Matrix-Vector Multiplication in Sub-Quadratic Time (Some Preprocessing Required)

Ryan Williams[*]

Abstract

We show that any $n \times n$ matrix A over any finite semiring can be preprocessed in $O(n^{2+\varepsilon})$ time, such that all subsequent vector multiplications with A can be performed in $O(n^2/(\varepsilon \log n)^2)$ time, for all $\varepsilon > 0$. The approach is combinatorial and can be implemented on a pointer machine or a $(\log n)$-word RAM. Some applications are described.

1 Introduction

Matrix-vector multiplication is an absolutely fundamental operation, with countless applications in computer science and scientific computing. Efficient algorithms for matrix-vector multiplication are of paramount importance. However, the sheer size of the matrix can be an issue: if the matrix is dense, then $\Omega(n^2)$ time is certainly required for an $n \times n$ matrix.

Suppose one allows for a slightly superquadratic ($n^{2+\varepsilon}$, for arbitrarily small $\varepsilon > 0$) preprocessing of the matrix. How quickly can matrix-vector multiplication be done then? This question has been studied since the beginning of scientific computing, but with an almost exclusive focus on special, structured matrices (such work is briefly surveyed in the next section). To our knowledge, the general case of matrix-vector multiplication has not received attention.

We first show how to matrix-vector multiply over the Boolean semiring in sub-quadratic time with preprocessing, for arbitrary matrices. More precisely, the following theorem is proved.

Theorem 1.1 *For all $\varepsilon \in (0, 1/2)$, every $n \times n$ Boolean matrix A can be preprocessed in $O(n^{2+\varepsilon})$ time such that every subsequent multiplication of A with an arbitrary Boolean n-vector x can be performed in $O(n^2/(\varepsilon \log n)^2)$ time, on a pointer machine or a $(\log n)$-word RAM.*

Both the preprocessing and matrix-vector multiplication algorithm are of the combinatorial/"non-algebraic" variety. At a high level, the preprocessing algorithm encodes $O(n^2/(\varepsilon \log n)^2)$ lookup tables for the matrix in a directed graph that "correlates" the tables; the multiplication algorithm looks at one entry from each of the tables, using the graph to merge results.

An immediate corollary of the theorem is a new combinatorial $O(n^3/(\log n)^2)$ Boolean matrix multiplication algorithm, obtained by performing a matrix-vector multiplication for n times. In fact, the theorem implies that Boolean matrix multiplication can be computed *on-line* in $O(n^3/(\log n)^2)$

[*]Computer Science Dept., Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh PA 15213. Email: `ryanw@cs.cmu.edu`. Partially supported by NSF CCR-0122581 and CCR-0085982 (ALADDIN), and by a grant from Google, Inc.

time, in the following sense: given an $n \times n$ matrix A, and supposing that the columns of an $n \times n$ matrix B are revealed one at a time, one can maintain the product of A with the currently known B in $O(n^2/(\log n)^2)$ time per column. Another application of our algorithm is that graphs can be preprocessed so that certain queries can be answered faster. For example, one can determine if a given vertex subset is a dominating set in $O(n^2/(\log n)^2)$ time.

Building on our results in the Boolean setting, we generalize our algorithms to show that matrix-vector multiplication over *any* finite semiring can be sped up with preprocessing.

Theorem 1.2 *Let $(R, +, \times)$ be a semiring on K elements. For all $\varepsilon \in (0, 1)$, every $n \times n$ matrix A over R can be preprocessed in $O(n^{2+\varepsilon \log_2 K})$ time such that every subsequent matrix-vector multiplication can be performed in $O(n^2/(\varepsilon \log n)^2)$ steps on a pointer machine or a $(\log n)$-word RAM, assuming operations in R take constant time.*

2 Preliminaries and Related Work

It would be infeasible to properly summarize all of the prior work on matrix preprocessing here; we shall settle for mentioning a few relevant (theory-based) references. For more information on this topic, the reader is referred to Pan [P01].

