RFreak–An R Package for Evolutionary Computation

Robin Nunkesser
TU Dortmund

Abstract

RFreak is an R package providing a framework for evolutionary computation. By enwrapping the functionality of an evolutionary algorithm kit written in Java, it offers an easy way to do evolutionary computation in R. In addition, application examples where an evolutionary approach is promising in computational statistics are included and described in this paper. The package is thus further supporting the use of evolutionary computation in computational statistics.

Keywords: R, evolutionary algorithms, evolutionary computation, association study, robust regression.

1. Introduction

In computational statistics like in every field of research involving computers, we are confronted with problems that are very hard to compute exactly or that are not computable at all in reasonable time. A typical approach to tackle these problems is to use heuristics. Evolutionary computation (see e.g. De Jong 2006) is a well established search heuristic in computer science and steadily gaining importance in computational statistics. Examples for the application of evolutionary computation in computational statistics include evolutionary clustering (Hruschka *et al.* 2006), association studies (Nunkesser *et al.* 2007), computation of robust estimators (Morell *et al.* 2008), time series modeling (Baragona *et al.* 2004), and many more.

In this paper, we describe an R (R Development Core Team 2007) interface to the Free Evolutionary Algorithm Kit FrEAK (Briest *et al.* 2004) which also includes numerous extensions to FrEAK. In addition, two extensive application examples from computational statistics are integrated. FrEAK is written in Java and provides an easy-to-use framework to build evolutionary algorithms out of modules. The approach of FrEAK to achieve an easy construction of evolutionary algorithms and how we integrate it in an R package is explained. Furthermore, the application examples from genetic association studies and robust regression are briefly

discussed.

This paper is organized as follows: Section 2 gives a short introduction to the concept of evolutionary computation and how it is reflected in FrEAK. Section 3 introduces the package and illustrates its use by application examples. Section 4 concludes the paper with a discussion.

2. Evolutionary computation and FrEAK

The main idea of evolutionary computation is to mimic Darwinian evolutionary processes in order to obtain efficient search heuristics. Numerous successful applications of evolutionary computation back up its importance as a search heuristic (see e.g. Banzhaf *et al.* 1998 for application examples). Taking a unified view of evolutionary computation like De Jong (2006) allows us to conceive evolutionary computation as a process build up by interchangeable modules. This modular concept is utilized by FrEAK to achieve high reusability.

The main principle of Darwinian evolution is that *populations* of *individuals* evolve through variational inheritance where a concept of *fitness* reflects the ability to survive. Individuals may be characterized by *genotypes* (the genetic makeup) or *phenotypes* (observed qualities). Essential modules of evolutionary algorithms are therefore the fitness function that maps individuals to fitness values, the genotype search space and an optional mapping between genotype and phenotype (called "Fitness Function", "Search Space", and "Genotype Mapper" in FrEAK).

Roughly speaking, solving an optimization problem with an evolutionary algorithm requires the definition of genotypes or phenotypes and a fitness function in such a way, that the individuals represent possible solutions and that the "best" solutions are mapped to the best fitness values and therefore have a higher probability to survive *selection*.

The basic evolutionary process used by evolutionary algorithms is described by the following algorithm.

Algorithm 1 (Basic Evolutionary Algorithm)

1. *Create an initial random population.*
2. *Perform the following steps on the current* generation *of individuals:*
   (a) *Select individuals in the population based on a selection scheme.*
   (b) *Adapt the selected individuals.*
   (c) *Evaluate the fitness value of the adapted individuals.*
   (d) *Select individuals for the next generation according to a selection scheme.*
3. *If the stopping criterion is fulfilled, then output the final population. Otherwise, set the next generation as current and go to step 2.*

The apparent way to adapt this process to specific needs by using interchangeable modules is to visualize the process as a flowchart. In Fig. 1 the flowchart approach FrEAK uses is depicted ("Algorithm Graph"). This reveals some modules of FrEAK not mentioned yet. More precisely, stopping criteria and population initialization are also interchangeable modules of

