On logarithmic-factor speedup algorithms for some classic problems

YeahPotato

January 7, 2025

Abstract

This survey discusses techniques that speed up the time complexity of algorithms for certain problems (mainly those under the scope of fine-grained complexity) by a logarithmic factor, and provides a brief overview of works aimed at improving the lower bound. Four main problems are discussed: boolean matrix multiplication (introducing bit-parallelization and the method of four Russians), longest common subsequence, 3-sum, and all pairs shortest path.

1 Introduction

Among the history of algorithmic improvements for specific problems, one type of speedup is often overlooked compared to optimizations on polynomial order. Typically, we are concerned with problems such as, "Can we reduce the time from $O(n^2)$ to $O(n \log n)$?" or "Is it possible to remove the $\log n$ factor from $O(n \log n)$?".

In this survey, however, we focus on algorithms that run in $O\left(\frac{n^k}{\text{polylog}(n)}\right)$ time, providing a logarithmic speedup compared to the trivial algorithms. Our discussion is based on the word RAM model—$w = \Omega(\log n)$ bits can be processed simultaneously, where n is the problem size. The techniques discussed are essentially complicated variations of the idea proposed by [ADKF70], mainly bit parallelization and preprocessing all possible scenarios of the locality of the unoptimized algorithm model.

This technique can also be applied to other problems not covered in this survey but worth mentioning. Using the method of four Russians, the lowest common ancestor (LCA) problem, range minimum query (RMQ) problem (see [BFC00]), level ancestor (LA) problem (see [BFC04]), and other related problems (see [Jin23]), have been optimized to $O(n)$ preprocessing and $O(1)$ query time, removing the $\log n$ factor from the common algorithms.

We consider these problems under the category of fine-grained complexity theory. Fine-grained complexity focuses on determining the exact polynomial order of time complexity achievable by the best algorithm for a given problem. Based on the following conjectures, a class of problems has been shown to be unlikely to have an algorithm with time complexity below a specific polynomial order bound (known as the conditional lower bound), via fine-grained reductions, which is analogous to polynomial-time reductions used to prove NP-hardness:

1. Strong Exponential Time Hypothesis (SETH): kSAT cannot be solved in $O(2^{(1-\epsilon)n})$ time.
2. APSP cannot be solved in $O(n^{3-\epsilon})$ time.
3. 3SUM cannot be solved in $O(n^{2-\epsilon})$ time.

The introduction to fine-grained complexity presented above is based on [Bri21]. We will not discuss the hardness results further in the subsequent sections, but present algorithms that optimize the algorithms for LCS, 3SUM, and APSP by a logarithmic factor, which do not violate the conjectures mentioned above. We will first discuss BMM to illustrate the basic idea of optimization.

In researching the papers and algorithms, the content of [Li23] was particularly helpful and is itself a high-quality survey of this topic.

2 Preliminaries

We analyze time complexity based on the word RAM model with word size w. We assume that equality, comparison, and addition between elements can be performed in $O(1)$ time. If not specified, A_i ($A_{i,j}$) denotes the i-th (row and j-th column) element of sequence (matrix) A, and indices start from 1. We will not omit the $\log\log n$ factor in complexity unless $\hat{O}$ is used.

3 The boolean matrix multiplication problem

Problem 1 (BMM). *Given two $n \times n$ boolean matrices A, B, compute $n \times n$ matrix C, such that*

$$C_{i,j} = \bigvee_{k=1}^{n} \left(A_{i,k} \wedge B_{k,j}\right).$$

This problem is weaker than the general matrix multiplication, which has an $O(n^\omega)$ algorithm, with $\omega \approx 2.37$ so far ([Wik24]). Here, however, we focus only on combinatorial algorithms.

[ADKF70] introduced a groundbreaking algorithm, widely known as "the method of four Russians". It works as follows: partition A into $n \times d$ and B into $d \times n$ smaller matrices, and consider optimizing the multiplication of such a pair, namely A_1 and B_1.

