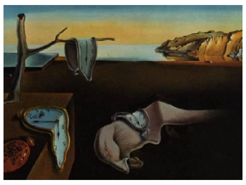


Journal of Statistical Software

*May 2010, Volume 34, Issue 12.* *http://www.jstatsoft.org/*

# glmulti: An R Package for Easy Automated Model Selection with (Generalized) Linear Models

**Vincent Calcagno**
McGill University

**Claire de Mazancourt**
McGill University

**Abstract**

We introduce **glmulti**, an R package for automated model selection and multi-model inference with `glm` and related functions. From a list of explanatory variables, the provided function `glmulti` builds all possible unique models involving these variables and, optionally, their pairwise interactions. Restrictions can be specified for candidate models, by excluding specific terms, enforcing marginality, or controlling model complexity. Models are fitted with standard R functions like `glm`. The $n$ best models and their support (e.g., (Q)AIC, (Q)AICc, or BIC) are returned, allowing model selection and multi-model inference through standard R functions. The package is optimized for large candidate sets by avoiding memory limitation, facilitating parallelization and providing, in addition to exhaustive screening, a compiled genetic algorithm method. This article briefly presents the statistical framework and introduces the package, with applications to simulated and real data.

*Keywords*: AIC, BIC, `step`, `glm`, **rJava**, variable selection, genetic algorithm, marginality.

## 1. Introduction

### 1.1. Generalized linear models

Generalized linear models (GLMs) provide a flexible framework to describe how a dependent variable can be explained by a range of explanatory variables (*predictors*). The dependent variable can be continuous or discrete (integer valued), and the explanatory variables can be either quantitative (*covariates*) or categorical (*factors*)[1]. The model is assumed to have linear effects on some transformation of the dependent variable, defined by the link function, and the error distribution can have various shapes, such as Gaussian, binomial or Poisson. The

[1]Note that this use of factor is more restrictive than the technical meaning of factor in R.

GLM framework encompasses many situations, like ANOVAs, multiple regressions, or logistic regression. Unsurprisingly GLMs are widely used in several fields of science (Venables and Ripley 1997; Grafen and Hails 2002).

The R language includes a built-in function to fit GLMs: `glm` (R Development Core Team 2010). This function takes as its main argument the specification of a model as a formula object, e.g., `y ~ f1 + c1`. Here y is the dependent variable, `f1` is a factor, and `c1` is a covariate. This formula refers to a specific model that `glm` will fit to the data. The formula can specify more complicated models, in which for instance some predictors have interactive effects on the dependent variable. For example, if one wants to fit a different slope for each level of the factor `f1`, in addition to a different intercept, one can use the formula `y ~ f1 + c1 + f1:c1`. The new term in the equation refers to the pairwise interaction between `f1` and `c1`, and uses the symbol `:`. Several such symbols exist, allowing to specify in detail what model should be fitted to the data, and usually one model can be expressed by several alternative formulae, so that the formula notation is redundant (Venables and Ripley 1997). After the fitting procedure, `glm` returns (up to a constant) the maximum likelihood of the model, together with the associated parameter estimates, and a range of other indicators. Other fitting functions work following the same scheme.

## 1.2. Stepwise model selection and beyond

Fitting a single model is not satisfactory in all circumstances. In particular, this assumes that the model used is true or at least optimal in some sense. Hence the resulting inference is conditional on this model (Buckland, Burnham, and Augustin 1997).

In many cases, one wants to decide, among all the terms that have been included in the model formula, which are *important* or *relevant* in some way to describe the dependent variable, in other words, which one should be retained, and which one could be dropped. Starting from a "full" model (containing all terms the investigator think could be important), we can actually define a broad family of models: all those models that include some of the potential terms. All of them are nested within the full model (Figure 1). The question is then to determine which of these models should be retained. A special case of this general problem is variable selection in the multiple regression framework (Miller 2002).

The usual method to select a model is stepwise. Typically one fits the full model and looks for terms that are not statistically significant, i.e., whose removal does not significantly reduce the fit of the model. One then removes these non-significant terms from the formula, or the least-significant one, thus obtaining a new "simplified" model. The procedure can be repeated until all effects in the formula are found significant. This approach is often called "backward simplification". A similar strategy is to start from the simplest model and to sequentially add the most significant effects (a "forward selection" approach).

Determining whether the removal or addition of a given term at each test is significant can be done in several ways. Hypothesis test tools, such as the $t$ test, $F$ test, or LR test, involve specifying a significance threshold for the $p$ values (e.g., 5%). Since the number of tests is typically high, this poses the problem of choosing a relevant significance level (Harrell 2001). Another approach is to use information criteria (IC) to compare the models obtained in the course of the simplification/complexification scheme. Several such criteria have been used, e.g., Mallow's Cp for multiple regression or the Akaike information criterion (AIC) (Venables and Ripley 1997; Miller 2002).

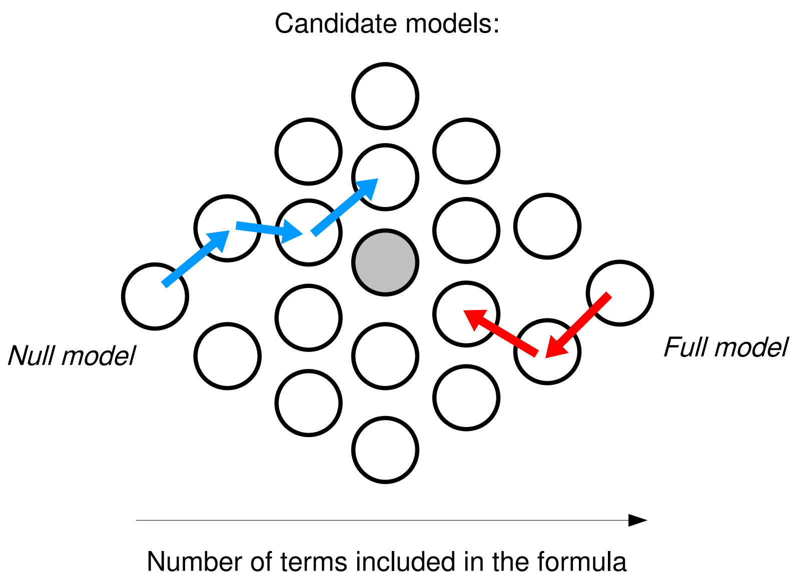


Figure 1: A schematic representation of the candidate set of models for a hypothetical "full model". There is one null model (left), one full model (right), and a range of models with some terms but not all of them (in between). Arrows represent stepwise model selection procedures: backward (red) and forward (blue). In this case both approaches would not converge to the same model, and it can be that none of them converges to the actual model that is optimal in terms of the IC used (grey circle).

Iterative procedures are dependent on the starting point and on the stopping rules though. For example backward and forward approaches are not generally expected to converge to the same model (Grafen and Hails 2002; Venables and Ripley 1997). Several model selection techniques have been developed to avoid these caveats, such as shrinkage regression methods (e.g., LASSO or LAR) or Bayesian methods.

One of these techniques consists in giving a weight to alternative models, based on some information criterion, and using all of them for inference (Burnham and Anderson 2002). Information criteria can indeed be used to compare the support for several models at a time, so using them to compare subsets of models in the course of a stepwise process does not use their full capacity. It is similar to using a threshold of two for significance with LRT, bringing back arbitrary cut-off values, whereas the appeal of IC comes in part from the possibility to avoid this kind of arbitrariness (Anderson 2008).

A full IC-based approach is to compare all candidate models and rank them based on their IC value. First, this ensures that the "best" model (according to the IC) is identified, whereas stepwise explorations do not (Figure 1). Second, and most importantly, this method allows to assess model-selection uncertainty or to perform multi-model inference, rather than using one and only one best model (Buckland *et al.* 1997; Burnham and Anderson 2002; Johnson and Omland 2004).

## 1.3. Automated model selection with R

It is increasingly common to deal with many candidate predictors, often with modest a priori information about their potential relevance (Ripley 2003). This commonly happens in exploratory analyses, or in experimental studies addressing complex systems. In such situations the number of candidate models can be fairly large, making manually browsing through them in an iterative complexification/simplification process tedious and inefficient (Ripley 2003). This calls for automated model selection procedures (Venables and Ripley 1997).

Several R packages have been created in the past years to carry out automated variable selection or model selection. Most of them address the issue of subset selection in multiple regression, i.e., choosing predictors in linear regression. The criterion for model ranking is typically the adjusted R squared or Mallow's Cp, and best subsets (i.e., combinations of predictors) are returned for every size. The search through the different models can be either exhaustive, as in **meifly** (Wickham 2009), or is optimized to some extent. For instance, **leaps** (Lumley and Miller 2009) uses a branch-and-bound algorithm to identify best subsets more quickly, while **subselect** (Orestes Cerdeira, Duarte Silva, Cadima, and Minhoto 2009) implements a simulated-annealing algorithm.

When it comes to model selection with GLMs, R users have fewer options. One is the model-selection oriented function `step`, now builtin in package **stats** (see Venables and Ripley 1997). This function uses a stepwise selection procedure, based on AIC (or an approximation of it). It is pretty fast (since it explores only a fraction of the candidate models) and can deal with complicated model formulas, including interaction terms. It nonetheless suffers from the above-mentioned drawbacks of stepwise procedures, and does not allow the assessment of model selection uncertainty. Also, it is not fully automated in the sense that the user is usually expected to have several interactions with the function and feedback to it.

A radically different option is the **glmpath** package (Park and Hastie 2007), which implements an algorithm to find paths for $L_1$-regularized GLMs (a shrinkage method, Miller 2002). This specific approach does not handle model formulas though (i.e., interactions between predictors and the associated complications) and, again, does not allow multi-model inference.

