# moca

**m**ultivariate **o**rganization of **c**ombinatorial **a**lterations

**0.80**

**User's manual**

**Contents**

Please contact David Masica (david.masica@gmail.com) with questions, comments, or bugs. Please read the manual in its entirety the Examples before contacting the "help desk" ☺. If you find potential bugs or runtime errors, please cut and paste the python error into an email, and attach your Arguments and Paths files.

## What is MOCA?

In the broadest sense, the Multivariate Organization of Combinatorial Alterations (MOCA) algorithm can find highly correlated features. Typically, a user wishes to find features that are good predictors of a response. At the time of this writing, MOCA has been most extensively employed to find clinical and genomic features that predict phenotype (e.g., disease, drug response, cancer subtype).

MOCA is flexible. MOCA normalizes and binarizes all data, so diverse data types can be combined with little inconvenience to the user. For example, say you have a patient cohort that exhibits variable response in a drug trial. You have also catalogued some clinical and genomic features across the cohort, and want to know which of these features predict response to the drug. MOCA would easily facilitate the inclusion of any data you might have from this trial into a single calculation. Gene mutation and expression could be easily combined with patient age, smoking habits, and occurrence of radiation therapy, even though some of these data types are inherently continuous, some binary, and some categorical in nature. Furthermore, MOCA makes the best possible use of your missing data points.

It is highly recommended that you read the three articles cited at the end of this manual before doing the examples, to familiar yourself with Boolean set operations, and how to interpret combinatorial and pairwise markers.

## Installation

The recommended installation follows. This is a simple installation with no automated installer; therefore, advanced users can tailor most steps to their own liking. This assumes a Linux environment, so if you're using a Mac or any flavor of Linux you are ready to proceed. A native Linux is included with recent versions of Windows 10; if you are running an earlier version of Windows you will need to install a VirtualBox (virtualbox.org) running Linux (I recommend Ubuntu). There are many great tutorials for accomplishing this, simply google "virtualbox ubuntu windows".

Download MOCA (github.com/KarchinLab/MOCA) to your home directory and unzip it.

Add the MOCA directory to your python path. On Linux this will likely be adding "export PYTHONPATH=/home/user/MOCA/:" to your *.bashrc* or *.bash_profile*. On a Mac you'll

probably be adding "export PYTHONPATH=/Users/user/MOCA/:" to your *.profile*. Of course, *user* would be your username on your machine. Then add the line "alias moca='python ~/MOCA/moca.py'" to either your *.bashrc*, *.bash_profile*, *.profile*. For all of these additions, the double quotes need be removed, single quotes stay.

Install NumPy (http://numpy.org), RPy2 (http://rpy.sourceforge.net/rpy2.html), and Fisher's exact test for python (https://pypi.python.org/pypi/fisher/). RPy2 requires that you have R installed. Your R installation will need access to certain non-default R libraries; if you are missing any required libraries for your desired MOCA implementation the runtime error will tell which library to install.

MOCA is written for Python2.7. If your machine's default python is not Python2.7, you need to add the correct interpreter to the moca.py file in the MOCA directory. As an example, the first line of your copy of moca.py could be "#!/home/user/Python-2.7/python", if you wanted to point MOCA to a local copy of Python2.7. Once you've done this, all MOCA code will know where to find the correct python installation.

## Input Data

Several input data files are included as .csv files in the MOCA/Database/Examples directory. MOCA will accept comma separated (must have .csv extension), tab delimited (must have .txt extension), and space delimited formats (cannot have .csv or .txt extension, anything else is fine). MOCA requires a feature-by-label (row-by-column) matrix for each datatype. Features are whatever you are selecting (genes, clinical parameters, etc.) and labels are whatever you're selecting features from (patients, tumor samples, etc.). Importantly, the zero-zero coordinate of the matrix needs to be the first label, NOT a description of the features (which is somewhat often the case with publically downloaded data).

## Running MOCA

If you've followed the installation instructions, then all you need to do is type *moca* on the command line and hit enter. Provided there is a *Paths* file and *Arguments* file in the current working directory, MOCA will run from wherever. You'll be introduced to the *Paths* and *Arguments* files in the examples.

**Recommended method:** All examples in the Examples section assume you are using this approach.

1. The MOCA directory should be in your home directory, and python needs to know where it is (see Installation). This step is quite arbitrary and can be tailored

to your liking with no impact on MOCA's ease of use; however, the Examples will assume this is where MOCA is.

2. Keep your MOCA input data in the MOCA/Databases folder. This way you only need one copy of any data file, and it can remain stationary in the Databases folder (the Paths file will tell MOCA where it is).
3. Run your calculations from the MOCA/Projects folder, making a new folder in MOCA/Projects for each project.

**Running on multiple processors (`MultiProcessMode`):**

For large calculations, the power user can run MOCA in parallel across multiple processors (cores, nodes, cpu's, etc.). This is achieved using the `MultiProcessMode` command and giving each processor its own Arguments file. The `MultiProcessMode` command takes two arguments: an integer telling MOCA which processor it is being executed on and an integer telling MOCA the total number of processors used for the calculation. The default is `MultiProcessMode = 0 1`, meaning MOCA is running on the zeroth processor and that is the only processor in use. Multi-process mode works by taking a list of items to compute and uses python's built-in extended-slicing functionality.

