A faster implementation of association mapping from k-mers

Reimplementing portion of an alignment free association mapping method using k-mers to improve execution time of whole method

Zakaria Mehrab[1,2], Jaiaid Mobin[1], Ibrahim Asadullah Tahmid[1], and Atif Rahman[1]

[1]**Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Bangladesh**
[2]**Department of Computer Science and Engineering, United International University, Bangladesh**

Abstract
Genome wide association studies (GWAS) attempt to map genotypes with phenotypes in organisms. This is typically performed using microarray data or by aligning whole genome sequencing reads to a reference genome. Both approaches require knowledge of a reference genome and it limits identifiable variant types. This caveat can be removed using non alignment based association mapping method based on k-mers from sequencing read obtained from individuals. Here we reimplement an alignment free association mapping method to improve its execution time. We have tested it on an *E. Coli* ampicillin resistance dataset and compared our implementation performance with original one.

Keywords
Association mapping, GWAS, alignment-free, Categorical, Phenotype

Introduction

Genome wide association is the process of associating physical traits with genotypes of an individual. DNA sequence or genome of a species is almost identical in the individuals belonging to the same species. However, there are some portions in the sequence which are different, giving rise to different *phenotype* (outward expression of genome sequence) for a particular trait. For example, in our work we used a dataset which has genome information about two different categories of *E.coli* showing different phenotype regarding ampicillin resistance. One is ampicillin resistant (case) and the other is not (control). Genome wide association study (GWAS) tries to find the responsible genotype for a particular phenotype.

Most GWAS methods that have been used so far address only one type of variant called single nucleotide polymorphism (SNP). This method uses microarrays which in turn needs reference genome and SNP position information. An alternate approach is to use whole genome sequencing reads where reads are first mapped to a reference genome to call variants and those are tested for association with the phenotype.

Recently, Rahman et al[1] have presented a way where the reference genome is not required. They have used k-mer counts as a feature to find associated k-mer and then assemble them to form the sequence responsible for a categorical phenotype. First, they use Jellyfish [2] to count k-mers in reads from each individual. Second, using likelihood ratio test, they find k-mers with significantly different counts in cases and controls. Next, population structure is determined from k-mer counts using Eigenstart [3, 4]. In the last step, association to k-mers after correcting for population structure is determined. Then, the k-mers found associated may be assembled to get a sequence for each associated loci. The strength of this work is that it is applicable in case of no or incomplete reference genome information.

We have worked on a portion of the above mentioned pipeline. The goal of our work is to reduce the execution time of author's implementation. Here, we have reimplemented the last step of the pipeline using C++ which was previously implemented in R. We have also extended support for Jellyfish 2 and implemented Benjamini–Hochberg procedure [5] which can be used to correct for multiple test when the study is underpowered for Bonferroni correction. We have tested our work with a given dataset on *E.coli* ampicillin resistance and have compared it's output with previous code's output. We have also compared the performance in runtime. Our code has made the execution faster and the output is quite similar to the original code's output.

Methods

Population stratification is a known confounder in association studies. Without correcting for this confounding factor, one may falsely associate non-significant genotypes with phenotypes.

In the previous literature, some confounders were found for the significant k-mers after the Poisson Distribution base likelihood ration test. To correct for these confounders, we fit two logistic regression models on the phenotype. First, against potential confounders and then against k-mer counts normalized by the total number of k-mers in the sample.

By default we include the first two principal components in our analysis. First we fit a model where we use the principal components and total number of k-mers in sequencing reads from each individual. This model corroborates with our null hypotheses that the k-mers are not significant for the phenotype. We refer to this as the null model.

Having fitted the null model, we fit additional models, which we refer to as alternate models, against the count of each k-mers as well as the confounding factors. These models correspond to our alternative hypotheses. Each of these models has an extra feature which was not in the null model. The extra feature is counts of the k-mer we are testing for association. If a k-mer is not significant, then adding the k-mer counts in the logistic regression model should not provide significantly improved fit. In other words, the likelihood of the phenotypes of the individuals should be similar in both null model and this alternate model. On the other hand, if the k-mer is significant, then the likelihood derived from the alternate model should be significantly higher than the null model. So, depending on the likelihood ratio, we can decide whether to reject the null hypotheses or not.

It should be noted that, by setting an alternate model's weight parameter associated with the k-mer count to 0, we obtain the null model. Therefore, the null model is just a special case of the alternate model and we can use likelihood ratio test to quantify the additional goodness of fit provided by each k-mer after the confounding factors.

So the work flow is as follows: a) A logistic regression model, the null model, is fitted against the confounders and probability of responses are used to compute likelihood. b) Another logistic regression model, the alternate model, is fitted against the confounders as well as k-mer counts and likelihood under this model is computed.

