Speeding up the Four Russians Algorithm by About One More Logarithmic Factor

Timothy M. Chan[*]

September 19, 2014

Abstract

We present a new combinatorial algorithm for Boolean matrix multiplication that runs in $O(n^3(\log\log n)^3/\log^3 n)$ time. This improves the previous combinatorial algorithm by Bansal and Williams [FOCS'09] that runs in $O(n^3(\log\log n)^2/\log^{9/4} n)$ time. Whereas Bansal and Williams' algorithm uses regularity lemmas for graphs, the new algorithm is simple and uses entirely elementary techniques: table lookup, word operations, plus a deceptively straightforward divide-and-conquer.

Our algorithm is in part inspired by a recent result of Impagliazzo, Lovett, Paturi, and Schneider (2014) on a different geometric problem, offline dominance range reporting; we improve their analysis for that problem as well.

1 Introduction

In this paper, we examine the well-known problem of multiplying two (dense) $n \times n$ Boolean matrices, and study "combinatorial" algorithms—in contrast to "algebraic" algorithms, like Strassen's and its successors [16, 18, 11], which work in some ring structure and exploit clever cancellations. The "Four Russians" algorithm, by Arlazarov, Dinic, Kronrod, and Faradzhev [2], is the earliest and most famous combinatorial algorithm for Boolean matrix multiplication. The time bound was originally stated as $O(n^3/\log n)$, but as noted subsequently, it can be made $O(n^3/\log^2 n)$ when implemented on the standard $(\log n)$-bit RAM model. On a RAM model with a larger word size $w > \log n$ with standard word operations, the time bound is $O(n^3/(w \log n))$.

The "Four Russians technique" has since become synonymous with logarithmic-factor speedups, a phenomenon that has spread to numerous other algorithmic problems. It was believed by some researchers (the present author included, embarrassingly) that the speedup of the two logarithmic factors was the best possible among combinatorial algorithms; indeed, the Four Russians algorithm was shown to be optimal for Boolean matrix multiplication under some restricted model of computation [1, 15].

That is why Bansal and Williams' FOCS'09 paper [3] came as a shock—they presented combinatorial algorithms for Boolean matrix multiplication that run in time $O(n^3(\log\log n)^2/\log^{9/4} n)$ for $w = \log n$, and $O(n^3(\log\log n)^2/(w\log^{7/6} n))$ for $w > \log n$. Their algorithms require advanced techniques, notably, regularity lemmas for graphs. Random sampling is also used, although the algorithms were later derandomized [19].

[*]Cheriton School of Computer Science, University of Waterloo (tmchan@uwaterloo.ca)

We present a still faster combinatorial algorithm for Boolean matrix multiplication that runs in time $O(n^3(\log\log n)^3/\log^3 n)$ for $w = \log n$,[1] and $O(n^3 \log^3 w/(w \log^2 n))$ for $w > \log n$. In contrast to Bansal and Williams', our algorithm is simple and our analysis is completely elementary.

Our work is indirectly inspired by a recent manuscript of Impagliazzo, Lovett, Paturi, and Schneider [13]. Motivated by a different problem, namely, 0-1 integer linear programming, they analyzed a simple divide-and-conquer algorithm for a computational geometry problem called *offline dominance range reporting*: given n red points and n blue points in d dimensions, report all K pairs (p, q) of a red point p dominating a blue point q, i.e., p having larger value than q in all d coordinates. In a previous work, the author [5] has shown the relevance of dominance range reporting to logarithmic-factor speedups in *min-plus matrix multiplication* and *all-pairs shortest paths*. Geometric divide-and-conquer that inductively reduces the dimension by 1 tends not to do well when the dimension exceeds $\log n$, because of exponential dependence on d—the so-called "curse of dimensionality". Impagliazzo et al.'s analysis shows surprisingly that their algorithm performs well even when the dimension goes slightly beyond $\log n$: they got about $n^{2-\Omega(1/(c^{15}\log c))} + K$ time for $d = c \log n$; for example, this means subquadratic time for dimensions as high as $\log^{1+1/15-\varepsilon} n$. (They probably did not try to optimize the exponent 15.) In the appendix, we give a finer analysis by a more straightforward induction proof, to show that the same algorithm has subquadratic complexity for dimensions as high as $(\log^2 n)/(\log\log n)^3$. This immediately implies a new combinatorial algorithm for all-pairs shortest paths in dense real-weighted graph with running time $O(n^3(\log\log n)^3/\log^2 n)$. This is simpler than the author's previous algorithm [6] with the same running time, in that we completely avoid any explicit use of "word tricks", although it is slower than Han and Takaoka's improved $O(n^3 \log\log n/\log^2 n)$-time algorithm [12] and Williams' recent $n^3/2^{\Omega(\sqrt{\log n})}$-time algorithm [17] (the latter requires algebraic techniques for rectangular matrix multiplication).