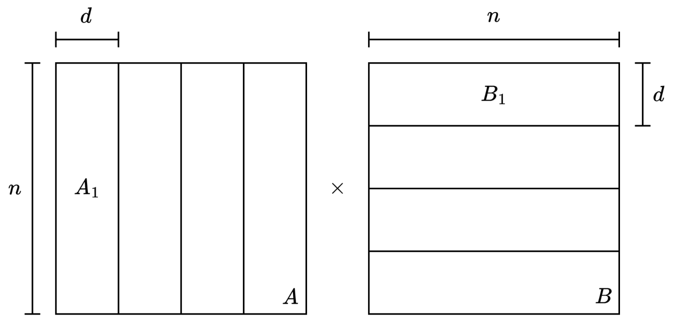

Figure 1: The partitioning.

Denote the rows of A_1 as $a_1, \cdots, a_n$ and the rows of B_1 as $b_1, \cdots, b_d$. Although we need to compute $a_1B, \cdots, a_nB$, there are only 2^d possibilities among them, so we can precompute these results by recursion. For a row vector a such that $a_t = 1$, we have

$$aB = (a \setminus t)B \vee b_t,$$

where $a \setminus t$ means setting a_t to 0, and $\vee$ represents the "or" operation applied element-wise to the two row vectors. Since we use the word RAM model, precomputation can be done in $O\left(2^d \frac{n}{w}\right)$ time, and using them to compute A_1B_1 takes $O\left(\frac{n^2}{w}\right)$ time. Finally, taking the $\vee$ of all the products requires $O\left(\frac{n^3}{wd}\right)$ time. Choosing $d = \log n$, the overall time becomes $O\left(\frac{n^3}{w \log n}\right)$.

After decades, [BW09] presented an algorithm based on advanced graph theory techniques with time complexity $O\left(\frac{n^3(\log\log n)^2}{(\log n)^{9/4}}\right)$, and [Cha15] subsequently gave an algorithm with time complexity $O\left(\frac{n^3(\log\log n)^3}{(\log n)^3}\right)$ using a relatively simple analysis. Then, [Yu15] introduced an algorithm with $\hat{O}\left(\frac{n^3}{(\log n)^4}\right)$ time complexity. The current best algorithm, presented by [AFK$^+$24], runs in $\frac{n^3}{2^{\Omega((\log n)^{1/7})}}$ time, which is a speedup strictly stronger than any polylogarithmic factor.

4 The longest common subsequence problem

Problem 2 (LCS)**.** *Given two sequences A and B of length n, find the length of the longest common subsequence (LCS) of A and B. A subsequence of a sequence is obtained by removing zero or more elements from it, while preserving the order of the remaining elements. Here, we assume the range of A_i and B_i is a finite set Σ.*

The common $O(n^2)$ algorithm is a dynamic programming approach: let $A[l, r]$ denote the consecutive subsequence $(a_l, \cdots, a_r)$ of A. We need to calculate $f_{i,j}$—the LCS of $A[1, i]$ and $B[1, j]$—for every $0 \le i, j \le n$. The recurrence is given by:

$$f_{i,j} = \begin{cases} 0, & i = 0 \vee j = 0 \\ \max\{f_{i-1,j}, f_{i,j-1}\}, & i, j \ge 1 \wedge A_i \ne B_j \\ f_{i-1,j-1} + 1, & i, j \ge 1 \wedge A_i = B_j \end{cases} .$$

Thus, $f_{n,n}$ gives the final answer. A practical $O\left(\frac{n^2}{w}\right)$ algorithm is presented in [CIPR01], which exploits the parallelized bitwise operations of the word RAM model. The brief description of [CIPR01]'s algorithm is as follows:

Let $d_{i,j} = f_{i,j} - f_{i,j-1}$. We can show that $0 \le d_{i,j} \le 1$ by considering the meanings of $f_{i,j}$ and $f_{i,j-1}$, so $(d_{i,1}, \cdots, d_{i,n})$ can be seen as a boolean vector, denoted as d_i. We want to compute d_i from d_{i-1}.

We calculate $d_{i,j}$ from $j = 1$ to n. The transition can be divided into three stages:

1. $f_{i,j-1} = f_{i-1,j-1}$, and $A_i \ne B_j$ or $d_{i-1,j} = 1$. Simply $f_{i,j} = f_{i-1,j}$ and $d_{i,j} = d_{i-1,j}$.
2. $f_{i,j-1} = f_{i-1,j-1}$, and $A_i = B_j$ and $d_{i-1,j} = 0$. $f_{i,j}$ will increase, $f_{i,j} = f_{i-1,j} + 1$ and $d_{i,j} = 1$.
3. $f_{i,j-1} = f_{i-1,j-1} + 1$. It must be $d_{i,j} = 0$. Also $f_{i,j} = f_{i-1,j} + 1$ until $d_{i-1,j} = 1$.

