Minimizing Time when Applying Bootstrap to Contingency Tables Analysis of Genome-Wide Data

Francesco Sambo and Barbara Di Camillo

Department of Information Engineering, University of Padova, Italy
francesco.sambo@dei.unipd.it

Abstract. Bootstrap resampling is starting to be frequently applied to contingency tables analysis of Genome-Wide SNP data, to cope with the bias in genetic effect estimates, the large number of false positive associations and the instability of the lists of SNPs associated with a disease. The bootstrap procedure, however, increases the computational complexity by a factor B, where B is the number of bootstrap samples. In this paper, we study the problem of minimizing time when applying bootstrap to contingency tables analysis and propose two levels of optimization of the procedure. The first level of optimization is based on an alternative representation of bootstrap replicates, *bootstrap histograms*, which is exploited to avoid unnecessary computations for repeated subjects in each bootstrap replicate. The second level of optimization is based on an ad-hoc data structure, the *bootstrap tree*, exploited for reusing computations on sets of subjects which are in common across more than one bootstrap replicate. The problem of finding the best bootstrap tree given a set of bootstrap replicates is tackled with best improvement local search. Different constructive procedures and local search operators are proposed to solve it.
The two proposed levels of optimization are tested on a real Genome-Wide SNP dataset and both are proven to significantly decrease computation time.

Keywords: Bootstrap, Contingency tables analysis, Genome-Wide SNP Data, Local Search

Introduction

In the past few years, the genetic basis of disease susceptibility has started to be explored through the novel paradigm of Genome Wide Association Studies (GWASs). A GWAS searches for patterns of genetic variation between a population of affected individuals (cases) and a healthy control population, for complex diseases arising from the interaction of a genetic predisposition with environmental risk factors [11].

The most common form of genetic variation among individuals is Single Nucleotide Polymorphism (SNP), a point variation at a single DNA locus across

members of the same species. Diploid individuals, such as human, have two homologous copies of each chromosome and the genetic variation can occur at the same locus in either of the two chromosomes: human SNPs, thus, are ternary variables, encoding the three possible configurations of nucleotide pairs, or *genotypes*, at a certain locus (AA, BB and AB).

Current technologies allow the simultaneous measurement of $O(10^6)$ SNPs for each individual and the usual number of individuals involved in a GWAS is $O(10^3)$: the size of the resulting dataset has thus induced the vast majority of studies to search only for univariate association between each single SNP and the disease [15, 16] or to rely on univariate SNP association as a ranking and/or pre-filtering step [6, 12].

All the methodologies proposed in the literature to test for the association between a SNP and a disease condition require the computation, for each SNP, of a 2×3 contingency table, containing the number of case and control individuals for each of the three genotypes of the SNP. Test statistics can then be exploited to select SNPs significantly associated with the disease or to rank SNPs in decreasing order of genetic effect on the disease [1].

The large number of tests involved in a GWAS, together with the low sample size relative to the number of variables tested for association, can give rise to bias in genetic effect estimates, to a large number of false positive associations and to instability in the ranked list of SNPs [4]. To cope with these limitations, one of the strategies adopted in the literature consists in coupling bootstrap with contingency tables creation [4, 12–14].

Bootstrap [2] is a data-based simulation method for statistical inference: given a dataset X, consisting of n observations of p variables, and a statistic $s(X)$, the bootstrap method consists in (i) generating B *bootstrap replicates* of the original dataset $(X^1, \ldots, X^B)$, each one obtained by sampling with replacement n observations from X, (ii) computing the test statistic for each bootstrap replicate and (iii) exploiting the B results for estimating some properties of the statistic, such as standard error or confidence intervals.

In the context of GWAS data analysis, SNPs are the variables and subjects are the observations; bootstrap is thus used to obtain B replicates of the dataset, each with the same set of SNPs and with subjects sampled with replacement from the original set. The statistic of association is then computed for each SNP in each sample replicate. In [4], bootstrap is applied to contingency tables analysis for computing point estimates and confidence intervals of the genetic effect of each SNP, exploiting the relative rank of each SNP in each bootstrap replicate. The same approach is further exploited in [14] for estimating the minimum sample size needed in replication studies. With the aim of estimating the total number of susceptibility SNPs of a complex disease from GWAS data, bootstrap is applied in [13] to contingency table analysis for computing confidence intervals of the estimate. Finally, in [12] an ensemble of Naïve Bayes classifiers is trained on as many bootstrap replicates of a GWAS dataset, and SNP ranking through test statistics is exploited for learning classification probabilities and for selecting the attributes of each Naïve Bayes classifier.

The bootstrap method has the appealing feature that it can be used "on top" of the statistic to be computed, exploiting the statistic computation as a black box and simply iterating it through the various bootstrap replicates. Suggested values for B are 50-100 when estimating bias or standard error and 1000 when estimating confidence intervals [2], thus the main drawback is the $O(B)$ increase in computational complexity due to the replication of the statistic computation.

