A faster implementation of association mapping from k-mers

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
[*]Corresponding author. Email: atif@cse.buet.ac.bd

Abstract

Genome wide association studies (GWAS) attempt to map genotypes to phenotypes in organisms. This is typically performed by genotyping individuals using microarray or by aligning whole genome sequencing reads to a reference genome. Both approaches require knowledge of a reference genome which limits their application to organisms with no or incomplete reference genomes. This caveat can be removed using alignment-free association mapping methods based on k-mers from sequencing reads. Here we present an implementation of an alignment free association mapping method [1] to improve its execution time and flexibility. We have tested our implementation on an *E. Coli* ampicillin resistance dataset and observe improvement in performance over the original implementation while maintaining accuracy in results.

Introduction

Association mapping is the process of associating phenotypes with genotypes. In genome wide association studies (GWAS), individuals are typically genotyped using mircoarrays or by aligning sequencing reads from individuals to a reference genome. However, both these approaches require a reference genome of the organism which makes them inappropriate for association mapping in non-model organisms with incomplete reference genome or none at all.

To address this issue, a number of alignment free methods for association mapping have been developed [1, 2, 3, 4, 5]. They are based on finding k-mers i.e. contiguous sequence of length k in sequenced reads and identifying k-mers associated with the phenotype.

Rahman et al. [1] have presented a tool named Hawk where they use frequency of k-mers to find associated k-mers and then assemble them to form the sequences associated with a phenotype. First, they count k-mers in reads from each individual using Jellyfish [6]. Second, using likelihood ratio test, they find k-mers with significantly different counts in case and control samples. Next, population structure is determined from k-mer counts using Eigenstrat [7, 8]. After that, association to k-mers after correcting for population structure is determined. Finally, the k-mers found associated may be assembled to get a sequence for each associated loci.

Here we re-implement Hawk with the goal to reduce its execution time and make it more user friendly. We have re-implemented the step for finding associated k-mers after population structure correction using C++, which was previously implemented in R. We have also

extended support for Jellyfish 2 and implemented Benjamini–Hochberg procedure [9], which can be used to correct for multiple tests when the study is underpowered for Bonferroni correction. We have tested our implementation with a dataset on *E.coli* ampicillin resistance and have compared its output with the output of the original implementation. We have also analyzed the execution times of the two implementations. Our implementation is faster and more flexible to run compared to the original implementation while maintaining accuracy.

Improvement and New Features

Here we will describe summary of the improvements and new features we have added. Results supporting the improvements is presented in the following section.

Re-implementation of correction for population structure

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 HAWK pipeline, population structure was estimated using Eigenstrat [7, 8] and subsequently p-values were adjusted for population structure using the glm function (for fitting logistic regression models) and ANOVA function (for testing the goodness of fit) in R. We re-implement this process in C++ and thereby improving the performance.

R uses IRLS (Iteratively Re-weighted Least Square) method to fit the model [10]. Therefore in our implementation of the glm function in C++, we also used IRLS for fitting the model. 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 $\mu(i) = \dfrac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}(i)}}$, the parameters $\mathbf{w}$ can be found using the following iterative algorithm:

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

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

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

is the dataset in matrix form, and $\mathbf{y}(i) = [y(1), y(2), \ldots]^T$ is 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 volume of data, this matrix can largely affect the performance of the implementation. However, we need to only 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 [11]. We used the learning rate value of 0.1.

The implementation also makes it easier for users to specify number of principal components to be used for population structure correction as well as additional covariates using command line parameters and input files.

Arguments: $\mathbf{X}$, $\mathbf{y}$, α, 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 $\mathbf{B}_k = \text{diag}(\mu(i)(1-\mu(i)))$ using $\boldsymbol{\mu}_k$;
  Compute $\boldsymbol{Error}$ using $\mathbf{y}$;
  if $\boldsymbol{Error} < \epsilon$ **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 $\mathbf{y}$ ($y_i \in \{0, 1\}$) 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

Bug fixes

An error was found in the old implementation regarding the order of samples during adjustment of p-values using the confounding factors. This has been corrected in the new C++ implementation.

Implementation of Benjamini–Hochberg Procedure

A number of corrections exist for p-values in multiple hypothesis testing. Two such methods are: Bonferroni correction and Benjamini-Hochberg correction. The previous implementation performed hypothesis testing on each k-mer and performed Bonferroni correction using the total number of k-mers to determine k-mers associated with the phenotype in question. However, Bonferroni correction is known to be conservative i.e. it may fail to reject the null hypothesis even when it should be rejected.

