Applying the Four Russians Technique to Banded Extension and X-Drop Sequence Alignment

Cristina Teodoropol

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-240
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-240.html

December 22, 2020

Applying the Four Russians Technique to Banded Extension and X-Drop Sequence Alignment

by Cristina Teodoropol

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Katherine Yelick
Research Advisor

12/21/2020

(Date)

* * * * * * *

Aydin Buluç
Second Reader

12/22/2020

(Date)

Applying the Four Russians Technique to Banded Extension and X-Drop Sequence Alignment

CRISTINA TEODOROPOL, UC Berkeley

The Four Russians technique is a method that divides a sequence alignment problem into tiles and uses a precomputed lookup table to quickly find the solution to each tile. By dividing the dynamic programming table into tiles of size $t_n \times t_m$, the running time of the alignment algorithm is reduced to $O(nm/(t_n t_m))$. We explore this technique as it applies to the problem of genomic sequence alignment. Beyond the simple global alignment problem, we implement the Four Russians technique for a banded extension algorithm, which only covers a diagonal band of the dynamic programming table and, similar to local alignment, extends an alignment ending anywhere in the table, and an early dropout algorithm similar to X-Drop. Using a $t_3 \times t_3$ sized tile, the average speedup was 5.15 for the Needleman-Wunsch full global alignment algorithm, 2.98 for the banded extension algorithm, and 2.77 for the X-Drop algorithm, running over data produced by the PBSIM synthetic generator.

1. INTRODUCTION

The problem of aligning a pair of strings arises in many areas of computer science, but it is of particular importance in computational biology, to align DNA, RNA, or protein sequences. The general approach to solving alignment problems is with dynamic programming. There exist many variations of the sequence alignment problems that use heuristics to cut down on the quadratic search of the full dynamic programming table, such as banded alignment and X-Drop. In this report I will be exploring a speedup technique known as Four Russians in the context of these alignment variants. In particular I believe this is the first application of Four Russians to an early dropout algorithm similar to X-Drop. The Four Russians approach was originally developed to speed up boolean matrix multiplication, and was then adapted to work for unit cost edit distance computation by Dan Gusfield [1997].

This report will cover the traditional global alignment algorithm, the Lookup Table (LUT) used in Four Russians, how the Four Russians algorithm progresses, and experimental results, as well as some boundary cases to consider.

2. PAIRWISE SEQUENCE ALIGNMENT

The classic problem of sequence alignment is that of global alignment where the entirety of both sequences is included in the resulting alignment. The algorithm for computing the optimal global alignment is called Needleman-Wunsch [1970]. For a sequence S_1 and a sequence S_2 over a finite alphabet Σ, of lengths $|S_1| = n$ and $|S_2| = m$, we want to find an optimal alignment in the presence of a maximizing or minimizing function and given a particular scoring matrix. An alignment is formed by inserting gap characters, represented by '-', as needed. There can be multiple optimal alignments with the same score. The score of an alignment is the sum of the scores of each pair of characters in the alignment according to the scoring matrix. The Needleman-Wunsch[1] algorithm is as follows:

$$M_{i,j} = max \begin{cases} M_{i-1,j-1} + score(S_1[i-1], S_2[j-1]) & \text{if } i > 0 \text{ and } j > 0 \\ M_{i-1,j} + d & \text{if } i > 0 \\ M_{i,j-1} + d & \text{if } j > 0 \\ 0 & \text{if } i = 0 \text{ and } j = 0 \end{cases} \tag{1}$$

where d is the gap penalty.

In this report we will consider the case where d is -1, and where

$$score(\alpha, \beta) = \begin{cases} 1 & \text{if } \alpha = \beta \\ -1 & \text{if } \alpha \,! = \beta \end{cases} \tag{2}$$

The scoring matrix can in general give different penalties for insertions, deletions, or different combinations of pairwise mismatches.

$M_{n,m}$ gives the optimal alignment score. The result can be computed by filling a $(n+1) \times (m+1)$ table using dynamic programming. This algorithm then runs in *O(nm)* time and space.