Even though the time needed for acquiring and pre-processing a GWAS dataset is much longer than the time needed for computing a statistic of association between all SNPs and the disease, an acquired dataset is seldom processed just once, both because multiple statistics are usually computed on the same dataset and because separate, smaller datasets are often joined together and re-processed in larger meta-analyses. Any attempt in reducing the computation time of data processing is thus definitely worth the effort.

In this paper, we explore the problem of minimizing computation time when applying bootstrap to contingency table analysis of Genome-Wide SNP data. The main contributions of the paper are two levels of optimization of the computational procedure: the first level of optimization derives from an alternative representation of bootstrap replicates as *bootstrap histograms*, which is exploited to avoid repeating computations for repeated subjects in each bootstrap replicate. The second level of optimization is based on an ad-hoc data structure, the *bootstrap tree*, exploited for reusing computations on sets of subjects which are in common across more than one bootstrap replicate. The problem of finding the best bootstrap tree, given a set of bootstrap replicates, is tackled with a best improvement local search approach [7] and different constructive procedures and local search operators are proposed to solve it.

We tested our optimized computational procedure on the WTCCC case-control study on Type 1 Diabetes [15] and indeed observed a significant decrease in computation time for both levels of optimization, with respect to a standard bootstrap approach.

The remainder of the paper is organized as follows: Section 1 describes in details the problem of applying bootstrap to contingency table analysis of GWAS data and presents the two levels of optimization, Section 2 describes the experimental dataset and reports performance results and Section 3 draws conclusions and presents some possible future directions.

1 Methods

Given a GWAS dataset X, consisting of p SNPs measured for $n = n_{cases} + n_{controls}$ subjects, and a binary vector of class labels Y of size n, computing a contingency table like the one in Table 1 for each SNP involves scanning all subjects and counting the occurencies of the three possible variants of the SNP. Iterating the process on the whole SNP set has thus computational complexity $O(pn)$.

The frequency counts in each contingency table are exploited to test for an association between the corresponding SNP and the disease condition, with a

	AA	AB	BB
cases	a	b	c
controls	d	e	f

Table 1: Example of a 2×3 contingency table for a particular SNP.

certain statistic s. SNPs can then be ranked according to the computed statistics and the topmost SNPs, or the SNPs whose statistic pass a certain threshold, are identified as associated with the disease.

The reliability of the statistic and the robustness of the list of associated SNPs can be assessed with bootstrap.

If we define $I = \{i_1, \ldots, i_n\}$ the original patient set (Figure 1a), the bootstrap procedure generates B bootstrap replicates $\{I^1, \ldots, I^B\}$, each of size n and sampled with replecement from I (Figure 1b). For each SNP, B contingency tables and B corresponding statistics are then computed. The $B \times p$ resulting statistics can be exploited either for calculating bias, squared error or confidence intervals for each SNP [4, 14], or for computing B ranked lists of SNPs, which can be merged to obtain a single, more robust list [10].

The pseudocode of the classic bootstrap procedure is given in what follows. As it is clear from the pseudocode, the computational complexity of the algorithm is $\mathrm{O}(Bpn)$.

ClassicBootstrap(X, Y, B)

```
1   I = {i_1, ..., i_n}, original patient set
2   Generate the sets {I^1, ..., I^B}, each sampled with replacement from I
    // results of the statistics for each SNP and each replicate
3   S = p × B matrix of zeros
4   for b in {1, ..., B}
5       for k in {1, ..., p}
            // contingency table
6           CT = 2x3 matrix of zeros
7           for j in {1, ..., n}
8               i_j = I^b[j]
9               CT[ Y[i_j], X[k, i_j] ]+=1
            // statistic of the association between SNP k and the disease
            // for the bootstrap replicate b
10          S[k, b] = s(CT)
11  Use S to estimate properties of the statistic or to obtain a robust SNP ranking
```

Bootstrap replicates I^b belong to the class of *multisets*, *i.e.* sets that allow the repetition of elements. In the next section, we introduce a convenient representation for multisets, which will lead to a first level of optimization of the bootstrap procedure.

I: i_1 i_2 i_3 i_4 i_5 i_6 i_7 i_8 i_9 i_{10}

(a)

I^b: i_1 i_1 i_3 i_5 i_5 i_5 i_7 i_8 i_9 i_9

(b)

$\overline{I^b}$: 2 0 1 0 3 0 1 1 2 0

(c)

Fig. 1: (a) example of a patient set I with 10 patients. (b) example of a bootstrap replicate of I. (c) boostrap histogram of the replicate I^b.

