GAPWM: A Genetic Algorithm Method for Optimizing a Position Weight Matrix

Leping Li, Yu Liang and Robert Bass
Biostatistics Branch and Computational Biology Facility
National Institute of Environmental Health Sciences
Research Triangle Park, North Carolina 27709
li3@niehs.nih.gov

GAPWM is a tool that derives an improved PWM via a genetic algorithm from an existing PWM, a set of sequences containing a motif (e.g., ChIP sequence) and a set of background sequences.

Usage:

```
gapwm -fm <motifSeqFile> -fb <backgSeqFile> -fpwm <initialPWMfile>
[optional_arguments]
```

-fm	motifSeqFile	file containing motif sequences in FASTA format, e.g., ChIP file
-fb	backgSeqFile	file containing background sequences in FASTA format
-fpwm	initialPWMfile	file containing initial PWM
-g	maxGeneration	maximal number of generations for a GA run (default: 1000)
-p	populationSize	population size (default: 100)
-rg	greaterMutationRate	mutation rate for first 100 generations (default: 0.05)
-rl	lesserMutationRate	mutation rate for the generations after 100 (default: 0.02)
-sp	specificity	for ROC integration from 0 to 1.0-specificity (default: 0.9)
-fo	output	name of the output file (default: output.txt)
-fc	constraintFile	file containing position constraints

Mandatory Arguments

Three files are required to run GAPWM: 1) file containing motif sequences as denoted by the ‘-fm’ option; 2) file containing background sequences as denoted by the ‘-fb’ option; and, 3) file containing initial PWM to be optimized as denoted by the ‘-fpwm’ option. Unless qualified, each file is assumed to reside in the current working directory.

Example:

```
gapwm -fm ../Seq/human_Oct4/hg17BoyerOct4LociMasked_training1.seq
-fb ../Seq/human_Oct4/backgCDS_650_split1_set1.seq -fpwm ../PWMs/knownOct4Sox2.mx
-g 1000 -p 10 -rg 0.05 -rl 0.02 -fc ../Others/Oct4Sox2_constraints.txt -fo
result.txt
```

Information on Required Files

Sequences

Sequences in both the motif set and the background set should be in case-insensitive FASTA format. For example:

```
>chr1:12610241-12610625 5'pad=0 3'pad=0 revComp=FALSE strand=
ggaagagttaatcggatcggctttggctgatagttcaggctccaaagttc
agtcccagtcagagccaccccggaggaattgtaaatctcagggcagtatt
taacaaaacaaaagcaacctggaattacatgcaggtttggttttctacag
tacatatttacttaatccccaaggtatgcggctccatgtcagatcagctg
gctttgcggccctttcaccccccctagttcacaacagtttaagtttcaaac
taattccctgttttcgctcttcctcttcacagggctggctggagacagcc
tggcctgcctccctctcctgatggctctggtcaccgcgtgagtcagcctg
gcctgggctgggagttgggtgacagcctgcccact
>chr1:18703077-18703668 5'pad=0 3'pad=0 revComp=FALSE strand=
cgcatcagcccgcacaacttctggccgaggccagccggcagaggcggact
tggggttggagtgtttgtttgtttgaacttcctcgtcgtcgccaccttcc
ctccccccaacctccaccccacctcacccccctccccagcttctggacgc
gtttgactgcagccaggggtggggggtgggggtagggagtgtgtgtggag
gggagggagaagaggttaaaaaaaagaagacgaagaagacggaaagaaag
agatcgcagcaggggtgaagggagcggacgggaagcgatttttgccgact
ttggattcgtccccggcgtgcgcaagaatggcggcccttcccggcacggt
accgagaatgatgcggccggctccggggcagaactaccccccgcacgggat
tccctttggaaggtaagaacgcccaggctggcctcgccgcgactccgccg
cccggaactcggggtccttggagaggctgcggtctccaggggacggtggc
ggcgccggcgatagcagagggatcccgttctcttctgggtcccagtccgg
gcgcggaacccagggagtttctgggacccatacttgtccgct
>chr1:52581595-52581889 5'pad=0 3'pad=0 revComp=FALSE strand=
caaagaacgaaacaagtagagtgctttacaaatgcagatggagggaaagt
catcactgagcatcagggtgcggagggcaggaatgctcctgcttctaggc
tgttggcttccgccttccccccctgcaaactcagttccctgcagcgcggga
agccttttaggaatcggagtgtggaacagaggaacgctcttaacagttcn
nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
nnnnngactgagtctacagagaacgtcctcagtttcagaggttcc
```

The background sequences may be randomly selected from coding regions, introns, or upstream regions. The average lengths of motif sequences and the background sequences should be comparable. Some background sequences are provided in gapwm package.

Initial position weight matrix (PWM)

GAPWM reads in the initial PWM from an input file with the argument –fpwm followed by the file name. The first line must contain two (ONLY two) integers that specify the dimension of the PWM. The first integer should always be 4 whereas the second integer should match the length of the motif/PWM. For example:

```
4  17
0  7  0  4  4  0  0 11  0  4 11  0 11 11  4  0  1
7  0  2  0  0  0 11  0  0  2  0 11  0  0  0  0  1
2  1  2  0  0 10  0  0  0  1  0  0  0  0  0 11  1
2  3  7  7  7  1  0  0 11  4  0  0  0  0  7  0  1
```

The value in each cell can be an integer or a float number, e.g., frequency. Standardization will be carried out by gapwm. The fields can be tab delimited or separated by white space(s).

Optional Arguments

Maximal number of generations

GAPWM will stop when the maximal number of generations is reached. This limit can be specified using the '-g *n*' option, where *n* is an integer representing the desired generations (eg., -g 100). By default, the maximal number of generations is 1000.