In conclusion, as long as stage 2 is triggered, it will continue in stage 3 until $d_{i-1,j} = 1$, after which it returns to stage 1, which is simply copying the elements of d_{i-1}. The entire process can be interpreted as a combination of addition and bitwise operations if d_i is viewed as the binary representation of an integer. Let $\bar{d}_i = \operatorname{not} d_i$, the "chain effect" can be seen as the carry bit of the addition process. We have

$$\bar{d}_i = \left[\bar{d}_{i-1} + \left(\bar{d}_{i-1} \operatorname{and} m_{A_i}\right)\right] \operatorname{or} \left(\bar{d}_{i-1} \operatorname{and} \bar{m}_{A_i}\right),$$

where $m_c = ([B_1 = c], \cdots, [B_n = c])$ and $\bar{m}_c = \operatorname{not} m_c$. Both m_c and $\bar{m}_c$ can also be viewed as integers.

b	b	c	b	c	a	c	b	b	a	c	b	a
f_{i-1}	1	1	2	2	2	2	3	4	4	4	4	4
f_i	1	2	2	3	3	3	3	4	4	5	5	5
m_c	0	1	0	1	0	1	0	0	0	1	0	0
d_{i-1}	1	0	1	0	0	0	1	1	0	0	0	0
d_i	1	1	0	1	0	0	0	1	0	1	0	0
$\overline{m}_c$	1	0	1	0	1	0	1	1	1	0	1	1
$\overline{d}_{i-1}$	0	1	0	1	1	1	0	0	1	1	1	1
$\overline{d}_i$	0	0	1	0	1	1	1	0	1	0	1	1

Figure 2: An example, $A = \texttt{aaaaababbccc}$, $B = \texttt{bcbcacbbacba}$, $i = 10$.
Blue, green and orange&red grids represent stage 1, 2, 3 respectively.

Since the involved operations can be performed in $O\left(\frac{n}{w}\right)$ time, the overall time complexity is $O\left(\frac{n^2}{w}\right)$, and the memory complexity is $O(n)$.

[MP80] uses a similar technique to the method of four Russians, leading to a speedup by a $\log^2 n$ factor, under the assumption that $|\Sigma| = O(1)$. For arbitrary $|\Sigma|$, [BFC08] achieves $O\left(\frac{n^2(\log\log n)^2}{\log^2 n}\right)$, and [Gra16] improves it to $O\left(\frac{n^2 \log\log n}{\log^2 n}\right)$.

The basic idea of [MP80] is to divide the table of d into $y \times y$ blocks, with adjacent blocks sharing borders. For a block with $i \in [i'y + 1, (i' + 1)y]$ and $j \in [j'y + 1, (j' + 1)y]$, it can be fully determined as long as its top row, leftmost column, and $A[i'y+1, (i'+1)y]$ and $B[j'y+1, (j'+1)y]$ are given. If $|\Sigma| = C \geq 2$, then $2y(1 + \log C)$ bits are needed to represent this information.

We can precompute the result (only the bottom row and rightmost column of d of the block are needed for subsequent calculation) for all these $\Theta(2^{2y}C^{2y})$ possibilities in $O(2^{2y}C^{2y}y^2)$ time, and then compute the values of all blocks in $O(l \cdot n^2/y^2)$ time, with lookups from the precomputed results done in $l = \Theta\left(\frac{y}{\log n}\right)$ time. Taking $y = \left\lfloor \frac{\log n}{4 \log C} \right\rfloor = \Theta(\log n)$ leads to $l = O(1)$, and the overall time complexity becomes $O\left(\frac{n^2}{\log^2 n}\right)$.