3. CLASSIC FOUR RUSSIANS

The basic idea of the Four Russians technique is to divide the dynamic programming table into small tiles, precompute a lookup table (LUT) for every possible tile, and then use the LUT to perform the

[1] Note that the code written actually uses a symmetric minimizing function, positive gap penalty, and negated scoring matrix.

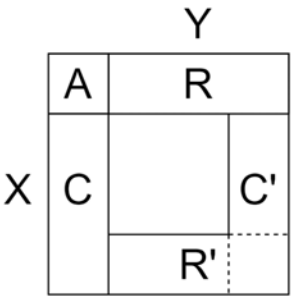

Fig. 1: The cells of interest in a t-block are the first cell A, the remaining first row R, the first column C, the last row R', and the last column C'. Additionally we consider the subsequences X and Y.

actual computation. By dividing the dynamic programming table into tiles of size $t_n \times t_m$, the running time of the algorithm is reduced to $O(nm/(t_n t_m))$, plus the cost of precomputing the LUT. Some of following subsections follow the presentation in the Master's thesis of Mikkelsen [2015], as the explanations naturally follow the same structure.

3.1 t-blocks

Rather than computing each cell in the table one by one, the table is divided into tiles called t-blocks, which overlap on their edges (the first and last rows and columns of the t-block). Each t-block could be viewed as a subtable and computed using the Needleman-Wunsch recurrence, but this would not improve the running time of the algorithm. Therefore we want to build a LUT where each entry represents a t-block. An entry is indexed by the first row and column in the t-block subtable, and by the subsequences of S_1 and S_2 covered by the t-block. Such subsequences $S_1[i...i + t]$ and $S_2[j...j + t]$ will hereafter be referred to as X and Y respectively. An entry then stores the parts of the last row and column of the t-block which do not overlap with the first row and column. See Figure 1. Note that in this report, the size of a t-block excludes the first row and column.

The LUT can then be precomputed by running the Needleman-Wunsch algorithm for every possible t-block input. Since the t-blocks overlap on their edges, the final alignment score can be found by adding the output of one t-block as the input to the next, and so on until the whole table is covered.

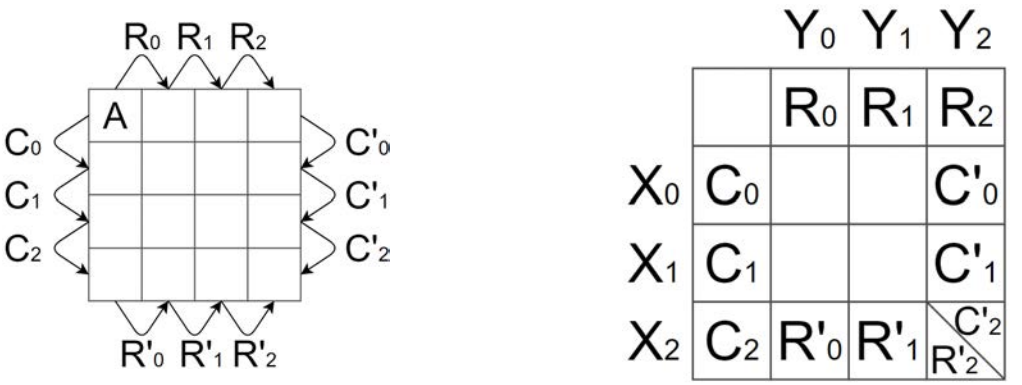

Fig. 2: Offsets between absolute cell values (left), and offsets as values in a t-block (right).

3.2 Encoding t-blocks

Precomputing the LUT for every possible combination of input values may not present an algorithmic advantage. Consider that for edit distance computation, each cell can have a value between 0 and n. Then an encoding of A, C, and R can have $(n+1)^{1+t_n+t_m}$ possible combinations, and for the finite alphabet Σ there are $|\Sigma|^{t_n+t_m}$ possible encodings of X and Y, for a total of $(n+1)^{1+t_n+t_m}|\Sigma|^{t_n+t_m}$ entries in the LUT. Since each t-block entry requires $O(t_n t_m)$ time to compute, this would be far less efficient than simply performing the $O(nm)$ alignment. Therefore we must find a more efficient way to encode t-blocks.