Here is a toy example, from the python terminal, illustrating how MOCA would farm out a computation based on ten Features:

```
>>> Features = ["F0", "F1", "F2", "F3", "F4", "F5", "F6",
"F7", "F8", "F9"]
>>> TotalNodes = 3
>>> for Node in range(TotalNodes):
...     print Node, Features[Node::TotalNodes]
...
0 ['F0', 'F3', 'F6', 'F9']
1 ['F1', 'F4', 'F7']
2 ['F2', 'F5', 'F8']
```

In this example, Node 0 is taking care of Features F0, F3, F6, and F9. Node 1 is taking care of Features F1, F4, and F7…etc. Ideally, the user will write a bash script (or something similar) to utilize a cluster resource manager. In the MOCA folder, there is an example two such scripts, written to utilize the Torque queuing system (PBS) and the Sun Grid Engine (SGE). To use these scripts, you'd do:

`./MultiProcessMode.PBS Integer String` (for Torque)

or

`MultiProcessMode.SGE Integer String` (for Sun Grid Engine)

where `Integer` is the number of processors you want to use and `String` is an extension used to name that results will be saved to.

**Customizing MOCA (`MyMOCA`):**

Advanced users and python coders can easily create and extend MOCA functionality. This will typically amount to writing one or more functions in the MyMoca.py script, and these functions will call existing MOCA functions that you will be able to tweak and combine in your own way. See the directions in MyMOCA.py to get started.

## Examples

All data for the following examples are included in MOCA/Databases/Examples, and were initially downloaded from cBio (www.cbioportal.org). This data is for the 279 head and neck squamous cell (HNSC) patients from The Cancer Genome Atlas (TCGA) study (*TCGA, Nature 2015* from the dialog box in the download tab). I included 48 genes by combining the p53, PI3K-AKTmTOR signaling, and Ras-Raf-MEK-Erk/JNK signaling pathways for the mutation, Gistic copy number, and methylation datatypes. The HPV (human papillomavirus) status of each patient was obtained from the clinical data section, which is found on cBio by mousing over the dataset and clicking "summary".

**Preprocessing your data.**

Your *Paths* will initially look like this:

```
MOCA.data = ./MOCA.data
MOCA.results = ./MOCA.results

Mutation = ${HOME}/MOCA/Database/Examples/Mutation.csv
CNA = ${HOME}/MOCA/Database/Examples/CNA.csv
Methylation = ${HOME}/MOCA/Database/Examples/Methylation.csv
HPV = ${HOME}/MOCA/Database/Examples/HPV.csv
```

The *MOCA.data* directory will store all data preprocessed and ready for MOCA calculations; all data in this directory will be pickled (python serialization), and is not human readable. During the following steps you'll also notice that all intermediate results files from MOCA calculations end up in the *MOCA.results* directory. Like MOCA data files, results files are also pickled for fast reading and processing by MOCA. As you will see below, MOCA results files are converted to human readable files during various post processing steps. You can place and access these directories wherever you like—I have chosen to put them in the current working directory to make the examples simple.

The second block of paths in the *Paths* file points MOCA to the data you will be using. The MOCA.zip you downloaded comes with example data in the MOCA/Database/Examples directory. These examples further assume you've put MOCA in your home directory. You can put the data wherever you like, so long as your *Paths* file points MOCA to it. As you can see in this example, MOCA accepts absolute paths by using the "{HOME}" convention, but you can also use relative paths as illustrated for the results and data directories in the first block of paths.

Your *Arguments* file will initially look like this:

```
Data = Mutation
DataType = Binary
Mode = PreProcess
```

These arguments tell MOCA that you want to process the mutation data, and that you want it processed as binary data; we are processing this mutation data as binary because we simply want to know if a particular patient is mutated in particular gene ("1") or not ("0"). The MOCA default Mode is PreProcess, so technically you could omit that line (I include it here for completeness).

If you followed the installation and setup instructions at the beginning of this manual, typing *moca* on the command line will preprocess the mutation data.

After running MOCA for the first time, you should see that the data and results directories have been created. If at any time you want to see the results of all your previous steps, you can run MOCA in *Reports* mode. Simply add the following line to your arguments file and run MOCA again:

```
Reports = Data
```

When you run the reports you should get the following output:

```
Processed data file = ./MOCA.data/Mutation

      Source data         /Users/{HOME}/MOCA/Database/Examples/Mutation.csv
      Label count         279
      Type                Binary
      Normalized          NA
      Threshold           NA
      Feature count       49
      Missing data        0 (0.0%)
```

This output tells you where the MOCA processed file is stored (the MOCA.data directory, of course), where the original data came from, the total number of labels (synonymous with patients, samples, etc.), the type of data, the total number of features (in this case, 49 mutated genes), and the count and fraction of missing data

(none in this case). You can see there is also a *Threshold* and *Normalized* entry: these entries only correspond to continuous-valued data, which will be clear below. You can turn this argument on at any time to get a report, but for now either delete that line or set `Reports = False` so that we can preprocess the rest of the data.