More all-purpose packages have also been contributed that can carry out IC-based model selection with GLMs: packages **MuMIn** (Bartoń 2009) and **bestglm** (McLeod and Xu 2009). Both offer to fit all possible models for a combination of predictors and rank models according to some IC. **MuMIn** additionally provides some facilities to carry out multi-model inference from the best models that have been returned. Those two packages remain limited in the way they explore the set of candidate models though. **MuMIn** can only perform an exhaustive screen of the models, with no optimization (in terms of speed or memory usage). **bestglm** can take advantage of the above-mentioned **leaps** package to optimize the search for best models, but because of this reliance on **leaps** this is only available for linear models and in the absence of factors with more than two levels. Otherwise a simple exhaustive screen is the only possibility.

Also, these two packages do not handle interactions among variables properly. Whereas **MuMIn** can handle formulas, it treats interactions as standard variables, which raises several issues (see below). **bestglm** takes as argument a matrix of predictors, so there is no notion of a formula or an interaction.

Here we introduce an alternative, implemented for R in our package **glmulti**. The goal of **glmulti** is to make the full IC-based model selection approach sketched above available transparently to R users. We have applied this approach before (Calcagno, Thomas, and Bourguet 2007; Prevost, Calcagno, Renoult, Heppleston, and Debruille 2010) and it proved insightful.

**glmulti** provides a general wrapper for `glm` and related functions. It is intended to be flexible so that support for any such function, and any Information criterion, is easy to implement. It supports formulas and interactions between variables. An obvious difficulty with this kind of approach is that the number of models to be considered easily becomes prohibitive. For this reason **glmulti** is optimized in order to minimize memory requirements and computation time, making it possible to manage very large candidate sets.

The ouput of a **glmulti** analysis is an object containing the confidence set of models (i.e., the $n$ best models) and their support. Standard R regression methods like `summary`, `coef` or `plot` can be used on the object in order to make model selection and multi-model inference a straightforward and automated task.

In the following we present the structure and functioning of the package, with a few applications to simulated and real data.

## 2. What is package glmulti?

### 2.1. Overview: What glmulti does

To summarize, **glmulti** provides a function, `glmulti`, to be used instead of `glm`, or any similar function (e.g., `lm`). In the following we will assume the function is `glm`.

`glmulti` is essentially a wrapper for `glm`: it generates all possible model formulas (from the specified effects and given some constraints), fits them with `glm`, and returns the best models (based on some information criterion) in an object of class `glmulti`. From this object, one can directly select the "best" model, build a confidence set of models, and produce model-averaged parameter estimates or predictions.

To achieve this goal, **glmulti** clearly has to generate all possible model formulas that involve some of the specified effects. More exactly, the user specifies which *main effects* are to be considered, and the program will generate all models including some of these effects and, optionally, their pairwise interactions. Three-way interactions and higher are not included in the current version. As shown in the introduction, pairwise interactions are usually not considered by selection tools, possibly because the number of models literally explodes with interactions (Grafen and Hails 2002; Orestes Cerdeira *et al.* 2009). Indeed, with main effects only the number of models to be compared increases as $2^n$, whereas with pairwise interactions too the increase is close to $2^{n^2}$.

As mentioned before, the formula notation for models is redundant. When some factors are included in interactions, many formulas actually specify the same model, and as the number of predictors increase, redundant formulas become very common.

The basic part of the package is therefore an enumerator that returns all possible non redundant formulas, which avoids fitting the same model many times. The enumerator also allows the user to specify some constraints on the formulas. Such constraints are: (1) particular effects that should *not* be considered (e.g., "consider all formulas except those containing the interaction `f3:c2`")[2], (2) minimum and maximum numbers of terms in the formula, or, most importantly, (3) minimal and maximum numbers of estimated parameters (model complexity).

The default method in `glmulti` (`method = "h"`) does this: fitting all candidate models. When more than five or six predictors are considered, together with interactions, the number of candidate models is so high that exhaustive search becomes unmanageable. `glmulti` returns the number of candidate models when used with `method = "d"` (see examples). Even with recent CPUs, computation time can be a concern. One solution is to reduce the number of candidate models by specifying constraints. Another option is to split the task across several cores/computers (arguments `chunk` and `chunks` allow this to be done easily), but even so computation time can be prohibitive.

For this reason **glmulti** provides an alternative to the "fit them all" strategy. It features a genetic algorithm (GA) method, used when argument `method` is set to `"g"`. By exploring only a subset of all possible models, randomly but with a bias towards better models thanks to selection, it makes computation much faster. Genetic algorithms are very efficient at exploring highly discrete state spaces, and have been used successfully in related optimisation problems (e.g., Orestes Cerdeira *et al.* 2009; Trevino and Falciani 2006). Unlike most of them, our genetic algorithm includes an immigration operator, which can improve convergence for complicated problems (Yang 2004; see below). This constitutes the most complicated part of the program, and its efficiency is studied in the examples section.

## 2.2. How glmulti is built

The R function `glmulti` is a simple front-end that calls other functions: the builtin `glm` (or similar) to fit models, and some background Java classes provided with the package (collected in the glmulti.jar archive). The dialogue between `glmulti` and the Java classes is handled by the **rJava** package (Urbanek 2009), which is hence required for **glmulti** to work. The code is portable and need not be compiled for each platform, provided the user has a Java Runtime Environment (JRE) installed.

The Java classes take care of generating the formulas that `glmulti` will feed to the fitting function. The fitting procedure is therefore exactly the same as one would use when calling `glm` or so, and the same options can be used (passed through `glmulti`).

**glmulti** is written using the S4 object specification scheme. It provides a class definition (`glmulti`), several S4 function definitions (including `glmulti`), and S3 standard functions for `glmulti` objects (such as `summary.glmulti`).

There are three Java classes in the package[3]:

1. `ModelGenerator`: The main class, used for interaction with R.

2. `GLMModel`: A class describing a particular model as an object (this is used with the genetic algorithm only). It implements the behavior of a model: producing an offspring model with mutation, recombining with another model, computing its complexity, etc.

[2]It is actually straightforward to allow for arbitrary constraints on formulas, as regular expressions, by writing in R a wrapper of the fitting function that returns very bad fit when passed a forbidden model. This is extremely flexible, although not optimal for computation time.

[3]All code written by Vincent Calcagno and can be used freelyaccording to the General Public License (GPL), version 3. You can find the source code, withlots of comments, at `http://redpath-staff.mcgill.ca/calcagno/stats.html`.

3. `Resumator`: An auxiliary class used to resume an interrupted Genetic Algorithm from the backup files that the main program writes at run time (serialized Java objects). It simply deserializes the objects.

### 2.3. How glmulti works

This section explains in greater details how to use **glmulti** and what the program does.

*How the models are fitted to data*

`glmulti` does not fit anything: it justs produces model formulas, and these are passed to a R fitting function; by default this function is `glm`, but the user can actually provide any other function that behaves the same way, for instance `lm`, through the argument `fitfunc`. Therefore, in terms of model fitting using `glmulti` instead of `glm` or `lm` does not change anything. Additional arguments can be passed to the fitting function, e.g., `maxit = 30` or `family = binomial(link = "probit")`, through `glmulti`.

Support for virtually any such fitting function can be achieved by defining a S4 method for the generic function `getfit`, used to access the coefficients from a fitted model (there is no standard way). The S3 `logLik` function (used to access likelihood) should also be available for the corresponding objects.

*Specifying what models will be considered*

`glmulti` has to be provided the building blocks (terms) from which to build model formulas. In other words, you have to specify what your candidate set of models will be. This can be done in several ways. The easier way is to pass a formula (i.e., `y ~ a + c + z + a:z`) as first argument, in which case all the terms found in the formula (up to pairwise interactions, higher order interactions are ignored) will be used as building blocks. Equivalently, a character string describing the formula can be used. Note that if the `level` argument is set to `1`, only the first order terms (i.e., main effects) will be extracted from the formula.

Predictors should either be actual variables defined in the environment of the formula, or correspond to a variable contained in a data frame, to be passed as `data` argument. This is the standard interface of regression functions in R.

Another option is to pass a fitted object as first argument. The formula, data, and fitting function (plus additional arguments to the fitting function if any) are then extracted from the object.

Lastly, the first argument can be the name of your dependent variable, here, `"y"`. The second argument is then a list of the predictors that should be used in the formulas, here `c("a", "c", "z")`. This list should contain *main effects* only. All variable names should be passed as strings (hence the quotes), corresponding to columns in the required argument `data` (the data frame containing your data). In this case, pairwise interactions will be automatically generated if the parameter `level` is set to 2 (the default). All pairwise interactions will be used by default. You can specify which terms (main effects and interactions) should be excluded using the `exclude` argument, for instance `exclude = c("a:c", "c:z")`. The undesired terms are given as strings, and the order of the predictors has no effect, i.e., `exclude = c("z:c", "c:a")` would be equivalent. By default, an intercept is included in all models (i.e., the model just above is explicitly `y ~ 1 + a + z + c`, where `1` denotes the intercept). It is possible to force the intercept to be omitted (equivalent to writing formulas like `y ~ a + z + c - 1`. This is done by setting the argument `intercept` to `FALSE`. Note that unless a model does not include any factor (as main effect and/or in an interaction), the intercept will have no effect on the fit.