Gusfield's original observation and proof that makes Four Russians efficient is of the fact that each cell's value is only separated by at most a fixed amount from its neighbor's value. Gusfield formally proved that horizontally or vertically adjacent values differ by at most 1 for edit distance computation, and Mikkelsen generalized the proof for arbitrary scoring matrices [1997; 2015]. These offsets between absolute cell values can then be used as the values in a t-block encoding. See Figure 2.

For example, suppose there is an edge $AR_0R_1R_2 = [3, 2, 2, 4]$. This is encoded as $[-1, 0, 2]$. Note that we may add a fixed integer to all the absolute values of the edge without changing the offsets. Therefore only the offset values are of interest to the encoding and A may be omitted. For computing the stored result of a t-block, we simply use 0 in the A position.

As Mikkelsen introduced, let I be the cardinality of the interval containing all the unique offset values possible with a given scoring matrix and gap penalty [2015]. For edit distance, $I = |\{-1, 0, 1\}| = 3$. For the scoring matrix (and gap penalty) introduced by equation (2) in section 2, $I = |\{-1, 0, 1, 2\}| = 4$.

Using the above encoding, the number of possible LUT entries is reduced to $(I|\Sigma|)^{t_n+t_m}$. I should be a small integer, and in this report we only consider the DNA nucleotides A, C, G, and T so likewise $|\Sigma| = 4$. This makes the Four Russians method practically feasible for sequence alignment.

Note that the boundaries of the interval can be found using the equations introduced by Mikkelsen:

$$I_{min} = d \tag{3}$$

$$I_{max} = max \begin{cases} d \\ cmp_{max} - d \end{cases} \tag{4}$$

where I_{min} is the smallest offset, I_{max} is the largest offset, and cmp_{max} is the largest score possible according to the scoring matrix. We refer the reader to [Mikkelsen 2015] for a proof of correctness for these equations.

3.3 Block function

The mapping of the LUT inputs and outputs is formalized as the block function:

$$F(X, C, R, Y) = R', C' \tag{5}$$

Here the X values take up the most significant bits of an entry address while the Y values use the least significant bits. Mikkelsen showed that the addressing layout matters for performance [2015]. Specifically it is most beneficial to have the X value in the most significant bits for row-wise iteration (and Y for column-wise) because the X subsequence remains constant, meaning that we are repeatedly indexing into a contiguous section of the LUT and loading it into the cache, making repeated queries faster.

3.4 Padding and final result

Since the lengths of input sequences S_1 and S_2 might not end on a t-block boundary, we employ the same padding strategy as in the work of Espeholt to ensure the sequences can be split into whole t-blocks [2013]. Namely, the same number of 'A's are prepended to both sequences so that the new S_1 is now divisible by the size of a t-block, and then 'A's are appended onto S_2 until the same condition is reached. We refer the reader to the work of Espeholt for an argument of correctness by induction.

Then we must take the padding into account when we obtain the final result, by using the following equation:

$$result = globalmax - prepadding * score(A, A) = globalmax - prepadding \tag{6}$$

Here $prepadding$ is the number of 'A's prepended to both sequences. $globalmax$ is the result at the end of the filled matrix, which we obtain by starting off with the value of the first cell, 0, and adding all the offsets until the end of the matrix along both axes. However since S_2 may have some padding appended, we only add offsets in the direction of S_2 until the end of the original S_2.

4. BANDED FOUR RUSSIANS

We now present our treatment of a banded Four Russians algorithm. t-blocks are only computed for a diagonal band of a certain width. For the t-blocks on the boundary of the band, about half of the t-block contains normal values and the other half contains a special offset representing an offset involving negative infinity. See Figure 3. Note since we need to use an additional bit value for this new offset, this may introduce wasted LUT entries if it now requires using an additional bit. Particularly for the interval of our scoring matrix, shown the be 4 in subsection 3.2 and meaning we need to represent 4 different bit values, we can encode it using $log_2(4) = 2$ bits, but if we add an additional bit value this forces us to now use 3 bits with waste. Thus the scoring matrix chosen (along with the size of the alphabet) greatly influences the size of the LUT. For this scoring matrix and a $t_3 \times t_3$ sized t-block, the wasted space due to the offset parts of the encoding is given by $\frac{available\ bit\ values}{used\ bit\ values}$, resulting in $(\frac{8}{5})^{t_n+t_m} = (\frac{8}{5})^{3+3} = (\frac{8}{5})^6 = 16.78$ times the optimal space used [Mikkelsen 2015]. This makes our scoring matrix a rather poor choice for a banded Four Russian application.

