Parallel Bootstrap Consensus Clustering

G Gentili Frias[1], M Saenz[1], C Fresno[1,2], EA Fernandez[1,2]
[1]Facultad de Ingeniería, Universidad Católica de Córdoba, Córdoba, Argentina. [2]UA. AREA CS. AGR. ING. BIO. Y S, CONICET, Universidad Católica de Córdoba, Córdoba, Argentina.

INTRODUCTION

Consensus clustering is a well-known approach for class discovery which has been extensively used in gene expression pattern discovery. However, since it's implemented as a serial procedure, its application to current high throughput databases makes it impractical for Big Omics Data approaches in terms of required time. Another drawback of current implementation is the user defined selection of variables/subject proportions that could impact in the discovered classes. The aim of this work is to improve ConsensusClusterPlus R library implementation in order to reduce execution time, as well as the proposal of a bootstrap sampling approach that eliminates user defined parameters.

The method was evaluated over two different gene expression datasets (17195 x 28 and 21770 x 105) running from 2 to 10 clusters, over 800 repetitions. They were evaluated using 1 to 23 cores. The bootstrap sampling was also compared against a 70% sampling approach for both samples and variables, as well as against the original implementation.

It is shown that just by doing a re-design of the algorithm there was an almost 20% speed improvement, increasing linearly up to 10 cores with a slope of 1.369e-10 per added core. It then tends to reach a plateau over 15 cores. Although the parallel implementation demanded an extra data postprocessing, it was not significant in comparison with the benefits obtained by the main changes. The Bootstrap implementation showed similar performance to the original implementation when comparing results, whereas speed up curves showed for both sampling alternatives a speed improvement of up to 9-fold, using up to 15 cores.

GOALS

- Reduce computational time, avoiding the *for* structure, which is correct, but expensive in resource usage.
- Implement Bootstrap sampling: an alternative sampling option to avoid used defined setting of sampling percentages.

METHODS

Work was focused on three main areas, and done in three steps:

Outer loop (repetitions) | **User defined sample/variables sampling** | **Inner loop (clusters)**

1) Re-engineering the loops

Being R an interpreted language, the *for* loops tend to be slower than using native iterators such as the *lapply* family. For this reason, the algorithm was re-engineered in order to re-implement the *for* lops

Inputs: ***M***: Expression matrix; ***maxK***: Maximum number of clusters to look for; ***rC***: number of clustering repetitions; ***pItem, pFeature***: Proportion of samples/variables used for clustering.

Original algorithm:

```
for (i in 1:rC){
    (....)
    sampled_M = sampleCols(...,pItem, pFeature)
    (...)
##use samples for each k/use same sample for all k
    for (k in 2:maxK){
      (...)
      Cluster[[k-1]] <- kmeans(sampled_M ,
                  k, ...)$cluster
     ml[[k]] <- connectivityMatrix(……)
    }## end for k
}## end for i

In this algorithm the conectivityMatrix is inside loops
```

Re-engineered algorithm:

```
sample<-lapply(1:rC, function(repet){
    sampled_M = sampleCols( ..,pItem,
                            pFeature,…)
    listaK<-lapply(2:maxK, function(k){
          (...)
          Cluster <- kmeans (sampled_M,
                             k,...)$cluster
                }# end function(k)
              )#end lapply
    return(list(K=listaK, sample[[3]] ...)
            } #end function(repet)
    ) #end lapply
```

In this algorithm the conectivityMatrix is outside loops

```
for(i in 1:rC){
  mCount <- connectivityMatrix(……)
  for(j in 2:maxK){
    ml[[j]] <- connectivityMatrix(……)
  }
}
```

2) Paralellizing the loops

In R, an alternative for parallelization only requires the loading of the "BiocParallel" library from www.bioconductor.org and the replacement of *lapply* by *bplapply* with some extra parameters relating to the number of used cores and OS.