Now, preprocessing the categorical datatypes. Change your arguments so they look like this, and run MOCA.

```
Data = CNA
DataType = Categorical
Mode = PreProcess
```

And again for HPV status.

```
Data = HPV
DataType = Categorical
Mode = PreProcess
```

And finally, we'll preprocess the methylation data, which is continuous valued and therefore requires thresholds so that MOCA know how to convert the data to binary.

```
Data = Methylation
DataType = Continuous
Threshold = >=0.75 <0.25
Mode = PreProcess
```

In this example, the threshold parameter tells MOCA to make two features for each gene: 1) every patient that has gene-specific methylation over 0.75 becomes a positive for over-methylation ("1"), and patients below that threshold are negative for over-methylation ("0"). The <0.25 argument has the opposite effect, generating an under-methylation feature for each gene.

Finally, run with `Reports = Data` to confirm you now have all four datasets preprocessed.

**Example 1: MOCA pairwise calculations.**

**1a**. Now we can do MOCA calculations. All calculations will be done using the data folder, so technically we only need the first two lines in the *Paths* file, but MOCA will ignore the other entries so it is fine to leave those.

To begin, we are going to do the simplest MOCA calculation, which is pairwise mode. Pairwise comparisons are exhaustive comparison of all possible feature pairs in whatever data you provide MOCA. Use the arguments below, to compare all pairs of

mutated genes, and see if there are any significantly co-occurring (or mutually exclusive) genetic mutations in HNSC patients.

```
Data = Mutation
Reports = False
Mode = MOCA
Pairwise = True
Filename = Example1a
```

Here, we tell it we just want to consider mutation as the data, we starting running the MOCA mode, rather than preprocessing mode. We just want pairwise MOCA (will do combinatorics later), so we have to set that to true. And finally, we provide a filename so we can track and view results—here I am just naming the files based on the exercises, but you can use any unique name here.

If you run with `Reports = Results` next, the following will be printed to the screen:

```
MOCA results file = ./MOCA.results/Example1a

      Pairwise                True
      Input data              Mutation
      Label count             279
      Supervised              False
      Correction method       BH
      FDR                     0.05
      Number of significant interactions  2
      Continuous-valued correlation       False
```

This gives us a record of how we ran MOCA, which can be useful especially when looking at work you ran weeks, months, or years ago. The first indicates the which pickled results file is being reported on. Next, we see that we ran a pairwise calculation, using the data called mutation, that we had 279 patients, that this is unsupervised because there was no phenotype, that we used the default Benjamini and Hochberg false discovery rate (FDR) to correct for multiple testing, that we used the default FDR of 0.05, there are only two significant pairwise interactions from this calculation, and that the data was not continuous valued.

Next let's see the results in human readable form. Run with the following arguments:

```
Data = Mutation
Reports = False
Mode = PostProcess
Pairwise = True
Filename = Example1a
```

As you can see, all we did was change the mode for post-processing, and turned reporting back off. After running these arguments you should have a file called

Example1a.csv. Open it using Excel (Word), Numbers (Mac), or Calc (Open Office; free). You should see the following:

| Feature 1 | Interaction | Feature 2 | P-value | FDR | Sample Count |
|---|---|---|---|---|---|
| CDKN2A:Mutation | Co-occurring | TP53:Mutation | 3.53E-07 | 1.15E-04 | 279 |
| HRAS:Mutation | MutuallyExclusive | TP53:Mutation | 1.18E-04 | 1.92E-02 | 279 |

As we knew from the report we just ran, we only have two significant interactions. The most significant is that CDKN2A and TP53 mutation are co-occurring (a well-known HNSC genotype). Also, HRAS and TP53 are significantly mutually exclusive. In addition to the significance (P-value and FDR), the file also shows how many patients this calculation applied to. Because the mutation data is not missing any data points, all 279 patients in our dataset were subject to this calculation.

**1b.** We'll do one more pairwise genetics calculation, before adding a phenotype—this time, with two data files. When MOCA is given two data files for pairwise comparison, it only compares features from one file with features from the other (i.e., no comparing features from the same file with one another). Run:

```
Data = Mutation CNA
Reports = False
Mode = MOCA
Pairwise = True
Filename = Example1b
```

Then

```
Data = Mutation CNA
Reports = False
Mode = PostProcess
Pairwise = True
Filename = Example1b
```

Feel free to open the resulting csv file, and look at the results. You should see 33 significant interactions.