Population size

The population size used by GAPWM can be specified using the '-p *pop*' argument, where *pop* represents an the desired population expressed as an integer (eg., -p 200). By default, the population size is 100.

Mutation rate

GAPWM uses two mutation rates: a greater mutation rate for the first 100 generation and a lesser mutation rate for the remaining generations. The default greater mutation rate of 0.05 can be overridden by the user through the use of the '-rg *m*' argument, where m is a floating point number. Similarly, the default lesser mutation rate of 0.02 can be altered using the '-rl *m*' argument.

Specificity

We assumed that each ChIP sequence in the training set contains at least one motif and each background sequence contains no motifs. Thus, given a threshold, one can calculate "true" positive and "false" positive rates, defined as the proportions of sequences scoring at or above the threshold in training and background sets, respectively. Further, we can define a fitness score of each PWM as the area under the receiver operating characteristic (ROC) curve across a particular range of false positive rates.

The user can specify the desired specificity using the '-sp *s*', where s is a floating point number. By default, the specificity is 0.9, or 90% of the ROC curve.

For example, -sp 0.8 will integrate the area under the ROC curve across a range of false positive rates from 0 to 0.2 as the fitness score.

Output file name

By default, the results will be saved into a file in the current working directory called 'output.txt'. Using the '–fo *output_filename*' argument, one can specify an alternate output file and location. The input parameters and other information are also saved in this file.

Constraints

Prior knowledge such as base conservation can be easily incorporated into our GAPWM algorithm. Perfectly conserved positions can be fixed at any stage of the search. Mutation can be restricted to specific bases (e.g., between two or among three bases). These features are useful for finding starting values for positions with no binding data. For instance, one might wish to explore the base pairs surrounding a known motif. The part of the PWM corresponding to the known motif can be fixed while the remaining part can be searched by a GA.

The constraints are read from an input file using the –fc argument followed by the file name. In the file, the first line specifies the number of constraints. Each of the following lines should have three elements: two integers and a string of characters (not case sensitive). The two integers specify the range of the generations for which a constraint will be imposed whereas the string specifies the domain of the constraint. The number of letters in the string must correspond to the length of the motif (PWM). A position with an 'a', 'c', 'g', or 't' is fixed during these generations. It does not matter whether the base corresponds to the consensus base at that position or not as long as it is one of the four bases. Yes, this means that natttgcataacaatgn and naaaaaaaaaaaaaaan have the same effect in that positions 2-16 are fixed! A position with a degenerate code (see below) is restricted to the corresponding bases. For instance, a position with a ‘w’ is restricted to an ‘a’ or ‘t’. A position with an 'n' is NOT constrained at all.

W = A or T
R = A or G
K = G or T
B = C, G, or T
H = A, C, or T
N = A, C, G, or T

S = C or G
Y = C or T
M = A or C
D = A, G, or T
V = A, C, or G

For Oct4/Sox2 PWM, the following constraints may be used:

```
  2
  1     10 natttgcataacaatgn
 11    100 nwnwwgcatnacaawnn
```

In this example, there are two constraints: 1) from generations 1 to 10, positions 2-16 are fixed; 2) from generations 11 to 100, positions 2, 4, 5, and 15 are restricted to 'a' or 't' and positions 6-9 and 11-14 are fixed. No constraints are imposed after generations 100.

Example 2:

```
  2
  1   10 rrrcwwgyyyrrrcwwgyyy
 11  100 nnncnngnnnnnncnngnnn
```

In this example, there are also two constraints: 1) from generations 1 to 10, positions 1, 2, 3, 11, 12, and 13 are constrained to r, r='a' or 'g', positions 8, 9, 10, 18, 19, and 20 to y, y='c' or 't', and positions 5, 6, 15, and 16 to w, w='a' or 't', positions 4, 7, 14, and 17 are fixed; 2) from generations 11 to 100, positions 4, 7, 14, and 17 are fixed and no constraints are imposed on any other positions . No constraints are imposed after generations 100.

Example 3:

```
1
   1   500 catttgcatnnnnnnnn
```

In this example, the first nine columns (corresponding to the Oct4 PWM) in the PWM are fixed whereas the last eight columns (corresponding to the Sox2 PWM) are being optimized during the first 500 generations.

Example 4:

```
1
   1   500 nnnnncatttgcataacaatggnnnnn
```

In this example, the first and last five columns in the PWM are being optimized whereas the 17 columns in the middle (corresponding to the Oct4/Sox2 PMW from MEME, in this case) are fixed. This example illustrates the type of constraints one might consider when exploring the flanking regions of a binding site. The initial PWM for this constraint might look like:

```
4  27
1 1 1 1 1 36 73  5  0 35 21   0 100  8 47 51  3 92 94 37 41 23 1 1 1 1 1
1 1 1 1 1 29  1  0 14  0  9 101   0  9  5  1 41  3  2  5 13 16 1 1 1 1 1
1 1 1 1 1 18  2  2  0  0 70   0   1  1 17  0 33  1  4  6 27 52 1 1 1 1 1
1 1 1 1 1 18 25 94 87 66  1   0   0 83 32 49 24  5  1 53 20 10 1 1 1 1 1
```

Programs for processing the gapwm output file

Three auxiliary programs (avepwm, fitnesspwm, and rocpwm) are also included in the gapwm package.

avepwm:	averages a set of PWMs in the gapwm output
fitnesspwm:	computes fitness score for each PWM in the gapwm output
rocpwm:	computes the ROC curve for a PWM

Details on how to use the auxiliary programs can be found in the corresponding directories.