Efficient Implementation of the Tandem Repeats Software (TRed) Final Report

1 Introduction

The most recent TRed program version, on which we worked this Spring 2020 semester, is TRed Version 3, which was written by Justin Tojeira and Professor Dina Sokol in 2009 [3]. As mentioned in the Proposal to this project, the average execution time of this program on a Homo Sapiens chromosome is 20 hours.

In this CISC 5001, we addressed the following goals, also mentioned in the Proposal:

1. Integrating the suffix array version of the Landau and Vishkin algorithm into the TRed software by extending its current version,
2. Analyzing how both the iterative and computational processes in TRed could be additionally simplified to increase the efficiency of the program, and applying these findings to the program as far as is practicable,
3. Running the improved version of TRed on the human chromosomes to compare its running time with this of the current version, and
4. Publishing this improved version on the Tandem Repeats Database website, providing global access to the program.

Below, we explained how each of these goals was addressed, and to what extent completing a goal can be labeled as successful.

2 Course of Work

During the semester, 6 meetings were scheduled with Professor Sokol to discuss the upcoming steps in changing the TRed program.

2.1 Increasing the Efficiency of TRed (Goal №2)

Some short-time goals of the project were evident already during the first meetings:

- Replacement of string reversal and concatenation methods with pointer operations. Since most pointer operations are on the order of $\mathcal{O}(1)$, while string functions are on the order of $\mathcal{O}(n)$, completing this replacement would considerably reduce execution time.
- Replacement of unrecommended programming techniques, such as `goto` statements, which appeared occasionally in the program.
- Improvement of difficultly readable lines, such as ones without spacing and parenthesization between intricate expressions to ease the reading of the source code. Also, giving variables meaningful names that relate to their purpose in the program.
- Compilation of the TRed Version 3 program in its current form, to ensure that the program works and to obtain output for the purpose of testing the changed program.

- Creation of a short program reading in, reversing, and finding the length of a chromosome's nucleotide sequence, and measure how much time each of these 3 tasks takes to run (in milliseconds). The purpose of this goal is to test whether C++ strings operate as fast as C strings.
- Adoption of FASTA formatted input files into the program.

The short-term goals listed above were completed as follows:

- The program reads the entire bio-sequence into a single buffer, whose size is computed using the standard library functions `fseek` and `ftell`. The concatenation is, therefore, not necessary anymore, so the corresponding `strcat` functions were eliminated. Every `strrev` function call was eliminated by creating a function that reads a given sequence excerpt backwards (that is, the index to the string is negative.) Any `strlen` function call became unnecessary since we store the length of every sequence excerpt is a corresponding integer (or long integer) variable.
- Every `goto` statement was eliminated by including a `bool` variable that indicates if we need to exit a loop or not. Every loop condition that is always true until it breaks (such as `while (true)`) was also replaced with a `bool` whose value changes within the corresponding loop.
- Proper spacing and indentation was added to each source file. Operators like `=, <=, >, +, -` are surrounded by spaces. Variables (except for temporary ones, such as `temp`, and common loop index variables, such as `i, j, k...`) are given mnemonic names.
- The program TRed Version 3 was compiled and run initially on a portion of the Y chromosome, then on the entire chromosome, and then on chromosome 2, and showed to work properly. The output of the program was preserved for testing purposes.
- A program that compares the running time of C++ string function versus C string function for various input sizes was created. It was shown that C string functions run at least twice as fast as most C++ string functions. Hence, we decided that the program will keep utilizing C strings to increase execution speed.
- A function names `read_file` that reads the sequence input file and clears FASTA comments and whitespaces was created. The function does not modify the input file, but filters the characters that are being read into the buffer that the program establishes for the cleared sequence. In addition, the `filter` and `nofilter` programs were modified such that they operate with FASTA formatted input files.

In addition, we made general changes to the source code, which could be summarized as follows:

- We removed function calls wherever it was possible. In other words, we tried to extract the code within relatively short functions and include it directly in the place where the function calls were initially given.
- In the presence of two loops that iterate over the same indices, we combined the two loops to reduce the running time that is involved with incrementing the indices.
- We removed any unused variable, and assigned one generic (temporary) variable to several tasks to reduce the number of declarations.
- To measure how long it takes for the program to execute, we included a function call from the `chrono` C++ library that gauges time and displays the number of milliseconds it takes the program to execute.