1.1 Level I optimization: bootstrap histograms

Given a multiset I^b with m elements, drawn from an underlying set $I = \{i_1, \ldots, i_n\}$ of n distinct elements, a *histogram* representation of the multiset is a vector $\overline{I^b}$ of length n containing, for each element $i_j \in I$, the number of times it appears in the multiset I^b (Figure 1c). We define *bootstrap histogram* the histogram representation of a bootstrap replicate.

Bootstrap histograms can be conveniently exploited to avoid unnecessary operations: when computing contingency tables for each bootstrap replicate $\overline{I^b}$, in fact, one needs only to process the elements j such that $\overline{I^b}[j] > 0$. Furthermore, given that each nonzero element of the boostrap histogram counts multiple copies of the same subject, one needs only to evaluate once the SNPs of the j-th subject and then add $\overline{I^b}[j]$ to the corresponding elements of the contingency tables. The pseudocode of the histogram-based bootstrap procedure is given in what follows.

HistogramBootstrap(X, Y, B)

```
1  I = {i_1, ..., i_n}, original patient set
2  Generate {Ī^1, ..., Ī^B}, sampled with replacement from I
3  S = p × B matrix of zeros
4  for b in {1, ..., B}
       // tmp storage of indices, values and labels of nonzero elements of Ī^b
5      tmpInd = ∅, tmpVal = ∅, tmpY = ∅
6      for j in {1, ..., n}
7          if Ī^b[j] > 0
8              tmpInd = tmpInd ∪ j
9              tmpVal = tmpVal ∪ Ī^b[j]
10             tmpY = tmpY ∪ Y[ Ī^b[j] ]
11     for k in {1, ..., p}
12         CT = 2x3 matrix of zeros
13         for j in {1, ..., length(tmpInd)}
14             CT[ tmpY[j], X[k, tmpInd[j]] ] += tmpVal[j]
15         S[k, b] = s(CT)
16 Use S to estimate properties of the statistic or to obtain a robust SNP ranking
```

The preprocessing routine at lines 5−10 extracts, for each bootstrap histogram $\overline{I^b}$, the indices, values and corresponding disease condition of the nonzero elements of $\overline{I^b}$; its computational complexity, $\mathrm{O}(Bn)$, can be considered negligible if $n \ll p$, as in our case.

Thanks to the preprocessing routine, the summation at line 14 is now executed only $B \cdot p \cdot \mathrm{length}(tmpInd)$ times, resulting in an expected relative gain of computation time equal to the average proportion of zero elements in a bootstrap histogram: for n sufficiently large, the relative gain approaches $e^{-1} \simeq 0.368$ [2].

A further level of optimization, obtained by exploiting the presence of common elements across multiple bootstrap histograms, is presented in the next section.

1.2 Level II optimization: bootstrap tree

For our second level of optimization, the aim is to group together bootstrap histograms sharing common elements, so to be able to reuse the results of contingency table computations for multiple bootstrap histograms. To this purpose, we define *intersection* of two bootstrap histograms $\overline{I^x}$ and $\overline{I^y}$ the boostrap histogram $\overline{I^z}$ such that:

$$\overline{I^z}[j] = \overline{I^x}[j] \cap \overline{I^y}[j] = \begin{cases} \overline{I^x}[j] & \text{if } \overline{I^x}[j] = \overline{I^y}[j], \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } j = 1 \ldots n.$$

Furthermore, we define *size* of a bootstrap histogram the number of its nonzero elements and *similarity* between two bootstrap histograms the size of their intersection.

Given a set of bootstrap histograms, we define *bootstrap tree* a data structure with the following features:

1. the bootstrap tree is a balanced binary tree,
2. each leaf of the tree, *i.e.* each node at level 0, contains one of the original bootstrap histograms,
3. each internal node of the tree, at level $l > 0$, contains the intersection of its two *children*, *i.e.* the pair of nodes connected to it at level $l - 1$.

An example of bootstrap tree is given in Figure 2.

For each node, we define *unique elements* the nonzero elements of its bootstrap histogram which are zero in the histogram of its parent node. Unique elements are marked in bold in the example tree of Figure 2.

A bootstrap tree can be effectively exploited to obtain a further decrease in computation time. Each bootstrap replicate can be processed by visiting the tree in a depth-first, left-first traversal, backtracking once leaves are reached. The intuition is that computations can be carried out while descending the tree, exploiting the bootstrap histogram of each internal node for computing partial results, which can then be reused for all the nodes in the corresponding subtree. The pseudocode of such an algorithm is given in what follows.