To summarize, we could indifferently call (among others):

```
R> output <- glmulti(y ~ -1 + c*a*z - c:z - a:c, data = myDataFrame,
+    maxit = 30)
```

or

```
R> mod <- glm(y ~ -1 + c + a + z + a:z, data = myDataFrame, maxit = 30)
R> output <- glmulti(mod)
```

or

```
R> output <- glmulti("y", c("a", "c", "z"), exclude = c("a:c", "c:z"),
+    data = myDataFrame, intercept = FALSE, maxit = 30)
```

This will call `glm` with the full model:

```
R> glm(y ~ -1 + a + c + z + a:z, data = myDataFrame, maxit = 30)
```

as well as simpler models, for instance

```
R> glm(y ~ -1 + a + c, data = myDataFrame, maxit = 30)
```

For performance, the Java classes encode formulas as compact bit strings. Currently two integers (32 bits each) are used for main effects, and two long integers (128 bits) are used for each category of interaction terms (factor:factor, covariate:covariate, and factor:covariate), to encode models. This means that there can be at most 32 factors and 32 covariates, and, if including interactions, at most 128 interactions of each category. The latter constraint necessitates that, if $x$ is the number of factors and $y$ the number of covariates:

$$\begin{aligned} x &< 16 \\ y &< 16 \\ xy &< 128 \end{aligned}$$

### *Redundancies in formulas*

So far we did not specify whether the predictors were factors (i.e., categorical) or covariates (i.e., continuous). If all predictors are covariates, 4 terms that can be included or not in the formulas (`a`, `c`, `z` and `a:z`), resulting in $2^4 = 16$ candidate models that `glmulti` will fit. As soon as `a` or `z` are factors though, some formulas are redundant.

As a rule, when an interaction between two factors is included in a model, then adding or not these factors as main effects does not change the model. We will say that these main effects are *implied* by the interaction term. Therefore, if `a` and `z` were factors, the following formulas would be redundant

```
y ~ a + z + c + a:z
y ~ a + c + a:z
y ~ z + c + a:z
y ~ c + a:z
```

They all refer to exactly the same model. Therefore, in our example, the number of models drops from 16 to 10 when a and z are factors.

Another rule is that an interaction term between a factor and a covariate implies the covariate as a main effect. Hence, if `a` were a factor and `z` a covariate, the following formulas would be redundant:

```
y ~ z + c + a:z
y ~ c + a:z
```

but adding a would make a difference. In this case, we have 12 models instead of 16. `glmulti` will treat all redundant formulas as equivalent, and will retain only one to be fitted: the most complicated, i.e., the one with all implied effects showed explicitly. Generally, this implies that the number of candidate models increases slower with the number of predictors when many of these predictors are factors (Figure 2A).

### *Setting limits on model complexity*

Setting exclusions of terms is an efficient way to refine the definition of candidate models. Candidate models should all be considered likely, or at least possible. If some terms are a priori meaningless, then they should be excluded from the candidate models. This step is an important one Burnham and Anderson (2002); Anderson (2008).

Constraints on the number of terms (parameters `minsize` and `maxsize`) that candidate formulas should have, or alternatively on the complexity of the candidate models (parameters `minK` and `maxK`), can also be specified. By default there is no constraint. Note that complexity is not a simple function of the number of terms: when there are factors, it strongly depends on the number of levels of these factors (Figure 2B). Note also that the error distribution used in the GLM usually consumes one or more degrees of freedom, i.e., increases model complexity. For instance, the default Gaussian family estimates the variance of the error, resulting in estimating one parameter more than one may expect from the model formula.

### *Considerations of functional marginality*

The principle of "marginality", advocated in textbooks like Venables and Ripley (1997) and Grafen and Hails (2002), states that when including an interaction term, the corresponding main effects should always be included, be they factors or covariates. This rule is a mathematical necessity when both effects are factors, as explained above: in such a case the different formulas would all specify the same mathematical model (they are redundant). But, when at least one of the predictors is a covariate, it leads to ignoring models that are mathematically different. The general principle of marginality is therefore not a mathematical necessity, but an a priori choice: those models including interactions but not the main effects can sometimes be considered irrelevant (Nelder 2000; Venables and Ripley 1997; Cederkvist, Aastvelt, and Naes 2007).

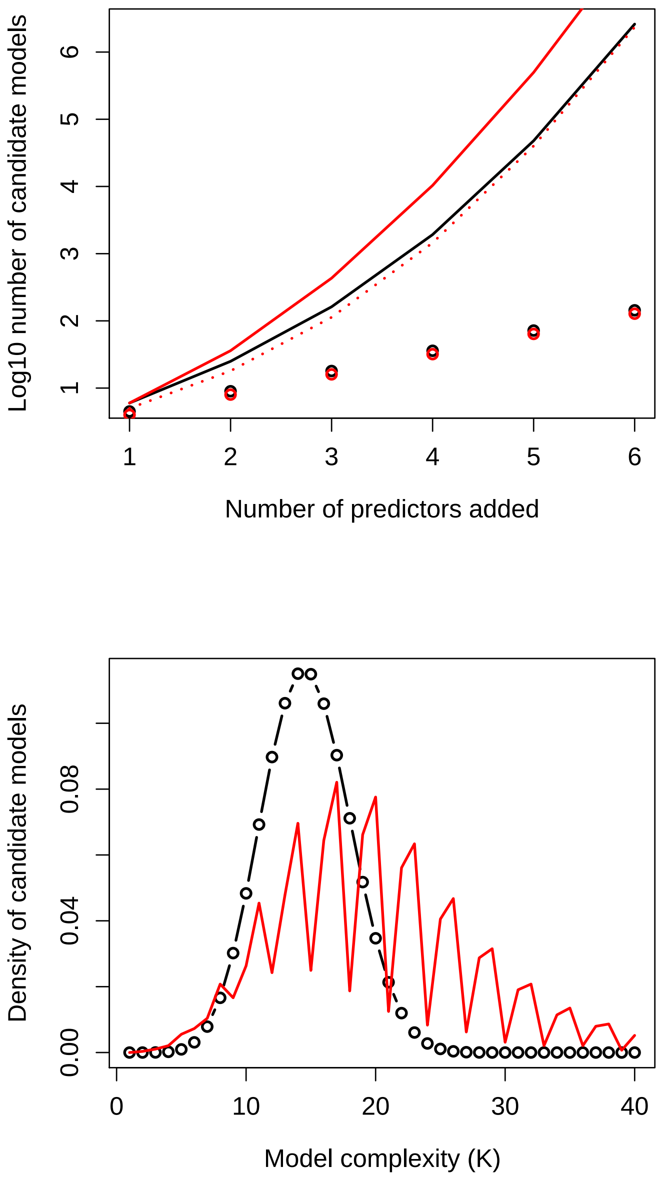


Figure 2: Candidate models. Top panel (A): Number of candidate models as the function of the number of predictors. Black: one factor plus increasing number of covariates (x axis), Red: one covariate plus increasing number of factors (x axis). Solid lines: with pairwise interactions, Dotted lines: with pairwise interactions and marginality, Dots: only main effects (black dots have been slightly moved to make then visible). Lower panel (B): the distribution of model complexity among candidate models. Black: with one factor and 5 covariates, Red: with three factors of three levels and two covariates (see main text).

Applying this rule amounts to setting an additional constraint on the candidate set of models; it incidentally reduces its size, making model selection easier (Figure 2A). One can use the marginality constraint by setting the argument `marginality` to `TRUE`.

### *What information criterion will be used?*

By default, `glmulti` uses AICc to compare models. Other options are available: `crit = aic` to use the original expression of AIC, `crit = bic` to use the Bayesian Information Criterion (or Schwartz Criterion), and `crit = qaic / crit = qaicc` to use QAIC/QAICc (for overdispersed data). These criteria are calculated from applying the function `logLik` on the `glm` object after fitting. The number of observations (used for AICc and BIC) is taken to be the number of observations actually used by `glm`, i.e., omitted observations (because of missing values) are not counted in (see functions `aicc` and `bic` provided in the package). Any function accepting a fitted model as argument and returning a numerical value can be used and passed as argument `fitfunc`.

### *Fitting all models*

Calling `glmulti` with `method = "h"` produces all non-redundant formulas (as explained above), and fits them with `glm`, which allows to compute the specified IC. The argument `confsetsize` indicates how many models should be returned: the `confsetsize` best models will have been identified when `glmulti` returns.

Technically, the R function `glmulti` calls the function `nextModel` of class `ModelGenerator`. This function returns the next non-redundant formula that satisfies the constraints. `glmulti` fits the model, computes its IC, and determines whether the model should be retained in the confidence set. Then it calls `nextModel` again until all models have been fitted.

Every 50 models, glmulti prints a short report of the current confidence set, including the IC of the best model so far and its formula (turned off using `report = FALSE`). By default it also plots the current "IC profile" (i.e., a plot of the ranked IC of the models in the confidence set), so that one can visualize the improvements. This can be turned off by setting argument `plotty` to `FALSE`.

`glmulti` returns an object of class `glmulti`. This S4 object contains a number of slots, including the formulas and IC values of the best models.

There are several ways to access the content of the object. A graphical summary can be obtained by using `plot`. A brief report is available through `print` and a more detailed account is returned by `summary`. Model-averaging and multi-model inference can be performed through the standard functions `coef` and `predict`. The object can be printed as a data frame using `write`. It is also possible to write it compactly as a R object on the disk (using `write` and appending `|object` to the `file` argument).

All these possibilities will be illustrated in the examples section.

### *The genetic algorithm approach*

Besides the "brute force" approach presented above, `glmulti` can explore the candidate set with a genetic algorithm (GA), which can readily find the best models without fitting all possible models. This approach is recommended when the number of candidate models make the previous method unapplicable. It is called when `method = "g"` is used.

Formulas are encoded as strings of zeros and ones, indicating which terms are present and which are absent. As mentioned earlier, this string is encoded in Java as two integers and six long integers, hence containing 448 bits. This string is the "chromosome" that will undergo adaptive evolution, and each bit in this string is a "locus".

The genetic algorithm maintains a population of models, whose size is determined by argument `popsize`. Every generation, models are fitted in R and IC values are used to compute the fitness of each, $w$. The fitness of the $i$th model is computed as:

$$w_i = \exp(-(IC_i - \mathit{ICbest}))$$

where *ICbest* is the best IC in the current population of models. Higher IC means lower fitness. For performance, a finite list of the most recently fitted models is maintained acts as a buffer to minimize the number of calls to R. Since fitting will often be the time-limiting step, models are fitted only if not found in the buffer (or in the current confidence set), and duplicates are only fitted once.