- We prevented the algorithm building the `KxK` matrix from assigning `-1` to a great portion of its elements by making the rest of the program focus only on the relevant (non `-1`) regions of the `KxK` matrix.

After the changes above were added to the program, we opined that adding multi-threading features to the program would increase its efficiency (if the computer that runs the program has more than one core.) The premise that allows using multithreading in this program is that the tandem repeats in one section with `DOUBLE_PERIOD` characters do not depend on the tandem repeats found in another, non-overlapping section of that size. Therefore, we can divide all such sections among the threads created in the program to do an individual work that will not interfere with the results of other threads.

We implemented the multi-threaded version as follows: the `main` function inside the `main.cpp` file became the thread managing function. It computes the size of the work that each thread has to complete (depending on the size of the of the sequence that was read,) creates the threads, and joins them after they complete their work. Finally, the `main` collects all the information produced by the threads and places it into a single output file. On the other hand, another source code file named `aux_main.cpp` that contains that functions on which each thread is working was created. Each thread receives its own portion of the input sequence to work on, information about the offset, and the name of a temporary output file into which the thread will write its results. After each thread returns, the main function accesses these temporary files and combines them into a single output file. The temporary files are erased from the directory after the output is combined. When testing the program with different number of threads activated, it was noticed that the multi-threaded program runs the fastest when the number of threads is equal to the number of cores on the machine.

We consider all the mentioned achievements above as part of the second goal that we mentioned in the introduction. That is, all the steps we took that are listed above intended to increase the efficiency in the execution of TRed aside from the addition of the suffix arrays algorithm.

2.2 Suffix Arrays Algorithm in TRed (Goal №1)

Regarding the suffix arrays algorithm, we used the implementation of the Landau-Vishkin algorithm [2] by Yocheved Levitan and Brani Cohen [1] to adopt it into the TRed program:

- We changed the programming techniques in the current C++ version of the algorithm. For example, we replaced all C++ strings by C strings (we proved earlier this semester that the code runs faster this way.) To speed the algorithm further, we extracted the code from the various class definitions and placed it into routines (this way, less function calls, which consume time, are made.)

- We created one C++ source file called `NewMatrixL.cpp` and a corresponding header file `NewMatrixL.h` in which we included routines that set up a suffix array and the associated longest common prefix array (`lcp`), update the suffix array from once "iteration" of the string to another (these are the `for` loop iterations inside the `oneiteration.cpp` source file,) and create the `L` matrix according to which the `KxK` matrix elements are generated.

- It was discovered that by changing the initialization of the `L` matrix, it is possible to generate elements of the `KxK` matrix. The initialization changes consist of creating arrow-like boundaries to the `L` matrix based on the Landau Vishkin algorithm [2].

The functions in `NewMatrixL.cpp` replaced one call of the original `KxK` matrix building algorithm, for the other 3 calls involve approaching the suffix array construction in a more intricate manner. We describe how the running time of this new algorithm is compared to the multi-threaded TRed program.

2.3 Running Time of TRed after Each Change (Goal №3)

Here, we compare the running time results of running the TRed program, after each cardinal group of changes, on the Y Chromosome:

1. After making the general changed described in the beginning of this report, and before adding multi-threading into the program, we noticed an increase of **23.5532%** in the speed of the program compared to the original TRed 3 version.

2. After adding multi-threading into the program and before adding the suffix arrays algorithm into the program, we noticed an increase of **80.3292%** in the speed of the program compared to the original TRed 3 version. The program was ran on a 4 core computer, which suggests the speedup of at most 75% in the running time. The combination of the 23.5532% speedup from before and the 75% here resulted in that 80.3292% speedup.

3. After adding the suffix arrays algorithm into the program, we notice a slowdown in the running of the program. For the single replacement of the call to the `KxK` matrix building algorithm by the suffix arrays algorithm, we noticed that the program ran slower by **15.5%** than the multi-threaded program. This is about the same execution time as the original TRed 3 version.

When running the corresponding programs on Chromosome 2, which is the currently longest chromosome sequence, we noticed speedups of similar percentage magnitude. However, we did not run the suffix arrays program on Chromosome 2 once we noticed that the running time is slow already for the shorter Y Chromosome.