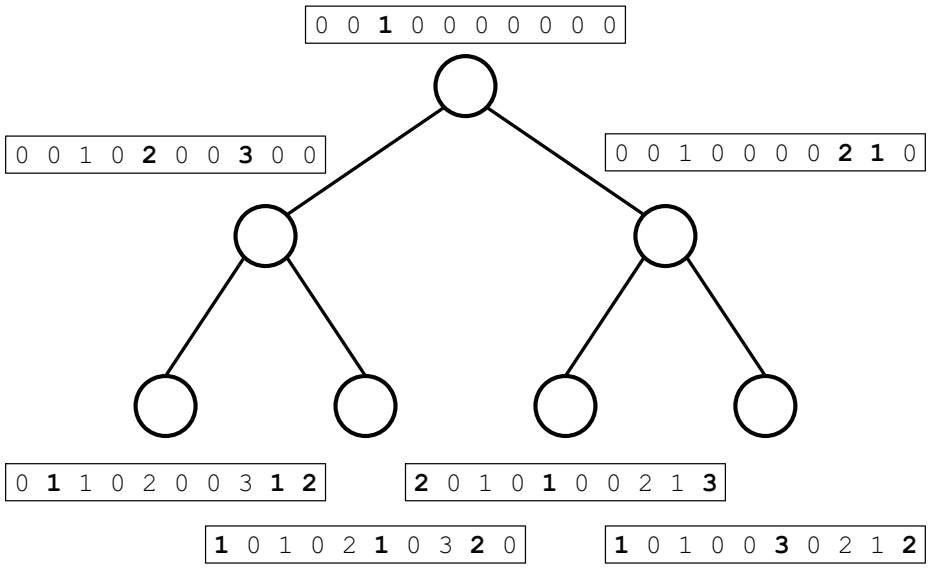

Fig. 2: Example of a bootstrap tree, for $B = 4$ bootstrap replicates. The leaves contain the bootstrap histograms of the 4 bootstrap replicates and each internal node contains the intersection of its children. Unique elements of each node, *i.e.* nonzero elements of its bootstrap histogram which are zero in the histogram of its parent node, are marked in bold.

TreeBootstrap$(X, Y, tree)$

1 CTs = set of p contingency tables, all elements initialized to zero
2 $S = p \times B$ matrix of zeros
3 RecTreeBootstrap$(X, Y, tree.root, tree.height, CTs)$
4 Use S to estimate properties of the statistic or to obtain a robust SNP ranking

RecTreeBootstrap$(X, Y, node, level, CTs)$

1 Increment CTs according to the unique elements of $node$
2 **if** $level > 0$
  // Proceed to the children
3 $newCTs$ = copy of CTs
4 RecTreeBootstrap$(X, Y, node.leftChild, level - 1, newCTs)$
5 RecTreeBootstrap$(X, Y, node.rightChild, level - 1, CTs)$
6 **else**
  // A leaf has been reached, compute the statistic
7 $b = node.b$ // Index of the bootstrap replicate
8 **for** k **in** $\{1, \ldots, p\}$
9 $S[k, b] = s(CTs[k])$

The TreeBoostrap algorithm creates a set CTs of p zero-valued contingency tables, allocates space for the results of the statistic and launches the

recursive RecTreeBoostrap algorithm from the root of the tree, passing the set CTs. Each node visited in the left-first descent receives a set of contingency tables, increments them according to its unique elements and passes a copy of them to its left child (RecTreeBootstrap, lines 1-4). When a leaf is reached, its corresponding contingency tables are complete and the statistic of association can be computed for all SNPs (lines 7-9). When backtracking from a left child to a right child, the set of contingency tables at the parent node is directly passed to the right child rather then copied (line 5), since it does not need to be stored anymore. The algorithm terminates when the last right child, *i.e.* the rightmost leaf, is visited.

The gain in computation time of the TreeBootstrap algorithm, relative to the HistogramBootstrap algorithm, can be computed as the sum of the sizes of the internal nodes over the sum of the sizes of the leaves. We prove this intuitively: each nonzero element in the bootstrap histograms of nodes at level 1 indicates an element for which computations can be spared, because it is in common between two histograms; nonzero elements at level 2 stands for further spared elements when computing nodes at level 1, and so on up to the root. The gain of the tree in Figure 2 is thus $7/24 \simeq 0.29$[1].

The gain in computation time comes at the cost of an increased memory occupation: the TreeBootstrap algorithm needs to keep in memory a number of contingency tables, for each SNP, equal to the height of the bootstrap tree plus one. Memory occupation is often critical when dealing with GWAS data. We thus impose the constraint on the bootstrap tree of being a balanced binary tree: among all possible binary trees of B nodes, balanced binary trees have the lowest height, $log(B)$.

Having defined the bootstrap tree, the TreeBootstrap algorithm and the concept of gain of a tree, we can now formulate the optimization problem of searching for the bootstrap tree with the maximum gain, given a GWAS dataset and a set of B bootstrap histograms. We chose to tackle the problem with a best improvement local search approach [7]: starting from an initial bootstrap tree, we generate a neighbourhood of trees by applying a local search operator, choose the tree with the highest gain among the neighbourhood and iterate the process until a local maximum is reached.

