`regevscape` Manual

Regulatory Evolution Landscape Simulator

Kevin Bullaughey
kbullaughey@gmail.com

February 11, 2011

Contents

1 Overview

This manual provides documentation of the `regevscape` program, which is short for **reg**ulatory **ev**olution land**scape** simulator. `regevscape` simulates sequence evolution of an enhancer using a computational model of regulatory function based on the Segal model [1], a fitness penalization for misexpression, and forward population simulations. It was originally used by Bullaughey [2] to investigate nucleotide substitution processes in a regime of stabilizing selection for a particular regulatory output of simple toy enhancers. For an overview of the model and details on the implementation, please refer to the original paper [2]. The purpose of this manual is to document usage of the program and aid interpretation of the output.

2 Terminology

Throughout this manual I invoke specific meanings of certain biological terms as they are relevant to the simulation.

A transcription factor's **activity** is the degree to which it contributes to activation of the basal transcription apparatus (for a repressor, this is a negative quantity). An **enhancer** is a specified length of DNA that is being modeled and is assumed to have regulatory function, and be in whatever chromatin configuration is necessary to be expressed. The enhancer is assumed to function in one or more discrete ***trans*-backgrounds**, each of which can be thought of as a particular cell type, developmental time point, or spatial position. Each *trans*-background is defined by the expression levels of the transcription factors (TFs) present. Each *trans*-background has an optimal expression level of the gene in that *trans*-background. The *trans*-backgrounds, together with the optima and a specification of the TF binding functions and activity levels, defines a **regulatory problem**, i.e., the particular task faced by the gene to express properly and what it has to work with.

Some terminology relates specifically to the Segal model and I follow my earlier conventions [2]. Occupancy of a nucleotide in the enhancer is the probability a TF is observed to be bound in a way overlapping the nucleotide. A configuration is a particular arrangement of TFs on the enhancer. The only valid configurations include the empty configuration (no TFs) or configurations with non-overlapping TFs.

3 Modes of operation

`regevscape` has two basic modes of operation. One is to do forward population simulations. The other mode is to compute various statistics related to the sequence fitness landscape. Possibilities here include computing expression profiles or various occupancy statistics. These statistics can be computed for either sequences sampled *iid* from sequence space or all sequences within some number of mutational steps of a provided or sampled sequence. Alternatively, one can simply sample *iid* from the distribution of configurations of TFs for a regulatory problem.

4 Inputs

There are several components that need specification in order to parameterize the regulatory landscape and optionally, simulate regulatory evolution of an enhancer. Parameters required for specification of the fitness landscape include the binding functions of the TFs, the TF activity levels, the expression levels of these TFs in each *trans*-background, the optimal expression levels in each *trans*-background, and the shape of the fitness function. For simulating regulatory evolution, additional parameters need specification including population size, number of generations, mutation rates, insertion and deletion rates and distributions, and the initial sequence representing the initially monomorphic population. Finally, parameters dictating what output is to be provided are also needed. All these parameters are specified on the command line in an order-independent way.

5 Invoking `regevscape`: an example

As an example, the following is one invocation of the `regevscape` command used in the original paper:

```
./regevscape --model=segal --steps=200000 --length=100 --seed=1 \
  --pwm=pwm_pair.pwms --tfs=2 --lambda=-1,2,-3 --gamma=2,-0.5,-0.5,2 \
  --ffunc=f4 --coef=0.6,0.041,0.786,0.999 \
  --condition=0.02,0.3 --condition=0.3,0.3 --condition=0.3,0.02 \
  --popsim -N 1000 -g 2000000 -u 5e-07 \
  --pstat=most_freq_seq,100 --pstat=mutational_effects --pstat=allele_loss \
  --seq=catttcggtcttgtttttggcgc...
```

Here the Segal model is used to simulate evolution of a 100 bp enhancer under three *trans*-backgrounds using a Gaussian kernel as the expression-fitness function.