From these results, we understand that the current fastest program is the one implementing multi-threading. However, we do not discard the suffix arrays version for future research purposes.

2.4 Making TRed Version 4 Publicly Available (Goal №4)

We named the upgraded program **TRed Version 4**. The multi-threading version is named `v4.0`, while the suffix arrays version is named `v4.1-alpha`.

Within the CISC 5001 repository that we built especially for this class, which can be found at https://github.com/mary060196/CISC5001_Research_Project_Implementing_TRed_Efficiently we updated the programs on which we worked this semester, including the running of the original TRed version 3 program, the comparison in the execution time of C++ and C strings, and a few of the upgrades that we made throughout the semester to the program. The multi-threading version and the suffix arrays version are placed within the `translation21` and `translation22` directories inside the repository, respectively.

To ensure easy access of researches to the programs, we created two `git` branches within the repository and named them `multi-threaded` and `suffix-array`. In these branches, we placed copies of the contents of `translation21` and `translation22`, respectively.

The `multi-threaded` branch can be found at:
https://github.com/mary060196/CISC5001_Research_Project_Implementing_TRed_Efficiently/tree/multi-threaded
and the `suffix-array` is at
https://github.com/mary060196/CISC5001_Research_Project_Implementing_TRed_Efficiently/tree/suffix-array

Furthermore, we created GitHub releases, which are special platforms that allow viewers to download zipped (`.zip`) or tarred (`tar.gz`) content directly, for each of the branches.

The `v4.0` release, pertaining to the `multi-threaded` branch is at
https://github.com/mary060196/CISC5001_Research_Project_Implementing_TRed_Efficiently/releases/tag/v4.0

while the `v4.1-alpha` release, pertaining to the `suffix-array` branch is at
https://github.com/mary060196/CISC5001_Research_Project_Implementing_TRed_Efficiently/releases/tag/v4.1-alpha

In each of the branches, we updated the `README.md` files to reflect the current specifications of the program, which include compilation and execution instructions, parameter description, and software/hardware requirements, all of which can help the user of the TRed version 4 program getting started.

The `README.md` file for the `multi-threaded` version is at
https://github.com/mary060196/CISC5001_Research_Project_Implementing_TRed_Efficiently/blob/multi-threaded/README.md
whereas the `README.md` file for the `suffix-array` version is at
https://github.com/mary060196/CISC5001_Research_Project_Implementing_TRed_Efficiently/blob/suffix-array/README.md

Last but not least, we constructed a GitHub Pages website, which carries the links to the two releases. The address of the website is:
https://mary060196.github.io/CISC5001_Research_Project_Implementing_TRed_Efficiently/
and it can be accessible by any internet-connected device.

3 Suggestions for Future Research

Since the not all the calls to the `KxK` matrix building algorithm were substituted by the suffix arrays algorithm, and since `v4.1-alpha` version takes more time to run than the `v4.0` version, we suggest the research directions below:

- Expansion of the API of the suffix arrays algorithm to encompass the `KxK` matrix building algorithm calls that remain in the current version of the program.
- Replacement of the currently used suffix arrays algorithms by faster ones. This includes the suffix array sorting algorithm (currently quicksort,) which, by analysis, is shown to execute on the order of $\mathcal{O}(n^2 \lg n)$. It could be replaced by a $\mathcal{O}(n)$ algorithm, which researches show exists.
- Investigation of whether suffix trees rather than suffix arrays cause faster execution time in the case of the TRed program.

References

[1] Cohen, B. (2015). Implementation of the Landau Vishkin Algorithm [GitHub Repository; Computer Software]. Retrieved from https://github.com/Brani/Implementation-of-the-Landau-Vishkin-Algorithm

[2] Landau, G. M., & Vishkin, U. (1989). Fast parallel and serial approximate string matching. *Journal of algorithms, 10(2), 157-169.* doi:10.1016/0196-6774(89)90010-2

[3] Sokol, D., Benson, G., & Tojeira, J. (2007). Tandem repeats over the edit distance. *Bioinformatics, 23(2), e30-e35.* doi: 10.1093/bioinformatics/btl309