subsets S_1,...,S_ℓ of vertices in G, for $i = 1, ..., \ell$, one can determine if S_i is a dominating set, an independent set, or a vertex cover of G before receiving S_{i+1} (in $i < \ell$ case) in $\tilde{O}(n(\ell^{1+o(1)} + ST_S))$ total time with high probability, where ST_S is the minimum cost of a spanning tree of the characteristic vectors representing the subsets in S under the extended Hamming distance. Using the same preprocessing, for an online sequence $v_1, ..., v_\ell$ of query vertices, for $i = 1, ..., \ell$, one can determine if v_i belongs to a triangle in G before receiving v_{i+1} (in case $i < \ell$) in $\tilde{O}(n(\ell^{1+o(1)} + ST_G))$ total time with high probability.*

Proof. Recall that a subset S_i of vertices in G can be easily represented by an n-dimensional Boolean column vector w_i with 1 on the j-th coordinate iff the j-th vertex belongs to S_i. Then, S_i is independent in G iff the vector u_i resulting from multiplying the adjacency matrix of G with w_i has zeros on the coordinates corresponding to the vertices in S_i. Next, S_i is a dominating set of G iff each vertex in $V \setminus S_i$ has a neighbour in S_i, i.e., iff u_i has ones on the coordinates corresponding to vertices in $V \setminus S_i$. Finally, S_i is a vertex cover of G iff $V \setminus S_i$ is an independent set of G, i.e., iff the vector resulting from multiplying the adjacency matrix of G with the complement of w_i has zeros on the coordinates corresponding to the vertices in $V \setminus S_i$.

Hence, it is sufficient to plug in our solution given in Theorem 5 to obtain the theorem. The preprocessing of G consists just in the construction of its adjacency matrix in $O(n^2)$ time. Note also that the extended Hamming distance between two 0-1 strings is equal to the extended Hamming distance between between the complements of these two strings. Thus, the upper bound in terms of ST_S is also valid in case of vertex cover. □ □

5 Final Remarks

Our results in Section 3 imply that to prove the conjectures OMv, AMv and MvP it is sufficient to consider $n \times n$ Boolean matrices where ST_M is almost quadratic in n.

Interestingly enough, our approximate nearest-neighbour heuristic for MST combined with the standard MST doubling and shortcuttings techniques immediately yields a corresponding online heuristic for TSP in metric spaces. By Theorem 4, it provides TSP tours TSP_s of length at most $2\lceil \log_2 s \rceil f(s)$ times larger than the optimum, where s is the number of input vectors and $f(s)$ is an upper bound on the approximation factor in the approximate nearest neighbour subroutine. The resulting TSP heuristic for $i = 2, ...$ simply finds an $f(i)$-nearest neighbour u of the new vector v_i and replaces the edge between u and its predecessor w by the path $\{w, v_i\}$, $\{v_i, u\}$ in TSP_{i-1} in order to obtain TSP_i.

Acknowledgements

CL, JJ and MP were supported in part by Swedish Research Council grant 621-2017-03750.

References

[1] A. Andoni and P. Indyk, *Nearest Neighbours in High-dimensional Spaces*. 43rd chapter in Handbook of Discrete and Computational Geometry, J.E. Goodman, J. O'Orourke and C.D. Toth (editors), 3rd edition, CRC Press, Boca Raton, FL, 2017.

[2] N. Bansal and R. Williams. *Regularity Lemmas and Combinatorial Algorithms.* Theory of Computing, Vol. 8, No. 1, pp. 69-94, 2012.

[3] A. Björklund and A. Lingas. *Fast Boolean matrix multiplication for highly clustered data.* Proc. of WADS 2001, LNCS Vol. 2125, pp. 258-263.

[4] D. Chakraborty, L. Kamma, and K. Green Larsen *Tight cell probe bounds for succinct Boolean matrix-vector multiplication.* Proc. of STOC 2018, pp. 1297-1306.

[5] P. Floderus, J. Jansson, C. Levcopoulos, A. Lingas, D. Sledneu. *3D Rectangulations and Geometric Matrix Multiplication.* Algorithmica 80(1): 136-154 (2018)

[6] L. Gąsieniec and A. Lingas. *An Improved Bound on Boolean Matrix Multiplication for Highly Clustered Data.* Proc. of WADS 2003, LNCS Vol. 2748, pp. 329-339.

[7] K. Green Larsen, R.R. Williams: *Faster Online Matrix-Vector Multiplication.* Proc. of SODA 2017, pp. 2182-2189.

[8] M. Henzinger, S. Krinninger, D. Nanongkai and T. Saranurak *Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture* Proc. of STOC 2015 (also presented at HALG 2016).

[9] M. Imase and B.M. Waxman, *Dynamic Steiner Tree Problem*. SIAM J. Discrete Math., 4(3), pp. 369-384.

[10] R. Williams, *Matrix-vector multiplication in sub-quadratic time (some preprocessing required).* Proc. of SODA 2007, pp. 995-2001.