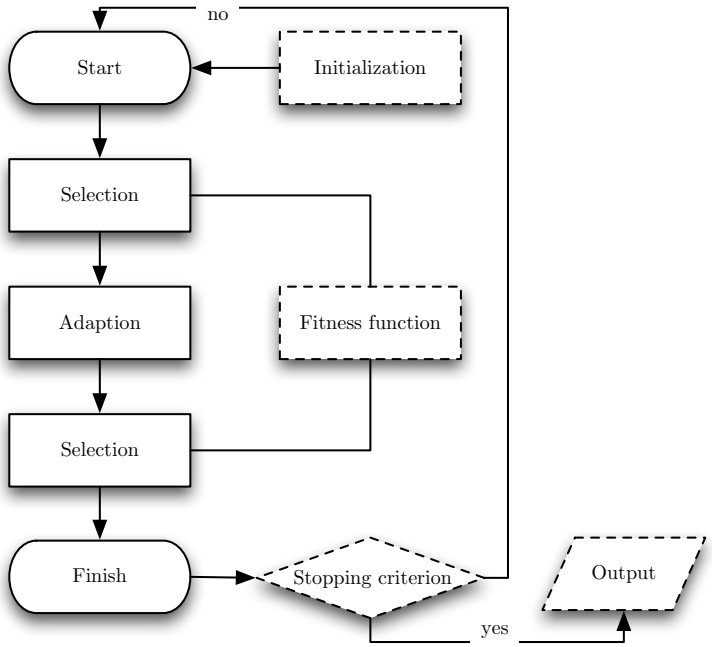

Figure 1: Flowchart containing the modules used in designing an evolutionary algorithm with FrEAK. The dashed modules implicitly influence the process and are no explicit part of a FrEAK algorithm graph.

an evolutionary algorithm (called "Stopping Criteria" and "Initial Population" in FrEAK). To summarize, the interchangeable and reusable modules needed in FrEAK to design an algorithm based on evolutionary computation are: "Search Space", "Fitness Function", "Genotype Mapper", "Algorithm Graph" (additionally consisting of adaption and selection modules), "Stopping Criteria", and "Initial Population".

3. The RFreak Package

The main function of the **RFreak** package is to provide an interface to use FrEAK from R with the help of the **rJava** package (Urbanek 2007).

3.1. Setting up an Evolutionary Algorithm

The function to build an evolutionary algorithm out of modules is called `launchScheduleEditor()`. When executed, it launches FrEAK's schedule editor from R.

One of the best known schoolbook examples of evolutionary algorithms is the $(1+1)$ EA on OneMax described in Algorithm 2, which optimizes the fitness function

$$\textsc{OneMax}(x) := \sum_{i=1}^{n} x_i$$

on genotypes $x \in \{0,1\}^n$.

Algorithm 2 ($(1+1)$ EA)

1. *Choose $x \in \{0,1\}^n$ uniformly at random.*
2. *Define y in the following way. Each bit of x is flipped independently of the other bits with probability $1/n$.*
3. *If $\textsc{OneMax}(y) \geq \textsc{OneMax}(x)$, replace x by y.*
4. *If the stopping criterion is fulfilled, then output the final population. Otherwise go to step 2.*

Obviously, the main merit of this example lies in its amenability to theoretical analysis. Nevertheless, suppose as an example that we would like to run the $(1+1)$ EA 5 times for $n = 30$ and stop each run after 50 generations.

In FrEAK, this setting corresponds to choosing the search space module "Bit String", the fitness function module "OneMax", the algorithm graph $(1+1)$ EA already containing the necessary adaption and selection modules, the stopping criterion "Generation Count", and the population initialization modules "Default Population Model" and "RandomInitialization". This choices determine the algorithm, but apart from that, some simulation specific settings are needed. In "Observers and Views" the observer "Result" and the view "R Return" – which we implemented as part of the interface – are preselected. Most of the other observers and views do not make sense in an R interface and can safely be ignored. The settings for "Batches of Runs" provide an overview of the so far selected modules and the possibility to set up the number of runs and additional batches. Setting up more than one batch also does not make sense in an R interface and is therefore not supported in **RFreak**. After finishing the schedule editor, we obtain a file `schedule.freak` containing the schedule which can be started by `executeSchedule()`. The result is an S4 object of class `FreakReturn`:

```
Result obtained from FrEAK:
  Run Generation Objective value                      Individual
1   1         50              26 10111110111111101111111011111
2   2         50              24 11100111110111110100111111111
3   3         50              22 10011110001110101111111111101
4   4         50              24 11011010111111110011011111111
5   5         49              24 01101011111111111011101111110
```

In addition to this general straight forward interface we added application examples from computational statistics to FrEAK.

3.2. Using RFreak for association studies

Single nucleotide polymorphisms (SNPs) are the most common type of genetic variations. A major goal of genetic association studies is to identify SNPs and SNP interactions that lead to a higher disease risk. Typically, individual SNPs have only a slight to moderate effect and the detection of SNP interactions is considered more important (Garte 2001; Culverhouse *et al.* 2002) but it is of course also computationally more demanding. GPAS (Genetic Programming for Association Studies) (Nunkesser *et al.* 2007) is a genetic programming algorithm

mainly designed to identify high-order interactions in genetic association studies. GPAS can additionally be employed for discrimination.

We integrated GPAS in **RFreak** by implementing new FrEAK modules. Using `launchScheduleEditor()` with the search space "Boolean Functions" offers the possibility to exchange modules and derive new genetic programming algorithms for association studies.