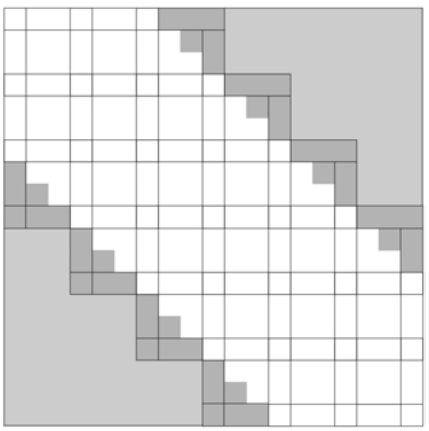

Fig. 3: A banded layout of t-blocks. All grey cells represent the value of negative infinity.

Additionally since we use a diagonal band that is symmetric around the main diagonal, we then use only square t-blocks to maintain the band around this diagonal.

5. EXTENSION

To work up to X-Drop, we first implemented an extension algorithm, which starts the alignment at the top-left corner of the matrix or the start of both sequences, but extends the alignment to anywhere in the matrix, not necessarily to the last (bottom-right) cell. In this regard the alignment starts similarly to a global alignment but ends like a local alignment.

5.1 Tracking the global max

In order to accomplish this extension, we must now keep track of the global max as we go through the matrix and return it at the end of the computation. But to do this we must know the local max of each tile so that we may update the global max. Since the t-blocks store things in a relative manner though, we need to store the relative max of each t-block in the LUT. We therefore present an updated block function:

$$F(X, C, R, Y) = R', C', max \tag{7}$$

max represents the maximal value of a t-block relative to A. Note our implementation uses packed, unaligned fields to preserve memory space.

Once we have the relative max of each t-block, we must get the absolute max values of each t-block in order to update the absolute global max. Therefore we must also compute the absolute A values, since the absolute max of a t-block is $A + max$. We do this by adding the offset edges of a previous t-block, either on the left of or above the current t-block, to that block's A value. Since we know the A value of the first t-block, which is 0, we can continually add the offsets along t-block edges as we fill the matrix and obtain each t-block's A value.

5.2 Naive fallback

As we fill the matrix, a new global max is found whenever

$$A + t\_block.max > global\_max. \tag{8}$$

However it may be the case that this max is found in a cell that is beyond the boundaries of the original S_2 sequence, if the cell is in the post-padding. Therefore whenever condition (8) is met in the last t-block of a row and that t-block contains post-padding, we fall back to naively computing the t-block. This is the same approach used by Mikkelsen [2015].

6. X-DROP

We additionally implemented an early dropout extension algorithm inspired by X-Drop, which ends the computation early if the score drops by more than X below the global max. Specifically our implementation checks a similar condition along the antidiagonal t-blocks as Zhang et al. [2000], that is $A + t\_block.max < prev\_global\_max - X$, but it does not shrink the size of the diagonal band dynamically. Therefore the algorithm terminates when the X-Drop condition is true for all t-blocks along an antidiagonal.