We explore the use of different constructive procedures for building the initial tree and of different local search operators for generating the neighbourhood. Constructive procedures and local search operators are described in the next sections.

1.3 Constructive procedures for bootstrap trees

We begin this section with a remark: since our final objective is to minimize computation time, which includes both the time for searching for the optimal

[1] To be precise, one should also consider the time spent for copying contingency tables. Copying p contingency tables, one for each SNP, has complexity $O(p)$ and the whole set of contingency tables is copied $B-1$ times: the computational complexity does not depend on n and can thus be considered negligible.

bootstrap tree and the time for the TreeBootstrap algorithm, simple but fast constructive heuristics can stand a chance against more complex but slower heuristics and should thus be considered.

The first constructive procedure we propose is a greedy agglomerative constructive heuristic, inspired by the literature on hierarchical clustering [5]: the GreedyAgglomerative heuristic builds the bootstrap tree starting from the leaves, *i.e.* the original boostrap histograms, by computing the mutual similarity between all pairs of histograms. Similarity between two bootstrap histograms, as defined in Section 1.2, is the size of the intersection between the two histograms. The two histograms with the highest similarity are joined as children of the first node at level 1, whose histogram is computed as the intersection of the two histograms. The heuristic keeps joining the pair of remaining leaves with the highest similarity, until no more leaves remain. The procedure is then iterated up to the root, by computing the mutual similarity between all pairs of nodes at level l and by iteratively joining the nodes with the highest similarity as children of the nodes at level $l+1$. The computational complexity of the GreedyAgglomerative heuristic is $\mathrm{O}(B^2 n)$.

The other constructive procedure we consider is RandomBuild, which builds the bootstrap tree by joining pairs of nodes at random up to the root. Despite the lower expected gain with respect to the GreedyAgglomerative heuristic, we choose to try also the RandomBuild procedure because of its lower computational complexity, $\mathrm{O}(Bn)$.

1.4 Local search operators for bootstrap trees

The first local search operator we define, TreeOpt, can be applied to all nodes at level $l \geq 2$ and operates by testing the two possible swaps between grandchildren of a node G (which stands for Grandparent), *i.e.* between the four nodes whose parents are the two children of G (Figure 3). The total number of nodes at level $l \geq 2$ is $B/2-1$, thus the size of the neighbourhood of the TreeOpt operator is $B-2$. The cost of a swap is the cost of updating the boostrap histograms for G's children, G itself and all the nodes in the path from G up to the root. The total cost of evaluating a TreeOpt neighbourhood is thus $\mathrm{O}(nB \log B)$.

The second operator we define, 2Opt, is inspired by the homonymous operator for the TSP problem [7]: the 2Opt operator tests all possible swaps between two leaves of the tree, excluding the swaps between the two children of the same node. The size of the neighbourhood of the 2Opt operator is thus $B^2/2-B$. The cost of a swap is the cost of updating the boostrap histograms of the nodes on the paths from the two swapped leaves up to the root: the total cost of evaluating a 2Opt neighbourhood is thus $\mathrm{O}(nB^2 \log B)$.

2 Experimental results

In this section, we present experimental results on the computational performance of the algorithms ClassicBootstrap, HistogramBootstrap and

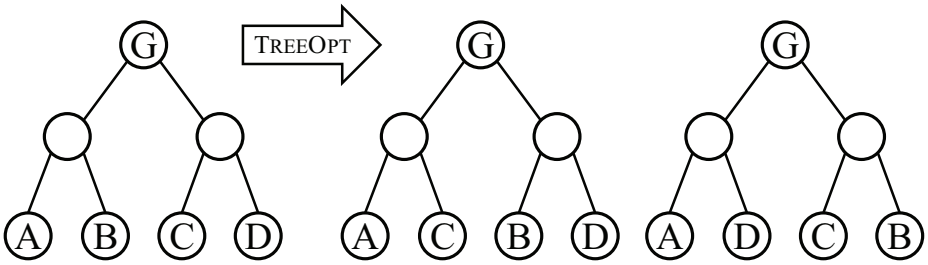

Fig. 3: Example of use of the TreeOpt operator: applied to the node G of the leftmost tree, the operator generates the two possible swaps of the nodes A, B, C and D, whose parents are the two children of G.

TreeBootstrap. Different combinations of the constructive procedures and of the local search operators are tested for the bootstrap tree of the TreeBootstrap algorithm.

As benchmark to assess the performance of the different algorithms, we choose the WTCCC case-control study on Type 1 Diabetes [15]: the dataset consists of 458376 SNPs, measured for 1963 T1D cases and 2938 healthy controls (after the application of all Quality Control filters reported in [15]). The numbers of SNPs and subjects involved are in line with the ones usually encountered in a GWAS [11], thus making the dataset a meaningful benchmark for our algorithms.