But what does all this have to do with Boolean matrix multiplication, where the goal is to get beyond speedup of two logarithmic factors? Following a similar line of thought, we propose and analyze a simple divide-and-conquer algorithm for multiplying an $n \times d$ and a $d \times n$ Boolean matrix. We show that our algorithm works well even when d is moderately large (e.g., polylogarithmic in n), *provided that* there is a subroutine to directly handle the case when one of the matrices is sparser by about a log factor. The overall running time is then determined by this sparser case, which can be solved by a variant of the Four Russians algorithm: to summarize, one log factor speedup comes from table lookup, another log from using word operations, and about one more log from sparseness as a result of the divide-and-conquer.

2 Boolean Matrix Multiplication

Preliminaries. We focus on the problem of computing the product of an $n \times d$ and a $d \times n$ Boolean matrix. We can just multiply by n/d to upper-bound the complexity of multiplying two $n \times n$ Boolean matrices.

We consider a slight generalization of this rectangular Boolean matrix multiplication problem, where the first matrix is $m \times d$ and the second is $d \times n$. We find it convenient to rephrase the problem as a d-dimensional "geometric" problem as follows: given a set A of m Boolean d-vectors

[1]As in previous algorithms, the algorithm for RAM with word size $w = \log n$ can be modified to work on pointer machines, with some extra effort. (Word operations for $w = \log n$ can be simulated by table lookup, and a batch of table lookup queries can be handled by a radix-sort-like technique [7].)

and a set B of n Boolean d-vectors, report all pairs in $A \times B$ that have Boolean inner product equal to 1. (More precisely, we want to output the labels to the vectors in each such pair; each pair is to be reported exactly once.)

We first describe a subroutine to solve the problem when one of the input sets is not fully dense. Incidentally, this gives us an opportunity to review the techniques behind the Four Russians algorithm, as our subroutine is a modification of that algorithm, using table lookup and bit packing.[2] This subroutine is the only place where we explicitly use word operations (bitwise-or).

Lemma 2.1. *In the case when B has a total of βdn 1's for some $\beta \leq 1$, the problem can be solved in $O\left(dn + dmn^{0.01} + \frac{\beta dmn \log d}{w \log n} + mn\right)$ time.*

Proof. First pre-compute the inner product of each vector of A with all possible vectors v that have at most $b := \frac{0.01 \log n}{\log(d+1)}$ 1's. The number of such vectors v is at most $(d+1)^b = n^{0.01}$; for each v, the answers are a sequence of m bits and can be packed in $O(m/w + 1)$ words. The pre-computation naively takes $O(dmn^{0.01})$ time. Now divide each vector of B into chunks with up to b 1's, use table lookup to find the answers for each chunk, and combine the answers by taking bitwise-or. The total number of chunks is $O(\beta dn/b + n)$, and each chunk is handled in $O(m/w + 1)$ time, for a total cost of $O((\beta dn/b + n)(m/w + 1))$. The initial division into chunks costs $O(dn)$ time. Reporting the labels to the output pairs costs an additional $O(mn)$ time. □