Models in the next generation are produced in three different ways: (1) asexual reproduction, (2) sexual reproduction, and (3) immigration. The relative importance of these processes are governed by arguments `sexrate` and `imm`, being the proportion of sexual reproduction and immigration, respectively.

A model produced by asexual reproduction is simply a copy of its parent (drawn randomly in the previous generation with a probability proportional to fitness) with the state of some loci changed by mutation. Each locus will be changed with a probability defined by argument `mutrate`. A model produced by sexual reproduction has two parents, each drawn as previously, and whose "chromosomes" are recombined. In addition to recombination, each locus can mutate as before.

A model produced by immigration has the state of each locus assigned randomly (zero or one with equal probability). Immigration, and to a lesser extent sexual reproduction, produce big changes in the structure of the models that will be fitted. This avoids being stuck around a local optimum. The immigration operator is not as standard in genetic algorithms as mutation and recombination are, but it can improve convergence in many cases (Yang 2004), and we have observed this in our own analyses (see examples section).

The last set of arguments to specify when using the GA are the stopping rules: they will tell the GA when it should stop looking for better models. Three arguments define these rules: `deltaB`, `deltaM` and `conseq`. The first two are target improvements in the best IC and the average IC, respectively. If the observed improvements are below these targets, then the GA is declared not to have significantly improved. This is checked every 20 generations. If it does not significantly improve `conseq` consecutive times, then it is declared to have converged and it returns.

As for the exhaustive screening method, graphical and printed reports of the progress can be displayed at run time. They can be turned off through arguments `plotty` and `report`, respectively.

The object returned is the same as with `method = "h"`, with additional values in the `params` slot: the elapsed time, the number of generations to reach convergence, and the dynamics of convergence (best and average IC in the confidence set).

When running the GA also writes two small files, with extensions `.modgenback` and `.modsback`. These files contain serialized Java objects that can be used to restore the analysis, should it be interrupted for any reason, or to continue the exploration with different parameter values. To do this one has to call `glmulti` with argument `method = "r"`: the GA will be restored from the files (their name can be specified with argument `recov` if it differs from `name`).

### 2.4. Availability

The **glmulti** package (version 0.6-1) is available from the Comprehensive R Archive Netwok at `http://CRAN.R-project.org/package=glmulti`.

## 3. Examples

This section provides examples of using **glmulti** with the corresponding R code. In particular, the second part uses simulated data with known structure in order to validate the approach and establishes the convergence properties of the genetic algorithm. The third part applies `glmulti` to real data (the standard *birth weight* dataset) and shows how the results compare to a previous comparable analysis (Venables and Ripley 1997). All the R code used in this section (and used to make the figures of this article) can be found in the accompanying example R code.

### 3.1. Growth of number of candidate models with number of variables

The number of models increases very fast with the number of predictors, though a little bit slower when there are many factors, as explained in the section "Redundancies in formulas" (in Section 2.3). Here we use `method = "d"` to show this.

The number of levels factors have does not affect the number of candidate models, only their complexity. We use a data frame `dod`, containing as a first column a dummy response variable, the next 6 columns are dummy factors with three levels, and the last six are dummy covariates.

To compute the number of candidate models when there are between 1 and 6 factors and 1 and 6 covariates, we call `glmulti` with `method = "d"` and `data = dod`. We use `names(dod)` to specify the names of the response variable and of the predictors. We vary the number of factors and covariates, this way:

```
R> dd <- matrix(nc = 6, nr = 6)
R> for(i in 1:6) for(j in 1:6) dd[i, j] <- glmulti(names(dod)[1],
+    names(dod)[c(2:(1 + i), 8:(7 + j))], data = dod, method = "d")
```

This will include the interaction terms. We can do the same but with parameter `level = 1`, to include only main effects. We can also add `marginality = FALSE` to investigate the effect of ruling out non-marginal models.

Looking at matrix `dd`, it is obvious that the number of candidate models increases faster when adding covariates than when adding factors. Note that when the number of models is greater than some threshold ($10^9$ in the current version), then `glmulti` returns -1. Plotting the first row and the first column of `dd` (and its equivalent with `level = 1` or `marginality = FALSE`) produces Figure 2A. Note how enforcing marginality reduces the number of models when there are many covariates, but not when there are mostly factors.

We can also look at the complexity of the candidate models, $K$, i.e. the number of parameters they estimate from the data. This will depend on the number of levels that the factors have. In our case they all have three.

Calling `glmulti` with arguments `minK = i` and `maxK = i` indicates how many candidate models have complexity `i`. We will do this with 1 factor and 5 covariates, using our previous dummy data frame:

```
R> dd <- integer(40)
R> for(i in 1:40) dd[i] <- glmulti(names(dod)[1],
+    names(dod)[c(2, 8:12)], data = dod, method = "d", minK = i, maxK = i)
R> plot(dd/sum(dd), type = "b", xlab = "Model complexity (K)",
+    ylab = "Density of candidate models")
```

This will plot the distribution of $K$ in the candidate models, as shown in Figure 2B. We can see that in this situation the distribution is of Gaussian shape, with most models having intermediate complexity. This distribution is expected when there are mostly covariates or when factors have only two levels.

With 3 factors and 2 covariates, the graph looks different (Figure 2B). There are two major differences: the distribution is skewed to the right, and it is much less smooth. The first difference stems from the fact that redundancies in the formulas make complex models more likely than simple models (marginality has the same effect). The second difference occurs because factors add several estimated parameters at once (one per level minus one). In our case all factors have three levels, hence the fluctuations of period two that we can observe in the distribution.

## 3.2. Application to simulated data

In this subsection we will apply `glmulti` on artificial data generated from a known model. Discussing the validity of the IC-based model selection approach is beyond the scope of this article, which simply introduces a way to carry out this approach for people willing to. To illustrate the functioning of the exhaustive screening method, we will nonetheless give some measures of statistical performance. We will subsequently validate the genetic algorithm approach by showing how it converges to the results of the exhaustive screen.

### *Data generation*

We used a linear model with gaussian noise, and five predictor variables: three covariates (`a`, `b` and `c`) and two two-level factors `f1` and `f2`). For each variable a given number of independent observations are produced. Covariates `a` and `b` are drawn uniformly on $(0, 1)$ while `c` comes from an exponential distribution with rate 1. The two factors are random permutations of 1/3 observations of the first level and 2/3 of the second level.

The response variable `y` is then calculated for each data point $i$ as:

$$\begin{array}{rl} -2 - 2a_ib_i + 3a_i + \epsilon_i & \text{if } \texttt{f1}_i = 1 \\ 2 - 2a_ib_i - 3a_i + \epsilon_i & \text{if } \texttt{f1}_i = 2 \end{array}$$

where $\epsilon_i$ is a Gaussian deviate with zero mean and a specified variance. The model therefore includes an effect of `a` and `f1` an interaction between `f1` and `a`, and an interaction between `f1` and `b` (but no main effect of `b`). `c` and `f2` are dummy variables with no effect at all. The corresponding formula reads `y ~ a + f1 + f1:a + a:b`, whereas the candidate set contain 5000 models.

The true model having finite dimensionality and being present in the candidate set, BIC is best adapted because it is dimensionally consistent (Burnham and Anderson 2002). `lm` will be used as fitting function, and marginality will not be required. Sample size will be varied between 20 and 150, and the error will have standard deviation 0.5, so that the full model explains 80% of the variance on average, with 16 estimated parameters.

*Exhaustive screening*

We first apply the exhaustive screening method. Calls look like:

```
R> obj <- glmulti(y ~ a*b*c*f1*f2, data = gaga(n, 0.5), fitfunc = lm,
+    crit = bic)
```

where `gaga` returns a simulated dataset of size $n$.

From the `glmulti` objects obtained, it is easy to plot graphical summaries (Figure 3):

```
R> plot(obj, type = "p", highlight = c("f1:f2"))
R> plot(obj, type = "w", highlight = c("f1:f2"))
R> plot(obj, type = "s")
```

The first type of `plot` draws the IC profile (i.e., the ranked IC values of models), together with a horizontal line two IC units above the best model. A common rule of thumb is that models below this line are worth considering. This can be used to decide whether enough models have been included in the confidence set: should the line be invisible, important models are likely to have been ignored, suggesting a bigger confidence set would be appropriate.

The second type plots the ranked relative evidence weight of the models. These are computed as $\exp(-\Delta IC/2)$, where $\Delta IC$ is the difference in IC between a model and the best model, and they normalized so that they sum up to one. They can be interpreted as probabilities for each model to be the best in the set. A red vertical line is shown where the cumulated evidence weight reaches 95%.

The third option plots for each term its estimated importance (or relative evidence weight), computed as the sum of the relative evidence weights of all models in which the term appears.

The `highlight` option is used in the first two cases to locate those models in which term a is found. The importance of `f1:f2`, as shown in the third panel, is for instance simply the sum of the relative weights of these models.

The model selection algorithm should identify the important terms (i.e., those included in our true model) and rule out the others. Hence we might expect the best model returned to be the true model. This uses IC model selection as a mean to compare multiple models and select the best one, but does not involve multi-model inference. It is similar to what function `step`, for instance, does.