To get a condensed help message detailing the possible command-line options to `regevscape`, simply invoke the program with no options, or the `--help` or `-h` options.

6 Parameters

Specification of the model is rather verbose at the moment, which allows for some consistency checking so that copy-paste errors can somewhat be avoided when running many variations of simulations.

Currently the only model that is implemented in its entirety is the Segal model. Originally, I experimented with several other models of fitness landscapes (completely neutral, Orr's block model), but these are not complete. So all invocations of the program should contain the parameter: `--model=segal`.

And when invoking the Segal model, one needs to specify how many samples, S, will be drawn (*iid*) from the distribution of configurations. This is to compute the following summation, which is estimated using Monte Carlo sampling:

$$P(E) = \sum_{c_k \in C} P(E|c_k)P(c_k) \approx \frac{1}{S}\sum_{i=1}^{S} P(E|c_i)$$

How many samples to draw is given with the `--steps=S` parameter. Naturally, the more samples, the more accurate the estimate of expression. This is particularly important when running evolutionary simulations, because misestimation of expression will lead to misestimation of fitness, and potentially allow mutations to fix that would not ordinarily fix, and potentially, create artificial local optima, due to an overestimation of fitness, making subsequent mutations look artificially unfit. I found that 200,000 samples was appropriate for a population size of $N = 1000$ and a selection tolerance parameter of $\sigma^2 = 0.6$ (see below). In this setting, estimation error was generally an order of magnitude smaller than $\frac{1}{N}$ which is the approximately the magnitude when selection is overwhelmed by drift.

The `--length=N` parameter gives the length of the enhancer sequence. This is not required if a sequence is specified, but must be provided if computing statistics of sequences sampled uniformly from sequence space (so the program knows to generate sequences of the desired length).

The seed of the random number generator can be set with `--seed=N`, where `N` is an non-negative integer. Starting with the same seed is guaranteed to run deterministically, and identically each time.

Finally, if either a population simulation starting with a particular enhancer sequence is desired or if landscape statistics of a certain enhancer sequence are desired, the enhancer sequence can be specified with `--seq=<dna>`. If multiple sequences are given, the requested operations are performed for each of the given sequences.

6.1 Regulatory problem

Specification of the regulatory problem requires providing the parameters: `--pwm`, `--tfs`, `--lambda`, `--gamma`, and one or more `--condition` specifications. In what follows, I reproduce equations from the paper using this implementation [2], which are very similar to the formulation given in the original formulation of the Segal model [1]. For a more complete description of the model, please refer to these papers.

`--pwm=<filename>` gives the file name of the position weight matrix file which specifies the binding affinity of the TFs. Here is an example of a valid pwm file:

```
pwm 6
0.3369 0.2508 0.9626 0.8526 0.6829 0.8748
0.0513 0.3186 0.0181 0.0449 0.2433 0.0072
0.5338 0.2232 0.0077 0.1021 0.0123 0.0902
0.0779 0.2074 0.0116 0.0004 0.0615 0.0278
pwm 6
0.0834 0.1921 0.4285 0.1172 0.9270 0.2437
0.1072 0.0138 0.1041 0.0154 0.0022 0.1425
```

```
0.3878 0.1982 0.3906 0.8614 0.0650 0.0247
0.4216 0.5958 0.0768 0.0059 0.0058 0.5891
```

For this file to be formatted properly, it must consist of one or more PWM records. Each record has a header line with the word `pwm` followed by a space followed by an integer, say r, indicating the width of the PWM in nucleotide positions. The next four lines must contain a matrix of site-specific affinities such that there are r columns and each column sums to one. These are the relative affinity of each position to each of the four possible nucleotides. Rows correspond to the affinities of each site to A, G, C, and T respectively.