**Example 2: MOCA pairwise marker selection.**

Now we'll select genetic markers of our phenotype, which is HPV status. In the arguments below we are including all four of our datatypes, and telling MOCA which one is the phenotype. This calculation will result in markers of HPV status, and because `Pairwise = True`, the makers will be single features associated with the phenotype.

```
Data = Mutation CNA Methylation HPV
Reports = False
Mode = MOCA
```

```
Pairwise = True
Phenotype = HPV
Filename = Example2
```

Because the original phenotype file (HPV.csv) was processed as categorical, we ended up with two features after preprocessing (one HPV+ and one HPV-). So, we need to post process each of these files separately. First:

```
Data = Mutation CNA Methylation HPV
Reports = False
Mode = PostProcess
Pairwise = True
Phenotype = HPV
Filename = Example2_HPV Status.HPV+
```

Then:

```
Data = Mutation CNA Methylation HPV
Reports = False
Mode = PostProcess
Pairwise = True
Phenotype = HPV
Filename = Example2_HPV Status.HPV-
```

Upon opening these files, you should notice two things: 1) results from marker selection have diagnostic parameters that were not present in the unsupervised results from Example 1. These include the odds ratio and effect size (calculated as the difference of proportions); the sensitivity, specificity, positive predictive value (PPV), negative predictive value, and the 95% confidence interval (CI) associated with each of these performance metrics; the accuracy and Mathew's Correlation Coefficient (MCC). The sample count and number of samples belonging to the positive class (Case Count) are also provided; these number can be very informative when trying to determine the origin or biases in any performance metrics. 2) These files are virtually the same, except that the patients defined as the cases and controls are switched. This is why the case count is 36 in the HPV+ file, and 243 in the HPV- file. This also means that the sensitivity, specificity, PPV, and NPV are reversed between the files.

Because our phenotype only has two class (HPV+ and HPV-), there is no point in looking at both classes as separate phenotypes. If we had three or more classes (e.g., cancer subtypes), then we would likely want to keep all classes as separate phenotypes. Here we remove one class to simplify subsequence analysis.

First, write a human readable version of the two-class HPV phenotype matrix:

```
Data = HPV
Reports = False
Mode = PostProcess
PostProcess = WriteMatrix
Pairwise = True
Phenotype = HPV
Filename = HPV
```

Open the results HPV.csv file created in your current working directory, remove row 3 (the *HPV Status.HPV+:HPV* feature), and for simplicity rename the feature in row 2 to just *HPV-*. Next, process this new phenotype file as we did before. Because the new, unprocessed phenotype file is in your current working directory, change your paths (note: only the last line changed):

```
MOCA.data = ./MOCA.data
MOCA.results = ./MOCA.results

Mutation = ${HOME}/MOCA/Database/Examples/Mutation.csv
CNA = ${HOME}/MOCA/Database/Examples/CNA.csv
Methylation = ${HOME}/MOCA/Database/Examples/Methylation.csv
HPV = ./HPV.csv
```

Then run the following arguments:

```
Data = HPV
DataType = Binary
Reports = False
Mode = PreProcess
Pairwise = True
Phenotype = HPV
Filename = HPV
```

This will overwrite your previous phenotype file in the data directory, which is OK because we won't be using the old one anymore. Test that you've done everything correctly:

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = MOCA
Pairwise = True
Phenotype = HPV
Filename = Example2
```

Postprocess:

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = PostProcess
Pairwise = True
Phenotype = HPV
Filename = Example2_HPV-
```

You'll get a file like you already had for the HPV- phenotype, though the file name and the naming convention for the phenotype in the spreadsheet will be simpler.

**Example3: Combinatorial marker selection.**

Now we'll try out the mode for deriving combinatorial markers of phenotypes. The following arguments tell MOCA we are running in the main mode (and without `Pairwise = True`), and that each marker can test up to three features using the union Boolean set operation, no features using the intersection operation, and up to three features using the difference operation. This will take about one minute to run.

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = MOCA
Phenotype = HPV
Filename = Example3
BooleanSets = 3 0 3
```

Make the results spreadsheet:

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = PostProcess
Phenotype = HPV
Filename = Example3_HPV-
BooleanSets = 3 0 3
```

Opening the Example3_HPV-.csv, you'll notice that the many of the markers now each contain multiple genetic features, and many of these multi-feature markers have much better performance than the pairwise markers you generated in Example 2. For example, the first marker indicates that if an HNSC patient has a TP53 mutation *OR* complete deletion of CDKN2A, then that patient is likely HPV-. This marker is highly significant, with a high odds ratio and effect size, and very high sensitivity and specificity. As before, the markers can be either mutually exclusive or co-occurring. Because multi-feature mutually exclusive markers can be difficult to interpret, we will eliminate these from our analysis by adding the `ForceCooccurring = True` argument going forward. If you become comfortable interpreting composite markers, and like working with *OR NOT* operations, you can always generate mutually exclusive markers by leaving `ForceCooccurring = False` (the default).

