

BACHELOR THESIS
OPTIMIZED PATTERN MATCHING IN GENOMIC DATA

Synopsis

MARTIN WESTH PETERSEN - MQT967
KASPER MYRTUE - VKL275

23. Februar 2015

1 Problem definition

Is it possible to re-implement/modify `scan_for_matches`, so that it has an equal or increased performance?

As a minimum, our product will contain the following features:

- Constructing literal pattern units (e.g. "AGUUG") that are used for finding a specific sub-sequence in the database sequence.
- Allowing for wildcards (e.g. "AGWUUG" where "W" matches multiple other bases).
- Allowing any pattern unit a specified number of insertions, deletion and mismatches (e.g. "AGGUAAA[2,0,3]").

In order for our program to meet in the minimum requirements for the users, the following core features will also be needed. If time allows, as many as possible will be implemented as well.

- Constructing ambiguous pattern units (e.g. "4..8") that match any sub-sequence with a possibility for a flexible range.
- Combining pattern units to a full pattern that defines the full search criteria when scanning the database sequence.
- Defining variables for referencing pattern units (e.g. "p1=5..6") that can be used to find related patterns that are unknown before search (e.g. p1=5..6 p1[1,0,0])
- The \sim symbol can be inserted in front of any pattern unit to indicate that we are looking for the reversed complement of that pattern unit.
- A more optimized way of searching for matches, given complex patterns consisting of multiple pattern units. E.g. the order of which the different pattern units are searched can be optimized to increase performance instead of going through the pattern units from one end to the other.

2 Limitations

In case of excess time, secondary features may be implemented:

- Is it possible to optimize the backtracking algorithm for string search, so that the order of the possibilities tried regarding insertions, deletion and mismatches results in a conclusion faster.
- Logical "or" between patterns (e.g. "(AUUG | AGGG)") that matches either of the sub-sequences.

- A possibility for defining custom "pairing rules" (e.g. "r1=au,ua,gc,cg,ga,ag") that can be used for defining allowances when comparing a reversed complement pattern unit (e.g. "r1~p1").
- An analysis of the complexity and running time of the program.

Every feature from `scan_for_matches` beyond what has been mentioned will not be implemented.

3 Motive

Pattern matching functionality for strings in genomic data is very useful, but requires a good performance due to huge amounts of data. `Scan_for_matches` serves this purpose, but big improvements in performance can be made. On top that, the code is poorly documented, lacks version control and the code is hard to read and maintain.

4 Tasks and schedule

Below is a list of the tasks that this project consists of:

- **Research**
 1. Why is `scan_for_matches` fast despite the use of a backtracking algorithm? This research task is about reading and understanding the vital parts of the `scan_for_matches` code so that we understand the overall ideas and algorithms, and can reuse them if needed.
 2. How should the interface be improved to best suit the users? E.g. Should the program be started and patterns specified in a command-line manner or reading of files? How should the output be displayed or stored?
 3. What functionality should we be focused on implementing. This task ensures that the most important functionality is identified, so a proper prioritization can be made. This task is already completed and the result is shown as the list of primary features and the list of secondary features.
 4. Researching what methods and algorithms for string-handling that exists, and exploring different options for choosing a programming language to work with.
- **Analysis**
 1. Choosing a language for implementation. (It will probably be C or C++).
 2. Figuring out the overall methods and algorithms that we are going to use.
 3. Deciding on how the user interface should work, e.g. how will patterns, input and output be provided, displayed and/or stored.
- **Design**

1. Designing the overall structure of the program.

- **Implementation**

1. First prototype that can search the database sequence for a single literal pattern unit and provide the output.
2. Second prototype allows ambiguous patterns with flexible length, and the possibility of combining pattern units into one single pattern.
3. Third prototype includes all the 6 core functions defined earlier.
4. The fourth prototype will include the optimized order of searching for the different pattern units.

- **Testing**

1. A thorough test of our implementation on the given data to verify that each of the implemented features work
2. Testing the running time of both simple and complex patterns with the original `scan_for_matches` against our implementation.

- **Documentation** - The documentation will start being written when we start the implementation period and finish before final report is due. It will consist of the comments written in the code and a user guide provided together with the program.

- **Report**

1. Midway report
2. Final report

Figure 1 shows our schedule for the project, with the the implementation period consisting of the 4 different milestone prototypes of the program. The midway report and final report is shown as dots in the bottom.

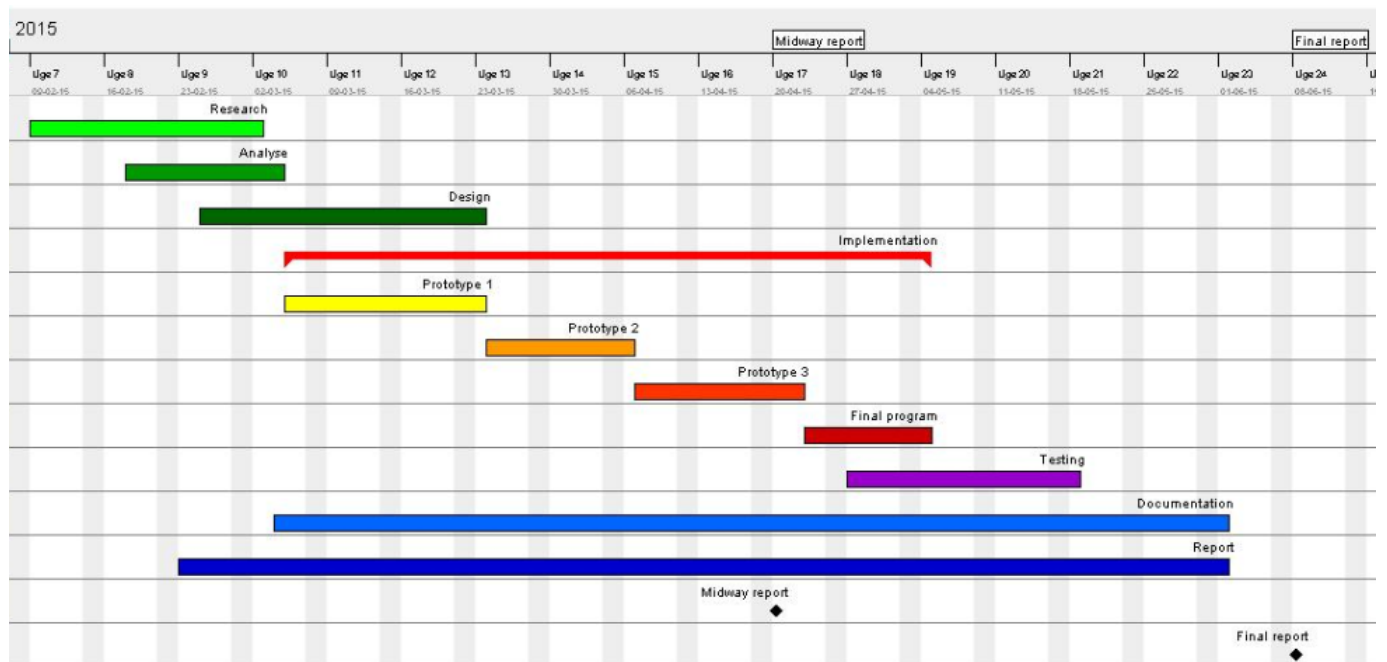


Figure 1: Schedule