The number of TFs that are being modeled is specified with the `--tfs=T` parameter, and this must match the number of PWM records provided. `--lambda=`$\lambda_0, \lambda_1, \ldots, \lambda_T$ gives the activity levels of the T TFs preceded by the basal activity level of the promoter, λ_0. These (floating-point) parameters come into play in the formulation of $P(E|c_k)$, which follows a logistic sigmoid:

$$P(E|c_k) = \frac{1}{1 + \exp(-\sum_{i=0}^{M} \lambda_{TF_i})}$$

Where there are M TFs in configuration c_k. The length of the `--lambda` vector must be one more than the number of TFs given in `--tfs`. A positive λ codes for an activator and negative λ codes for a repressor. Similarly, a negative λ_0 codes for a generally repressive basal promoter and a positive one codes for a generally activated one. A present limitation of this implementation is that the λ parameters are the same across all *trans*-backgrounds, not allowing for a TF to sometimes be an activator and sometimes a repressor, a situation known to be important in some true biological settings.

Interaction properties of the TFs are given by `--gamma=`$\gamma_{11}, \gamma_{21}, \ldots, \gamma_{TT}$, which lists a matrix (iterating over rows before columns) of pair-wise TF-TF cooperativity parameters. It probably makes most sense for this to be a symmetric matrix so that the interaction is the same for both orderings of a pair of adjacent TFs. Omitting this parameter means there are no interactions among TFs. $\gamma_{jk} \in [-1, \infty)$, with $g_{jk} = 0$ indicating no interaction. These γ_{jk} parameters have the effect of altering the probability of a configuration in a way that is a function of the distance, d, between adjacently bound TF_{i-1} and TF_i, such that the probability of a configuration, c_k is:

$$P(c_k) \propto w(c_k) = \prod_{i=1}^{M} \tau_{\mathrm{TF}_i} \frac{\mathrm{PWM}_{\mathrm{TF}_i}(s_x, s_{x+1}, \cdots, s_{x+r-1})}{\mathrm{PWM}_b(s_x, s_{x+1}, \cdots, s_{x+r-1})} \prod_{i=2}^{M} \gamma(\mathrm{TF}_i, \mathrm{TF}_{i-1}, d)$$

where γ_{jk} is given by this function of the distance, d, between the adjacent TFs, $j = \mathrm{TF}_{i-1}$ and $k = \mathrm{TF}_i$:

$$\gamma(\mathrm{TF}_i = j, \mathrm{TF}_{i-1} = k, d) = 1 + g_{jk} e^{\frac{-d^2}{v}}$$

Currently, v is hard-coded to 80, which means that interactions only extend maybe 20 bp or so for modest g_{jk} (e.g.,. $-0.9 < g_{jk} < 9$, a range covering up to a 10-fold decrease or increase in $P(c_k)$ due to the interaction).

Finally, the last requirement for a regulatory problem is specifying the expression levels of the TFs in each of the *trans*-backgrounds. These expression profiles are given by the

`--condition=`$\tau_1, \ldots, \tau_T$ parameters, where $\tau_i > 0$. So, if a regulatory problem has three distinct *trans*-backgrounds, then three `--condition` vectors must be given.

6.2 Fitness function

The original purpose of the Segal model was to provide a predictive model of expression and was essentially an optimization/inference problem. In order to use it to study evolution, I consider several parameterizations of expression-to-fitness mappings. Presently there are six fitness functions coded. The desired fitness function is given by the `--ffunc=<choice>` parameter where `<choice>` is a character string selecting one of the following:

Choice	Functional form	Conditions	Coeff.	Notes
`linear`	$f = \mathbf{x}^T\mathbf{c}$	B	B	linear combination
`nop`	$f = 1$	arbitrary	none	neutral
`f1`	$f = c_1(x_1 - c_2)^2 + c_3(x_2 - c_4)^2 + c_5$	2	5	
`f2`	$f = c_1^{(-c_2 * (x_1 - c_3)^2)} c_4^{(-c_5 * (x_2 - c_6)^2)}$	2	6	
`f3`	$f = \prod_{i=1}^{B} (c_1^{-c_2(x_i - c_{i+2})^2})^{\frac{1}{C}}$	B	$B + 2$	generalization of `f2`
`f4`	$f = \exp(-\sum_{i=1}^{B} \frac{(x_i - c_{i+1})^2}{c_1})$	B	$B + 1$	Gaussian kernel

Each fitness function (with the exception of `nop`) requires a vector of coefficients, `--coef=c1,c2,...`, parametrizing the selected function.