An important consideration with multi-feature markers of this type, is that it is relatively trivial for MOCA to generate markers that pass the default FDR but really have no biological significance. Therefore, it is a good idea to set a lower FDR based on results from permutations of the phenotype. Permuting the phenotype simply means randomly switching the labels on the cases and controls; if MOCA can find markers that pass your

significance threshold when the data is permuted (and biologically meaningless), then you have a problem. To do permutation testing, change your arguments (I have added force co-occurring here as well, though that has nothing to do with permutation testing):

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = MOCA
Phenotype = HPV
BooleanSets = 3 0 3
ForceCooccurring = True
PermutePhenotype = True
```

Then from the command line, use the following bash script:

```
x=1; while [ $x -le 10 ]; do message=$( moca | grep failed ); if
[ $( echo $message | wc -w ) -gt 1 ]; then echo $message; x=11;
else echo trial $x passed; x=$[ $x + 1 ]; fi; done
```

Hit Enter and you'll see that MOCA is running up to ten rounds of permutation testing. If MOCA finds markers that surpass your FDR cutoff in this mode, it will stop and tell you to run with a lower threshold.

Running the above command line will likely result in a couple of "passed" trials, with an eventual permutation test failed message, which instructs you that markers were found that had an FDR of ≈ 0.02. My personal rule of thumb is to decrease any failed threshold by an order of magnitude. Change the default FDR:

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = MOCA
Phenotype = HPV
BooleanSets = 3 0 3
ForceCooccurring = True
PermutePhenotype = True
FDR = 0.002
```

Then run again:

```
x=1; while [ $x -le 10 ]; do message=$( moca | grep failed ); if
[ $( echo $message | wc -w ) -gt 1 ]; then echo $message; x=11;
else echo trial $x passed; x=$[ $x + 1 ]; fi; done
```

This time you will likely see that all ten trials pass, so we use FDR = 0.002 going forward.

**Example 4: Optimizing markers for a desired performance metric.**

MOCA's multi-feature marker selection algorithm employs an evolutionary routine. As such, the user can define the selective pressure applied during marker selection, thus optimizing a desired performance metric. Choice of optimization will depend on the clinical needs of the resulting markers. The MOCA default is to optimize balanced accuracy, but this can be changed using the `RankMethod` argument. Results from Example 3 included markers with high performance by all measures except NPV (this is a typical result when the fraction of controls is small). Let's see if we can improve that by optimizing the NPV. We are also going to reduce output to the top-25 performing markers by this criteria.

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = MOCA
Phenotype = HPV
BooleanSets = 3 0 3
Filename = Example4
ForceCooccurring = True
FDR = 0.002
RankMethod = NPV
TopInteractions = 25
```

Then make the results spreadsheet:

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = PostProcess
Phenotype = HPV
BooleanSets = 3 0 3
Filename = Example4_HPV-
ForceCooccurring = True
FDR = 0.002
RankMethod = NPV
TopInteractions = 25
```

This spreadsheet only has 25 entries, as requested, and a couple markers achieved much higher NPV. However, this came at a serious loss of specificity, which is not ideal for this scenario; we will go back to the balanced-accuracy default for the remaining examples.

**Example 5: Cross validation.**

Now, instead of doing simple marker selection, we are going to try leave-some-out cross-validation selection. This is much more rigorous than our previous examples, and is often the standard for publication. This is achieved using the `LeaveSomeOut` argument. The number of cross-validation data splits is determined by the total number of samples divided by the integer you provide the `LeaveSomeOut` argument. In this

example, we are going to do 10-fold cross validation, so we divide 279/10, and round up (i.e., 28). This took about 3 minutes on a newish Mac Book Pro.

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = MOCA
Phenotype = HPV
LeaveSomeOut = 28
BooleanSets = 3 0 3
Filename = Example5
ForceCooccurring = True
FDR = 0.002
```

Now we post-process all 10 cross validation files, by setting `Mode = PostProcess` and leaving the LeaveSomeOut command (this is how MOCA knows to look for multiple cross validation files). MOCA will only return markers that pass the significance threshold on all cross-validation data splits.

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = PostProcess
Phenotype = HPV
LeaveSomeOut = 28
BooleanSets = 3 0 3
Filename = Example5
ForceCooccurring = True
FDR = 0.002
```

Lastly, post-process the combined file to get the spreadsheet.

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = PostProcess
Phenotype = HPV
BooleanSets = 3 0 3
Filename = Example5_HPV-
ForceCooccurring = True
FDR = 0.002
```

The resulting spreadsheet reveals that our best marker from previous selection (TP53 mutation *OR* CDKN2A deletion) didn't standup to cross-validation. While it can be disappointing to lose a high-performing marker, this result indicates that this marker might provide little benefit when testing in new datasets or clinical settings, a result that is best to discover up front, as we did here. Also, 279 patients, of which only 36 are HPV+, might be too few samples to draw conclusions from. Before proceeding, generate a results report (`Reports = Results`). For the results generated from a cross-validation calculation, MOCA reports which cases and which controls were included in each data split, which can be useful for subsequent analysis.

**Example 6: cross validation with a separate holdout.**

For this final example, we'll do an even more rigorous cross-validation calculation. First, a small set of 24 HPV- patients and 4 HPV+ patients are set aside for a separate holdout. We do 10-fold cross validation on the remaining 251 patients, and see how the resulting markers perform on our holdout validation set. The `LabelList` argument tells MOCA were to look in the *Paths* file to find which samples to use in each step. Repeating the initial steps as in Example 5.

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = MOCA
LeaveSomeOut = 25
Phenotype = HPV
BooleanSets = 3 0 3
Filename = Example6
ForceCooccurring = True
FDR = 0.002
LabelList = Train
```