For the case where $|\Sigma|$ is not constant, [Gra16] designs a more delicate structure: consider $A[i'y + 1, (i' + 1)y]$, a snippet of length y at a time. Relabel the elements of B by equality relation with elements of $A[i'y + 1, (i' + 1)y]$ so that a block of d with dimensions $x_1 \times x_2$, where the first dimension is fixed, can now be determined using $x_1 + x_2(1 + \log(y + 1))$ bits. Precompute all the possibilities for each snippet. The overall time complexity is:

$$O\left(\frac{n^2}{y}\log y + \frac{n}{x_1}2^{x_1+x_2}(y+1)^{x_2}x_1x_2 + \frac{n^2}{x_1x_2} \cdot \frac{(x_1+x_2)\log y}{\log n}\right).$$

Taking $y = \frac{\log^2 n}{2}$, $x_1 = \frac{\log n}{4}$, $x_2 = \frac{\log n}{4 \log\log n}$, the resulting time complexity will be $O\left(\frac{n^2 \log\log n}{\log^2 n}\right)$. No better algorithm is known.

5 The 3-sum problem

Problem 3 (3SUM). *Given a set $A \subseteq U$, determine whether $\exists a, b, c \in A$ such that $a + b + c = 0$.*

An $O(n^2)$ algorithm is based on the two-pointers technique. First, sort the elements of A, denoted as $a_1, \cdots, a_n$, in increasing order. We can run the following algorithm:

Algorithm 1 Brute force for 3SUM

```
for $i \leftarrow 1$ to $n$ do
    $k \leftarrow n$
    for $j \leftarrow i$ to $k$ do
        while $k > j \land a_i + a_j + a_k > 0$ do
            $k \overset{-}{\leftarrow} 1$
        if $a_i + a_j + a_k = 0$ then
            Report finding $(a_i, a_j, a_k)$
Report not found
```

We focus on $U = \mathbb{R}$. [GP18] developed several faster algorithms based on this method, initially presenting a slow algorithm but with only $O(n^{3/2}\sqrt{\log n})$ decision tree complexity. The main idea is to partition $a_1, \cdots, a_n$ into groups of size g, namely $A_1, \cdots, A_{\lceil n/g \rceil}$, where $A_i = \{a_t \mid (i-1)g < t \leq ig\}$, and then keep the first loop. Instead of enumerating j and k as single elements, we enumerate them as two groups, and check if $-a_i \in A_{j,k} = A_j + A_k = \{x + y \mid x \in A_j, y \in A_k\}$.

In the following analysis, we denote $A_j(x)$ as the x-th element of A_j. We assume there are no duplicate elements in the definitions of A_j and $A_{j,k}$, as ties can be broken by perturbation or by taking the index into account.

This algorithm uses binary search to speed up the checking process to $O(\log g)$, resulting in a total complexity of $O\left(\frac{n^2 \log g}{g}\right)$. Sorting each $A_{j,k}$ is then required. The number of possible orders of $A_{j,k}$ has an upper bound of $(g^2)!$. Consider enumerating these permutations, denoted as $\pi_1, \pi_2 : [g^2] \to [g]$, which corresponds to $A_j(\pi_1(t)) + A_k(\pi_2(t))$ being the t-th smallest element. A permutation sorts some $A_{j,k}$ correctly iff for all t,

$$\begin{aligned} &A_j(\pi_1(t)) + A_k(\pi_2(t)) < A_j(\pi_1(t+1)) + A_k(\pi_2(t+1)) \\ \iff &A_j(\pi_1(t)) - A_j(\pi_1(t+1)) < A_k(\pi_2(t+1)) - A_k(\pi_2(t)). \end{aligned}$$

Thus, we can regard $(A_j(\pi_1(1)) - A_j(\pi_1(2)), \cdots, A_j(\pi_1(g^2-1)) - A_j(\pi_1(g^2)))$ and $(A_k(\pi_2(2)) - A_k(\pi_2(1)), \cdots, A_k(\pi_2(g^2)) - A_k(\pi_2(g^2-1)))$ as two g^2-dimensional points (red and blue). Now finding all the $A_{j,k}$s that can be sorted by (π_1, π_2) is equivalent to finding all pairs consisting of a red and a blue point, where every coordinate of the red point is less than that of the blue point.