As mentioned earlier, the major theoretical work in matrix preprocessing has focused on structured matrices. For example, consider the case of Vandermonde matrices. In 1955, Motzkin [M55] introduced the idea of preprocessing a polynomial such that subsequent evaluations of the polynomial can be done more efficiently. Note that, after verifying a given matrix is Vandermode, computing a matrix-vector product with that matrix is equivalent to evaluating a univariate degree-n polynomial at n points. Hence Motzkin's ideas applied to preprocessing for Vandermonde matrix-vector multiplication. Later work on polynomial evaluation used ideas similar to the Fast Fourier Transform, obtaining algorithms for Vandermonde matrix-vector multiplication that run in $O(n \cdot poly(\log n))$ time after preprocessing (cf. [MB72, F72, BM75]).

In more recent work, Gohberg and Olshevsky [GO94] have generalized previous results, showing that several classes of structured matrices (including Vandermonde) admit matrix-vector multiplication in $O(n \log^2 n)$ time, after preprocessing. Substantial works continuing in this direction have discovered fast matrix-vector multiplication algorithms for a large variety of structured matrices (*e.g.* Olshevsky and Shokrollai [OS00] and Pan [P00, P01]).

Besides the work on structured matrices, another related result is that of Savage [S74] who showed thirty years ago that $n \times n$ matrix-vector multiplication over semirings with s elements can be performed in $O(n^2/\log_s n)$ arithmetic operations, *without* preprocessing(!). The key phrase here is *"arithmetic operations"*: while only $o(n^2)$ additions and multiplications are indeed performed, all n^2 entries of the matrix must obviously be read[1]. Thus Savage's algorithm is not sub-quadratic when one counts its steps on a typical machine model. (In interesting contrast, Winograd [W70] showed that in general, the number of arithmetic operations necessary to multiply an $n \times n$ matrix A with an n-vector x is the optimal $\Omega(n^2)$ bound. The catch is that his proof requires the underlying algebra to have an unbounded number of elements.)

[1]Very briefly, Savage's approach is to generate a table of all $O(\log n)$-vectors over the semiring in a Gray-code ordering, and compute the dot product of each vector on the list with various components of the input vector. The Gray code ensures that each dot product takes $O(1)$ additions and multiplications, amortized.

Let us briefly review some past combinatorial (non-algebraic) approaches to matrix multiplication. The "Four Russians" algorithm of Arlazarov *et al.* [ADKF70] performs Boolean $n \times n$ matrix multiplication in $O(n^3/\log n)$ time. (For a reference in English, cf. Aho, Hopcroft, and Ullman's book [AHU74].) Savage [S74] and Santoro [S80] observed that this time bound extends to a wide range of algebraic structures, assuming constant time arithmetic. This was eventually improved slightly to $O(n^3/\log^{3/2} n)$ time by Atkinson and Santoro [AS88]. Rytter [R85] (and independently Basch, Khanna, and Motwani [BKM95]) gave an $O(n^3/\log^2 n)$ algorithm for Boolean matrix multiplication on the $(\log n)$-word RAM. Chan [C06] has recently suggested (see the Discussion in the citation) that the original Arlazarov *et al.* algorithm may be modified to run in $O(n^3/\log^2 n)$ on a pointer machine, although the (allegedly messy) details are not provided.

Why Our Approach is Different. To our knowledge, none of the above algorithms can be modified to solve the matrix-vector multiplication problem in $O(n^2/\log^2 n)$ time. We shall attempt to give some brief intuition as to why this is the case. All above algorithms partition *both* input matrices into small blocks, where multiplications for the blocks are preprocessed and solved in advance. For example, the $\log^2 n$ speedup in matrix multiplication comes from a product of *two* $\log n$ speedups, one from each input matrix. In our problem, as only one matrix is being preprocessed, a $\log^2 n$ speedup of matrix-vector multiplication is not possible in the same manner[2]. Therefore we believe our approach to be a truly different method, in this regard.

A Note on Word Tricks. It is sometimes the case that logarithmic speedups come from "word tricks" that exploit the word parallelism of a RAM. We do not explicitly use word tricks in our algorithms; however, we do need a form of table lookup. The only "suspicious" operation required is that a list of pointers to the neighbors of a node v in a graph can be obtained in $O(deg(v))$ time, where $deg(v)$ is the degree of v. In the literature, this is known as a *neighborhood query* and can be implemented on a pointer machine or $(\log n)$-word RAM using a simple adjacency list representation. Our algorithm uses this operation to encode multiple lookup tables in a common graph. If one assumes that a neighborhood query requires $\Theta(deg(v) \cdot \alpha(n))$ time for some function α, our algorithm still takes only $O(n^2\alpha(n)/\log^2 n)$ to perform a matrix-vector multiplication.