With the modified Paths file to point MOCA to the training patients:

```
MOCA.data = ./MOCA.data
MOCA.results = ./MOCA.results

Mutation = ${HOME}/MOCA/Database/Examples/Mutation.csv
CNA = ${HOME}/MOCA/Database/Examples/CNA.csv
Methylation = ${HOME}/MOCA/Database/Examples/Methylation.csv
HPV = ./HPV.csv

Train = ${HOME}/MOCA/Database/Examples/TrainingPatients
```

Make the combined file:

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = PostProcess
LeaveSomeOut = 25
Phenotype = HPV
BooleanSets = 3 0 3
Filename = Example6
ForceCooccurring = True
FDR = 0.002
LabelList = Train
```

Then the results spreadsheet:

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = PostProcess
Phenotype = HPV
BooleanSets = 3 0 3
```

```
Filename = Example6_HPV-
ForceCooccurring = True
FDR = 0.002
LabelList = Train
```

The resulting spreadsheet reveals that no multi-feature marker passes the cross-validation with the reduced dataset of only 251 patients, but we do have a few single-feature markers. Arguably, we are probably stretching the data a little thin by doing a rigorous, multi-step cross validation with so few patients (and so few features, for that matter). But, it serves to illustrate MOCA's functionality. Finally, we see how well these markers perform in the in our validation set of 28 patients. The *Paths* file needs to point to the new `LabelList` of validation patients and the markers we wish to test in those patients (the Example6_HPV-.csv file), and we add the `PostProcess = ValidateMarkers` argument.

Paths file:

```
MOCA.data = ./MOCA.data
MOCA.results = ./MOCA.results

Mutation = ${HOME}/MOCA/Database/Examples/Mutation.csv
CNA = ${HOME}/MOCA/Database/Examples/CNA.csv
Methylation = ${HOME}/MOCA/Database/Examples/Methylation.csv
HPV = ${HOME}/MOCA/Database/Examples/HPV.csv

Test = ${HOME}/MOCA/Database/Examples/ValidationPatients
ValidateMarkers = ./Example6_HPV-.csv
```

Arguments file (provide a new filename and patient list):

```
Data = HPV Mutation CNA Methylation
Reports = False
Mode = PostProcess
Phenotype = HPV
BooleanSets = 3 0 3
Filename = Example6_Validate
ForceCooccurring = True
FDR = 0.002
LabelList = Test
PostProcess = ValidateMarkers
```

Comparing the two results spreadsheets for Example 6, TP53 mutation, and MAPK4 and CDNK2A deletion validate well by confidence interval (CI) standards, considering how small the validation set was.

## Commands

Use any of the following commands to override the MOCA defaults. In the Arguments file use the `Command = Option(s)`.

**`Bandwidth`**: By default, MOCA doesn't allow multiple features with the same base name (the part before the ":" when MOCA names features) to prevent multi-feature markers from comprising seemingly redundant individual features. For instance, a markers that said a phenotype is true if EGFR is amplified (EGFR:CNA+) provided EGFR is not deleted (EGFR:CNA-) might be meaningful numerically to MOCA but not particularly meaningful biologically to a user; therefore, MOCA's default is to prevent such a marker. By running in Bandwidth mode you can override this default.

- Options: `True`/`False`
- Default: `False`

**`BooleanSets`**: Max number of features to be combined using the union, intersection, and difference Boolean set operations. Every possible combination will be tested. So if you specify 3 2 1, the total setworks per trial (cycle) would be 62. For 3 3 3 it would be 504. And for 5 4 3 the total number of setorks sampled per trial would be 4088. Multiply this by the number of trials specified by the `Optimizations` command and you can see the number of calculations can explode quickly!

- Options: Three integers, separated by commas or spaces.
- Default: `0 0 0`

**`CorrectionMethod`**: Select the method to correct the Fisher's exact P-value and estimate the false-discovery rate (FDR) or familywise error rate. Adjusted P-values (Q-values) are always returned.

- Options: `BH` (Benjammini & Hochberg), `BY` (Benjammini & Yekutieli), `bonferroni`, `holm`, `hochberg`, `hommel`, `qvalue` (Tibsharani, PNAS 2003), `None` (bypass method).
- Default: `BH`

**`Data`**: What is the name(s) of the data being supplied to MOCA.

- Options: Any string(s)
- Default: `None`

**`DataType`**: During preprocessing the user must tell MOCA if the data is to be processed as a binary, continuous valued, or categorical.

- Default: `False` (default raises error to prevent user from processing data in unintended way.
- Options: `Binary`, `Continuous`, `Categorical`

**`FDR`**: Set the False-discovery-rate threshold for accepting interactions as significant. Choice of correction method set using the `CorrectionMethod` command.

- Options: Any floating-point number.
- Default: `0.05`

**`FeatureList`**: If you want to restrict a MOCA calculation to a subset of the total features in the provided data, make a list of the features you want to use and point MOCA to that list.