For this problem, [GP18] presents a divide-and-conquer algorithm. If the input consists of n d-dimensional points, it runs in $O\left(\left(\frac{2^\epsilon}{2^\epsilon - 1}\right)^d n^{1+\epsilon} + \text{output size}\right)$ time for all $\epsilon > 0$. We omit the details here. Taking $\epsilon = \frac{1}{2}$, the sorting step takes $O\left((g^2)!\left(2+\sqrt{2}\right)^{g^2}\left(\frac{2n}{g}\right)^{3/2} + \frac{n^2}{g^2}\right)$ time, since each $A_{j,k}$ will be correctly sorted once. For the binary search part, since the query entries $[-a_i \in A_{j,k}]$ are known before sorting, we can simply process the queries right after $A_{j,k}$ is found sorted. The time complexity remains $O\left(\frac{n^2 \log g}{g}\right)$.

Taking $g = \frac{1}{2}\sqrt{\frac{\log n}{\log\log n}}$, the dominating term for the sorting time will be $\frac{n^2}{g^2}$, which results in a time complexity of $O\left(\frac{n^2(\log\log n)^{3/2}}{(\log n)^{1/2}}\right)$. By [Fre76a], there are actually only $O(g^{8g})$ possible orders of all $A_{j,k}$, so taking $g = \frac{\sqrt{\log n}}{2}$ reduces the time complexity to $O\left(\frac{n^2 \log\log n}{(\log n)^{1/2}}\right)$.

Further optimization from [GP18] reduces the time complexity to $O\left(\frac{n^2(\log\log n)^{2/3}}{(\log n)^{2/3}}\right)$ by preventing the sorting of the entire $A_{j,k}$. This optimization narrows down the region to the space between two contours. A randomized algorithm based on the same approach runs in $O\left(\frac{n^2(\log\log n)^2}{\log n}\right)$ time. [KLM18] further lowers the decision tree complexity to $O(n\log^2 n)$, and the SOTA algorithm [Cha19] runs in $O\left(\frac{n(\log\log n)^{O(1)}}{\log^2 n}\right)$ time using a geometric approach. These algorithms are more complicated, so we do not delve into the details here.

6 The all-pairs shortest path problem

Problem 4 (APSP). *Given a directed graph $G = (V, E)$ with $|V| = n$, and assuming there is a weighted edge between every ordered pair of vertices, where the weights are nonnegative real numbers, find the shortest path between each pair of vertices.*

The basic $O(n^3)$ algorithm for this problem is the Floyd-Warshall algorithm [Flo62], which essentially computes the closure of the adjacency matrix. In this case, the addition and multiplication operations in the definition of matrix multiplication are replaced by minimum and addition operations. This is known as min-plus matrix multiplication. [AH74] states that:

> The time necessary to compute the closure of a matrix of nonnegative reals, with operations MIN and + playing the role of addition and multiplication of scalars, is of the same order as the time to compute the product of two matrices of this type.

Thus, we only need to focus on the min-plus matrix multiplication problem.

Problem 5 ((min,+) MM). *Given two $n \times n$ real matrices A, B, calculate matrix C, such that*

$$C_{i,j} = \min_{k=1}^{n}\{A_{i,k} + B_{k,j}\}.$$

The first breakthrough in this problem was made by [Fre76b], who developed a decision tree with $O(n^{5/2})$ complexity and an algorithm with time complexity $O\left(\frac{n^3(\log\log n)^{1/3}}{(\log n)^{1/3}}\right)$. The algorithm starts by dividing matrix A into $b \times m$ stripes and matrix B into $m \times b$ stripes. The original problem can then be reduced to performing $\frac{n^3}{b^2 m}$ multiplications between $b \times m$ and $m \times b$ matrices, alongside $O\left(\frac{n^3}{m}\right)$ min operations. For a $b \times m$ matrix A_1 and an $m \times b$ matrix B_1, multiplying them needs comparisons between $a_{i,r} + b_{r,j}$ and $a_{i,s} + b_{s,j}$ for every $1 \le i, j \le b$ and $1 \le r < s \le m$, where $a_{\cdot,\cdot}$ and $b_{\cdot,\cdot}$ denotes the elements of A_1 and B_1. We can use transposition of terms, to change the values needed for comparison to $a_{i,r} - a_{i,s}$ and $b_{s,j} - b_{r,j}$. So we just sort $\{a_{i,r} - a_{i,s}\} \cup \{b_{s,j} - b_{r,j}\}$ for every r, s to obtain $\binom{m}{2}$ lists $\{L_{r,s}\}$, and no more information is needed. Sorting requires $O(m^2 b \log b)$ comparison in total, so we can precompute a decision tree of this depth, to obtain the result of all matrix multiplications efficiently, i.e. we can directly know $\operatorname{argmin}_{k=1}^{m}\{a_{i,k} + b_{k,j}\}$ without enumerating through k.