Since the former model is a special case of the latter, negative logarithm of the likelihood ratio is asymptotically chi-squared distributed with one degree of freedom which is then used to calculate a p-value.

Implementation Details

The previous work had been implemented using R's glm function (for fitting logistic regression models) and ANOVA function (for testing the goodness of fit). The whole process was quite time consuming. Our focus is to implement this whole pipeline in C++ and thereby improving the performance.

R uses IRLS (Iteratively Re-weighted Least Square) method to fit the model [6]. Therefore in our implementation of the glm function in C++, we also used IRLS for fitting the model. Initially we attempted to implement using gradient descent method as it was more convenient. However, gradient descent was subject to local minima or late convergence. As our prime target was obtaining a result similar to R while keeping the running time efficient, we decided to discard gradient descent.

Iteratively re-weighted least squares for finding the MLE (Maximum Likelihood Estimate) for logistic regression is a special case of Newton's algorithm. If the problem is written in vector matrix form, with parameters $w^T = [\beta_0, \beta_1, \beta_2, \ldots]$, explanatory variables $x(i) = [1, x_1(i), x_2(i), \ldots]^T$ and expected value of Bernoulli distribution $\mu(i) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}(i)}}$, the parameters $\mathbf{w}$ can be found using the following iterative algorithm:

$$\mathbf{w}_{k+1} = w_k - \alpha\left(\mathbf{X}^T\mathbf{B}_k\mathbf{X}\right)^{-1}\mathbf{X}^T\left(\boldsymbol{\mu}_\mathbf{k} - \mathbf{y}\right)$$

where α is the learning rate, $\mathbf{B} = \mathrm{diag}(\mu(i)(1-\mu(i)))$ is a diagonal weighted matrix, $\boldsymbol{\mu} = [\mu(1), \mu(2), \ldots]$ the vector of expected values,

$$\mathbf{X} = \begin{bmatrix} 1 & x_1(1) & x_2(1) & \ldots \\ 1 & x_1(2) & x_2(2) & \ldots \\ \vdots & \vdots & \vdots & \end{bmatrix}$$

the dataset in matrix form, and $\mathbf{y}(i) = [y(1), y(2), \ldots]^T$ the vector of response variables.

It can be observed that the $\mathbf{B}$ matrix is of dimensionality $N \times N$, where N is the number of instances. For large number of data, this matrix can largely affect the performance of the implementation. However, we need only to keep the values along the diagonal as this is a diagonal matrix; thereby precluding the potential performance drawbacks. The pseudo-code of both glm and our implementation are given in Algorithm 1 and 2 respectively.

In our implementation, there are two hyper parameters those need to be tuned before running the process. One is the learning rate of the logistic regression model and the other is the number of maximum iterations allowed for convergence. We used maximum iteration as 25 because we found that the glm implementation of R has 25 maximum iteration by default [7]. We used different values as the learning rate and found 0.1 to give the most similar result to the result produced by R. However, we suspect a better learning rate may co-exist.

```
Arguments: $\mathbf{X}$, $\mathbf{y}$, $\alpha$, MaxIter
Result: Weight vector $\mathbf{w}$
Initialize $\mathbf{w}_0$
$k := 0$
while $k < MaxIter$ do
    Compute $\boldsymbol{\mu}_k$ using $\boldsymbol{w}_k$ ;
    Compute $\mathbf{B}_k = \mathrm{diag}(\mu(i)(1-\mu(i)))$ using $\boldsymbol{\mu}_k$ ;
    Compute $Error$ using $\mathbf{y}$ ;
    if $Error < \epsilon$ then
        break;
    $\mathbf{w}_{k+1} := w_k - \alpha\left(\mathbf{X}^T\mathbf{B}_k\mathbf{X}\right)^{-1}\mathbf{X}^T\left(\boldsymbol{\mu}_\mathbf{k} - \mathbf{y}\right)$ ;
    $k := k + 1$ ;
end
return $\mathbf{w}$
```

Algorithm 1: glm

```
Inputs:
A (Phenotype of each individual)
B (Count of each k-mers in each individuals)
Z (Principal Components)
total (Total number of k-mers)
Read A, B , Z, total from respective files
$\alpha := 0.1$
$MaxIter := 25$
Compute y ($y_i \in \{0,1\}$) using A
$X_{null} := \{Z[1], Z[2], total\}$
$Model_{null} := glm(X_{null}, \mathbf{y}, \alpha, MaxIter)$
foreach k-mer $k_i \in \boldsymbol{B}$ do
    Compute the proportion of $k_i$, $\boldsymbol{count}_i$, in each
      individual using B
    $X_{alt} := \{Z[1], Z[2], \boldsymbol{count}_i, total\}$
    $Model_{alt} := glm(X_{alt}, \mathbf{y}, \alpha, MaxIter)$
    Compute $\boldsymbol{Likelihood}_{null}$ of A using $Model_{null}$
    Compute $\boldsymbol{Likelihood}_{alt}$ of A using $Model_{alt}$
    $\Lambda := \frac{\boldsymbol{Likelihood}_{alt}}{\boldsymbol{Likelihood}_{null}}$
    $p := chisq(2ln\Lambda, 1)$
    print p
end
```