Semirings. We recall the definition of a semiring. A semiring is a triple $(R, +, \times)$ such that R is a set with distinguished elements 0 and 1, and $+$ and $\times$ are binary operations over R, satisfying the axioms:

- $(R, +)$ is a commutative monoid with identity 0,
- $x \times 0 = 0 \times x = 0$
- $(R, \times)$ is a monoid with identity 1,
- $\times$ distributes over $+$.

More succinctly, a semiring is essentially a ring that is not required to have additive inverses. (Thus every ring is also a semiring.) For brevity we often refer to the semiring $(R, +, \times)$ as just "R" when there is no chance of confusion.

[2]Avrim Blum has pointed out to us that a $\Theta(\log n)$-speedup of matrix-vector multiplication is indeed possible using a "Four Russians" approach. We invite the reader to verify this.

3 Fast Boolean matrix-vector multiplication

We begin by considering matrix-vector multiplication in the Boolean semiring (where $R = \{0, 1\}$, $+$ is OR, and $\times$ is AND). The main ideas introduced here carry over to the general case. In the following, let n be a positive integer, A be an $n \times n$ Boolean matrix, x be a Boolean n-vector, and $V = \{1, \ldots, n\}$.

Definition 3.1 *For $S \subseteq V$, the* neighborhood of S in directed graph (V, E) *is the set*

$$N(S) := \{j \in V \mid (\exists k \in S)[(k, j) \in E]\}.$$

Recall there is a simple correspondence between Boolean matrix-vector multiplication and neighborhood computations in a directed graph.

Lemma 3.1 *Let $G_A = (V, E)$ be the directed graph corresponding to A, and let x be an indicator vector for a subset $S \subseteq V$. Then $A^T x$ is the indicator vector for $N(S)$.*

Proof. Let $j \in V$. Then

$$\begin{aligned} (A^T x)[j] = 1 &\iff \textstyle\bigvee_{k=1}^{n}(A^T[j,k] \wedge x[k]) = 1 \\ &\iff \textstyle\bigvee_{k=1}^{n}(A[k,j] \wedge x[k]) = 1 \\ &\iff (\exists k \in S)[(k,j) \in E \text{ and } k \in S] \\ &\iff j \in N(S). \end{aligned}$$

□

Therefore a Boolean matrix-vector multiplication is equivalent to computing the neighborhood of a given node subset. From here on, we focus on the problem of preprocessing a graph such that neighborhood queries for a given node subset can be done in $O(n^2/(\log n)^2)$ time.

3.1 Neighborhood Queries for Node Subsets

In this section, we establish Theorem 1.1 from the Introduction. Given a graph G, the preprocessing phase constructs a new graph H on $O(n^{1+\varepsilon}/(\varepsilon \log n))$ nodes and $O(n^{2+\varepsilon}/(\varepsilon \log n)^2)$ edges. Neighborhood subset queries shall be handled by performing local operations on portions of H. Intuitively, the graph H encodes $O(n^2/(\varepsilon \log n)^2)$ different lookup tables, one for each $(\varepsilon \log n) \times (\varepsilon \log n)$ block of the adjacency matrix of G.

Without loss of generality, assume that $\lceil \varepsilon \log n \rceil$ divides n. For $i = 1, \ldots, n/\lceil \varepsilon \log n \rceil$, define

$$P_i = \{(i-1) \cdot \lceil \varepsilon \log n \rceil + 1, \ldots, i \cdot \lceil \varepsilon \log n \rceil\},$$

and define Π to be the partition $\{P_1, \ldots, P_{n/\lceil \varepsilon \log n \rceil}\}$ of $[n]$.

The graph H has two layers of nodes, both having $O(n^{1+\varepsilon}/(\varepsilon \log n))$ nodes. In particular both layers have $n/\lceil \varepsilon \log n \rceil$ "groups", of $O(n^{\varepsilon})$ nodes each[3]. Each group corresponds to a part P_i of Π, and each of the $O(n^{\varepsilon})$ nodes in a group of H represents one of the possible subsets of P_i. Therefore the number of nodes is $O(n^{1+\varepsilon}/(\varepsilon \log n))$.