The two algorithms described in Nunkesser *et al.* (2007) are directly accessible from **RFreak** with the function `GPASInteractions` and `GPASDiscrimination`, respectively. An example data set `data.logicfs` taken from the package **logicFS** (Schwender 2006) is also included. A small example like

```
> data(data.logicfs)
> GPASInteractions(cl.logicfs,data.logicfs,runs = 10,occurences=4)
```

returns an S4 object of class `GPAS` (inherits from `FreakReturn`) and saves a GraphViz (Gansner and North 2000) graph in a file `interactions.dot`. Objects of class `GPAS` outwardly contain the same information as objects of class `FreakReturn`. The individuals contained are representations of multi-valued logical expressions in disjunctive normal form. An example individual from the run is

```
(SNP1==3) | ((SNP4==3) & (SNP2==1)) |
   ((SNP3==3) & (SNP5==3) & (SNP6==1))
```

which actually is the true model of `data.logicfs`. When used as a predictor in a case-control study, a patient would be classified as case if the logical expression is true, i.e. if all SNPs in at least one of the three expressions `(SNP1==3)`, `(SNP4==3) & (SNP2==1)`, and `(SNP3==3) & (SNP5==3) & (SNP6==1)` show the genotypes indicated.

The graph `interactions.dot` depicted in Fig. 2 reveals further interesting interactions.

It is also possible to test the individuals on other data samples with the S4 method `predict`.

3.3. Using RFreak for robust regression

Another application example is an evolutionary algorithm for robust regression methods (Morell *et al.* 2008), where we concentrate on least trimmed of squares (LTS) regression (Rousseeuw 1984). Bernholt (2005) showed, that computing robust regression estimators fulfilling the exact-fit-property is NP-hard. Therefore, under the widely believed assumption that P≠NP, we cannot hope for exact algorithms with practical running times for large data sets. The implemented evolutionary algorithm is better suited to overcome local optima than other existing algorithms for this problem.

Consider the linear model $y = x\beta + e$. For n data points $(y_1, x_1), \ldots, (y_n, x_n)$ and a parameter h, the objective of the LTS estimator is to minimize

$$\sum_{i=1}^{h} (r^2)_{i:n}$$

where $(r^2)_{1:n} \leq \cdots \leq (r^2)_{n:n}$ are the ordered squared residuals $y_i - x_i\beta$.

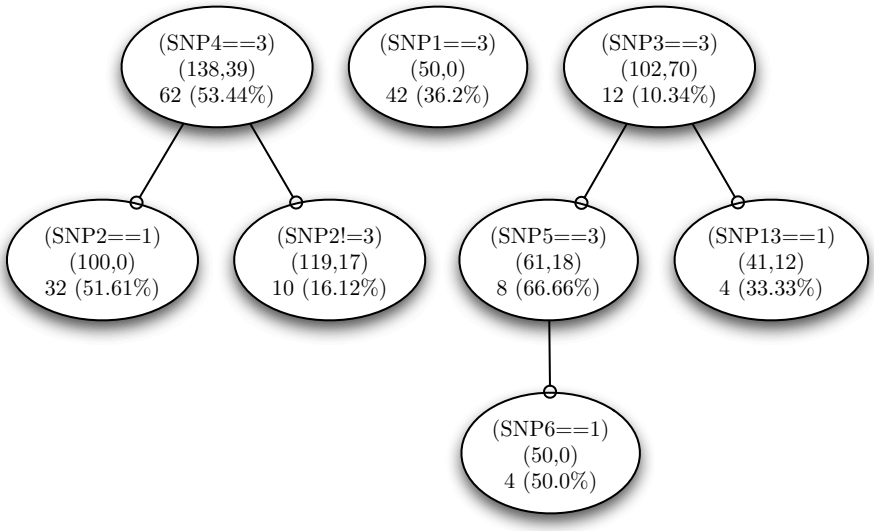

Figure 2: Tree visualization of the individuals. Each path from the root to an inner node or leaf represents an interaction occurring in the final population. The first line in each node contains the literal represented by the node. The second line shows the number of cases and controls explained by the corresponding interaction. The third line consists of the number of monomials containing the corresponding interaction and the percentage of monomials consisting of the ancestral interaction that also contain the literal represented.

Like GPAS, the LTS evolutionary algorithm may be executed with the help of `launchScheduleEditor()` (search space "Subsets of Points", fitness function "LTS", mapper "Point Set to Bit String") or with a specific function. This function is called `LTSevol`. The example

```
> data(stackloss)
> LTSevol(stackloss[,4],stackloss[,1:3],adjust=TRUE)

Result obtained from FrEAK:
  Run Generation Objective value            Individual
1   1        983       -2.932391 00000010001000010100

Chosen subset:
 [1]  7 17  6 11 19  5 12  9 18 10  8 15 16

Coefficients:
[1] -37.32332647   0.74092106   0.39152672   0.01113454

Criterion:
[1] 2.932391
```

returns an object of class `ltsEA` (inherits from `FreakReturn`). Objects of class `ltsEA` contain some additional information. The "Chosen subset" contains the h points with the smallest

squared residuals. Apart from this subset, the coefficients of the fitted hyperplane and the objective value are also contained.

