Evaluation plan

## Design evaluation

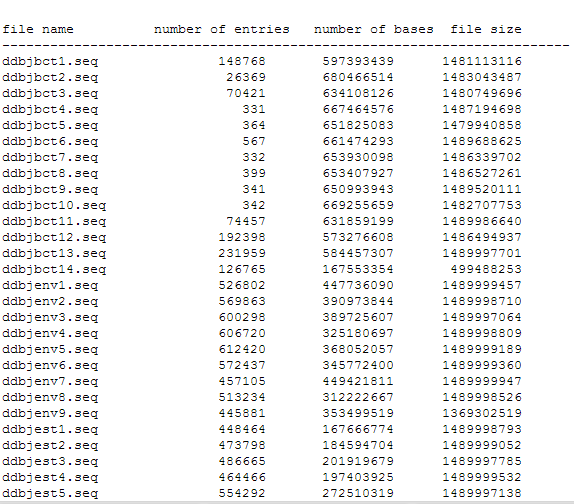
We are working on the traditional online approximate string matching problem which is stated as follows : Given a character string S over alphabet σ, a pattern string q, and an error bound k, find all substrings ,s, in S such that ed(s;q)<=k, where ed(s,q) is the edit distance between s and q.  
Edit distance here is defined as “the minimal number of insertions, deletions and substitutions to make two strings equal”(this is known as the Levenshtein distance). We aim to optimize performance for the case in which k<<|q| and σ is small. The text is not available to preprocess beforehand.

Searching for specific gene patterns in DNA sequences is a common real-world application of this problem. In this case the alphabet size is relatively small, and errors occur infrequently due to mutation in the organism or inaccuracies in measurement. Since this application matches our goals we decided to use it for testing our implementation.

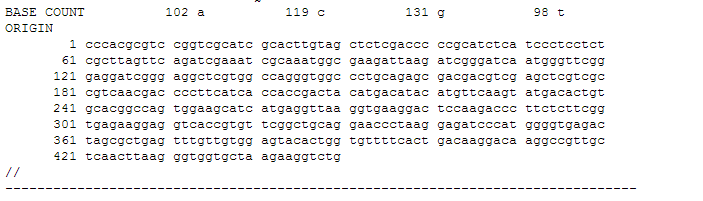
## Test set

DNA sequences consist of 4 types of characters (representing the base pairs A, C T,G) The DNA Data Bank of Japan (DDBJ) is a biological database that collects DNA sequences and the only nucleotide sequence data bank in Asia which is officially certified to collect nucleotide sequences from researchers. Nucleotide sequence records organismic evolution more directly than other biological materials. One of the sequences available for download from DDBJ is the human genome. In the human genome, about 3 billion bases are arranged along the chromosomes in a particular order for each unique individual. Fig. 1 shows the files available and their sizes. Fig. 2 shows a sample sequence.

Fig.1



The length of the search pattern can potentially be several thousands of characters. For purposes of testing these patterns are randomly generated from input data, and errors are introduced with a Poisson distribution. The error bound k is at most 20.

Fig.2

The parameters to vary in the algorithm are pattern size, edit distance, block size, size of input text. Since memory bandwidth is a bottleneck, one of the design principles is to minimize the time spent on memory transfer s to global memory. Measurable quantities would be memory footprint, speedup in execution time, CPU idle time (whether CPU is left free to do other work in the meantime). The following criteria are used to evaluate the algorithm:

## Accuracy

Results obtained from our GPU parallelized code is compared against theresults of standard application, AGREP, which is an open source fast , flexible and efficient tool for approximate string matching. Both applications are run on same input and pattern strings so that output can be verified [Wu 89].

## Speedup

Our GPU parallelized version of code is compared against standard CPU BitParallel NFA version[Hyyro Navarro 91] to calculate performance speedup in execution time. The input text is assumed to be loaded into main memory. The average time for execution is measured for each choice of parameters.

## Scalability

We wish to determine whether the problem scales well with input size/addition of more processors. As mentioned before, one key factor would be maximizing the ratio of the time spent in memory transfers to the time spent in processing. The algorithm is run with increasing input sizes from 4Mb to 1GB. By plotting the speedup obtained, we would get a picture of whether it scales linearly. The largest input sizes are such that the text cannot be accommodated entirely within the GPU’s shared memory. Increasing input size beyond this point is not expected to degrade the processing time for a unit of data significantly.

## References

[Hyrko Navarro 89]Faster Bit-Parallel Approximate String Matching, Heikki Hyrko and Gonzalo Navarro Proceeding of IEEE Symposium on Parallel Computing 1981, p160-189

[Wu 89] Agrep-A Fast Approximate Pattern-Matching Tool, Sudi Wu and David Manber, Proceedings of USENIX conference 1989, p23-69.