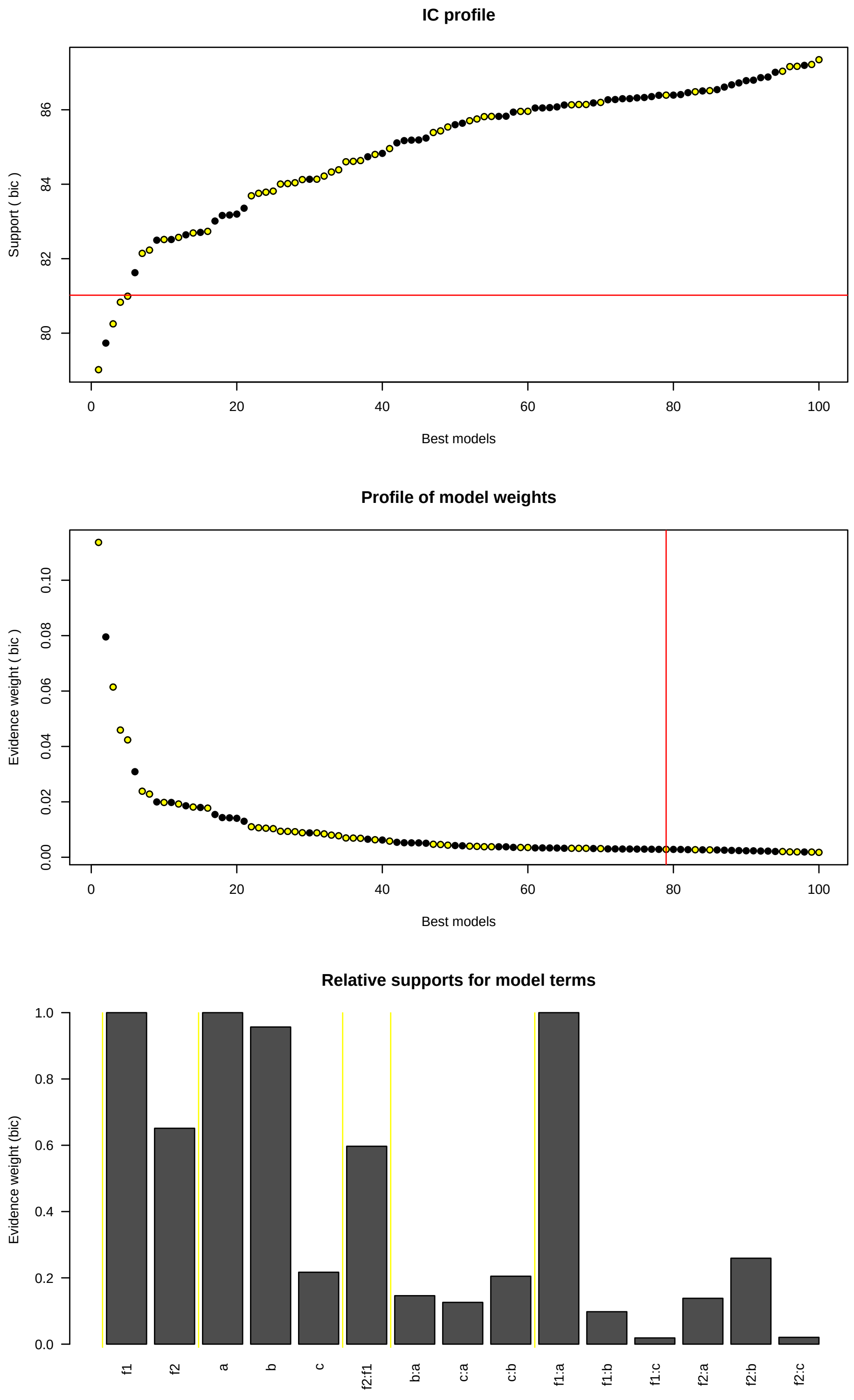


Figure 3: The three types of graphics provided by the `plot` method. Top: `type = "p"`, the IC profile. Middle: `type = "w"`, the relative weights of models. Models containing the highlighted term (`f1:f2`) appear in yellow. Bottom: `type = "w"`, the relative weights (importance) of model terms. The figure was obtained from a generated of size 100.

In practice IC values will be random variables because of finite sample size. Hence the identity of the best model may be subject to some fluctuations, especially with small sample size and/or when many models are compared. For this reason it is recommended to use all models rather than solely the best (Burnham and Anderson 2002; Anderson 2008). This can be done by estimating the importance of a given term as the summed IC weight of all models in which the term appears (e.g., Buckland *et al.* 1997).

In order to get an idea of the statistical performance of these approaches, we replicate analyses 100 times. Each time the formula of the best model and the estimated importance of terms can be obtained as:

```
R> bestmodel <- obj$@formulas[1]
R> importances <- summary(obj)$termweights
```

If we consider the best model only, we pick the true model 11%, 66%, 76% and 90% of times, depending on the number of observations in simulated datasets (20, 50, 100 and 150, respectively). This stresses the fact that focusing on a single best model may not be very robust.

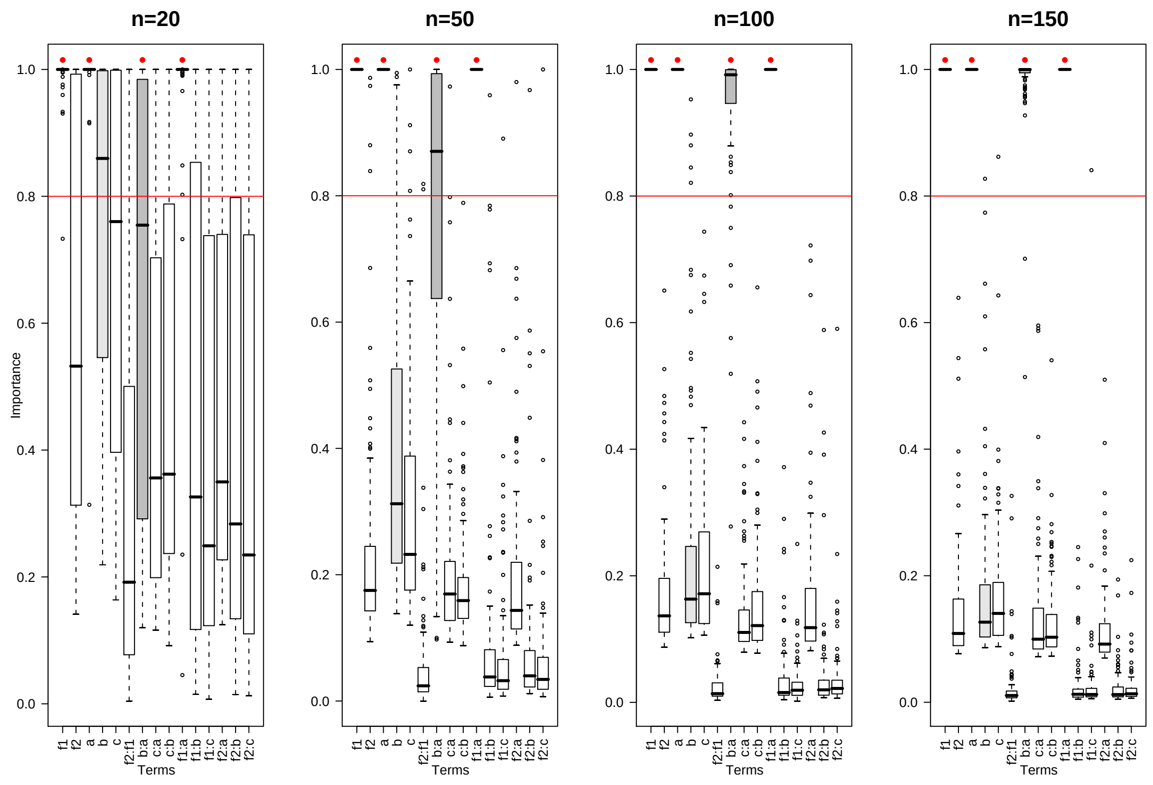


Figure 4: Exhaustive screening approach on simulated data. For different sample sizes (from left to right: 20, 50, 100 and 150) the distribution of the estimated importances of terms over 100 replicate datasets is presented as box-whisker plots. The terms that are effectively in the true model are marked with a red dot. The `a:b` interaction is shown in gray, the main `b` effect in light gray. The red line illustrates a 80% threshold for importance.

If instead we take the multi-model approach and compute the importance of each term across all models, we obtain the results in Figure 4. It can be seen that important terms are most of the time identified, even at small sample size (their inferred importance is close to one). At very small sample size importance is quite variable though, and some non-important terms have a non-negligible probability to be picked as important. As sample size increases, the discrimination between important and non-important terms quickly becomes sharp.

An interesting thing to observe is the behavior of the `a:b` interaction. It requires larger sample sizes than the other important identified in order to be identified. Even with $n = 150$ it is still given relatively small importance in about 2% of replicates (Figure 4). The reason is that the two variables `b` and `a*b` are highly correlated ($r = 0.7$ in our samples), even though `b` has no intrinsic effect. This makes it tricky to discriminate them statistically. Clearly, at small sample sizes, the estimated importance of `b` is rather high because of this correlation, and this in turn weakens the estimated importance of `a:b`. Sample size must be increased for the method to be able to favor the interaction over the main effect. This suggests that colinearity among variables will tend to reduce statistical power, in the sense that the importance will be "diluted" over the different correlated variables, reducing their individual weight. This is a classical issue (Graham 2001).

In deciding which terms are actually retained we must decide some threshold importance (say, 80%) below which terms will be regarded as unimportant. In doing so we take the risk to retain non-important terms (type I error) and to discard important effects (type II error). These two risks, especially the second, are difficult to quantify in most model selection approaches, for instance because of multiple non-independent hypothesis tests. Although IC-based model selection does not involve the computation of p-values, it is not immune to these problems. Clearly, the bigger the number of models compared for a given amount of data, the higher the chance to include spurious effects, and the lower the sample size, the higher the chance to miss important effects. Hopefully new research will help clarify these issues, but one should clearly bear them in mind and, for instance, think carefully about which candidate models/terms should be included as candidates (Anderson 2008).

For illustrative purposes, we computed the overall probability to make type I and type II errors for each sample size, as a function of the minimal importance a term should have in order to be retained (Figure 5).

Despite the high number of model comparisons (5000) statistical risk is generally well controlled, except at very small sample size. Using a 80% threshold in general yields good properties, with less than 5% overall type I risk when sample size is 150. Type II risk is somewhat higher, but in fact this is mostly driven by the difficulty to distinguish the main effect b from the interaction a:b. The main effect can be retained instead of the interaction, resulting in both type I and type II mistakes. Overall, dumb variables are consistently eliminated.