4. Discussion

In this paper we have illustrated the use of **RFreak** for evolutionary computation in R. To our knowledge, the package is the first publicly available software providing an evolutionary computation framework for R. The purpose of the interface is manifold. Analyzing evolutionary algorithms build with FrEAK empirically is a lot easier when FrEAK is integrated in R. We have also shown and implemented application examples from computational statistics that are very useful on their own and show how easy it is to extend FrEAK.

As an example, all that is needed to extend the robust regression facilities by further estimators are new fitness functions. For many other problems from computational statistics only an additional new genotype phenotype mapper or a new search space is needed. Thus, the reusability of written code is very high because of the modular layout of FrEAK.

Evolutionary computation already gained considerable importance in computational statistics and hopefully this easy-to-use framework will support that trend.

Furthermore, we are currently working on integrating more application examples and therefore more reusable modules to **RFreak**.

Sources, binaries, and documentation of **RFreak** and the extensions to FrEAK are available for download under the GNU Public License from the Comprehensive R Archive Network `http://cran.r-project.org/` and `http://ls2-www.cs.tu-dortmund.de/~nunkesser/#Software`, respectively.

Acknowledgements

Financial support of the Deutsche Forschungsgemeinschaft (SFB 475, "Reduction of Complexity in Multivariate Data Structures") is gratefully acknowledged. The author would also like to thank Holger Schwender, Roland Friend, and Oliver Morell for helpful discussions and Thorsten Bernholt, Melanie Schmidt, Sebastian Ruthe, and Dominic Siedhoff for their help in implementing FrEAK extensions.

References

Banzhaf W, Francone FD, Keller RE, Nordin P (1998). *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 1-55860-510-X.

Baragona R, Battaglia F, Cucina D (2004). "Fitting Piecewise Linear Threshold Autoregressive Models by Means of Genetic Algorithms." *Computational Statistics & Data Analysis*, **47**(2), 277–295.

Bernholt T (2005). "Robust Estimators are Hard to Compute." *Technical Report 52/2005*, SFB 475, Universität Dortmund.

Briest P, Brockhoff D, Degener B, Englert M, Gunia C, Heering O, Jansen T, Leifhelm M, Plociennik K, Röglin H, Schweer A, Sudholt D, Tannenbaum S, Wegener I (2004). *FrEAK: Free Evolutionary Algorithm Kit*. Version 0.2, URL `http://sourceforge.net/projects/freak427/`.

Culverhouse R, Suarez BK, Lin J, Reich T (2002). "A Perspective on Epistasis: Limits of Models Displaying No Main Effect." *Am. J. Hum. Genet.*, **70**, 461–471.

De Jong KA (2006). *Evolutionary Computation: A Unified Approach*. MIT Press.

Gansner ER, North SC (2000). "An Open Graph Visualization System and its Applications to Software Engineering." *Softw. Pract. Exper.*, **30**(11), 1203–1233. ISSN 0038-0644.

Garte S (2001). "Metabolic Susceptibility Genes as Cancer Risk Factors: Time for a Reassessment?" *Cancer Epidemiol. Biomarkers Prev.*, **10**, 1233–1237.

Hruschka ER, Campello RJ, de Castro LN (2006). "Evolving Clusters in Gene-Expression Data." *Information Sciences*, **176**(13), 1898–1927.

Morell O (2006). *Vergleich von Algorithmen für die Least-Trimmed-Squares-Schätzung mittels Statistischer Versuchsplanung*. Diploma thesis, Universität Dortmund.

Morell O, Bernholt T, Fried R, Kunert J, Nunkesser R (2008). "An Evolutionary Algorithm for LTS-Regression: A Comparative Study." In "Proceedings of Compstat 2008," Accepted.

Nunkesser R, Bernholt T, Schwender H, Ickstadt K, Wegener I (2007). "Detecting High-Order Interactions of Single Nucleotide Polymorphisms using Genetic Programming." *Bioinformatics*, **23**(24), 3280–3288.

R Development Core Team (2007). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Rousseeuw PJ (1984). "Least Median of Squares Regression." *Journal of the American Statistical Association*, **79**, 871–880.

Schwender H (2006). *logicfs: Identification of SNP Interactions*. R package version 1.4.0.

Urbanek S (2007). *rJava: Low-level R to Java interface*. R package version 0.5-1, URL http://www.rforge.net/rJava/.