We measure the computation time spent by the following procedure: generate B bootstrap replicates, compute a contingency table and a univariate statistic of association (described in [12]) for each SNP in each replicate and rank SNPs according to the computed statistic. To remove a possible source of noise, we exclude from the measurements the time needed for loading the dataset in RAM.

The number of bootstrap replicates, B, is varied among all powers of 2 in the range $\{2^0 \dots 2^{10}\}$: for each B, we generate 20 sets of B bootstrap replicates and repeat the whole procedure on each set.

All algorithms are written in C++ and all computations are carried out on a single 3.00 GHz Intel Xeon Processor E5450.

We first assess the effectiveness of the two levels of optimization by comparing the computation time of ClassicBootstrap, HistogramBootstrap and TreeBootstrap, the latter tested with either the GreedyAgglomerative or the TreeOpt constructive procedure and without local search.

Results are shown in Figure 4, top panel: the figure reports, for each value of B, the median over the 20 runs of the average time needed to process one bootstrap replicate, computed as the total time over B. For each point, whiskers extend from the first to the third quartile. We preferred to plot the median rather than the mean because of the presence of a small number of random outliers, which were however included in all the tests for significance. As it is clear from the figure, both levels of optimization result in a significant decrease in computation time (p-values of ClassicBootstrap *vs* HistogramBootstrap

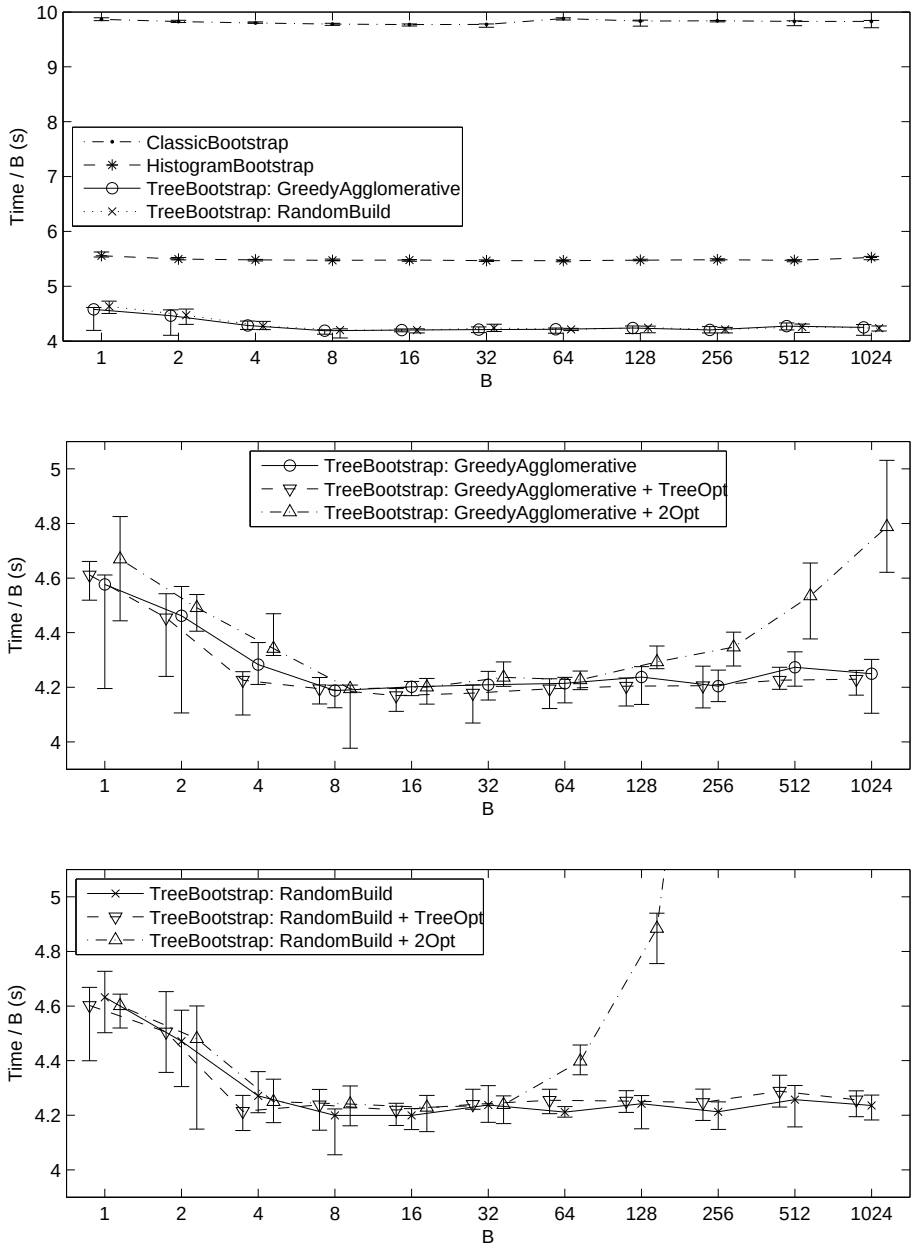

Fig. 4: Medians across 20 runs of the time for processing one bootstrap replicate (total time / B) versus the number of replicates B. Whiskers extend from first to third quartile. Top panel: NaiveBootsrap, HistogramBootstrap and TreeBootstrap with the two constructive procedures and without local search. Middle panel: TreeBootstrap with the GreedyAgglomerative constructive procedure, without local search and with the two local search operators. Bottom panel: TreeBootstrap with the RandomBuild constructive procedure, without local search and with the two local search operators.