But what exactly is $prev\_global\_max$? To answer that we must first go over some differences from the extension algorithm in the previous section. Notably, the extension algorithm calculated each t-block's absolute max, $A + t\_block.max$, and updated the global max in a row-wise iteration direction. The X-Drop algorithm instead just calculates each t-block's absolute max in a row-wise iteration direction and stores it in an array, but then to more closely follow the original X-Drop algorithm, it updates the

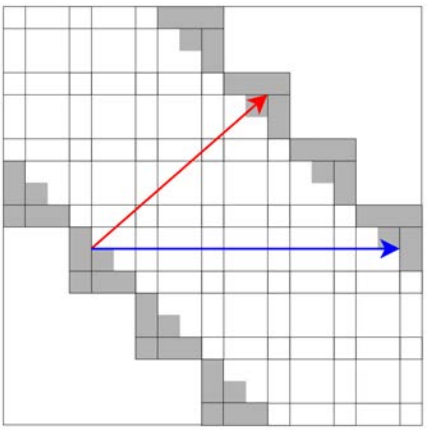

Fig. 4: Iteration directions for the X-Drop algorithm. The absolute max for each t-block is calculated row-wise along the blue arrow. The X-Drop condition is calculated antidiagonal-wise based on the global max along the red arrow.

global max by iterating over the antidiagonals. See figure 4. So $prev\_global\_max$ is the global max up to and including the t-blocks in the previous antidiagonal.

This then introduces some inconsistency in the computation, because we evaluate the naive fallback condition of equation (8) row-wise but based on the global max of the previous antidiagonal, so the final resulting score may only be an approximation of the original X-Drop algorithm.

7. EXPERIMENTS

7.1 Setup

The implementation started off using the code from Espeholt [2013], which only contained a basic Four Russians implementation that covered the whole matrix, performing a global alignment.

The experiments were run on an AMD Ryzen 9 3950X 16-core processor with a base clock of 3.5 GHz, max boost clock of up to 4.7 GHz, and with Precision Boost Overdrive enabled. In total it has 1 MiB of L1 cache, 8 MiB of L2 cache, 64 MiB of L3 cache, but since the code in single-threaded, each core has up to 64 KiB L1 cache, 512 KiB L2 cache, and 16 MiB L3 cache available. The system also had 32 GiB of DDR4-3200, CL14 RAM. The code was compiled with g++ 9.3.0 using c++17.

To verify correctness of the algorithms, each algorithm was tested against its naive counterpart on random sequences of up to 1000 base pairs long.

Banded and X-Drop experiments were run with a diagonal band size of 30 and with an X value of 15. The max t-block size used was $t_3 \times t_3$ since any larger size produced a LUT that could not fit in main memory for the banded case.

Benchmarks were run on reads produced with the PBSIM synthetic generator [Ono et al. 2013], using the provided sample genome and with Continuous Long Reads, model-based simulation, and the default coverage depth of 20. Alignments were skipped if an alignment was to the reverse-complemented reference sequence.

Note that LUT construction time is not included in these benchmarks, as building the LUTs is considered a one-time cost and thereafter the LUTs would be used for multiple alignments. Additionally Mikkelsen showed that an alignment using $t_3 \times t_3$ t-blocks surpasses the time required to construct the LUT (for non-banded alignment), using the same scoring system, at a length of 4,985 base pairs, and even less for smaller t-blocks [2015].

7.2 Results

The results of the synthetic benchmarks for different base pair lengths are shown in Figure 5. Each speedup is calculated over its naive counterpart, so e.g. the banded extension speedup is over the naive banded algorithm. Of note is the fact that running Four Russians with t-blocks of size $t_1 \times t_1$ is slower than running the naive algorithm. The overhead associated with such small t-blocks outweighs the potential benefit. All algorithms show a quadratic effect from scaling the input, although the X-Drop time is dependent on the values in the sequences, which makes the speedup more variable. The average speedups are shown in Table I. It can be seen that speedups improve with larger t-block sizes, though this would be limited by the ability to store the LUT in memory and fit used portions of the LUT in the cache. Notably the runtimes for the X-Drop variant are similar or slightly worse than the banded extension runtimes, which we believe is due to the overhead from iterating in both the row-wise and antidiagonal-wise directions.

