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_mathces, so that it has an equal or increased performance?

Our program will contain the following core features from scan_for_matches:

- Constructing literal pattern units (e.g. "AGUUG") that are used for finding a specific sub-sequence in the database sequence.
- Allowing 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]").
- 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. (E.g. "AUUA 4..8 AUCCUCCC[2,1,0] each pattern unit separated by a space).
- 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 ∼ symbol can be inserted in front of any pattern unit to indicate that we are looking for the reverse complement of that pattern unit.
- An optimized order of which the different pattern units are searched, instead of going through the pattern units from one end to the other. This optimization should speed up the search, especially with complex patterns.

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 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.

We will not do the following tasks:

- Testing with users (E.g. biologists at the university or other potential users of the program).
- Looking at other tools for dna, rna pattern matching (E.g. grep or other tools).

Other features from scan_for_mathces beyond what has been mentioned will not be implemented.

3 Motivation

Pattern matching functionality is essential and unavoidable when looking through genomic data such as DNA or RNA, for example in order to identify DNA from a known organism/animal. Huge amounts of data makes it extremely inefficient to manually find these patterns, so there's a need for clever and efficient software to do this.

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 as much of the good parts as possible.
  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.
- **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 the use. How much code can be reused and what will need to be re-implemented.

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. This phase is of course completely dependent on the result of the analysis phase. At this point it will be more clear to which degree we will simply modify scan_for_mathces or make a full re-implementation or (most likely) somewhere in between.

- **Implementation**

  1. First prototype/milestone can search the database sequence for a single literal pattern unit and provide the output.
  2. Second prototype allows searching with an allowed number of mismatches, insertions, and deletions.
  3. In the event that the second prototype lives up to all expectations, third prototype includes 2 more functionalities from the list given in the problem definition.
  4. In the event that the third prototype lives up to all expectations, the fourth prototype will include the rest of the core functionality from the problem definition.

- **Testing**

  1. A thorough test of our implementation on the given data to verify that each of the implemented features work.
  2. Bench marking the running time of the original scan_for_mathces against our modified 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. The design phase shown in the figure refers to the overall ideas of design. We will re-iterate the choice of design along with the implementation. Same argument goes for the testing phase. Testing on a smaller scale will be an integrated part of the implementation.

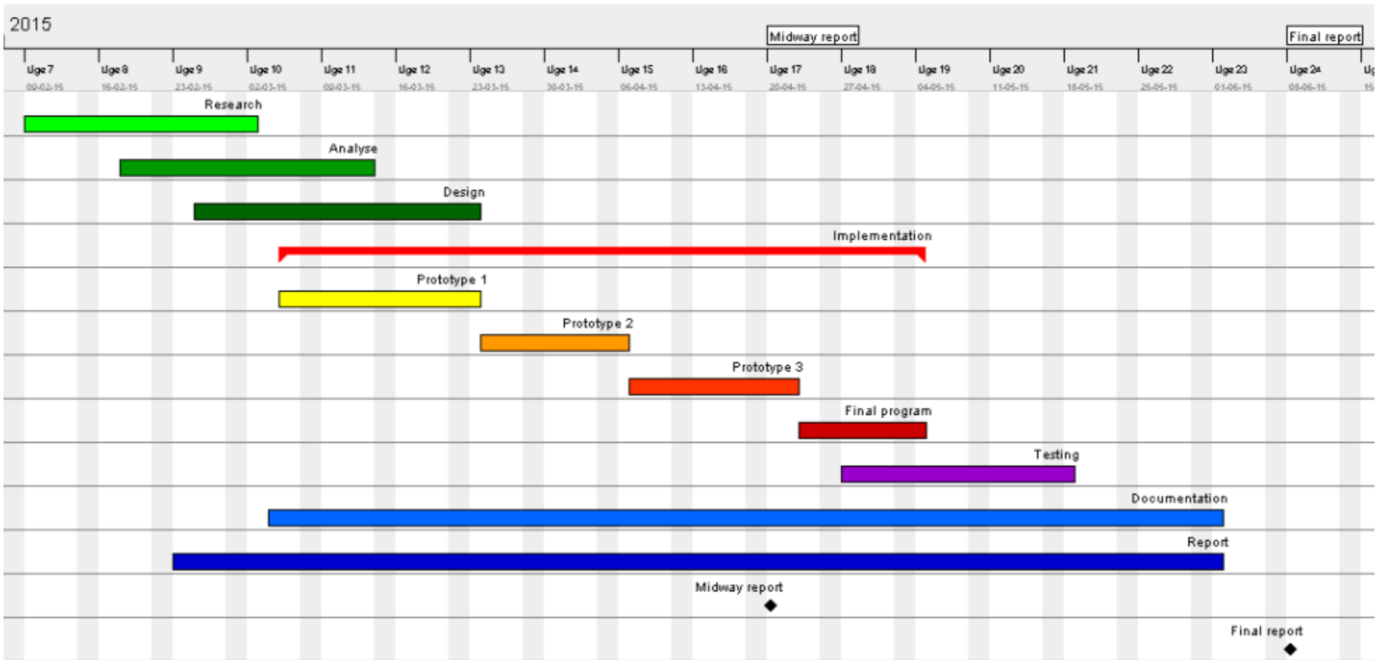

Figure 1: Schedule