The Algorithm. Let β_0 be a fixed parameter to be set later. Our algorithm is based on divide-and-conquer and is deceptively simple:

0. If B has at most $\beta_0 dn$ 1's, then directly solve the problem by Lemma 2.1 and return.
1. Otherwise, find a coordinate position that has at least $\beta_0 n$ 1's among the vectors of B. We may assume that it is the first coordinate, by swapping coordinates in all vectors.
2. Recursively solve the problem for all the vectors of A that start with 0 and all the vectors of B; the first coordinate can be dropped from all vectors.
3. Recursively solve the problem for the vectors of A that start with 1 and the vectors of B that start with 0; the first coordinate can be dropped from all vectors.
4. Directly report all pairs of vectors of A that start with 1 and vectors of B that start with 1.

The Recurrence. Let $T_d(m, n)$ denote the running time of the algorithm. After step 1, let αm be the number of vectors of A that start with 1 and βn be the number of vectors of B that start with 1, with $0 \leq \alpha \leq 1$ and $\beta_0 \leq \beta \leq 1$. Then step 2 takes $T_{d-1}((1-\alpha)m, n)$ time and step 3 takes at most $T_{d-1}(\alpha m, (1-\beta)n)$ time, excluding a cost of $O(dn + dm)$ to form the input to both

[2]Feder and Motwani [9] obtained a similar result, but for multiplying two square $n \times n$ Boolean matrices rather than two rectangular matrices. They used combinatorial techniques on graph compression to obtain a time bound $O(\beta n^3 \log(1/\beta)/(w \log n))$ when one of the matrices has βn^2 1's; Bansal and Williams' algorithm also requires this as a subroutine. Our lemma can provide an alternative derivation of Feder and Motwani's result.

subproblems. Step 4 takes $O(\alpha\beta mn)$ time. To summarize, we get the recurrence

$$T_d(m,n) \;\le\; \max\begin{cases} \displaystyle\max_{0\le\alpha\le 1,\,\beta_0\le\beta\le 1} [T_{d-1}((1-\alpha)m,n) + T_{d-1}(\alpha m,(1-\beta)n) + dn + dm + \alpha\beta mn] \\ \displaystyle dn + dmn^{0.01} + \frac{\beta_0 dmn\log d}{w\log n} + mn, \end{cases} \tag{1}$$

ignoring constant factors and the trivial base cases $T_d(m,0) = T_d(0,n) = T_0(m,n) = 0$.

Solving the Recurrence. The solution to the recurrence is not obvious. We resort to a "guess-and-check" approach: for a certain choice of parameters β_0, δ, and ε to be set later, we claim that

$$T_d(m,n) \;\le\; d^2(1+\delta)^d nm^{1-\varepsilon} \;+\; d^2mn^{0.01} \;+\; \frac{\beta_0 dmn\log d}{w\log n} \;+\; mn \tag{2}$$

and confirm the guess by induction.

If the maximum in (1) is attained by the bottom case, the claim is trivial. Otherwise, suppose that the maximum in the top case is attained by α and $\beta \ge \beta_0$. Assume inductively that (2) is true when (d,m,n) is replaced by $(d-1,(1-\alpha)m,n)$ and by $(d-1,\alpha m,(1-\beta)n)$. Then $T_d(m,n)$ is

$$\begin{aligned} \le\;\; & (d-1)^2(1+\delta)^{d-1}n((1-\alpha)m)^{1-\varepsilon} + (d-1)^2(1-\alpha)mn^{0.01} + \frac{\beta_0 d(1-\alpha)mn\log d}{w\log n} + (1-\alpha)mn \\ & + (d-1)^2(1+\delta)^{d-1}(1-\beta)n(\alpha m)^{1-\varepsilon} + (d-1)^2\alpha mn^{0.01} + \frac{\beta_0 d\alpha mn\log d}{w\log n} + \alpha(1-\beta)mn \\ & + dn + dm + \alpha\beta mn \\ \le\;\; & (d-1)^2(1+\delta)^{d-1}[(1-\alpha)^{1-\varepsilon}+\alpha^{1-\varepsilon}(1-\beta)]nm^{1-\varepsilon} + (d-1)^2mn^{0.01} + \frac{\beta_0 dmn\log d}{w\log n} \\ & + dn + dm + mn. \end{aligned}$$