Individuals are haploid, and so only the expression of a single allele need be computed.

6.3 Evolution simulation

The population simulation is a standard forward population simulation with Wright-Fisher multinomial sampling of alleles based on the fitness of the alleles. Individuals are haploid, and the expression profile of a single enhancer is the sole determinant of fitness for the individual. Fitness of each allele is computed when it arises, and each newly arisen allele is given a unique integer ID even if this particular allele has arisen before by a separate mutational event. I assume a Jukes-Cantor mutation model with a single mutation parameter, thus mutating to/from any base is equiprobable. There is no recombination implemented at the moment.

To run a forward population simulation, give the parameter `--popsim` (no argument required). The population size is indicated with `-N` and the number of generations with `-g` and the per-base-pair mutation rate is given with `-u`. The mutation rate can be given either in decimal or scientific notation.

To include insertions and deletions into the mutation model, provide the `--duplication=u,n,p` and `--deletion=u,n,p` parameters. As the name suggests, insertions are modeled as tandem duplications. Both duplications and deletions occur with some rate, `u`, and then conditional on occurring, the length distribution follows a negative binomial distribution with size parameter, `n`, and probability parameter, `p`. When the draw from the negative binomial is 0, this is equivalent to no mutation occurring (and is not recorded). Thus the actual probability of no mutation (of the particular type) is not exactly 1-`u`, it's actually a bit higher, including the zero-class of the negative binomial distribution.

6.4 Population statistics

Population statistics are only permissible when running a population simulation by specifying `--popsim`. There are two types of population simulation statistics: those printed at regular intervals and those printed when certain events occur. The former are summaries of the state of the simulation at the generation when they're printed, while the latter detail the particular event and give the generation when the event occurred. The below table summaries the possible statistics.

Statistics printed every `N` generations:	
`--pstat=most_freq_seq,N`	Most frequent allele summary with sequence
`--pstat=most_freq_noseq,N`	Most frequent allele summary, but no sequence printed
`--pstat=mean_fitness,N`	Population mean fitness
`--pstat=all_alleles,N`	Details of each allele, sorted by number of copies
`--pstat=allele_counter,N`	Number of alleles queried so far
Real-time statistics:	
`--pstat=mutational_effects`	Print fitness info for new mutants
`--pstat=allele_loss`	Print the generation and allele ID for alleles when they're lost

6.5 Landscape statistics

Landscape statistics can only be computed in the absence of `--popsim`, in which case one or more `--lstat` statistics must be requested (otherwise there is nothing to do). Each landscape statistic has the form: `--lstat=<stat>,N` where `<stat>` is the name of the requested statistic, and `N` is the number of sequences about which to provide the statistic (i.e., when sampling). Landscape statistics can be computed for either individual sequences (given with one or more `--seq` parameters) or for the indicated number of replicate sequences (which are sampled *iid* from sequence space). If multiple `--lstat` parameters are given it makes most sense for each to have the same number or replicates, in which case all statistics are computed for each of the `N` sampled sequences.

If `--neighbors=R` is given, then starting from each given or sampled sequence, all neighbors within a mutation radius of `R` will also be included in the output. By mutation radius, I mean sequences that can be reached with `R` or fewer point mutations. When `--neighbors` is given and a landscape statistic is requested with more than one replicate, then statistics for all neighbors of each replicate will be computed.