Our query algorithm uses the simple fact that a subset of nodes of G can be represented as a set of $n/(\varepsilon \log n)$ nodes in a layer of H.

[3]In the following, we omit the ceilings for notational convenience.

Definition 3.2 *Let x be a Boolean n-vector, and let $\ell \in \{1,2\}$. The* **ℓth layer node representation** *of x is the unique list of vertices $v_1, \ldots, v_{n/(\varepsilon \log n)}$ in H such that*

- *for all i, v_i is in the ith group of layer ℓ, and*
- *the indicator vector for the subset corresponding to v_i is equal to $x[(i-1)\lceil \varepsilon \log n \rceil + 1, \ldots, i\lceil \varepsilon \log n \rceil]$,* i.e. *bits $(i-1)\lceil \varepsilon \log n \rceil + 1$ through $i\lceil \varepsilon \log n \rceil$ of x.*

That is, each v_i corresponds to a distinct set of $\varepsilon \log n$ bits of x.

Definition 3.3 *Let S be a set of $n/(\varepsilon \log n)$ nodes, from either the first or second layer of H, such that each node is from a different group. The* **vector representation of** *S is the indicator vector for the set obtained by taking the union of all node sets corresponding to the nodes in S.*

We remark that on a computational model with $\Theta(\mathit{poly}(\log n))$ cost per random access, the two representations above can be computed in $O(n \cdot \mathit{poly}(\log n))$ time. In one matrix-vector multiplication, both of these representations are computed only once.

We now specify where the edges of H are placed.

First Layer Edges of H. Let v be a node in the first layer. Recall v corresponds to a subset S_v of V, of size at most $\varepsilon \log n$. Note that $N(S_v)$ can easily be determined in $O(n \cdot \varepsilon \log n)$ time[4]. Let $x_1, \ldots, x_{n/(\varepsilon \log n)}$ be the second-layer node representation of $N(S_v)$. Then the edges out of v are defined to be (v, x_1), …, $(v, x_{n/(\varepsilon \log n)})$. The number of outgoing edges from nodes in the first layer is $O(n^{1+\varepsilon}/(\varepsilon \log n) \cdot n/(\varepsilon \log n)) = O(n^{2+\varepsilon}/(\varepsilon \log n)^2)$. As there are $O(n^{1+\varepsilon}/(\varepsilon \log n))$ nodes in the first layer, the above edges can be determined in $O(n^{1+\varepsilon}/(\varepsilon \log n) \cdot n \cdot \varepsilon \log n) = O(n^{2+\varepsilon})$ time.

Second Layer Edges of H. In the second layer, there is an edge (u, v) between u and v in the same group if and only if, when construed as subsets of V, u is a subset of v. Therefore, the second layer consists of $n/(\varepsilon \log n)$ disjoint copies of the same n^{ε}-node DAG. (In particular, this DAG is the transitive closure of the directed hypercube on $2^{\varepsilon \log n}$ nodes.) Thus the number of edges between nodes in the second layer is $O(n/(\varepsilon \log n) \cdot n^{2\varepsilon}) = O(n^{1+2\varepsilon}/(\varepsilon \log n))$.

The figure below gives a bird's eye view of H.

Processing Neighborhood Subset Queries. Neighborhood queries for a node subset are performed on H as follows.

1. Given a node subset S as a Boolean n-vector, determine the first-layer node representation of S. Let T be the set of $n/(\varepsilon \log n)$ nodes in this representation.
2. Put a *mark* on every node of the set $N(T)$ in H. Recall that the degree of each node in the first layer is $n/(\varepsilon \log n)$. Assuming the neighbors of a node v can be marked in $O(\mathit{deg}(v))$ time, the nodes of $N(T)$ can be marked in $O(|T| \cdot \frac{n}{(\varepsilon \log n)}) = O(n^2/(\varepsilon \log n)^2)$ time. (Note this stage is the bottleneck in the algorithm's runtime.)

[4]With possibly a $\log n$ factor extra in computational models with limited random access.