Algorithm 2: FastHawk

Our full implementation can be found in Github. [8]

Results

In this section we will describe our experimental setup and result of our experiment. Our target was to re implement the portion which works to remove significantly associated read to phenotype due to population rather than genotype. The target of this reimplementation is to improve execution time. There are two program in our implementation one is **log_reg_case.out** and another is **log_reg_control.out**. Corresponding original scripts are **log_reg_case.R**. and **log_reg_control.R**.

We have tested our code on an *E.coli* dataset on ampicillin resistance phenotype. Our input is k-mers information from the previous steps of the pipeline, count of the k-mers, total number of k-mers and information about the read (case/control) from the dataset. To test the correctness of the code we have taken original R script's output as ground truth and compared Mean Square Error (MSE) and standard deviation of square error. 1 and 2 show the similarity in p-value of case and control, respectively, between the two outputs. Also we report randomly chosen 10 line from our implementation output and corresponding R script output. To measure the time performance we have run both code in the same environment.

Experimental Setup

All the reported results are obained from the execution on a system with Intel Core i3 CPU M370 @2.40GHz, 2GB Ram, Ubuntu18.04. The CPU of the system has 2 physical cores. The implementation is based on C++ language. It has used alglib[9], numerical analysis library, for statistical test. All programs are compiled using g++ 7.3.0 with

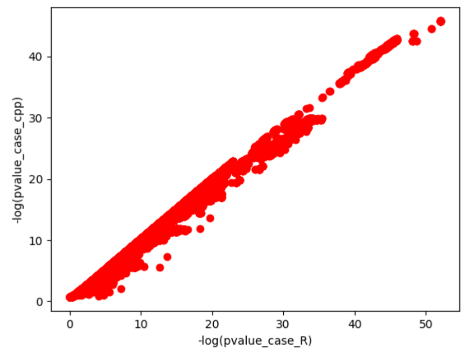

Figure 1. A comparison of the P-values in case samples found from implementation in R and implementation in C++

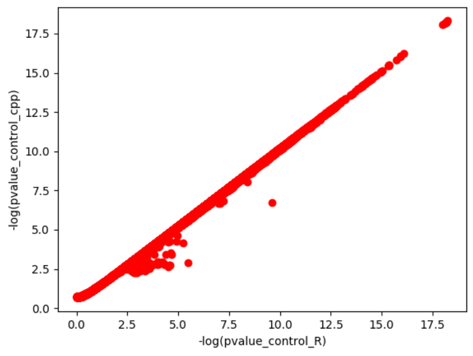

Figure 2. A comparison of the P-values in control samples found from implementation in R and implementation in C++

no optimization flag. Time of execution is measured by time cmd of linux .

Test Dataset

Our portion of interest is a step of the whole pipeline. It uses the output of previous steps. We have got this pre-processed dataset by running the original pipeline. The dataset is *E.coli* ampicillin dataset.
This dataset contains 200,000 kmers. which means The output of dataset will contain 200,000 p-values which gives the significance of association of the k-mers.

Code Hyper Parameters

There are 3 hyper parameters in code. They are described below

learning_rate learning rate of the logistic model fitted in the code

max_iter maximum number of iteration the logistic model use

CHUNK_SIZE number of sample the model reads at a time while fitting the model

Used values of the parameters are given in table1

Table 1. Hyperparameter used in code

Hyper parameter Name	Value
learning_rate	0.1
max_iter	25
CHUNK_SIZE	10000

Comparison with Original Output

Both of our programs give a sequence of p-value (thus signifies k-mers association). As, this outputs are gained from logistic model which needs convergence there are some randomness in the output and it is likely that two implementation will produce non-identical results. In our experiment we have compared the output with original implementations output which are two R scripts.
As the sequence of numbers are like signals/image we have measured their similarities using Mean of Square Error (MSE) and standard deviation of square error σ^2. Mean of square will signify that how much near our outputs to the R output on average and std. deviation will show if there are many/less sample which show results further than the mean. So, both MSE and std. deviation should be small. To show individual sample value quality we have shown 10 randomly chosen line from our output and corresponding R output.
If the sequence of p value from our program is $< p_1, p_2, ..., p_n >$ and from R script output p-value sequence is $< R_1, R_2, ..., R_n >$, where n =number of p-value

$$MSE = \frac{1}{n} \sum_{i=1}^{i=n} (R_i - p_i)^2$$

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^{i=n} (MSE - p_i)^2$$

Table 2. Output comparison with R script

Program	MSE	σ^2
log_reg_case.out	0.000812	0.000047
log_reg_control.out	0.000639	0.000043

Execution Time

We can see from Figures 3 and ,4 that for both program our implementation outperform R implementation by almost 3× for single worker. In case of multiple R worker (process) our program still beats R implementation. Another thing to note that, this programs are CPU intensive tasks. So, multi threading more than number of CPU core may not bring significant improvement. We can see that from our data. For two workers run time almost halved but increasing more than two doesn't bring improvement like that and our experiment environment has two CPU cores.