- Options: Any string that matches a file key in the *Paths* file.
- Default: `False`

**`FeatureMin`**: Set the minimum number of positive elements ("1"s) a binary feature vector must have to be considered in the MOCA calculation. Particularly useful as a timesaver if you have many vectors too sparse to participate in a significant interaction (e.g., your matrix has 100 labels and some vectors have 2 or fewer positive elements); this threshold can be set to eliminate those vectors from the MOCA calculation *a priori*.

- Options: Any integer.
- Default: `3`

**`Filename`**: Specify the filename for MOCA output.

- Options: Any string indicating the name you want to use for input/output files.
- Default: Just the word "default". Don't do this.

**`ForceCooccurring`**: Admittedly, multi-feature markers can get a bit difficult to interpret when they are mutually exclusive with the phenotype (which is indicated in the "interaction" field in all results files). Therefore, MOCA allows you to focus the calculation on interactions that are more interpretable (i.e., co-occurring only).

- Options: `True`/`False`
- Default: `False`

**`LabelList`**: If you want to restrict a MOCA calculation to a subset of the total labels (patients, samples, etc.) in the provided data, make a list of the labels you want to use

and point MOCA to that list. This is particularly use full if you want to leave some labels out for a later holdout validation.

- Options: Any string that matches a file key in the *Paths* file.
- Default: `False`

**`LeaveSomeOut`**: Performs leave-some-out cross-validation; you specify the integer number to leave out. For the common leave-one-out cross-validation, you'd just specify "1". If you want to do, say, 10-fold cross-validation, divide the total number of labels (e.g., samples, cell lines, tumors, patients, etc.) by 10 and round up to the nearest integer. So if you have 416 labels, you'd do a leave-42-out cross-validation. Beholden to your label imbalance (number of cases vs. number of controls), MOCA will try to keep the same number of cases and controls in the testing (hold-out) group. And of course, each label appears in the testing group only a single time.

- Options: Any integer.
- Default: `False`

**`MinimumPerformance`**: Sets the minimum predictive performance a marker can have to be returned to the user or for post processing. Useful to make output less verbose or use only the most predictive markers when post-processing cross-validation output.

- Options: `Sensitivity (or sens), specificity (or spec), PPV (`positive predictive value`), or NPV (`negative predictive value`).`
- Default: `False`

**`Mode`**: Tell MOCA what mode you want to run.

- Options: `Preprocess`, `MOCA`, `PostProcess`
- Default: `Preprocess`

**`MultiProcessMode`**: Use this command to parallelize calculations across multiple processors. This is useful for large calculations.

- Options: Two integers. The first specifies the node that that particular job is running on and the second specifies the total number of nodes to be used. See Running MOCA for more details.
- Default: `False`

**`MyMOCA`**: For the experienced user/Python programmer that wishes to create and expand upon existing MOCA functionality. See Running MOCA for more detail.

- Options: Any user-defined function.

- Default: `False`

**`NA`**: Tell MOCA how missing data points are defined in your data (e.g., `NA` or `nan`)

Options: Any string.
Default: `NA`

**`Normalize`**: Normalize continuous-valued data before applying the desired `Threshold(s)`. Currently only z-score normalization is available.

- Options: `True`/`False`
- Default: `False`

**`Optimization`**: The number of trials (or cycles) during which MOCA will randomly select features to combine using Boolean set operations; The frequency with which to repopulate the feature pool with top-predicting features; The percentage of top-predicting features to put in the feature pool. For the MOCA defaults (below), MOCA would make 1000 random feature selections, then every 100 times individual features from the most predictive 1% of setworks would repopulate their respective feature pools (union features go to the union pool, intersection to the intersection pool, etc.). The number of possible combinations that can be made from even 10 features is very large, so 1% of 100 random samples can be large. By default, "top predicting" is defined as the balanced accuracy (arithmetic mean of specificity and sensitivity). The defaults are only sufficient when the number of features is small.

- Options: Two integers and a float, separated by
- Default: `1000 100 0.01`

**`Pairwise`**: Run MOCA's pairwise mode (see Example 1 and 2).

- Options: `True/False`
- Default: `False`

**`PermutePhenotype`**: Randomly switch the labels on the phenotype, thus making biological nonsense out of the cases ("1") and controls ("0"). Great way to test whether there arguments and data your supplying MOCA are prone to returning false positives (i.e., you shouldn't be getting significant results during permutation testing if your arguments are sufficient for the data your using).

- Options: `True`/`False`
- Default: `False`

**`Phenotype`**: Tell MOCA which of your Data types, if any, is the phenotype for which you wish to select predictive biomarkers. For multi-feature markers you must set this. For `Pairwise` calculations you want to set this if you are selecting predictive biomarkers, rather than making a network of all correlated interactions (see Example 1 and 2, for instance).