### *Genetic algorithm*

We now apply the genetic algorithm method to determine whether it converges to the exact solution (i.e., model ranking) obtained by exhaustive screening. For one random dataset used before (with $n = 100$), 500 algorithms were run with random parameter combinations (Figure 6). Calls are exactly the same as before except that we use `method = "g"`:

```
R> objga <- glmulti(y ~ a*b*c*f1*f2, data = onedataset, fitfunc = lm,
+    crit = bic, method = "g")
```

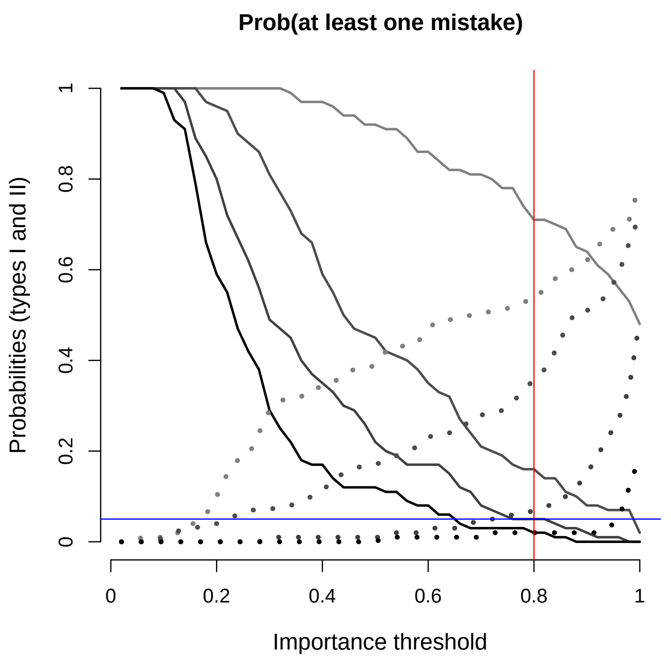


Figure 5: Type I and type II statistical risks. Probability to retain a non-important effect (type I error, solid lines) and to drop an important term (type II error, dotted lines), as a function of the threshold level of importance (x axis). The commonly accepted 5% level is shown as a horizontal blue line. Sample size increases with the shade of gray (light gray: 20, dark gray: 150). The 80% threshold is shown as before by a red line.

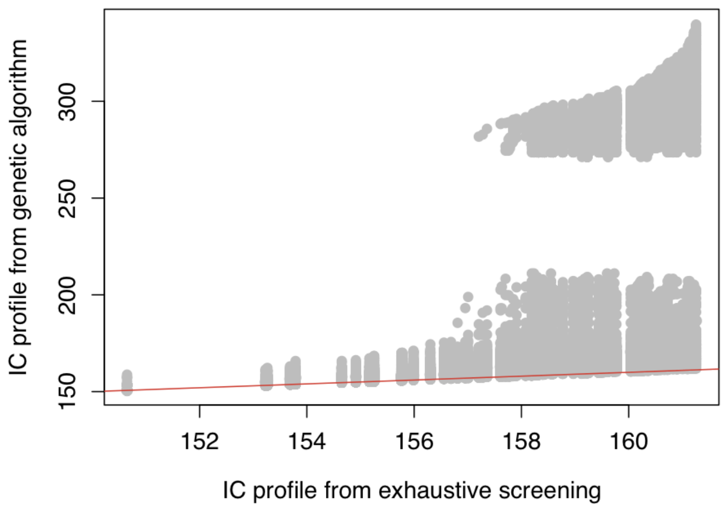


Figure 6: Genetic algorithm on a simulated dataset of size 100. The IC profile (the IC values of the 100 returned models, in ascending order) is plotted against the exact IC profile (obtained through exhaustive screening). 500 genetic algorithms are shown superposed. They had random parameter values for `popsize` (between 10 and 150), `mutrate` (between $10^{-4}$ and $10^{-2}$), `sexrate` (0 to 0.2), `imm` (0 to 0.5), `deltaB` (0 to 0.5), `deltaM` (0 to 0.5) and `conseq` (2 to 5). The red line represents $y = x$, i.e., a perfect agreement between genetic algorithm and exhaustive screening.

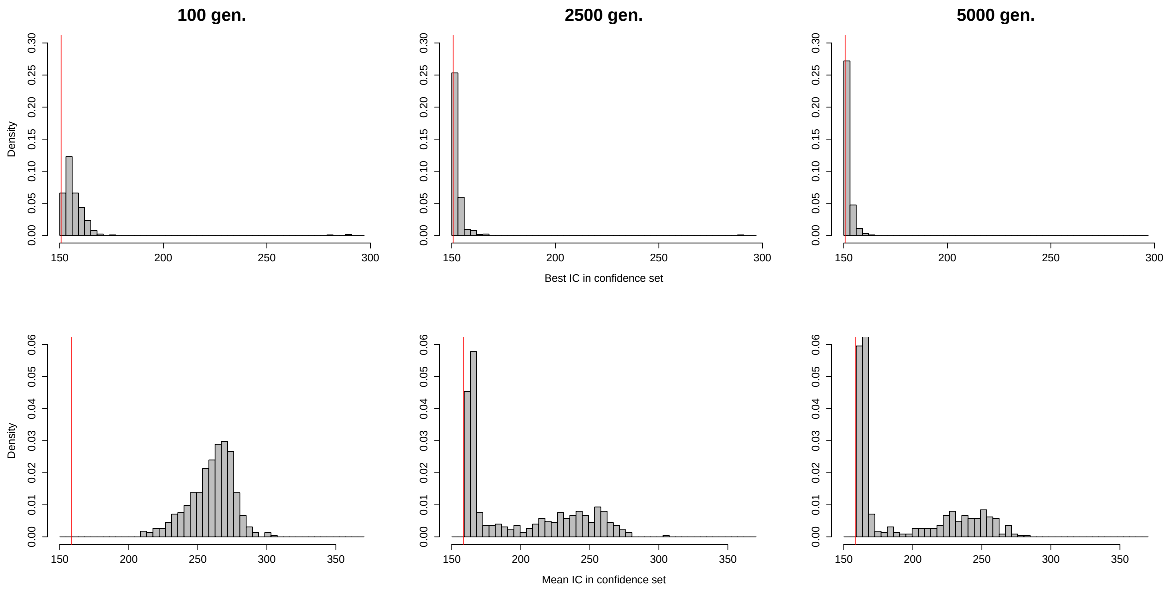


Figure 7: Convergence dynamics of the genetic algorithm in terms of the best model found (top) and the average IC in the confidence set (bottom). The frequency distributions are over the 500 genetic algorithms.

The objects obtained can be handled as usual. In addition, the number of generations before convergence and the time it took can also be obtained through `summary`:

```
R> summary(objga)$generations
R> summary(objga)$elapsed
```

Clearly the problem posed here is small (5000 models) and the a genetic algorithm is not expected to rival with an exhaustive screen. Despite that most algorithms returned within 2 minutes, which is only 20 seconds slower than the exhaustive screen on our machine.

The actual best model was found 48.6% of the time, but there is clearly a lot of variance across simulations, especially in the tail of the confidence set (Figure 7). The best model was identified pretty fast but, unsurprisingly, resolving the whole confidence set took longer (Figure 7). Actually the distribution of the mean IC is bimodal even after 5000 generations. This bimodality concerns the tail of the confidence set, as can be seen in Figure 6.

This indicates that there can be suboptimal but locally stable states in which the algorithm may be stuck, a common issue in optimization. To avoid this it is recommended to always use the immigration operator when using the genetic algorithm, since it is very efficient at avoiding local optima and pushing the algorithm towards the global optimum (see Figure 8).

Default parameters will work in most cases, but we recommend testing different parameter combinations with a subset of the candidate models before starting a massive genetic algorithm. This allows to find parameter values that are optimal in terms of performance and convergence properties for a given type of data.

Fortunately, in our case alternative states do not affect inference since they differ only for

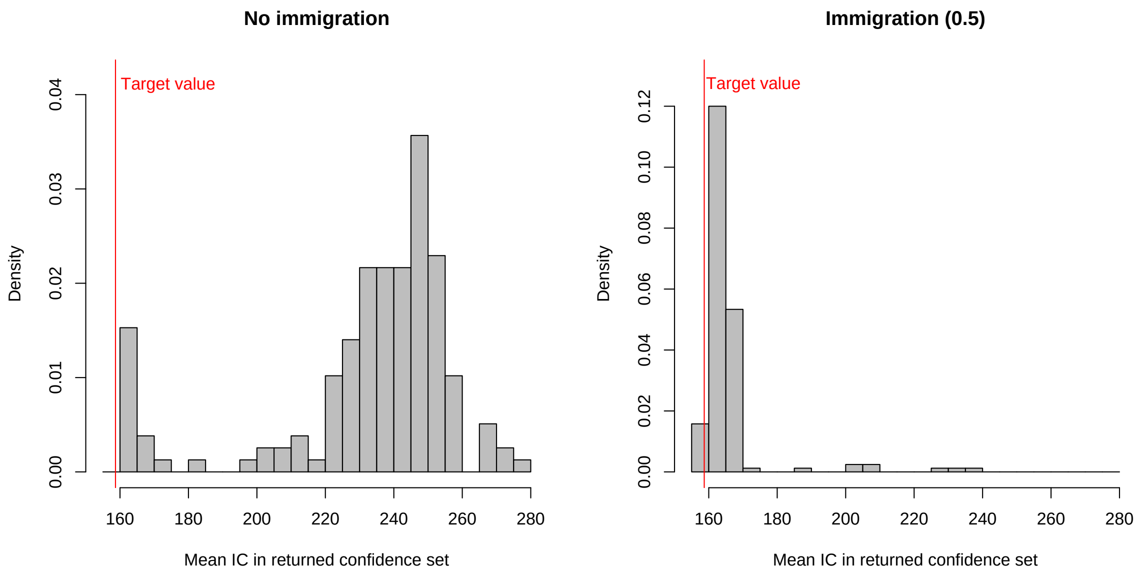


Figure 8: The distribution of mean IC value in the confidence sets returned by the genetic algorithm with no immigration (left) and with immigration (`imm = 0.5`, right). Many simulations keep locked at a suboptimal state (rightmost peak) in the absence of immigration. Including immigration efficiently prevents this.