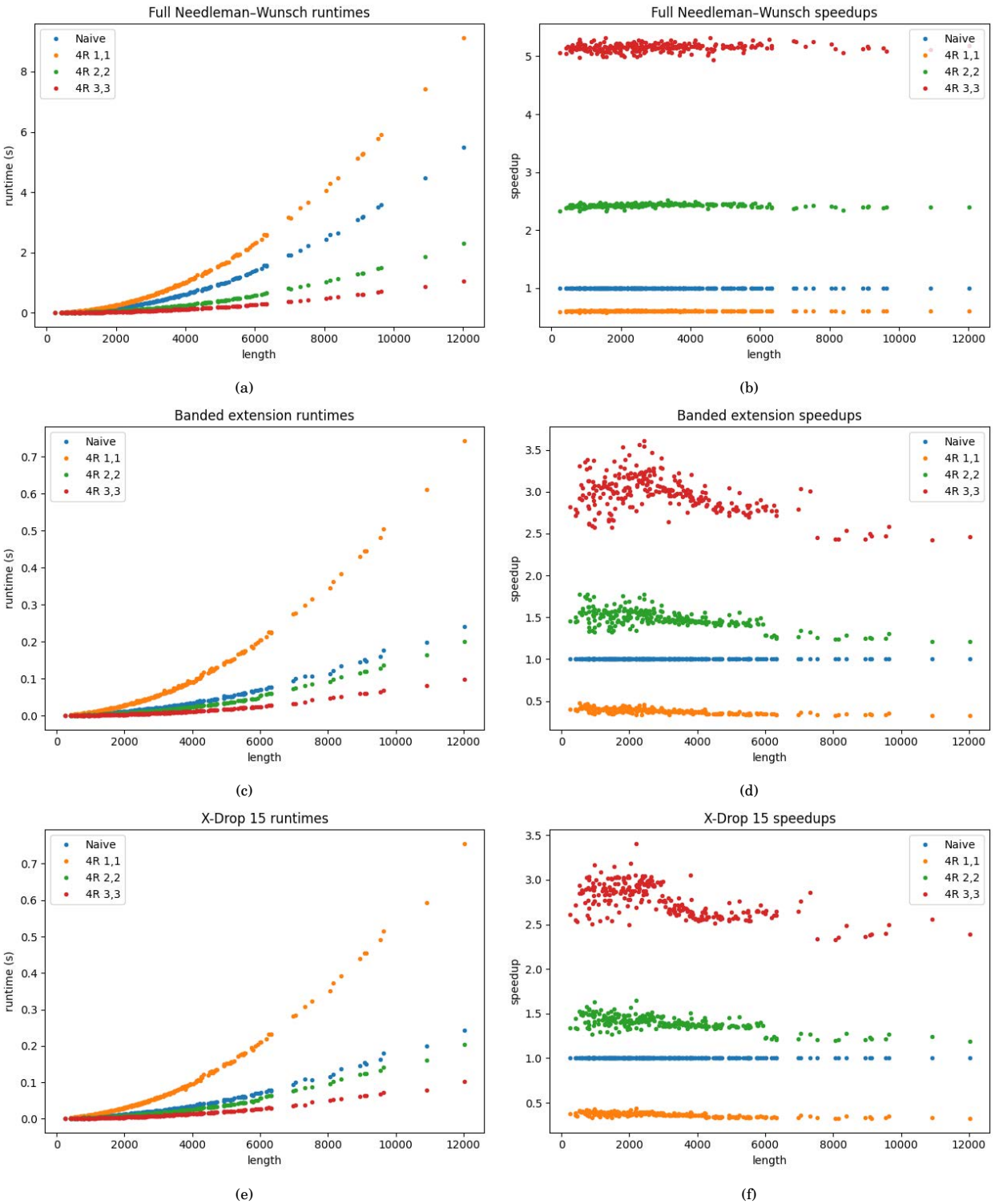

Fig. 5: Benchmark runtimes and speedups.

Average Speedup			
Algorithm	$t_1 \times t_1$	$t_2 \times t_2$	$t_3 \times t_3$
Needleman-Wunsch	0.60	2.43	5.15
Banded Extension	0.38	1.49	2.98
X-Drop	0.37	1.40	2.77

Table I. : Average speedups using the different algorithms for t-blocks of size $t_1 \times t_1$, $t_2 \times t_2$, and $t_3 \times t_3$.