and HistogramBootstrap *vs* both versions of TreeBootstrap $< 8.9 \times 10^{-5}$ for each B, Wilcoxon signed-rank test)[2].

Oberving a significant difference between HistogramBootstrap and TreeBootstrap for $B = 1$ is somehow unexpected, since the number of operations executed by the two algorithms is practically the same. The only notable difference is that HistogramBootstrap allocates just one contingency table and reuses it, while TreeBootstrap has to allocate the whole set of p contingency tables: rather than resulting in an increased overhead, the second strategy seems to be more fit to be optimized by the compiler and thus results in a significant gain.

Further, significant increases in the time gain of the TreeBootstrap algorithm can be observed up to $B = 8$ (p-values of the differences between consecutive samples < 0.02, Wilcoxon rank-sum test). This means that the algorithm is effectively able to avoid unnecessary computations, by reusing common elements of the bootstrap histograms. For $B \geq 16$, no further gain can be observed: the identification of groups of 16 or more bootstrap histograms of length 4901 (the total number of subjects in the dataset) sharing many common elements seems to have a computational cost which is too high to result in an effective improvement of the overall procedure.

Interesting is also the fact that no significant difference can be observed between the two constructive procedures for TreeBootstrap (p-value > 0.21 for each B). The higher tree gain otained by the GreedyAgglomerative heuristic, thus, seems to be compensated by its higher computational complexity, with respect to the RandomBuild constructive procedure.

We then tested the effect on the TreeBootstrap algorithm of the two local search operators, when combined with either the GreedyAgglomerative and the RandomBuild constructive procedures (Figure 4, middle and bottom panel). As it is clear from the figures, local search with the 2Opt operator has a negative effect on performance, significant for $B \geq 256$ when the initial tree is built with GreedyAgglomerative and for $B \geq 64$ when RandomBuild is used (p-values $< 6 \times 10^{-3}$, Wilcoxon signed-rank test). This probably means that the rate at which the 2Opt operator increases the gain of the bootstrap tree is too slow to be able to improve the overall performance of the TreeBootstrap algorithm.

On the other hand, the TreeOpt local search operator has different effects when coupled with the GreedyAgglomerative and the RandomBuild constructive procedures: in the first case, the performance with local search tends to be consistently better than without local search for all values of B; in the second case, the performance is consistently worse for each $B \geq 8$. Differences, however, are not statistically significant, with the exception of one case.

[2] For all tests throughout the paper, we consider significant a p-value < 0.05.

3 Conclusions and future directions

In this paper, we studied the problem of minimizing computation when applying bootstrap to contingency table analysis of Genome-Wide SNP data. We proposed two levels of optimization of the procedure, which both result in a significant improvement in computation time: altogether, the optimization strategies presented in this paper allow us to reduce computation time by a factor of approximately 2.5, a result which can be considered definitely valuable in the context of GWAS data analysis.

The first level of optimization, implemented in the HistogramBootstrap algorithm, is based on an alternative representation of bootstrap replicates as bootstrap histograms. The bootstrap histogram representation is not new in the literature: for example, in [3] the same representation is proven more effective than the standard representation for estimating the bias of order invariant statistics. As far as we know, however, the idea of exploiting the bootstrap histogram representation for reducing computation time has never been proposed in the literature.

The second level of optimization, implemented in the TreeBootstrap algorithm, is based on an ad-hoc data structure, the *bootstrap tree*, which is exploited for reusing partial results on sets of subjects shared by multiple replicates.

Once defined the bootstrap tree and the algorithm for exploiting it, we formulated the optimization problem of finding the tree leading to the highest gain in computation time and tackled the problem with a best improvement local search approach. Two constructive procedures and two local search operators were specifically designed for the problem and several combinations of them were tested. Experimental results show that simpler but faster approaches to tree construction and refinement are competitive with (and, in some cases, significantly more effective than) more powerful, yet slower approaches. As far as we know, the idea of exploiting common elements of the boostrap samples for reducing computation time of boostrap has never been proposed in the literature.

The algorithms designed for the second level of optimization require the number of bootstrap replicates, B, to be a power of two. We do not see this requirement as a big limitation: guidelines for the choice of the number of bootstrap replicates are present in the literature [2], but they usually give indications on the order of magnitude of B rather than on its exact value, the choice of which is left to the experimenter.