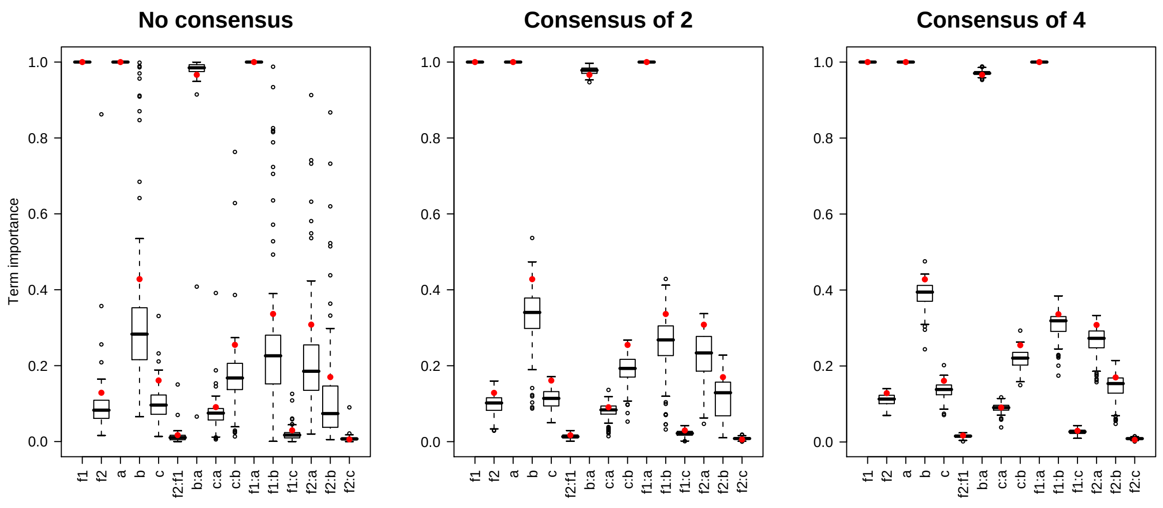


Figure 9: Estimated importance of model terms across all 500 genetic algorithms. The actual value obtained from an exhaustive screen are shown as red dots. Left: Individual simulations (no consensus), Middle: Consensus of two simulations, and Right: Consensus of four simulations.

very weakly supported models, having virtually no weight. Across all genetic algorithms, the estimated importance of terms are very close to the exact values obtained from the exhaustive screen (Figure 9).

A very effective way to improve convergence is to make a consensus of several replicate genetic algorithms, i.e., taking the best models found over all replicates, removing possible duplicates. This is easy to realize with to the function `consensus`:

```
R> consensusobj <- consensus(list(objga1, objga2, ...), confsetsize = 100)
```

As shown in Figure 9, making inferences from a consensus of two simulations greatly improves convergence, and from a consensus of four simulations the results are almost indistinguishable from the true ones. We strongly recommend to run replicate genetic algorithms and working on a consensus of them rather than using a single simulation with extremely conservative stopping rules (e.g., high `conseq` value).

## 3.3. Application to real data

We now turn to real data, with a different model structure (`glm` with binomial distribution). We use a standard dataset from the **MASS** package, used in Venables and Ripley (1997) to illustrate automated model selection with GLMs: the birth weight dataset. This will allow to validate `glmulti` through a comparison with an earlier analysis using `step`. In line with Venables and Ripley (1997), we will try to explain the probability of babies having low weight at birth (binary variable `low`) from 8 predictors, using a binomial GLM. The data frame `bwt` is prepared exactly as in this book. These variables are collected in the `birthwt` data frame.

The analysis presented in Venables and Ripley (1997) starts with dropping main effects only. We similarly call:

```
R> test1 <- glmulti(low ~ ., data = bwt, family = binomial,
+    confsetsize = 300, crit = aic)
```

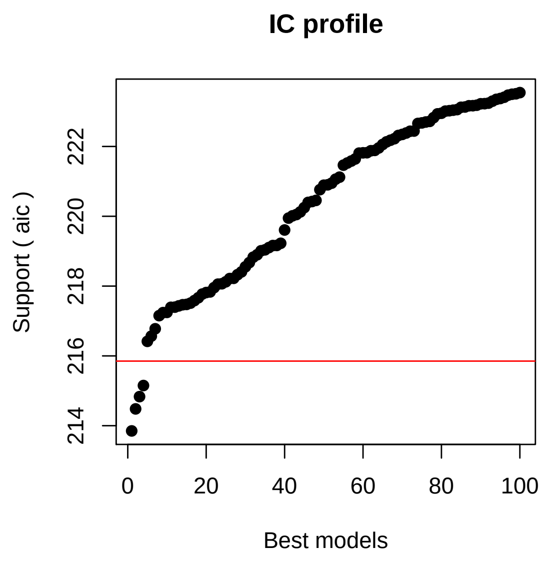
This call returns almost instantaneously since there are only 256 candidate models. Our setting `confetsize` to 300 means that all possible models are in the output.

The IC profile is plotted in Figure 10. Four models are clearly better than the others, with a sharp discontinuity; all other models are beyond two IC units. The best model is `low ~ 1 + race + ptd + lwt + smoke + ht + ui`, and this is exactly the model that `step` had selected. The three other models are well supported though, and consequently the five variables do not have the same importance: three (smoke, ht and hwt) are clearly supported, whereas the others (especially age) have considerably less support.

We now run the full analysis, with interaction terms and, to be consistent with Venables and Ripley (1997), the marginality rule. The number of candidate models is now much higher (about 286 millions[4]), to such a point that an exhaustive screening approach is hardly feasible in a reasonable amount of time.

A better option is to use the GA approach, as we will do just after, but for the purpose of this article we have carried out the exhaustive screening approach, parallelizing the task between 20 processes with:

[4]Not applying the general marginality rule results in some 338 millions models; the difference is slight here because there are many factors.



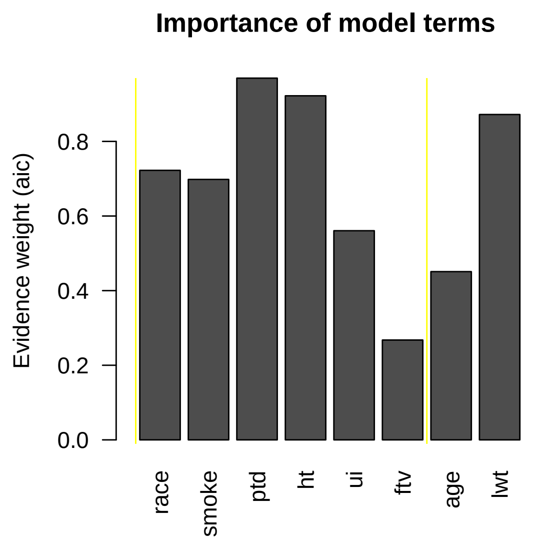


Figure 10: Exhaustive screening for the birth weight dataset with main effects only. Top panel (A): AIC profile (all possible 256 models). The horizontal red line represents the $\Delta AIC = 2$ limit. Lower panel (B): Estimated importance of predictors.

```
R> partobj <- glmulti(low ~ .*., data = bwt, family = binomial, chunk = 1,
+    chunks = 20, plotty = FALSE, report = FALSE, marginality = TRUE,
+    crit = aic, name = "exhausting")
R> write(partobj, file = "|object")
```

and similar calls with `chunk` between 2 and 20. We turned off any runtime report and wrote outputs as objects on the disk. This example took 11 days on our G5 cluster. This is when the GA approach proves very useful, as we will see below.

Each object is automatically named according to its task (i.e., `exhausting.1.20`, `exhausting.2.20`, etc.). When all calculations are over we just have to create a consensus object from all these files, using `consensus` as before:

```
R> fullobj <- consensus(list.files(pattern = "exhausting"),
+    confsetsize = 100)
```

The resulting IC profile and term importances are plotted in Figure 11. The arbitrary choice `consensus = 100` was maybe too small, since it is just enough to cover a span of two IC units. This is expected given the much higher number of candidate models; the proportion of models within two IC units from the best remains tiny though (about 110 out of 286 millions).

We observe that two models have identical and minimal IC values; hence there are two "best" models. Their formulas are:

```
low ~ 1+ smoke + ptd + ht + ui + ftv + age + lwt + ui:smoke + ftv:age
low ~ 1+ smoke + ptd + ht + ui + ftv + age + lwt + ui:smoke + ui:ht + ftv:age
```

The analysis presented by Venables and Ripley (1997) identified the first (simpler) one as the best. This indicates that with this dataset a stepwise search does converge rather well, while providing an independent validation of **glmulti**. Having two best models and not one is an extreme case where taking model selection uncertainty into account rather than looking for a single best model is certainly recommended!

In Figure 11 we see that eight effects are highly supported (above 80%): seven main effects (those found in the best model) and one interaction: age:ftv. The second interaction term included in both best models (ui:smoke) is much less supported, and has similar support as about three other interactions, *two of them not included in the best models* (around 40% estimated importance, see Figure 11). This shows again as focusing only on "best" models (and especially on a single best model) for inference may lead one to miss the full picture.

Further applications can be computing model-averaged estimates of the effects (model parameters) and model-averaged confidence intervals. These can easily be obtained through `coef`, which returns the model-averaged coefficients and their unconditional variance (Buckland *et al.* 1997; Burnham and Anderson 2002; Johnson and Omland 2004). `coef` also returns the estimated importance of model *coefficients*, so that one can take the same selection approach but with model coefficients rather than model terms.