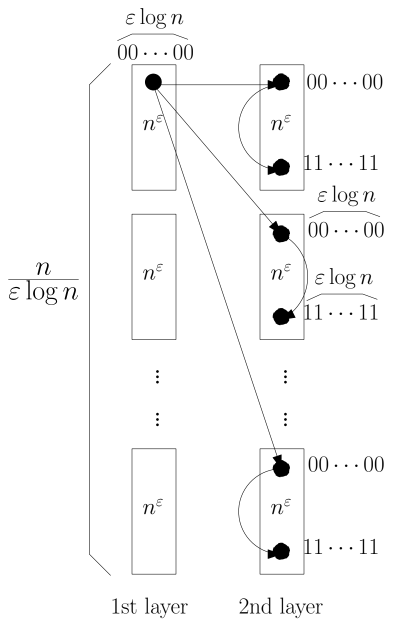

Figure 1: The graph H. Both layers have $n^{1+\varepsilon}/(\varepsilon \log n)$ nodes, divided into $n/(\varepsilon \log n)$ groups. Each node in H corresponds to a subset of nodes in G, with cardinality at most $\varepsilon \log n$. In the example above, the vertices labelled $00\cdots00$ represent empty sets, and the vertices labelled $11\cdots11$ represent the set of all $\varepsilon \log n$ nodes in that respective part. Since the empty set node in the first layer has no neighbors, its edges point to the empty sets of the nodes in the second layer.

3. Fix a topological order on nodes to be used for all groups (recall that each group in the second layer is a copy of a certain DAG). Each group of n^{ε} nodes in the second layer is processed separately as follows. *Specially mark* the first node u in the topological order such that all marked nodes in the group have an edge to u. To do this, obtain a count t of the total number of marked nodes in the group, then for each node in order, count its marked predecessors and compare that count with t. This needs at most $O(n^{2\varepsilon} \log n)$ time per group, as each edge is accessed at most once. The total runtime for this stage is $O\left(\frac{n}{\varepsilon \log n} \cdot n^{2\varepsilon} \log n\right) = O(n^{1+2\varepsilon})$. For $\varepsilon < 1/2$, this is less than $O(n^2/(\varepsilon \log n)^2)$.

4. Observe that each group has exactly one specially marked node. Compute the vector representation of the set of specially marked nodes, and finally, erase all node markings.

This concludes the description of the neighborhood query algorithm. Observe that the runtime of the query algorithm is dominated by the second step. The next lemma proves correctness.

Lemma 3.2 *The vector representation of the set of specially marked nodes is $N(S)$.*

Proof. For $j = 1, \ldots, n/(\varepsilon \log n)$, let S_j denote the subset of S restricted to part P_j. For each $i = 1, \ldots, n/(\varepsilon \log n)$, consider the ith group of nodes in the second layer of H, representing a subset of nodes of G in part P_i. When a node v in the ith group is marked, then there is a j such that the node representing S_j in the first layer has an edge to v. By construction of H, the nodes of G in the subset corresponding to v are exactly the neighbors of S_j that lie in P_i.

It follows that the collection of all marked nodes in the ith group corresponds to *all* neighbors of S that lie in the part P_i. More precisely, the *union* of the node sets in G corresponding to the marked nodes in group i of H gives bits $(i-1)\cdot\lceil\varepsilon\log n\rceil+1$ through $i\cdot\lceil\varepsilon\log n\rceil$ of the indicator vector for $N(S)$. Over all $n/(\varepsilon\log n)$ groups, these unions give all the bits in the indicator vector for $N(S)$.

We claim that the third step in the algorithm accomplishes exactly this, in that for each group, the algorithm specially marks the node corresponding to the union of the sets represented by marked nodes in that group. By finding a node u in a group such that all marked nodes in that group have edges to it, this implies that the node subset in G corresponding to u is a superset of all node sets in G corresponding to the marked nodes. By finding the first node in the topological ordering with this property, the node u is the smallest cardinality node subset in G that is a superset of all sets in G corresponding to the marked nodes. That is, u corresponds to the union of the sets represented by marked nodes in the current group. $\square$

3.2 Application: Faster Graph Subset Queries

Beyond on-line matrix multiplication, another application of faster Boolean matrix-vector multiplication is that certain types of graph queries can be answered more rapidly than one might initially believe is possible. Here we give a few examples of such queries.