8. CONCLUSIONS AND FUTURE WORK

In this work we have introduced two new variants of the Four Russians technique for sequence alignment, namely a banded extension algorithm and an X-Drop algorithm with early cutoff for sequences with poor alignments. The results show greater speedups with larger t-block sizes, up to average speedups of 5.15 for Needleman-Wunsch, 2.98 for banded extension, and 2.77 for X-Drop with a t-block of size $t_3 \times t_3$. Of note is that the runtimes with the X-Drop algorithm do not exhibit an improvement over the banded extension runtimes. This is likely caused by the iteration of both the rows and antidiagonals in the X-Drop case.

There are many potential continuations of this work. A basic step would be to experiment with the tuning: trying other scoring matrices, diagonal band sizes, and X values. Next we would perform a cache analysis to observe how the different t-block sizes affect the portion of the LUT that fits in the cache and in turn the runtimes. We would also experiment with different iteration directions, such as only iterating over the antidiagonals rather than the current combined row-wise and antidiagonal-wise iteration. Further optimization such as SIMD vectorization could be applied to the offset edge sums. A different encoding scheme similar to [Kim et al. 2016] could be explored to enable the use of this Four Russian technique with larger alphabet sizes, such as for protein sequences. Another improvement to the algorithm would be to only use linear space, and to return the alignment rather than just the score, using Hirschberg's algorithm [Hirschberg 1975].

A significantly different direction would be to adapt the algorithms to run on the GPU. Finally, in regards to the experiments, we would like to run the benchmarks on real genomic data, and compare results to the X-Drop implementation in Seqan [Döring et al. 2008; Reinert et al. 2017]. A comparison to the alternate banded Four Russians design in [Brubach et al. 2017; Brubach and Ghurye 2018], which overlaps t-blocks on more than just their edges, could also prove insightful.

REFERENCES

Brian Brubach and Jay Ghurye. 2018. A Succinct Four Russians Speedup for Edit Distance Computation and One-against-many Banded Alignment. In *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Brian Brubach, Jay Ghurye, Mihai Pop, and Aravind Srinivasan. 2017. Better greedy sequence clustering with fast banded alignment. In *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Andreas Döring, David Weese, Tobias Rausch, and Knut Reinert. 2008. SeqAn an efficient, generic C++ library for sequence analysis. *BMC bioinformatics* 9, 1 (2008), 11.

Lasse Espeholt. 2013. *Exploring the practicality of the Four Russian Method for sequence alignment*. Master's thesis. https://docs.google.com/file/d/0B1ZO_s5g90zHVEtiNXpMV0ZUUEE

Dan Gusfield. 1997. Algorithms on stings, trees, and sequences: Computer science and computational biology. *Acm Sigact News* 28, 4 (1997), 41–60.

Daniel S. Hirschberg. 1975. A linear space algorithm for computing maximal common subsequences. *Commun. ACM* 18, 6 (1975), 341–343.

Youngho Kim, Joong Chae Na, Heejin Park, and Jeong Seop Sim. 2016. A space-efficient alphabet-independent Four-Russians' lookup table and a multithreaded Four-Russians' edit distance algorithm. *Theoretical Computer Science* 656 (2016), 173–179.

Anders Høst Mikkelsen. 2015. *Local alignment using the Four-Russians technique*. Master's thesis. Aarhus University, Aarhus, Denmark.

Saul B Needleman and Christian D Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology* 48, 3 (1970), 443–453.

Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. 2013. PBSIM: PacBio reads simulator—toward accurate genome assembly. *Bioinformatics* 29, 1 (2013), 119–121.

Knut Reinert, Temesgen Hailemariam Dadi, Marcel Ehrhardt, Hannes Hauswedell, Svenja Mehringer, René Rahn, Jongkyu Kim, Christopher Pockrandt, Jörg Winkler, Enrico Siragusa, and others. 2017. The SeqAn C++ template library for efficient sequence analysis: A resource for programmers. *Journal of biotechnology* 261 (2017), 157–168.

Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb Miller. 2000. A greedy algorithm for aligning DNA sequences. *Journal of Computational biology* 7, 1-2 (2000), 203–214.