[Fre76b] further notes that the information complexity can be reduced. Consider a $2mb$-dimensional space, where each dimension corresponds to an element of A_1 or B_1. We construct $\binom{m}{2}\binom{2b}{2} = \Theta(m^2b^2)$ hyperplanes: $a_{i,r} - a_{i,s} = a_{i',r} - a_{i',s}$, $b_{s,j} - b_{r,j} = b_{s,j'} - b_{r,j'}$, and $a_{i,r} - a_{i,s} = b_{s,j} - b_{r,j}$. Then, each possible order corresponds to a region. According to [Buc43], an r-dimensional space can be partitioned into at most $O(rn^r)$ regions by n hyperplanes, so the information complexity is reduced to $O(mb \log mb)$.

[Fre76a] provides an algorithm to construct a decision tree for sorting a sequence of length n, under the constraint that only the orders in a subset Γ of n-permutations are possible. The tree has depth $\log|\Gamma| + O(n)$, and its construction can be done in polynomial time in $|\Gamma|$. In this case, $n = \Theta(m^2b)$, so the depth is $O(m^2b)$, and the construction time is $\exp(O(mb\log mb))$.

Now if Γ can be obtained in time T, then the total time will be

$$O\left(T + \exp(O(mb\log mb)) + n^3\left(\frac{m}{b} + \frac{1}{m}\right)\right).$$

To obtain Γ, we consider sampling at least one point from each region. It can be proven that such points can all be obtained by solving the intersections of hyperplanes: $a_{i,r} - a_{i,s} = a_{i',r} - a_{i',s} \pm 1$, $b_{s,j} - b_{r,j} = b_{s,j'} - b_{r,j'} \pm 1$, $a_{i,r} - a_{i,s} = b_{s,j} - b_{r,j} \pm 1$, $a_{i,r} = 0$, and $b_{j,r} = 0$. For the intersections, we only need to calculate the $\{L_{r,s}\}$s and remove duplicates. Since there are $O((m^2b^2)^{2mb})$ intersections, T will also be $\exp(O(mb\log mb))$.

Finally, we need to choose m and b such that the $\exp(O(mb\log mb))$ term doesn't dominate, and the term $\frac{m}{b} + \frac{1}{m}$ is minimized. By choosing $m = c\frac{(\log n)^{1/3}}{(\log\log n)^{1/3}}$ and $b = m^2$ for a sufficiently small constant c, we get the time complexity:

$$O\left(\frac{n^3(\log\log n)^{1/3}}{(\log n)^{1/3}}\right).$$

For further improvements, [Tak92] works under the same framework and achieves a time complexity of $O\left(\frac{n^3(\log\log n)^{1/2}}{(\log n)^{1/2}}\right)$. The result of $O\left(\frac{n^3}{\log n}\right)$ was achieved in [Cha08], while [HT12] achieves $O\left(\frac{n^3\log\log n}{(\log n)^2}\right)$. The current SOTA result [Wil14], based on tools from circuit complexity, runs in time $\frac{n^3}{2^{\Omega((\log n)^{1/2})}}$.

Here we shall mention another very common problem closely related to the problem above:

Problem 6 ((min,+) Conv). *Given two real sequences A, B of length n, compute sequence C of length n, such that*

$$C_i = \min_{j+k=i}\{A_j + B_k\}.$$

We make indices start from 0 for alignment. This problem can be reduced to (min,+) MM. By [BCD+06], we can construct the following multiplication between a $\frac{n}{d} \times d$ and a $d \times n$ matrix:

$$\begin{bmatrix} A_0 & A_1 & \cdots & A_{d-1} \\ A_d & A_{d+1} & \cdots & A_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n-d} & A_{n-d+1} & \cdots & A_{n-1} \end{bmatrix} \begin{bmatrix} B_0 & B_1 & \cdots & B_{n-1} \\ +\infty & B_0 & \cdots & B_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ +\infty & +\infty & \cdots & B_{n-d} \end{bmatrix} = \begin{bmatrix} P_{0,0} & P_{0,1} & \cdots & P_{0,n-1} \\ P_{1,0} & P_{1,1} & \cdots & P_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ P_{\frac{n}{d}-1,0} & P_{\frac{n}{d}-1,1} & \cdots & P_{\frac{n}{d}-1,n-1} \end{bmatrix},$$

$$C_i = \min_{j=0}^{\lfloor i/d \rfloor} \{P_{j,i-jd}\}.$$

Take $d = \sqrt{n}$. If $n \times n$ $(\min, +)$ matrix multiplication can be done in $\mathsf{M}(n) = \frac{n^3}{\mathsf{R}(n)}$ time, then the convolution can be done in $O(\sqrt{n}\mathsf{M}(\sqrt{n})) = O\left(\frac{n^2}{\mathsf{R}(\sqrt{n})}\right)$ time.

7 Conclusion

From the algorithms discussed above, we can observe that those following the method of four Russians generally work by identifying the gap between information complexity and actual computational complexity, and try to precompute the results of all possible cases. However, since the number of possibilities grows exponentially with n, they can only divide the input into small groups or pieces and make trade-offs, seeking a balance between precomputation and actual computation. The decision tree complexity cannot be fully realized.

While different approaches to breaking down the model or varying the processing order may lead to significant improvements, most of the algorithms require a careful analysis of information complexity—that is, counting the possible cases. As a result, combinatorics, information theory, and more advanced mathematical tools are increasingly used. Some SOTA methods even rely on tools from circuit complexity, and surprisingly, shaving logs is also related to computation hardness. [AHWW16] shows that

> If Edit Distance or LCS on two binary sequences of length n can be solved in $O(n^2/\log^c n)$ time for every $c > 0$, then $\mathsf{NTIME}[2^{O(n)}]$ does not have non-uniform polynomial-size log-depth circuits.

Finally, regarding parallelized bitwise operations, actually every algorithm implicitly or explicitly uses this property. It's needed for encoding and addressing the possibilities, or at least for performing basic operations like additions. Algorithms for BMM and LCS (the first one) explicitly leverage parallelization, and these are the two algorithms actually practical. Other algorithms have large constant factors. To prevent precomputation time from exceeding the desired complexity, the group/block sizes need to be kept very small, and the optimization will barely be effective or may even lead to deoptimization.

References

[ADKF70] V. L. Arlazarov, Y. A. Dinitz, M. A. Kronrod, and I. A. Faradzhev, *On economical construction of the transitive closure of an oriented graph*, Dokl. Akad. Nauk SSSR, vol. 194, 1970, pp. 487–488.

[AFK$^+$24] Amir Abboud, Nick Fischer, Zander Kelley, Shachar Lovett, and Raghu Meka, *New graph decompositions and combinatorial boolean matrix multiplication algorithms*, Proceedings of the 56th Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC 2024, Association for Computing Machinery, 2024, p. 935–943.

[AH74] Alfred V. Aho and John E. Hopcroft, *The design and analysis of computer algorithms*, 1st ed., Addison-Wesley Longman Publishing Co., Inc., USA, 1974.

[AHWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams, *Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made*, Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '16, Association for Computing Machinery, 2016, p. 375–388.

[BCD$^+$06] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian, *Necklaces, convolutions, and $x + y$*, Proceedings of the 14th Conference on Annual European Symposium - Volume 14 (Berlin, Heidelberg), ESA'06, Springer-Verlag, 2006, p. 160–171.

[BFC00] Michael A. Bender and Martín Farach-Colton, *The lca problem revisited*, Proceedings of the 4th Latin American Symposium on Theoretical Informatics (Berlin, Heidelberg), LATIN '00, Springer-Verlag, 2000, p. 88–94.

[BFC04] ______, *The level ancestor problem simplified*, Theoretical Computer Science **321** (2004), no. 1, 5–12, Latin American Theoretical Informatics.

[BFC08] Philip Bille and Martín Farach-Colton, *Fast and compact regular expression matching*, Theoretical Computer Science **409** (2008), no. 3, 486–496.