Recall that a subset S of vertices is *dominating* if and only if every node in the graph is either in S or has a neighbor in S, S is *independent* if and only if there is no edge between any two nodes in S, and S is a *vertex cover* if every edge has an endpoint in S. Note S is independent if and only if $V-S$ is a vertex cover. The following proposition is straightforward.

Proposition 1 *$S\subseteq V$ is a dominating set if and only if $N(S)\cup S=V$. $S\subseteq V$ is independent if and only if $N(S)\cap S=\varnothing$.*

Hence the Boolean matrix-vector multiplication algorithm can be used to more efficiently determine if a given query subset S is dominating, independent, or a vertex cover.

Corollary 3.1 *A graph G can be preprocessed in $O(n^{2+\varepsilon})$ time such that one can determine in $O(n^2/(\varepsilon\log n)^2)$ time if a given subset of nodes is dominating, independent, or a vertex cover.*

The problem of finding an independent set or dominating set is quite general. As a result, certain other graph queries can also be performed in subquadratic time as well. For example, one can determine if a given query node participates in a triangle.

Corollary 3.2 *A graph G can be preprocessed in $O(n^{2+\varepsilon})$ time such that one can determine in $O(n^2/(\varepsilon\log n)^2)$ time if a given query node is in a 3-cycle.*

Proof. Let v_u be the neighborhood vector for the query node u, obtainable in $O(n)$ time. Determine if the set of vertices denoted by v_u is independent, in $O(n^2/\log^2 n)$ time. But this set is independent if and only if every pair of neighbors of u do not have an edge between them, *i.e.* if and only if u does not participate in a 3-cycle. $\square$

Note that in the absence of preprocessing, these tasks require $\Omega(n^2)$ time on dense graphs.

4 Matrix-vector multiplication over finite semirings

The matrix-vector multiplication scheme can be extended to all finite semirings, with a few modifications. We recall the statement of Theorem 1.2 from the Introduction for convenience:

THEOREM 1.2: *Let $(R, +, \times)$ be a semiring on K elements. For all $\varepsilon \in (0, 1)$, every $n \times n$ matrix A over R can be preprocessed in $O(n^{2+\varepsilon \log_2 K})$ time such that every subsequent matrix-vector multiplication can be performed in $O(n^2/(\varepsilon \log n)^2)$ steps, assuming operations in R take constant time.*

Proof. As in the Boolean case, we build a graph H with two layers, and each layer has $n/(\varepsilon \log n)$ groups of nodes, where each group represents a part P_i. However, four changes are made to the preprocessing phase:

1. The number of nodes in a group is now $K^{\varepsilon \log_2 n} = n^{\varepsilon \log_2 K}$, for all of the possible values of a vector over R with $\varepsilon \log n$ components.

2. There are no edges between nodes in the second layer.

3. The edges from the first layer to the second layer are redefined as follows. Each $\varepsilon \log n$ part of an input vector, when multiplied with the corresponding $n \times \varepsilon \log n$ submatrix of A, contributes an n-vector to the overall matrix-vector product. For a node u in the first layer, let x_u be its corresponding $\varepsilon \log n$ vector over R, and let y_u be the corresponding n-vector obtained from left-multiplying x_u by the proper $n \times \varepsilon \log n$ submatrix of A. Finally, let $v_1, \ldots, v_{n/(\varepsilon \log n)}$ be the second layer node representation of the vector y_u. Then u has outgoing edges to $v_1, \ldots, v_{n/(\varepsilon \log n)}$. For each node u, its neighbors can be computed in $O(n \cdot \varepsilon \log n)$ steps. Therefore $O(n^{1+\varepsilon \log_2 K}/(\varepsilon \log n) \cdot n \cdot \varepsilon \log n) = O(n^{2+\varepsilon \log_2 K})$ semiring operations and computation steps suffice for determining the edges.

4. For every node u in the second layer, maintain a variable c_u that is set to $0 \in R$ at the start of a query.

Multiplication of A with an n-vector x is performed as follows:

1. For all $n/(\varepsilon \log n)$ nodes in the first layer node representation of x, starting with the node in group 1, compute the node's list of neighbors $u_1, \ldots, u_{n/(\varepsilon \log n)}$ in the second layer, and set $c_{u_i} := c_{u_i} + 1$ for all i, where $+$ is over the semiring. These counts and neighbor queries can be computed in $O(n^2/(\varepsilon \log^2 n))$ additions and computation steps, using fast neighborhood queries and the assumption that operations in R take constant time.