The possible landscape statistics are described in the following table:

Statistic	Models	Description
`--lstat=fitness,N`	All	Print the enhancer fitness
`--lstat=sequence,N`	All	Print out the sequence
`--lstat=expression,N`	Segal	Print vector of expression levels
`--lstat=tf_occupancy,N`	Segal	Print list of TF occupancy levels for each condition and TF combination
`--lstat=wsum,N`	Segal	Print the sum of all configuration weights
`--lstat=configs,1`	Segal	Print out `N` configurations, where here `N` is given by the `--steps=N` parameter

7 Output

The program always prints two summary lines at the beginning. The first, beginning with `cmd:` is the particular invocation of the command that was run, and the second, beginning with `params:` gives the values of all parameters, including those that were not specified but are set using the default values for those parameters (note, some of these are not documented here). In the examples below, some of the output is replaced by an ellipsis (...) for brevity and lines ending with a slash are not broken in the actual output and do not contain the slash. In (nearly) all cases, the statistic gives the generation at which point it is printed, the name of the statistic and one or more columns some of which may be prefaced by labels. Nucleotide positions count from zero on up from left to right.

All of the output given below is available in its entirety in the `examples` directory of the `regevscape` distribution. In each case, there is a shell script with the original command and the output from that command under the same root file name (with a `.out` extension).

7.1 `--pstat=most_freq_seq,N`

This results in a summary giving the most frequent sequence. The columns following the statistic name are: the allele ID, the number of substitutions on the lineage leading to this allele, the allele's sequence, the fitness, and the number of copies of the allele at this generation (Here N=1000).

```
gen: 0 pstat_most_freq_seq: 0 0 catttcggtc... 0.976249 1000
gen: 100 pstat_most_freq_seq: 0 0 catttcggtc... 0.976249 978
gen: 200 pstat_most_freq_seq: 0 0 catttcggtc... 0.976249 757
gen: 300 pstat_most_freq_seq: 13 1 catttcggtc... 0.996808 873
gen: 400 pstat_most_freq_seq: 13 1 catttcggtc... 0.996808 930
.
.
```

As we can see, a substitution occurred somewhere between generations 200 and 300, and this was the 13th new allele to segregate in the population. For the full example see: `examples/popsim1.sh`

7.2 `--pstat=most_freq_noseq,N`

Similar to `--pstat=most_freq_seq,N` but omitting the DNA sequence.

7.3 `--pstat=mean_fitness,N`

Print just the mean fitness of the population every N generations. Here's an example from `examples/popsim5.sh`:

```
gen: 0 pstat_mean_fitness: 0.603736
gen: 200 pstat_mean_fitness: 0.746087
gen: 400 pstat_mean_fitness: 0.827466
gen: 600 pstat_mean_fitness: 0.947273
gen: 800 pstat_mean_fitness: 0.956373
gen: 1000 pstat_mean_fitness: 0.962493
```

In the above simulation, a sequence was not specified, so instead one was sampled uniformly from sequence space, and thus the initial allele is not very fit at the beginning of the simulation, but the population quickly evolves adaptively toward a much more fit solution.

7.4 `--pstat=all_alleles,N`

Instead of just the most frequent allele, print a line for each allele in the population. Summary lines are the same as in `--pstat=most_freq_seq`:

```
gen: 500 pstat_all_alleles:
0 0 catttcggtcttgtttttggcgggaagatgttctcgactgtgtgccgcgt 0.976822 1000

gen: 750 pstat_all_alleles:
39 1 catttcggtattgtttttggcgggaagatgttctcgactgtgtgccgcgt 0.978885 3
0 0 catttcggtcttgtttttggcgggaagatgttctcgactgtgtgccgcgt 0.976822 997

gen: 1000 pstat_all_alleles:
50 1 catttcggtgttgtttttggcgggaagatgttctcgactgtgtgccgcgt 0.982094 4
0 0 catttcggtcttgtttttggcgggaagatgttctcgactgtgtgccgcgt 0.976822 996
```

This example is provided in `examples/popsim4.sh`.

