README of the stab-forest source code release

Jelle Hellings

July 12, 2020

1 Introduction

This document describes the licensing details, content of the package, the environment in which we have tested the software, documentation on how to build the software, documentation on how to use the software, documentation on how to prepare the datasets, and documentation on all known issues.

2 Licensing details

The software package does not rely on other software packages to build and/or run (except for the compiler and the standard language library). The software written for this project is free to use under the terms of the well-known 2-clause BSD license:

3 Content of the package

The root directory contains license details, this readme file, and the following directories:

source Contains the main stab-forest source code, the source code for the experiments, and the source code of supporting `C++` tools. This directory contains the following files:

`algorithm.hpp`	supporting header file containing a three-way merge algorithm.
`block_list.hpp`	the block-list data structure, used internally by the stab-forest.
`interval.hpp`	the interval data type used to represent events together with the necessary comparison and predicate objects.
`raw_array.hpp`	a raw uninitialized array, used internally by the stab-forest.
`stab_forest.hpp`	the stab-forest data structure together with stab and multi-stab functionality.
`temporal_join.hpp`	the temporal join algorithms (sweep-join and skip-join).
`dataset.hpp`	supporting header file containing functions for reading dataset files.
`measure_join.hpp`	supporting header file for measuring the duration of temporal joins.
`measure_insert.cpp`	the 'constructing stab-forests' experiment.
`measure_jump.cpp`	the 'the threshold constant' experiment.
`measure_part_join.cpp`	the 'temporal joins' experiment.
`measure_window.cpp`	the 'multi-window-queries and temporal joins' experiment.
`min_max.cpp`	a dataset analysis tool to find the events with minimal and maximal duration.
`tool_split.cpp`	a dataset split tool to select a fragment of a dataset and split this fragment in two equal parts.

script Contains the Java-programs that are used to preprocess the datasets used in the experiments.

measure Contains the raw measurements as published in the paper.

4 Environment

The software was developed and tested on a workstation with an Intel Core i5-4670 CPU with 16GB of main memory and running Microsoft Windows 10, version 1703 (build 15063.608). To the best of our knowledge, the software does not rely on any system-specific behavior and should build and run without any issues on any other modern platform.

4.1 Software requirements

1. *A modern `C++` compiler (with `C++14` support).*
   To the best of our knowledge, the entire stab-forest source is fully `C++`14 standard-compliant. We have checked this using the appropriate compiler

switches (`-pedantic` for G++ and `/permissive-` for Visual C++). We have tested with the following compilers:

(a) g++ (GCC) 8.1.0, part of the GNU Compiler Collection, via Cygwin64.

(b) Microsoft C/C++ Compiler Version 19.13.26132 for x64, part of Visual Studio 2017.

All programs used for the experiment measurements, as published in the paper, have been compiled using Visual C++.

2. *Java JDK and JRE (Java SE 7 or higher).*
These are only used for tools that prepare the datasets. We have used `javac 1.8.0_144` and `java version "1.8.0_144"`.

5 Build instructions

The programs do not depend on any special libraries or compiler options that need setup prior to compilation. Below are the specific details for compiling each file.

5.1 Building the C++ source using G++

For each program (each `.cpp` file), a simple direct invocation of the compiler suffices:

```
g++ x.cpp -std=c++14 -o output_name
```

To enable optimizations and the use of system-specific instruction sets, we have used the standard options `-O3 -march=native`.

5.2 Building the C++ source using Visual C++

To build the stab-forest source code using Visual C++, one needs to disable warnings related to what Visual C++ deems unsafe parts of the standard C++ library (which includes basic algorithms such as `std::copy`). One does so by setting the preprocessor definition `_SCL_SECURE_NO_WARNINGS`. One does so in a Visual Studio project by going to the project properties and then navigating to Configuration Properties → C/C++ → Preprocessor → Preprocessor Definitions → `<edit...>` and add `_SCL_SECURE_NO_WARNINGS` to the first text field.

5.3 Building the script sources

The files `ExtractFlightData.java` and `SelectFlightDays.java` can be compiled using a standard java compiler:

```
javac ExtractFlightData.java
javac SelectFlightDays.java
```

6 Using the software

Each program is a simple command line tool, next follows brief documentation on how to use each program:

6.1 `measure_insert`

```
measure_insert data_file step_size runs
```

Measure append performance of the stab-forest and compare it with other data structures. The dataset is read from the file `data_file` and should be a single set of events in the standard dataset format. The `step_size` parameter controls the increments in which the measurement is (first `step_size` events are appended, then 2·`step_size`, and so on). The measurements are repeated `runs` times. The program writes measurements on the duration of each operation to the standard output (in milliseconds).

6.2 `measure_jump`

```
measure_jump runs
```

Measure temporal join performance on gap datasets (that are generated by the program). The measurements are repeated `runs` times. The program writes measurements on the duration of each operation to the standard output (in milliseconds).

6.3 `measure_part_join`

```
measure_part_join f_file s_file runs
```

Measure temporal join performance by joining the dataset in file `f_file` with the first 10%, . . . , 100% of the dataset in file `s_file`. The second dataset is sorted before use. It is assumed that the first dataset is already sorted lexicographically on (start, end)-time order. The measurements are repeated `runs` times. The program writes measurements on the duration of each operation to the standard output (in milliseconds).