The key lies in the following inequality:

$$(1-\alpha)^{1-\varepsilon} + \alpha^{1-\varepsilon}(1-\beta_0) \;<\; 1+\delta. \tag{3}$$

To prove (3), split into two cases. If $\alpha \le \delta^2$, then the left-hand side of (3) is at most $1+\alpha^{1-\varepsilon} \le 1+\delta$, assuming that $\varepsilon < 1/2$. If $\alpha > \delta^2$, then the left-hand side is at most

$$\begin{aligned} [1-(1-\varepsilon)\alpha] + \alpha(1-\beta_0)e^{\varepsilon\ln(1/\alpha)} \;&\le\; [1-(1-\varepsilon)\alpha] + \alpha(1-\beta_0)(1+3\varepsilon\ln(1/\delta)) \\ &\le\; 1 - \alpha\beta_0 + \alpha\varepsilon(1+3\ln(1/\delta)) \;=\; 1, \end{aligned}$$

by setting $\beta_0 = \varepsilon(1+3\ln(1/\delta))$, assuming that $\varepsilon\ln(1/\delta) < 0.1$.

Continuing the earlier chain of inequalities, we obtain from (3):

$$\begin{aligned} T_d(m,n) \;&\le\; (d-1)^2(1+\delta)^d nm^{1-\varepsilon} + (d-1)^2mn^{0.01} + \frac{\beta_0 dmn\log d}{w\log n} + dm + dn + mn \\ &\le\; d^2(1+\delta)^d nm^{1-\varepsilon} \;+\; d^2mn^{0.01} \;+\; \frac{\beta_0 dmn\log d}{w\log n} \;+\; mn. \end{aligned}$$

Conclusion. Putting $\delta = 1/d$, $\varepsilon = 2\log d/\log n$, $\beta_0 = \varepsilon(1 + 3\ln(1/\delta)) = \Theta(\log^2 d/\log n)$, and $d = w\log^2 n/\log^3 w$ into (2) yields a time bound of $T_d(n, n) = O(n^2)$. This is assuming that $\varepsilon\ln(1/\delta) = o(1)$, i.e., $\log^2 w = o(\log n)$. (Note that if $\log^2 w = \Omega(\log n)$, we would have $d = o(w\log n)$ and can switch to the original Four Russians algorithm.) We have thus shown:

Theorem 2.2. *For $d = w\log^2 n/\log^3 w$, there is a combinatorial algorithm to multiply any $n \times d$ and any $d \times n$ Boolean matrix in $O(n^2)$ time.*

Corollary 2.3. *There is a combinatorial algorithm to multiply any two $n \times n$ Boolean matrices in time $O(n^3\log^3 w/(w\log^2 n)) \le O(n^3(\log\log n)^3/\log^3 n)$.*

Many applications follow. For example, we immediately obtain a combinatorial algorithm with the same time bound for the problem of detecting triangles in a (dense) graph with n vertices.

3 Final Remarks

Our algorithm is still substantially slower, in theory, than "algebraic" algorithms with running time $O(n^\omega)$ for $\omega < 2.373$ [18, 11]. Formally we still lack a precise, natural definition of a model that includes algorithms like the Four Russians and the one from this paper but excludes Strassen's and its successors. On the other hand, the result from Theorem 2.2 on multiplying rectangular Boolean matrix multiplication is new, regardless of the distinction between combinatorial and algebraic algorithms (according to the current best algebraic results, we can multiply an $n \times n^{0.302}$ and $n^{0.302} \times n$ matrix in $O(n^{2+o(1)})$ arithmetic operations [10], or an $n \times n^{0.172}$ and $n^{0.172} \times n$ matrix in $O(n^2\log^2 n)$ operations [8], or an $n \times \log n$ and $\log n \times n$ matrix in $O(n^2)$ operations [4]).