7.5 `--pstat=mutational_effects`

This prints each time a new mutation occurs and following the statistic name are the background allele sequence on which the mutation occurs, the ID of this background allele, the ID of the new allele, the number of copies of the background when the mutation occurs, the type of mutation (point, duplication, deletion), the site at which the mutation occurs (for deletions and duplications, this is the left-most nucleotide involved in the event), the number of nucleotides involved, the bases at the site(s) involved on the original background, the bases this is replaced with, the fitness of the background allele, the fitness of the new allele, the new DNA sequence resulting from the mutation, and whether this is the first time this allele has been seen. Here is an example:

```
gen: 17 pstat_mutational_effects: background: catttcggtc... old_id: 0 \
  new_id: 6 copies: 999 type: point site: 27 len: 1 from: 'a' to: 't' \
  bfit: 0.976249 mfit: 0.996041 new: catttcggtc... isnew: 1
```

The above is from the example: `examples/popsim2.sh`.

7.6 `--pstat=allele_loss`

Tracking of when alleles are lost from the population (due to not getting sampled in the subsequent generation) gives output like:

```
gen: 18 pstat_allele_loss: 6
```

which indicates that at generation 18 allele with ID 6 was lost.

7.7 `--lstat=fitness,N` and `--lstat=sequence,N`

In the same way that any number of `--pstats` can be given, one can also request multiple `--lstats`. This allows one to get multiple types of information about the requested sequences or sampled sequences without multiple invocations of the program. Here is an example (`examples/lstat1.sh`) in which both `--lstat=fitness,4` and `--lstat=sequence,4` were given with no `--seq` provided (so all outputs are for randomly sampled sequences):

```
rep: 0 lstat_fitness: 0.425137
rep: 0 acgagtgcgtcggatgccctcactttctatgttgtgcggcagttcaaaca
rep: 1 lstat_fitness: 0.194325
rep: 1 gagtgagttgtgacagattatgtggctgtgggcgcgcagcccgactttcc
rep: 2 lstat_fitness: 0.532711
rep: 2 acgggccgacggggcggaggcagaatccgcgcgtacctatcaagacctgg
rep: 3 lstat_fitness: 0.564029
rep: 3 tgaacctatataactcacactgtattctcggtggccctcgaacaaataat
```

7.8 `--lstat=expression,N`

This is given to compute the expression in each *trans*-background (in the same order as the `--condition` arguments). In the following example, both `--lstat=expression,1` and `--lstat=fitness,1` are used along with one enhancer sequence. Here the fitness function is parameterized with

`--ffunc=f4` and `--coef=0.6,0.041,0.786,0.999`, with the later three coefficients indicating the optimal expression. In the output below, we can see that the sequence is near the fitness optimum (1.0) but due to slight misexpression, it is not perfectly fit:

```
rep: 0 lstat_expression: 0.059615 0.90464 0.99958
rep: 0 lstat_fitness: 0.976889
```

7.9 `--lstat=tf_occupancy,N` and `--lstat=wsum,N`

One interesting summary of the enhancer sequence is the amount of binding of each TF in each *trans*-background at each nucleotide position. This can be obtained with `--lstat=tf_occupancy,N` producing output like the following:

```
rep: 0 lstat_tf_occupancy: condition: 0 tf: 0 oc: 0.000345 0.182335 0.25777 ...
rep: X lstat_tf_occupancy: condition: 0 tf: 1 oc: 0.00062 0.10839 0.26971 ...
rep: X lstat_tf_occupancy: condition: 1 tf: 0 oc: 0.001075 0.656755 0.916395 ...
rep: X lstat_tf_occupancy: condition: 1 tf: 1 oc: 6.5e-05 0.00614 0.01471 ...
rep: X lstat_tf_occupancy: condition: 2 tf: 0 oc: 0.001005 0.664515 0.928995 ...
rep: X lstat_tf_occupancy: condition: 2 tf: 1 oc: 5e-06 0.00042 0.00097 ...
rep: 0 wsum: 8005.96 3.33561e+07 1.9619e+07
```

The first six lines give the occupancy profile for the indicated *trans*-background (condition) and TF. One the occupancy levels for the first three nucleotide positions are shown in this output. The `X` simply indicates it's printing a continuation of the previous replicate as this statistic is a multi-line entry.