Concerning future directions, we intend to further explore the design of other constructive heuristics and local search operators, together with other stochastic local search techniques, to search for the optimal bootstrap tree.

The extent to which bootstrap performance can be improved with our two levels of optimization has, however, a theoretical limit. In each bootstrap replicate, all observations are sampled with equal probability $1/n$: the expected number of elements in common among B bootstrap histrograms is thus limited. Several authors, however, have proposed to exploit importance sampling for reducing the variance of certain bootstrap estimates [8, 9, 17]. The idea beyond importance sampling is to sample observations with nonuniform weights: such

an approach dramatically increases the number of elements in common among subsets of bootstrap histograms and can thus further benefit from our optimization strategies. One of our future directions is thus to study how to adapt our two levels of optimization to importance sampling in bootstrap estimates.

Finally, the enhancement of bootstrap with the two levels of optimization can be extended to other application domains, as long as the function to be iterated on each bootstrap replicate has the two following features: i) the sets of observations to be processed can be split in m distinct subsets and the function can be independently applied to each subset, ii) the computational cost of processing each subset is considerably higher than the cost of assembling the m results. One of our future directions is thus to study the application of our optimization strategies to different problem domains, characterized by the two aforementioned features.

Acknowledgements

This study makes use of data generated by the Wellcome Trust Case-Control Consortium. A full list of the investigators who contributed to the generation of the data is available from www.wtccc.org.uk. Funding for the project was provided by the Wellcome Trust under award 076113 and 085475.

Francesco Sambo would like to thank Prof. Silvana Badaloni for her precious advices and support.

References

1. Balding, D.J.: A tutorial on statistical methods for population association studies. Nature Reviews Genetics 7(10), 781–791 (2006)
2. Efron, B., Tibshirani, R.J.: An Introduction to the Bootstrap. Chapman & Hall, New York (1993)
3. Efron, B.: More Efficient Bootstrap Computations. Journal of the American Statistical Association 85(409), 79–89 (1990)
4. Faye, L., Sun, L., Dimitromanolakis, A., Bull, S.: A flexible genome-wide bootstrap method that accounts for ranking- and threshold-selection bias in GWAS interpretation and replication study design. Statistics in Medicine 30(15), 1898–1912 (2011)
5. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning, Second Edition: Data Mining, Inference, and Prediction. Springer Series in Statistics, Springer (Feb 2009)
6. He, Q., Lin, D.Y.: A variable selection method for genome-wide association studies. Bioinformatics 27, 1–8 (January 2011)
7. Hoos, H.H., Stützle, T.: Stochastic Local Search : Foundations & Applications (The Morgan Kaufmann Series in Artificial Intelligence). Morgan Kaufmann (September 2004)
8. Hu, J., Su, Z.: Short communication: Bootstrap quantile estimation via importance resampling. Computational Statistics and Data Analysis 52, 5136–5142 (August 2008)

9. Johns, M.: Importance sampling for bootstrap confidence intervals. Journal of the American Statistical Association 83, 709–714 (1988)
10. Jurman, G., Merler, S., Barla, A., Paoli, S., Galea, A., Furlanello, C.: Algebraic stability indicators for ranked lists in molecular profiling. Bioinformatics 24(2), 258–264 (2008)
11. Ku, C.S., Loy, E.Y., Pawitan, Y., Chia, K.S.: The pursuit of genome-wide association studies: where are we now? Journal of Human Genetics 55(4), 195–206 (Mar 2010)
12. Sambo, F., Trifoglio, E., Di Camillo, B., Toffolo, G., Cobelli, C.: Bag of Naïve Bayes: biomarker selection and classification from Genome-Wide SNP data. In: Clinical Bioinformatics. 11th Workshop on Network Tools and Applications in Biology NETTAB2011. Pavia, Italy (October 2011)
13. So, H.C., Yip, B.H.K., Sham, P.C.: Estimating the total number of susceptibility variants underlying complex diseases from genome-wide association studies. PLoS ONE 5(11), e13898 (11 2010)
14. Sun, L., Dimitromanolakis, A., Faye, L., Paterson, A., Waggott, D., Bull, S.: Br-squared: A practical solution to the winner's curse in genome-wide scans. Human Genetics 129(5), 545–552 (2011)
15. The Wellcome Trust Case Control Consortium: Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. Nature 447(7145), 661–678 (Jun 2007)
16. Zeggini, E., et al.: Meta-analysis of genome-wide association data and large-scale replication identifies additional susceptibility loci for type 2 diabetes. Nature Genetics 40(5), 638–645 (March 2008)
17. Zhou, H., Lange, K.: A fast procedure for calculating importance weights in bootstrap sampling. Computational Statistics and Data Analysis 55, 26–33 (January 2011)