The outer *lapply* loop was then modified as follows:

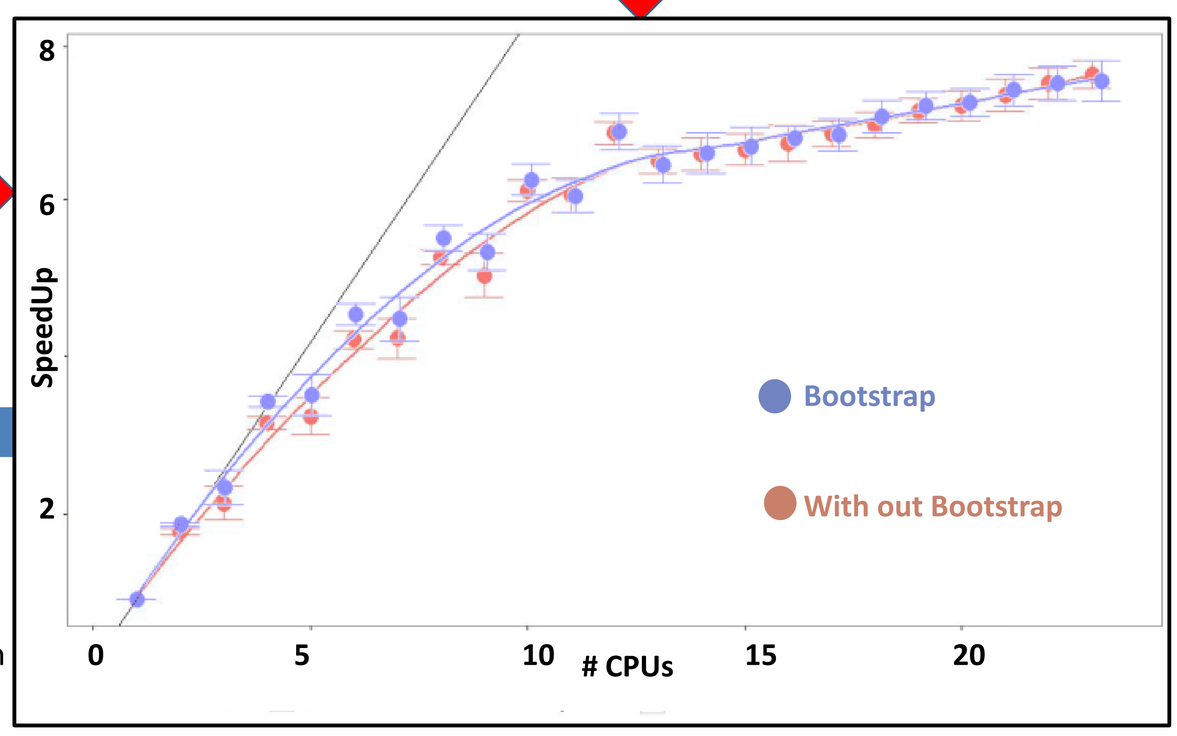
```
sample<-lapply(1:rC, function(repet){
      (...)
) # end lapply
```

```
sample<-bplapply(1:rC, function(repet){
      (...)
) # end lapply
```

3) Boostrap re-sampling

Diminishing the number of user defined parameters provides consistent results across users. The original CCP algorithm requires the user to set the percentage of variables and/or samples to be used inside the inner loop (the cluster loop).

Here, we propose the use of a bootstrap sampling for both variables and subjects selection. The bootstrap algorithm randomly choses a sample with replacement, keeping the same dimension as the original, but with some repeated subjects/variables.



CONCLUSION

The proposed parallelized implementation yielded an efficient alternative to reduce time consumption in class discovery problems, enhancing the original ConsensusClusterPlus serial implementation in more than 50% of speedUp just by a re-engineering procedure. The speedUp scales linearly up to 5 cores, and then tends to reach a plateau around 15 cores, for both user defined percentage selection of variables/subjects and the boostrap alternative. This can be seen in the speed up graphs, which also compare implementation with and without bootstrap sampling.

In addition to reducing the number of user-defined parameters needed, bootstrap sampling also proved to be beneficial in regards to time consumption, without showing significant/any differences in the results.

Finally, it was also established that when the algorithm uses 1 core, it's still more efficient than the serial implementation: there's an average of 20% time improvement.

Original Algorithm	Re-engineered algorithm
1) 427.789	1) 374.932
2) 431.811	2) 362.010
3) 434.320	3) 362.588
4) 432.921	4) 368.748
5) 435.626	5) 368.780
6) 445.200	6) 357.745