Our work undermines the main message from Bansal and Williams' paper [3], that advanced graph-theoretic techniques such as regularity lemmas provide a fruitful avenue to obtain new results for Boolean matrix multiplication. To be fair, better time bounds such as $n^3/2^{\Omega(\sqrt{\log n})}$ might still be possible if there is further progress on regularity lemmas, as remarked in their paper (although no such improvements have been reported in the intervening few years).

Our approach and Bansal and Williams' exploit the special nature of the Boolean matrix multiplication problem; for example, they currently do not extend to matrix multiplication in the field F_2. However, they at least raise the possibility for improvements beyond two logarithmic factors in combinatorial algorithms for other problems, such as all-pairs shortest paths.

Acknowledgement. I thank Seth Pettie for asking a question about dominance range reporting which indirectly leads to this work.

References

[1] D. Angluin. The four Russians' algorithm for Boolean matrix multiplication is optimal for its class. *SIGACT News*, 1:29–33, 1976.

[2] V. Z. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzhev. On economical construction of the transitive closure of a directed graph. *Soviet Mathematics Doklady*, 11:1209–1210, 1970.

[3] N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. *Theory of Computing*, 8:69–94, 2012.

[4] R. W. Brockett and D. Dobkin. On the number of multiplications required for matrix multiplication. *SIAM Journal on Computing*, 5:624–628, 1976.

[5] T. M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50:236–243, 2008.

[6] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM Journal on Computing*, 39:2075–2089, 2010.

[7] T. M. Chan. All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time. *ACM Transactions on Algorithms*, 8(4):34, 2012.

[8] D. Coppersmith. Rapid multiplication of rectangular matrices. *SIAM Journal on Computing*, 11:467–471, 1982.

[9] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *Journal of Computer and System Sciences*, 51:261–272, 1995.

[10] F. Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 514–523, 2012.

[11] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the International Symposium on Algorithms and Computation*, pages 296–303, 2014.

[12] Y. Han and T. Takaoka. An $O(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths. In *Proceedings of the Scandinavian Symposium and Workshops on Algorithm Theory*, pages 131–141, 2012.

[13] R. Impagliazzo, S. Lovett, R. Paturi, and S. Schneider. 0-1 integer linear programming with a linear number of constraints. arXiv:1401.5512, Feb 2014.

[14] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer–Verlag, 1985.

[15] J. E. Savage. An algorithm for the computation of linear forms. *SIAM Journal on Computing*, 3:150–158, 1974.

[16] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.

[17] R. Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 664–673, 2014.

[18] V. Vassilevska Williams. Multiplying matrices faster than Coppersmith–Winograd. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 887–898, 2012.

[19] V. Vassilevska Williams and R. Williams. Subcubic equivalence between path, matrix and triangle problems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 645–654, 2010.

A Appendix: Offline Dominance Range Reporting

Impagliazzo et al.'s Algorithm. Given m red points and n blue points in d dimensions (which are assumed to be in general position for simplicity), Impagliazzo et al. [13] proposed a variant of a standard divide-and-conquer algorithm [5, 14] that can report all pairs (p, q) with a red point p dominating a blue point q. (The innovation lies in step 1.)

0. If $d = 0$, then directly report all pairs of red and blue points and return.

1. Find a value x such that [the number of red points with first coordinate at most x]$/m$ is equal to [the number of blue points with first coordinate at least x]$/n$. (The existence of x is guaranteed by the intermediate value theorem; we allow a point with first coordinate exactly x to contribute a fraction to either count.)

2. Recursively solve the problem for the red and blue points with first coordinate at least x.

3. Recursively solve the problem for the red and blue points with first coordinate at most x.

4. Recursively solve the problem for the red points with first coordinate at least x and the blue points with first coordinate at most x; the first coordinates can be dropped from all the points.