6.4 `measure_window`

```
measure_window flights_file filter_file runs
```

Measure temporal join performance on the flight dataset in file `flight_file` by joining it with the dataset in file `filter_file`. It is assumed that the second dataset is a sequence of non-overlapping intervals (as used by the multi-window-query). The measurements are repeated `runs` times. The program writes measurements on the duration of each operation to the standard output (in milliseconds).

6.5 `min_max`

```
min_max data_file
```

Read a dataset from `data_file` and write to the standard output the details of the events with minimal and maximal duration.

6.6 `tool_split`

```
tool_split data_file f_file s_file size
```

Split the dataset in file `data_file` into two files containing, each exactly half of the events (randomly chosen). Write these resulting datasets to `f_file` (sorted lexicographically on (start, end)-time order) and the second dataset to `s_file` (randomly shuffled). If `part` is set, then before splitting the dataset, the number of events is reduced to the specified size. Else, the entire dataset is used. This tool is used to produce the input datasets for the `measure_part_join` program.

6.7 `ExtractFlightData`

```
java ExtractFlightData file_1 ... file_N
```

Script to parse the ‘Airline On-Time Performance Data’ dataset (see Section 7.1. This program accepts as parameters the file names the original CSV files containing a flight dataset. The parser expects the fields `YEAR`, `MONTH`, `DAY_OF_MONTH`, `DEP_TIME`, and `AIR_TIME`. The script filters out flights with incomplete records, translates the date and time of departure to the start-time (in minutes), and computes the end-time (in minutes) using the start-time and the air-time. The resulting dataset is sorted lexicographically on (start, end)-time order and written to standard output.

6.8 `SelectFlightDays`

```
java SelectFlightDays
```

Script that produces events that represent one-day periods in each month (compatible with the dataset representation of the ‘Airline On-Time Performance Data’ as produced by `ExtractFlightData`. The 7th day of every month is selected, starting with July, 2007, and ending with June, 2017.

7 Obtaining and preparing the datasets

All `C++` programs that process datasets expect the input to be a plain text file with, on each line, a pair of whitespace separated positive integers indicating start and end-time of an event. Depending on the tool, the dataset should be sorted lexicographically on (start, end)-time order (See Section 6 for details).

7.1 Airline On-Time Performance Data

The raw CSV data files can be obtained from

`https://www.transtats.bts.gov/Tables.asp?DB_ID=120`,

where also the manual to the structure of these raw data files can be found. We have only selected the fields named `YEAR`, `MONTH`, `DAY_OF_MONTH`, `DEP_TIME`, and `AIR_TIME`. We observe that the departure information is in local time (of the airport), and there does not seem to be readily available a way to look up the associated time zone information. Consequently, we interpreted each time as being in the same time zone (which should not matter for the purpose of benchmarking). We have downloaded the 120 data files for the period July, 2007–June, 2017. Next, we have fed all these files to the `ExtractFlightData` script, which outputs a dataset in the format expected by the various `C++` programs.

Using the script `SelectFlightDays`, we have constructed the 120 windows used in the 'multi-window-queries and temporal joins' experiment. Using the `tool_split` program with `size` set to $2,500,000$, we constructed the fragment of this dataset used in the 'temporal join' experiment.

7.2 Civil Unrest Event Data

The raw data file `ssp_public.csv` can be obtained from

`http://www.clinecenter.illinois.edu/data/event/speed/`,

where along the raw data files the file 'Codebook for Event Data File (PDF)' can be found that documents the structure of the raw data files. We imported this raw data file into Microsoft Excel 2016, removed all columns except for the following four columns:

`JUL_PSD`	Precise date of event start
`JUL_PED`	Precise date of event end
`JUL_EED`	Estimated date of event start
`JUL_LED`	Estimated date of event end

These dates are in the format: days since January 1st, 1945.

We combined these two event tables by selecting the precise start or end time when it is available, and otherwise selecting the estimated start or end time. The resulting columns we have stored in a plain text file (in the dataset format expected by the `C++` programs).

Using the `tool_split` program with `size` not set, we constructed the fragment of this dataset used in the 'temporal join' experiment.

8 Known issues

There are no checks on unsigned integer overflows while computing array sizes and allocating arrays. Hence, when dealing with very large datasets, these allocations will allocate wrongly sized objects which will lead to out-of-bound behavior.

Although explicitly dealing with this issue by raising an exception is not hard, it would clutter a portion of the forest-point merge code. Hence, for clarity, we have chosen to not incorporate all necessary checks. To fix all issues, small changes are necessary in the file `stab_forest.hpp`, where the following fragment of the method `void stab_forest::merge_forest_points` needs to be rewritten:

```
/* Compute the sizes of the new left-list and max-list. */
size_type dll_size = std::distance(dll_it, dll_ed_end(left));
size_type nll_size = std::distance(nll_it, nll_ed_end(left));
size_type ml_add_size = left.ll_size - (dll_size + nll_size);

/* Construct new raw arrays to hold the data. */
event_array raw_left_list(dll_size + 2 * nll_size);
event_array raw_max_list(right.nll_size + right.ll_size +
                            2 * ml_add_size);
```