Here, we implemented Benjamini-Hochberg correction which controls the false discovery rate (FDR). This can be used in studies underpowered for Bonferroni correction. The new implementations gives the provision for performing either correction according to user preference.

Incorporation of Jellyfish 2

The original implementation of HAWK used a modified version of Jellyfish [12]. Subsequently, Jellyfish2 has been released which provides better performance. The present implementation of HAWK allows k-mer counting using a modified version of Jellyfish2 available through our Github repository.

Re-implementation of post-processing

Once sequences corresponding to each loci associated with the phenotype are obtained, information such as average p-values of constituent k-mers as well as average number of times they are present in case and control samples could be looked up using scripts provided with the original implementation. However, these scripts used a combination of C++ codes and shell commands, and was found to be slow in some cases [4]. Here, we re-implement the script using hash tables in C++ to speed up the look-up.

Results

To assess the performance and accuracy of our implementation, we use the *E.coli* dataset on ampicillin resistance which was analyzed using the original implementation of HAWK [1].

All the experiments are performed on a machine with CPU Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz, 386GB memory with OS Ubuntu 18.04.3. There are two CPUs in the system with total 48 logical cores. The C++ implementation are compiled with g++-7.4.0 with no optimization flag. Execution time is measured using the "date" command.

Hyper parameters

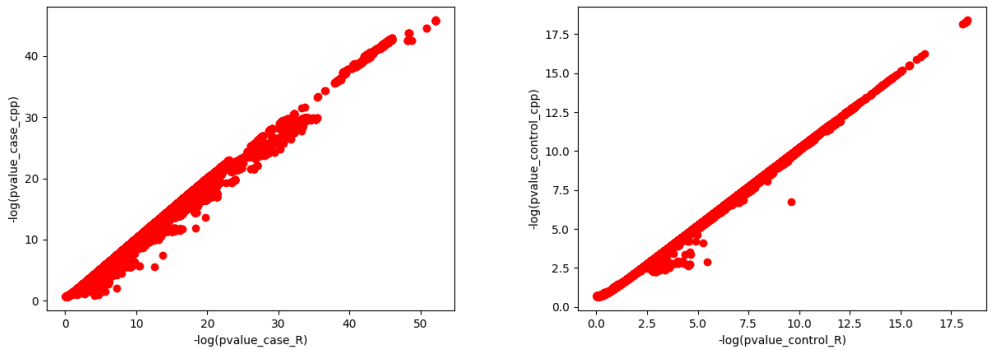
There are 3 hyper parameters in the code. They are described below along with the values used in Table 1.

Table 1. Hyper parameter values used in the implementation

Hyper Parameter Name	Description	Value
learning_rate	Learning rate of the logistic model fitted in the code	0.1
max_iter	Maximum number of iterations the logistic model uses	25
CHUNK_SIZE	Number of samples the model reads at a time while fitting the model	10000

Comparison of outputs

We have compared the output of our implementation and the output of the previous implementation by plotting logarithm of p-values obtained using C++ implementation against that of p-values obtained using the R implementation in Figure 1. An identical result should give a line with slope 1. Graph produced by our implementation are almost linear with small deviation in few points.



(a) **(b)**

Figure 1. Comparison of log of p-values computed using the previous implementation in R and new implementation using C++ for k-mers of potential association with cases and controls

The k-mers found significant using the old implementation (corrected version) were mapped to *Escherichia coli* strain DTU-1 genome [GenBank: CP026612.1] and *Escherichia coli* strain KBN10P04869 plasmid pKBN10P04869A sequence [GenBank: CP026474.1]. The positions in the reference genomes and the p-values are shown in Manhattan plots in Figures 2(a) and (b). We find associations near the *β-lactamase TEM-1 (blaTEM-1)* gene, the presence of which is known to confer ampicillin resistance, as in [1]. However, some of the associations outside of this gene detected previously in [1], that are likely to be spurious, is no longer observed after the correction of error.

The above analysis was also performed with the k-mers found significant using the new C++ implementation and the Manhattan plots are shown in Figures 2(c) and (d). We observe that associations are detected in same regions as those found using the R implementation.

Controlling FDR using the Benjamini-Hochberg procedure

Hawk uses Bonferroni correction to address multiple testing by default. However, we provide the option to control false discovery rate (FDR) using the Benjamini-Hochberg procedure. Figures 2(e) and (f) show Manhattan plots when k-mers are considered associated with ampicillin resistance after controlling FDR using the Benjamini-Hochberg procedure. We observe that many k-mers outside of the *β-lactamase TEM-1* gene are considered significant. We therefore recommend using Bonferroni correction and using the Benjamini-Hochberg procedure only if the study is under-powered for Bonferroni correction.