The Recurrence. Let $T_d(m, n)$ denote the running time of the algorithm, excluding the $O(K)$ cost for reporting the K output pairs. Step 1 can be done in linear time by applying a weighted median algorithm (by assigning each red point a weight of $1/m$ and each blue point a weight of $1/n$). Let α be the equal value in step 1. Then step 2 takes $T_d((1-\alpha)m, \alpha n)$ time, step 3 takes $T_d(\alpha m, (1-\alpha)n)$ time, and step 4 takes $T_{d-1}((1-\alpha)m, (1-\alpha)n)$ time, excluding an $O(dn + dm)$ cost to form the input to these subproblems.[3] To summarize, we get the recurrence

$$T_d(m,n) \;\le\; \max_{0\le\alpha\le 1} \left[T_d((1-\alpha)m, \alpha n) + T_d(\alpha m, (1-\alpha)n) + T_{d-1}((1-\alpha)m, (1-\alpha)n) + dn + dm\right], \tag{4}$$

ignoring constant factors and the trivial base cases $T_d(m, 0) = T_d(0, n) = T_0(m, n) = 0$.

Solving the Recurrence. For a certain choice of parameters δ and ε to be set later, we claim that

$$T_d(m,n) \;\le\; d(1+\delta)^d(nm^{1-\varepsilon} + mn^{1-\varepsilon})(\log(mn) + d) \tag{5}$$

and confirm the guess by induction.

Assume that (5) is true when (d, m, n) is replaced by $(d, (1-\alpha)m, \alpha n)$, by $(d, \alpha m, (1-\alpha)n)$, and by $(d-1, (1-\alpha)m, (1-\alpha)n)$. Then $T_d(m, n)$ is

$$\begin{aligned}
\le\;& d(1+\delta)^d[\alpha n((1-\alpha)m)^{1-\varepsilon} + (1-\alpha)m(\alpha n)^{1-\varepsilon}](\log((mn)/4) + d)\\
&+ d(1+\delta)^d[(1-\alpha)n(\alpha m)^{1-\varepsilon} + \alpha m((1-\alpha)n)^{1-\varepsilon}](\log((mn)/4) + d)\\
&+ d(1+\delta)^{d-1}[(1-\alpha)n((1-\alpha)m)^{1-\varepsilon} + (1-\alpha)m((1-\alpha)n)^{1-\varepsilon}](\log(mn) + d - 1)\\
&+ dn + dm\\
\le\;& d(1+\delta)^d\left[\alpha(1-\alpha)^{1-\varepsilon} + \alpha^{1-\varepsilon}(1-\alpha) + \frac{(1-\alpha)^{2-\varepsilon}}{1+\delta}\right](nm^{1-\varepsilon} + mn^{1-\varepsilon})(\log(mn) + d - 1)\\
&+ dn + dm.
\end{aligned}$$

The key lies in the following inequality:

$$\alpha(1-\alpha)^{1-\varepsilon} + \alpha^{1-\varepsilon}(1-\alpha) + \frac{(1-\alpha)^{2-\varepsilon}}{1+\delta} \;<\; 1. \tag{6}$$

[3]Technically, there should be a "+1" in the arguments because of the extra "fractional" point with first coordinate exactly x, but we can remove that point by directly reporting all dominating pairs involving it in $O(dn + dm)$ time.

To prove (6), split into three cases. If $\alpha \le \delta^2$, then the left-hand side of (6) is at most $\delta^2 + \delta^{2(1-\varepsilon)} + 1/(1+\delta) < 1$, assuming that $\delta, \varepsilon < 0.1$. If $\alpha \ge 2/3$, then the left-hand side is at most $(1/3)^{1-\varepsilon} + (1/3) + (1/3)^{2-\varepsilon}/(1+\delta) < 1$, assuming that $\delta, \varepsilon < 0.1$. If $\delta^2 < \alpha < 2/3$, then the left-hand side is at most