2. For each node u in the second layer, let y_u be its corresponding $\varepsilon \log n$ vector over R. Set $y'_u := c_u \times y_u$. That is, y'_u is obtained by semiring-multiplying the scalar c_u with each component of y_u. This takes $O(\varepsilon \log n)$ arithmetic operations for each node in the second layer.

3. For each group $i = 1, \ldots, n/(\varepsilon \log n)$, compute the sum of all y'_u. That is, determine

$$z_i = \sum_{u \text{ in group } i} c_u y_u,$$

   and output the block vector $[z_1\ z_2\ \cdots\ z_{n/(\varepsilon \log n)}]^T$.

The multiplications take $O(n^{\varepsilon} \cdot poly(\log n))$ time per group, and the sum takes $O(n^{\varepsilon} \cdot poly(\log n))$ time as well. (Note that no table lookup is required to achieve these runtimes.) Hence the number of steps taken by stages 2 and 3 is $O(n^{1+\varepsilon} \cdot poly(\log n))$.

The runtime analysis is already embedded in the description of the algorithm. We now prove that the output vector is indeed Ax. We need to show that for the ith group of $\varepsilon \log n$ components of Ax, the corresponding sum z_i is equal to it. Without loss of generality, let us only consider group 1. Then formally our objective is to show $(Ax)[1, \ldots, \varepsilon \log n] = z_1$.

Observe that by construction, the number of times that an increment occurs to some c_u in group 1 is $n/(\varepsilon \log n)$. Let $u'_1, \ldots, u'_{n/(\varepsilon \log n)}$ be the temporal sequence of second layer nodes in group 1 whose $c_{u'_i}$'s were incremented, *i.e.* u'_1 is a neighbor of a first layer node in group 1, u'_2 is a neighbor of a first layer node in group 2, *etc.* (Notice that the sequence has repetitions, for $n^{\varepsilon} < n/(\varepsilon \log n)$.) Then, letting $y_{u'_i}$ be the vector corresponding to u'_i, $y_{u'_i} = A_i x_i$, where A_i is the matrix A restricted to rows $1, \ldots, \varepsilon \log n$ and columns $(i-1)\lceil \varepsilon \log n \rceil + 1, \ldots, i\lceil \varepsilon \log n \rceil$, and

$$x_i = x[(i-1)\lceil \varepsilon \log n \rceil + 1, \ldots, i\lceil \varepsilon \log n \rceil].$$

By distributivity of R, and commutativity of $+$,

$$\sum_{i=1}^{n/(\varepsilon \log n)} y_{u'_i} = \sum_{u \text{ in group } 1 : \exists i . u = u'_i} c_u y_u = \sum_{u \text{ in group } 1} c_u y_u,$$

the second equality following since all other c_u are 0.

Finally, by definition of A_i and x_i,

$$(Ax)[1, \ldots, \varepsilon \log n] = \sum_{i=1}^{n/(\varepsilon \log n)} (A_i x_i) = \sum_{i=1}^{n/(\varepsilon \log n)} y_{u'_i},$$

therefore the output of group 1 is indeed the first $\varepsilon \log n$ components of Ax, *i.e.* $(Ax)[1, \ldots, \varepsilon \log n] = z_1$. □

5 Conclusion

We have demonstrated how preprocessing makes it possible to achieve sub-quadratic matrix-vector multiplication for all constant-sized semirings. Our method implies an *on-line* combinatorial matrix multiplication algorithm, in the sense that a $\Theta(\log^2 n)$ speedup is achieved even if the columns of one input matrix are only revealed one at a time.

We conclude with three interesting open problems. First, it might be of more practical importance if one could obtain a speedup of matrix-vector multiplication for *sparse* matrices. A time bound of the form $O(m/poly(\log n) + n)$ is desirable, but it is not clear how to extend our ideas to this case. Another interesting question is whether or not the algebraic methods for matrix multiplication (such as Strassen's [S69], Coppersmith-Winograd's [CW90], *etc.*) can be applied to our problem of matrix-vector multiplication with preprocessing. This possibility seems unlikely to us, but we have not rigorously ruled it out. The power of algebraic matrix multiplication algorithms stems from the redundancy in multiplying different pairs of the same collections of vectors. Such redundancy is not present in matrix-vector multiplication.