Comparison of running time

The execution of times of the old and new implementations are compared in Figure 3. Reported time is obtained by running both implementation using 32 threads. We find that the C++ implementation of the confounder correction phase is approximately three times faster than the precious implementation in R. However, the overall execution time is dominated by the k-mer counting step.

Comparison of Jellyfish versions

Table 2 shows comparison of execution times of Jellyfish and Jellyfish 2. We observe that, although Jellyfish 2 is faster overall, the performance improvement it provides is not substantial.

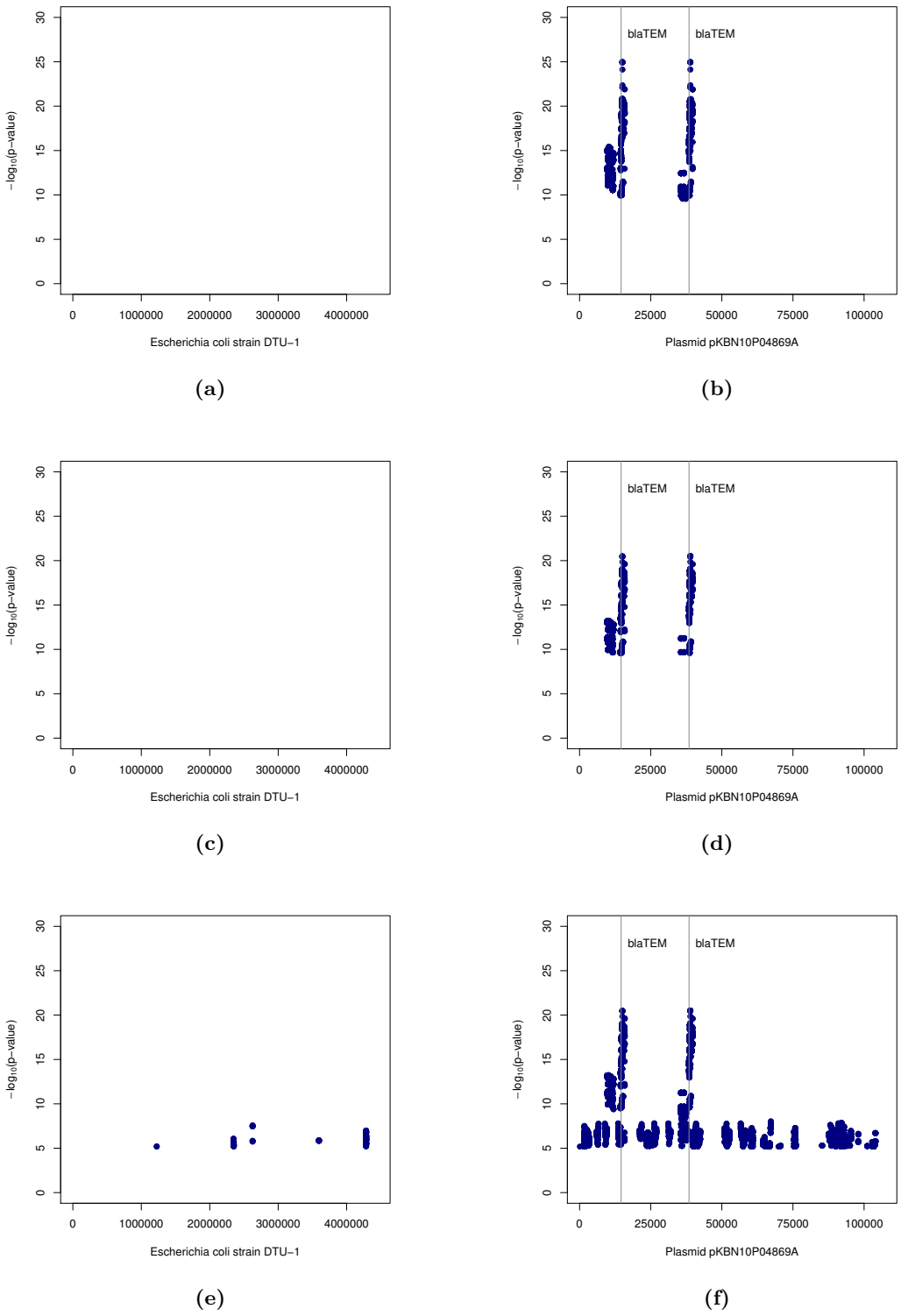
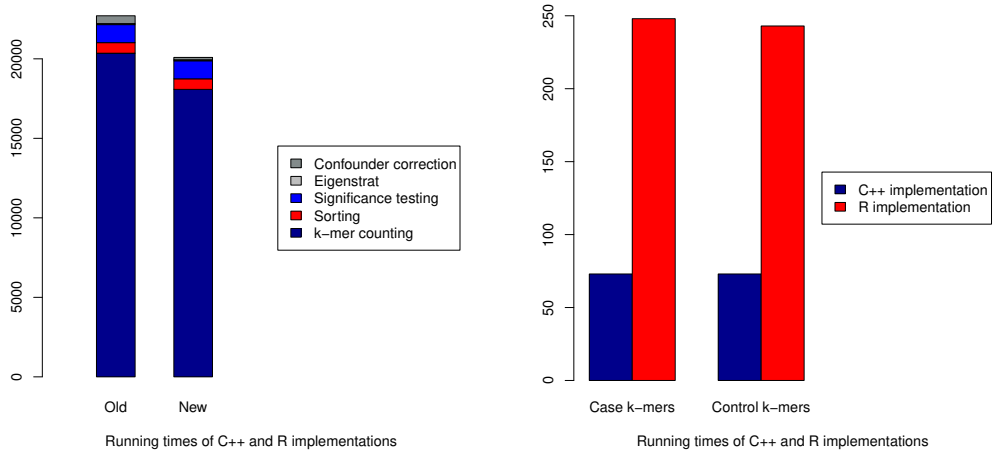