$$\begin{aligned}
&\alpha(1-\alpha)^{1-\varepsilon} + \alpha^{1-\varepsilon}(1-\alpha) + (1-\alpha)^{2-\varepsilon}(1-\delta/2) \\
&\le \alpha(1-\alpha)^{1-\varepsilon} + \alpha^{1-\varepsilon}(1-\alpha) + (1-\alpha)^{2-\varepsilon} - (1/3)^{2-\varepsilon}\delta/2 \\
&\le (1-\alpha)^{1-\varepsilon} + \alpha^{1-\varepsilon}(1-\alpha) - \delta/18 \\
&\le [1-(1-\varepsilon)\alpha] + \alpha(1-\alpha)e^{\varepsilon\ln(1/\alpha)} - \delta/18 \\
&\le [1-(1-\varepsilon)\alpha] + \alpha(1-\alpha)(1+3\varepsilon\ln(1/\delta)) - \delta/18 \\
&\le 1 - \alpha^2 + \alpha\varepsilon(1+3\ln(1/\delta)) - \delta/18.
\end{aligned}$$

assuming that $\varepsilon\ln(1/\delta) < 0.1$. If $\alpha \ge \varepsilon(1+3\ln(1/\delta))$, the above expression is less than 1. On the other hand, if $\alpha < \varepsilon(1+3\ln(1/\delta))$, it is less than $1 + \varepsilon^2(1+3\ln(1/\delta))^2 - \delta/18 = 1$, by setting $\varepsilon = \sqrt{\delta/18}/(1+3\ln(1/\delta))$.

Continuing the earlier chain of inequalities, we obtain from (6):

$$\begin{aligned}
T_d(m,n) &\le d(1+\delta)^d(nm^{1-\varepsilon} + mn^{1-\varepsilon})(\log(mn) + d - 1) + dn + dm \\
&\le d(1+\delta)^d(nm^{1-\varepsilon} + mn^{1-\varepsilon})(\log(mn) + d).
\end{aligned}$$

Conclusion. Putting $\delta = 1/(c^2\log^2 c)$, $\varepsilon = \sqrt{\delta/18}/(1+3\ln(1/\delta)) = \Theta(1/(c\log^2 c))$, and $d = c\log n$ into (5) yields $T_d(n,n) \le n^{2-\Theta(1/(c\log^2 c))}(c\log n)^{O(1)}$. We thus have the following theorem; the second sentence follows by setting c a small constant times $\log n/(\log\log n)^3$.

Theorem A.1. *There is a combinatorial algorithm to solve the offline dominance range reporting problem for two sets of n points in $d = c\log n$ dimensions with K output pairs in time $O(n^{2-\Omega(1/(c\log^2 c))}(c\log n)^{O(1)} + K)$. For d being a sufficiently small constant times* $\log^2 n/(\log\log n)^3$, *the time bound can be made $O(n^2/\log^{100} n + K)$.*

Many applications in computational geometry follow. For offline dominance range counting (counting the number of red points dominating each blue point), we obtain the same result but without the K term. We obtain similar results for general offline orthogonal range searching (which reduces to dominance range searching after doubling the dimension), offline exact L_∞-nearest neighbor search (which reduces to orthogonal range searching by binary search), and exact L_∞-minimum spanning tree (which reduces to offline nearest neighbor search by Borůvka's algorithm).

The result has implications beyond computational geometry. The author [5] has shown how to reduce the problem in the following corollary to d instances of the d-dimensional dominance range reporting problem, where the total number of output pairs in the d instances is $O(n^2)$.

Corollary A.2. *For d being a sufficiently small constant times* $\log^2 n/(\log\log n)^3$, *there is a combinatorial algorithm to compute the min-plus product of any $n \times d$ and $d \times n$ real-valued matrix in $O(n^2)$ time.*

Corollary A.3. *There is a combinatorial algorithm for computing the min-plus product of any two $n \times n$ real-valued matrices, and for solving the all-pairs shortest paths problem in any real-weighted graph, in $O(n^3(\log\log n)^3/\log^2 n)$ time.*

As noted in the introduction, the above result is not new, but the algorithm is simpler.