[Bri21] Karl Bringmann, *Fine-grained complexity theory: Conditional lower bounds for computational geometry*, p. 60–70, Springer International Publishing, 2021.

[Buc43] R. Creighton Buck, *Partition of space*, American Mathematical Monthly **50** (1943), 541–544.

[BW09] Nikhil Bansal and Ryan Williams, *Regularity lemmas and combinatorial algorithms*, Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science (USA), FOCS '09, IEEE Computer Society, 2009, p. 745–754.

[Cha08] Timothy M. Chan, *All-pairs shortest paths with real weights in $O(n^3/\log n)$ time*, Algorithmica **50** (2008), no. 2, 236–243.

[Cha15] ______, *Speeding up the four russians algorithm by about one more logarithmic factor*, Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (USA), SODA '15, Society for Industrial and Applied Mathematics, 2015, p. 212–217.

[Cha19] ______, *More logarithmic-factor speedups for 3sum, (median,+)-convolution, and some geometric 3sum-hard problems*, ACM Trans. Algorithms **16** (2019), no. 1, 1–23.

[CIPR01] Maxime Crochemore, Costas S. Iliopoulos, Yoan J. Pinzon, and James F. Reid, *A fast and practical bit-vector algorithm for the longest common subsequence problem*, Information Processing Letters **80** (2001), no. 6, 279–285.

[Flo62] Robert W. Floyd, *Algorithm 97: Shortest path*, Commun. ACM **5** (1962), no. 6, 345.

[Fre76a] Michael L. Fredman, *How good is the information theory bound in sorting?*, Theoretical Computer Science **1** (1976), no. 4, 355–361.

[Fre76b] ———, *New bounds on the complexity of the shortest path problem*, SIAM J. Comput. **5** (1976), no. 1, 83–89.

[GP18] Allan Grønlund and Seth Pettie, *Threesomes, degenerates, and love triangles*, J. ACM **65** (2018), no. 4, 1–25.

[Gra16] Szymon Grabowski, *New tabulation and sparse dynamic programming based techniques for sequence similarity problems*, Discrete Applied Mathematics **212** (2016), 96–103, Stringology Algorithms.

[HT12] Yijie Han and Tadao Takaoka, *An $O(n^3 \log\log n/\log^2 n)$ time algorithm for all pairs shortest paths*, Algorithm Theory – SWAT 2012 (Berlin, Heidelberg) (Fedor V. Fomin and Petteri Kaski, eds.), Springer Berlin Heidelberg, 2012, pp. 131–141.

[Jin23] Huai'en Jin, 【理性愉悦】如何优雅地玩转线性-常数时间复杂度, `https://return20071007.blog.uoj.ac/blog/8573`, Jul 2023, [Online; accessed 7-January-2025].

[KLM18] Daniel M. Kane, Shachar Lovett, and Shay Moran, *Near-optimal linear decision trees for k-sum and related problems*, Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (New York, NY, USA), STOC 2018, Association for Computing Machinery, 2018, p. 554–563.

[Li23] Baitian Li, 一些经典问题比暴力快一点点的算法, `https://www.cnblogs.com/Elegia/p/slightly-faster-than-brute-force.html`, Aug 2023, [Online; accessed 7-January-2025].

[MP80] William J. Masek and Michael S. Paterson, *A faster algorithm computing string edit distances*, Journal of Computer and System Sciences **20** (1980), no. 1, 18–31.

[Tak92] Tadao Takaoka, *A new upper bound on the complexity of the all pairs shortest path problem*, Information Processing Letters **43** (1992), no. 4, 195–199.

[Wik24] Wikipedia contributors, *Computational complexity of matrix multiplication — Wikipedia, the free encyclopedia*, `https://en.wikipedia.org/w/index.php?title=Computational_complexity_of_matrix_multiplication&oldid=1263768929`, 2024, [Online; accessed 7-January-2025].

[Wil14] Ryan Williams, *Faster all-pairs shortest paths via circuit complexity*, Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing (New York, NY, USA), STOC '14, Association for Computing Machinery, 2014, p. 664–673.

[Yu15] Huacheng Yu, *An improved combinatorial algorithm for boolean matrix multiplication*, Automata, Languages, and Programming (Berlin, Heidelberg) (Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, eds.), Springer Berlin Heidelberg, 2015, pp. 1094–1105.