Figure 2. Manhattan plots showing negative logarithm of adjusted p-values of k-mers found significantly associated with ampicillin resistance against their start positions, computed using the R implementation with Bonferroni correction (a),(b); using C++ implementation with Bonferroni correction (c),(d); and using C++ implementation with Benjamini-Hochberg procedure (e),(f); in *Escherichia coli* strain DTU-1 genome (a), (c), (e), and plasmid pKBN10P04869A sequence (b), (d), (f). The vertical lines denote start positions of *β-lactamase TEM-1* gene, the presence of which is known to confer resistance to ampicillin.



(a) **(b)**

Figure 3. Comparison of execution times of old and new implementations (a) for the entire pipelines, and (b) correction of confounding factors using C++ and R implementations

Table 2. Comparison of running times of Jellyfish and Jellyfish 2

subcmd \ tool	Jellyfish (sec)	Jellyfish 2 (sec)
histo	20	674
count	19852	16580
dump	481	824
Total	20353	18078

Conclusions

We have re-implemented portions of HAWK, which is a tool for association mapping using k-mers. The re-implementation in C++ makes it faster and more convenient to use while retaining accuracy. We have also added support for the new version of k-mer counting tool Jellyfish and correction for multiple testing using the Benjamini-Hochberg procedure. The k-mer counting step remains the bottleneck in the pipeline which may be addressed by adding support for other k-mer counting tool.

Availability and User Support

Scripts, source code, and user manual for HAWK is available in `https://github.com/atifrahman/HAWK`. The system is available for and tested in Linux (Ubuntu14.04 and Ubuntu18.04).

Acknowledgments

We like to thank Guillaume Marcais for providing the patch to modify Jellyfish 2 [12] according to our need.

References

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

2. John A Lees, Minna Vehkala, Niko Välimäki, Simon R Harris, Claire Chewapreecha, Nicholas J Croucher, Pekka Marttinen, Mark R Davies, Andrew C Steer, Steven YC Tong, et al. Sequence element enrichment analysis to determine the genetic basis of bacterial phenotypes. *Nature communications*, 7:12797, 2016.

3. Sarah G Earle, Chieh-Hsi Wu, Jane Charlesworth, Nicole Stoesser, N Claire Gordon, Timothy M Walker, Chris CA Spencer, Zamin Iqbal, David A Clifton, Katie L Hopkins, et al. Identifying lineage effects when controlling for population structure improves power in bacterial association studies. *Nature microbiology*, 1:16041, 2016.

4. Magali Jaillard, Leandro Lima, Maud Tournoud, Pierre Mahé, Alex Van Belkum, Vincent Lacroix, and Laurent Jacob. A fast and agnostic method for bacterial genome-wide association studies: Bridging the gap between k-mers and genetic events. *PLoS genetics*, 14(11):e1007758, 2018.

5. Yoav Voichek and Detlef Weigel. Identifying genetic variants underlying phenotypic variation in plants without complete genomes. *Nature Genetics*, 2020.

6. 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`.

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

8. 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.

9. 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.

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

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

12. 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.