The final line shows the sum configuration weight over all configurations: $\sum_{c_k} w(c_k)$. This calculation, as well as the occupancy calculations are exact in the sense that they use a dynamic programming algorithm to sum over the combinatorially large number of configurations. In contrast, expression calculations are estimates based on a sample from the distribution of configurations. The output above is from `examples/lstat3.sh`.

7.10 `--lstat=configs,1`

To simply view a sampling of configurations, one can use the `--lstat=configs,1` argument. Here the number of configurations is given in the `--steps=N` argument. N configurations are sampled under each *trans*-background. The following example is given in `examples/lstat4.sh`.

```
rep: 0 lstat_configs: condition: 0 printing configs
2:33, 2:23, 2:16,
2:33, 2:26, 1:20, 1:14,
2:43, 2:33, 2:26, 2:16, 2:8,
2:43, 2:33, 2:26, 2:8,
rep: X lstat_configs: condition: 1 printing configs
1:44, 1:36, 1:28, 1:20, 1:14, 1:7,
1:43, 1:34, 1:20, 1:14, 1:7,
2:45, 1:36, 1:28, 1:20, 1:14, 1:7,
1:36, 1:28, 1:20, 1:14, 1:7,
rep: X lstat_configs: condition: 2 printing configs
1:36, 1:28, 1:20, 1:14, 1:8,
1:45, 1:36, 1:28, 1:20, 1:14, 1:8,
1:36, 1:28, 1:20, 1:14, 1:7,
1:41, 1:34, 1:28, 1:20, 1:14, 1:7,
```

Each configuration is given as a line, with the tuples having the structure `TF:position`.

7.11 `--neighbors=R`

Here I provide an example of the use of `--neighbors=R` (see above). For this run the arguments `--lstat=expression,1` and `--lstat=sequence,1` are given. Since the enhancer in this example is $L = 20$ bp, there are $L * 3$ neighbors and the original sequence, resulting in $2(3L + 1)$ records for the two statistics:

```
rep: 0 lstat_expression: 0.02123 0.251142 0.33598
rep: 0 aagagtgcgtcggatgccct
rep: 0 lstat_expression: 0.0207075 0.243028 0.33882
rep: 0 aggagtgcgtcggatgccct
rep: 0 lstat_expression: 0.0423825 0.498242 0.623918
rep: 0 acaagtgcgtcggatgccct
rep: 0 lstat_expression: 0.0111825 0.155055 0.36141
rep: 0 acgaatgcgtcggatgccct
rep: 0 lstat_expression: 0.03893 0.446225 0.55018
rep: 0 acgagagcgtcggatgccct
.
.
```

8 Obtaining source code

The entire package is coded in C++ and makes use of the Standard Template Library and object-oriented programming, so it's reasonably accessible if you're looking to modify it or extend it. I will be happy to answer your questions (kbullaughey@gmail.com).

So far I have tested it on Linux (Red Hat Enterprise 5) MacPro (10.5) and MacBook (10.6). It comes with a `configure` script, and so if you're having trouble porting it to another platform or it is not compiling, I can try and generalize the build further. See the `README` that comes with the source distribution for information on compiling the software and for information about its dependencies.

All the source code is available under the MIT License, with the additional request that if you're publishing anything using my software that you cite my original paper [2].

Finally, there are several undocumented uses of this package that are only partially implemented including Orr's block model and a method to do inference based on expression data from multiple enhancers along the lines of the original Segal paper but using parallel tempered simulated annealing. I never came back and finished this feature and there are large chunks of code related to this goal that can be ignored.

References

[1] Segal E, Raveh-Sadka T, Schroeder M, Unnerstall U, Gaul U (2008) Predicting expression patterns from regulatory sequence in drosophila segmentation. Nature 451: 535–40.

[2] Bullaughey K (2011) Changes in selective effects over time facilitate turnover of enhancer sequences. Genetics 187: 567–82.