- Options: Any of the strings supplied to the Data argument.
- Default: `False`

**`PostProcess`**: A few different methods for analyzing MOCA output.

- Options: `WriteMatrix`, `ValidateMarkers`, `WriteSetworks`, `MakePriors`
  - `WriteMatrix`: Makes a human readable version of any data matrix in your MOCA.data file.
  - `WriteSetworks`: Reads in a .csv results file and writes a human readable matrix from the features in that results file. Particularly useful if you want to combine multi-feature markers into more complex and well-performing multi-feature markers via further MOCA runs.
  - `ValidateMarkers`: Reads in a .csv results file and tests those markers on a new dataset.
  - `MakePriors`: If using priors, you have the option of extracting those priors from previous MOCA multi-feature marker. This is useful if you really want to focus the search around some features previously determined to be good markers. When running this command, make sure your Paths file's "Priors" tag points to a .csv results file.
- Default: `False`

**`Priors`**: Specify any priors (features), to include for each Boolean set operation, when making multi-feature markers. If `Priors` argument is used, MOCA looks for a `Priors` tag in the *Paths* file; this is how MOCA knows the path/name to the priors file. Example from the Paths file:

```
Priors  =  ${HOME}/MOCA/Projects/ThisProject/ThesePriors.txt
```

Then in "ThesePriors.txt", you'd have something like:

```
Union = Feature1:ExpressionPositive Feature20:Mutation
Difference = Feature13:MethylationPositive
```

- Options: `Strict`, `weighted`, or `stochastic`
  - `Strict`: You specify exactly which features to add for each Boolean set operation. For the above example, every setwork would include Feature1 and Feature20 in the union and Feature13 in the difference, in addition to any other features selected.

    - `Weighted`: If you use `PostProcess = MakePriors`, to make the Priors file from a previous `Setworks` or `ValidateBiomarkers` run, you will likely have many copies of some features. Using this command will add those features, to their respective Boolean set operations, in a fashion that is biased by the relative differences in the Priors file. Here, priors are added at random, but will be biased (weighted) by their relative counts. So, if you have 50 copies of one Feature1 and 10 copies of Feature2, there is a five-fold likelihood that Feature1 will be randomly selected during any cycle of `Setworks`, relative to Feature2. `Weighted` and `stochastic` options add anywhere between one and 10 of your priors, to a setwork; that number is chosen at random during each cycle, but is biased towards lower numbers.
    - `Stochastic`: Like the `weighted` command, except that prior selection is truly random because the list of priors is uniquified before selection. Compared to the "weighted" example, Feature1 and Feature2 would have an equal probability of getting selected each time. `Weighted` and `stochastic` options add anywhere between one and 10 of you priors, to setwork; that number is chosen at random during each cycle, but is biased towards lower numbers.
- Default: `False`

**`RankMethod`**: Multi-feature selection in MOCA utilizes an evolutionary algorithm. As such, enrichment for particular features in the multi-feature markers is guided by applying selective pressure during optimization. The default selective pressure is balanced accuracy, but you can change that to several other diagnostic measures. Also, MOCA orders markers using `RankMethod` when writing results spreadsheets.

- Options: `Sensitivity`, `Specificity`, `BalancedAccuracy`, `PPV`, `NPV`, `MeanPredictiveValue`, `Accuracy`, `MCC`.
- Default: `BalancedAccuracy`

**`Reports`**: If you are in a directory with a MOCA.data and/or MOCA.results directory, you can request reports regarding the files in those folders. This will provide you with much useful information, which is not normally accessible in these pickled files.

- Options: `Data` or `Results`.
- Default: `False`

**`Seed`**: For any calculation that is too large to be exhaustive, there will be some random sampling of features. Use this command to control these stochastic elements of the calculation and make your calculation reproducible. If you don't assign a seed, MOCA will assign a random seed and make note of it. If you ever need to reproduce results, run

`Reports = Results` to see the see MOCA randomly assigned to the calculation of interest. Note: this might only guarantee reproducibility on a single machine or operating system, but will probably be consistent across operating systems.

- Options: Any integer.
- Default: `False`

**`Threshold`**: For continuous-valued datatypes, you must tell MOCA how to binarize the data by providing one or more thresholds during preprocessing. For instance, one of your features might be patient age. You could tell MOCA to make four binary feature vectors using the following:

```
Threshold = <25 <50 >=50 >=75
```

This would result in one vector for patients under 25, one for patients under 50, one for patients 50 and over, and one for patients over 75. The only operators accepted at this time are < and ≥.

**`TopInteractions`**: Tell MOCA how many of the top-performing interactions you want returned to results spreadsheets. For instance, if this was set to 1,000 and RankMethod was set to PPV, results would be ordered by positive predictive value, and as many as 1,000 markers written to the results spreadsheet (provided that many passed your significance threshold).

- Options: Any integer.
- Default: `100`

## References


Masica, David L., and Rachel Karchin. "Correlation of somatic mutation and expression identifies genes important in human glioblastoma progression and survival." *Cancer research* 71.13 (2011): 4550-4561.

Masica, David L., and Rachel Karchin. "Collections of simultaneously altered genes as biomarkers of cancer cell drug response." *Cancer research* 73.6 (2013): 1699-1708.

Masica, David L., Marco Dal Molin, Christopher L. Wolfgang, Tyler Tomita, Mohammad R. Ostovaneh, Amanda Blackford, Robert A. Moran et al. "A novel approach for selecting combination clinical markers of pathology applied to a large retrospective cohort of surgically resected pancreatic cysts." *Journal of the American Medical Informatics Association* (2016): ocw069.