Finally, is it possible to combine our preprocessing techniques with others to achieve a non-algebraic $o(n^3/\log^2 n)$ Boolean matrix multiplication algorithm? Achieving such an algorithm is an important open problem at the intersection of theory and practice that has resisted many efforts. The answer would be *yes* if, for example, it were possible to preprocess an $n \times n$ matrix so that it can be multiplied with an arbitrary $n \times \log n$ matrix in $o(n^2/\log n)$ time.

6 Acknowledgements

I thank Avrim Blum for introducing me to the problem of preprocessing a graph such that vertex cover queries can be efficiently computed. This investigation led to the present work. Thanks to the SODA referees for helpful comments and references to the literature.

References

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, 1974.

[ADKF70] V.Z. Arlazarov, E.A. Dinic, M.A. Kronrod and I.A. Faradzev. On economical construction of the transitive closure of a directed graph. *Doklady Adamedii Nauk SSSR* 194:487-488, 1970.

[AS88] M.D. Atkinson and N. Santoro. A practical algorithm for Boolean matrix multiplication. *Inf. Proc. Letters* 29:37–38, 1988.

[BKM95] J. Basch, S. Khanna, and R. Motwani. On Diameter Verification and Boolean Matrix Multiplication. Technical Report No. STAN-CS-95-1544, Department of Computer Science, Stanford University, 1995.

[BM75] A. Borodin and I. Munro. *The Computational Complexity of Algebraic and Numerical Problems.* Elsevier, New York, 1975.

[C05] T.M. Chan. All-Pairs Shortest Paths with Real Weights in $O(n^3/\log n)$ Time. *Proc. Workshop on Algorithms and Data Structures* (WADS), 318–324, 2005.

[C06] T.M. Chan. All-Pairs Shortest Paths for Unweighted Undirected Graphs in $o(mn)$ Time. *Proc. ACM-SIAM Symposium on Discrete Algorithms* (SODA), 514–523, 2006.

[CW90] D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. *J. Symb. Comput.* 9(3):251–280, 1990.

[F72] C.M. Fiduccia. Polynomial evaluation via the division algorithm– the Fast Fourier Transform revisited. *Proc. ACM Symposium on Theory of Computing* (STOC), 88–93, 1972.

[GO94] I. Gohberg and V. Olshevsky. Fast algorithms with preprocessing for matrix-vector multipication problems. *Journal of Complexity* 10(4):411–427, 1994.

[M55] T.S. Motzkin. Evaluation of polynomials and evaluation of rational functions. *Bulletin of American Mathematical Society* Vol. 61, 163, 1955.

[MB72] R. Moenck and A. Borodin. Fast modular transforms via division. *Proc. IEEE 13th Annual Symposium on Switching and Automata Theory*, 90–96, 1972.

[OS00] V. Olshevsky and A. Shokrollahi. Matrix-vector product for confluent Cauchy-like matrices with application to confluent rational interpolation. *Proc. ACM Symposium on Theory of Computing* (STOC), 573–581, 2000.

[P00] V.Y. Pan. Nearly optimal computations with structured matrices. *Proc. ACM-SIAM Symposium on Discrete Algorithms* (SODA), 953–962, 2000.

[P01] V.Y. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms.* Birkhäuser-Springer, Boston and New York, 2001.

[R85] W. Rytter. Fast Recognition of Pushdown Automaton and Context-free Languages. *Information and Control* 67(1-3):12–22, 1985.

[S74] J.E. Savage. An algorithm for the computation of linear forms. *SIAM J. Comput.* 3(2):150–158, 1974.

[S80] N. Santoro. Extending the Four Russians' Bound to General Matrix Multiplication. *Inf. Proc. Letters* 10(2):87–88, 1980.

[S69] V. Strassen. Gaussian elimination is not optimal. *Numer. Math.* 12:354–356, 1969.

[W70] S. Winograd. On the number of multiplications necessary to compute certain functions. *Comm. Pure and Applied Math.* 23:165–179, 1970.