Table 3. Randomly chosen 10 lines from log_reg_case.out and log_reg_case.R

sample#	our method	R
82930	0.0090807	0.01123317
55080	0.0015562	0.001867564
36901	2.88744e-05	3.070927e-05
164295	0.00449161	0.005660282
92016	0.00115249	0.001027722
153789	0.00188703	0.002325312
137286	0.00254658	0.003145136
168576	0.00220118	0.002672731
89865	0.0246166	0.02368169
136616	0.00883812	0.01116087

Table 4. Randomly chosen 10 lines from log_reg_control.out and log_reg_control.R

sample#	our method	R
191295	0.00166103	0.002013221
87019	0.00734845	0.009029687
178206	0.00298255	0.003646629
47273	0.000438268	0.0005345342
109416	0.00185097	0.002271831
97158	0.000678963	0.0008223507
3460	0.00629517	0.007707163
56539	0.00587483	0.007294636
192390	0.0285446	0.03743725
197894	0.00113821	0.001382621

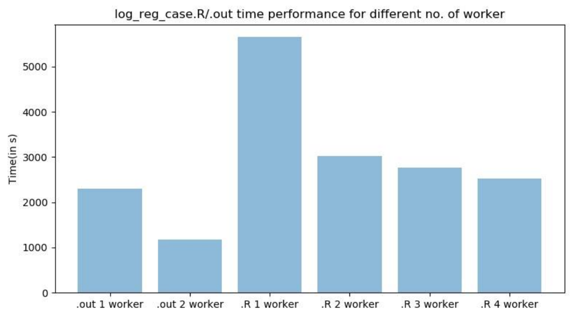

Figure 3. A comparison of execution times of implementations in R and in C++ on case samples

Discussion

In this work we have re-implemented a method to make it more efficient with respect to execution time. Our obtained results has shown that results from our implementation is showing small deviation from the original implementation's output while reducing it's execution time (nearly 3× faster).

Although our implementation is showing promising result

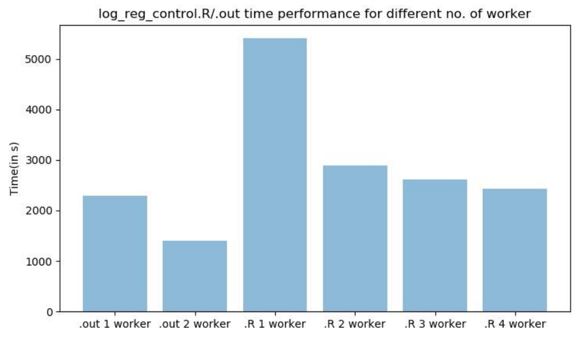

Figure 4. A comparison of execution times of implementations in R and in C++ on control samples

there are some rooms to improve. For example, changing learning rate is showing some variance in our result. So, we can tweak the learning rates to find a more suitable result.

References

[1] Atif Rahman, Ingileif Hallgrímsdóttir, Michael Eisen, and Lior Pachter. Association mapping from sequencing reads using k-mers. *eLife*, 7:e32920, 2018.

[2] Guillaume Marçais and Carl Kingsford. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770, 2011. doi: 10.1093/bioinformatics/btr011. URL `http://bioinformatics.oxfordjournals.org/content/27/6/764.abstract`.

[3] Nick Patterson, Alkes L Price, and David Reich. Population structure and eigenanalysis. *PLoS genetics*, 2(12):e190, 2006.

[4] Alkes L Price, Nick J Patterson, Robert M Plenge, Michael E Weinblatt, Nancy A Shadick, and David Reich. Principal components analysis corrects for stratification in genome-wide association studies. *Nature genetics*, 38(8):904, 2006.

[5] Yoav Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the royal statistical society. Series B (Methodological)*, pages 289–300, 1995.

[6] R manual. Fitting Generalized Linear Models. URL `https://stat.ethz.ch/R-manual/R-devel/library/stats/html/glm.html`.

[7] DataScience StackExchange. Number of Iterations in R glm. URL `https://datascience.stackexchange.com/a/16811`.

[8] J. Mobin Z. Mehrab. Faster Association Mapping using K-mer Counts. URL `https://github.com/dmehrab06/Fast_HAWK`.

[9] Sergey Bochkanov. ALGLIB. URL `www.alglib.net`.