Note that in our GLM model selection framework all terms are more or less equally frequent in the candidate models (up to implications and the marginality rule), so that the candidate set is "balanced" in this sense Anderson (2008). This is a nice property when doing such multi-model inferences.

We now use the genetic algorithm approach to evaluate how much faster it is. We ran 20 replicates of the algorithm with these calls:

```
R> glmulti(low ~ .*. data = bwt, family = binomial, method = "g",
+    plotty = FALSE, report = FALSE, marginality = TRUE,
+    deltaB = 0, deltaM = 0.01, conseq = 6, sexrate = 0.15, imm = 0.2)
```

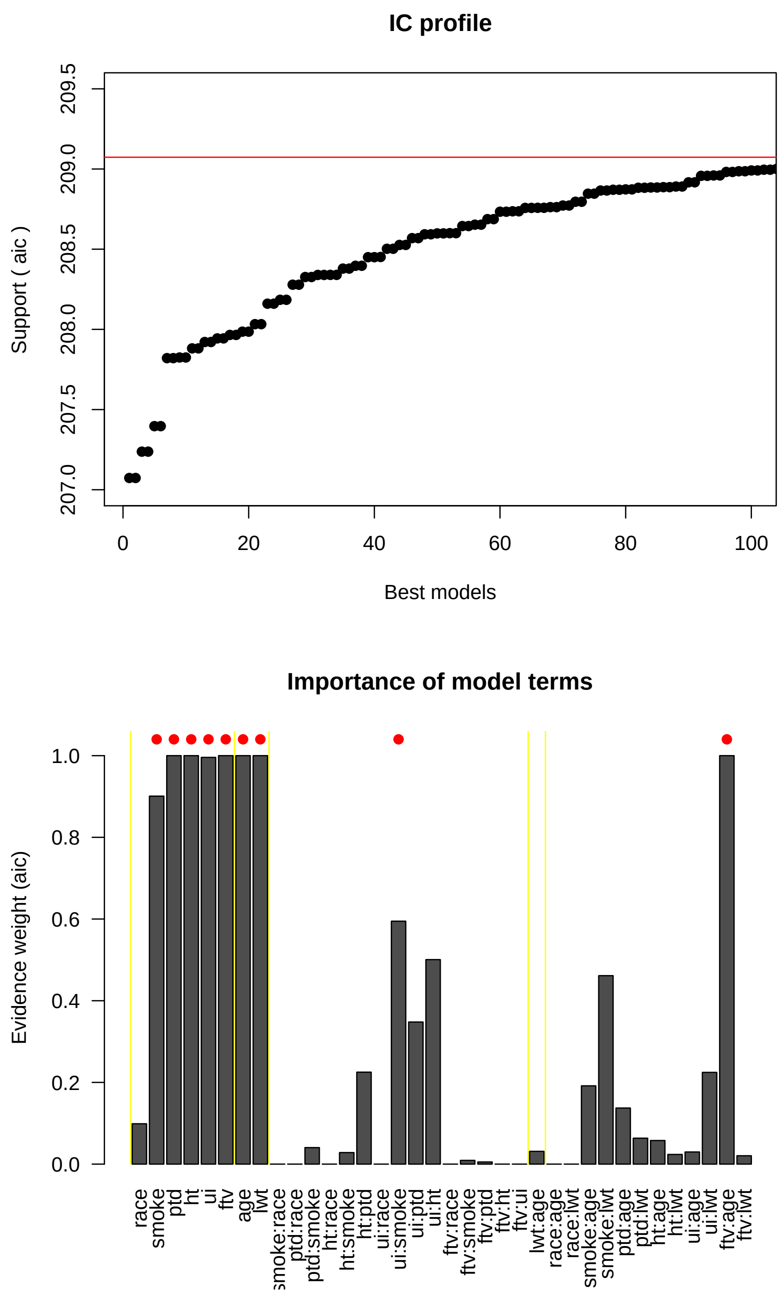


Figure 11: Exhaustive screening for the birth weight dataset with interactions. Top panel (A): AIC profile. Lower panel (B): Estimated importance of predictors. Terms found in the two best models are shown with a red dot.

As shown in Figure 12, all twenty replicates converged to the same values after less than 3000 generations. Inspecting the variability across replicates, before taking a consensus, is a good way to know whether simulations are likely to have converged to the actual solution.

Most replicates returned within a dozen of minutes on our computer. This is to be compared with the dozens of days it took to carry out the exhaustive screen (on similar CPUs). The

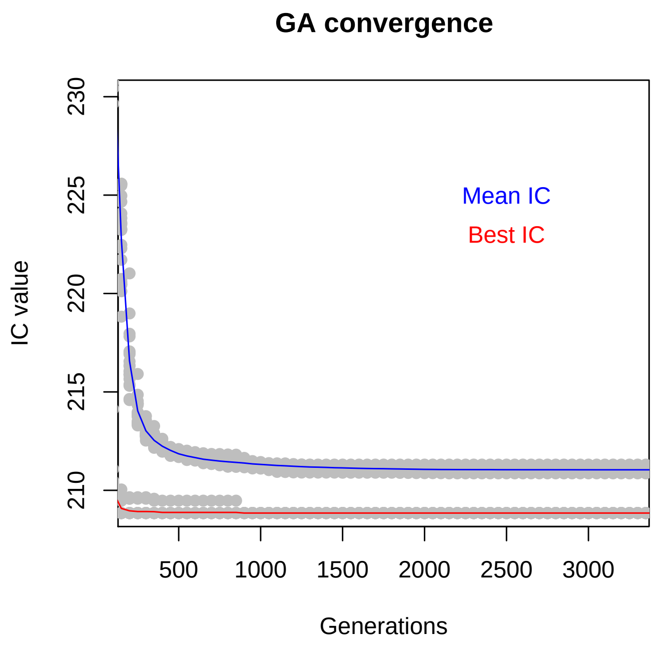


Figure 12: Convergence dynamics of the genetic algorithm. Blue: mean AIC value in the confidence set, averaged over all 20 simulations. Red: Mimimum AIC value in the confidence set, averaged over all 20 simulations. In both cases all individual values are plotted as gray dots.

results obtained, especially when taking a consensus of the 20 replicates, are indistinguishable from the ones obtained in Figure 11. This demonstrates the computational efficiency of the genetic algorithm with large candidate sets.

## 4. Conclusions

**glmulti** provides R users with a new and flexible way to carry out automated model selection and multi-model inference with GLMs. The default method (exhaustive screening of all candidate models) allows to have a complete view of the fit of the candidate models rather than using only one best model. This allows, in particular, the inclusion model-selection uncertainty in statistical inference. This approach can be be very time consuming, but the task can be split into an arbitrary number of parts, so that multi-core architectures and/or multiple computers can be taken advantage of.

The faster genetic algorithm approach allows to consider extremely large candidate sets in reasonable amounts of time. It converges very efficiently on the actual best models. Identifying the complete confidence set of models takes longer, but even when convergence is not perfect, the relative support for the different variables (or terms in model formulas) is quite robust. This is consistent with what is known of IC-based model selection: the identity of the single "best" model or of the few best models may be subject to model-selection bias and are random variables over samples. Synthetic statistics of the confidence set of models (e.g., multi-model parameter estimates, relative support for the variables, etc.) are expected to be more robust (Anderson 2008).

The goal of this package is not to replace `step` and other iterative approaches already available, but rather to provide R users with an alternative which, at the price of being more computationally demanding, is more robust and provides additional insight. We expect **glmulti** will help investigators to use IC-based model selection and multi-model inference.

## Acknowledgments

We are grateful to all those who provided comments and encouragements while the package was being written, to Andy Gonzalez and all attendants of the R4All course at McGill in March 2009, and to Simon Urbanek for developing the **rJava** package.

## References

Anderson DR (2008). *Model Based Inference in the Life Sciences.* Springer-Verlag, New York.

Bartoń K (2009). ***MuMIn**: Multi-Model Inference.* R package version 0.12.2/r18, URL `http://R-Forge.R-project.org/projects/mumin/`.

Buckland ST, Burnham KP, Augustin NH (1997). "Model Selection: An Integral Part of Inference." *Biometrics*, **53**, 603–618.

Burnham KP, Anderson DR (2002). *Model Selection and Multimodel Inference.* Springer-Verlag, New York.

Calcagno V, Thomas Y, Bourguet D (2007). "Sympatric Host Races of the European Corn Borer: Adaptation to Host Plants and Hybrid Performance." *Journal Of Evolutionary Biology*, **20**(5), 1720–1729.

Cederkvist HR, Aastvelt AH, Naes T (2007). "The Importance of Functional Marginality in Model Building – A Case Study." *Chemometrics And Intelligent Laboratory Systems*, **87**(1), 72–80.

Grafen A, Hails R (2002). *Modern Statistics for the Life Sciences.* Oxford University Press, New York.

Graham MH (2001). "Confronting Multicolinearity in Ecological Multiple Regression." *Ecology*, **84**, 2809–2815.

Harrell FE (2001). *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis.* Springer-Verlag, New York.

Johnson JB, Omland KS (2004). "Model Selection in Ecology and Evolution." *Trends In Ecology & Evolution*, **19**(2), 101–108.

Lumley T, Miller A (2009). ***leaps**: Regression Subset Selection.* R package version 2.9, URL `http://CRAN.R-project.org/package=leaps`.

McLeod A, Xu C (2009). ***bestglm**: Best Subset GLM.* R package version 0.20, URL `http://CRAN.R-project.org/package=bestglm`.

Miller AJ (2002). *Subset Selection in Regression.* Monographs on Statistics and Applied Probability, Norwell, MA.

Nelder JA (2000). "Functional Marginality and Response-Surface Fitting." *Journal Of Applied Statistics*, **27**(1), 109–112.

Orestes Cerdeira J, Duarte Silva AP, Cadima J, Minhoto M (2009). ***subselect**: Selecting Variable Subsets.* R package version 0.10-1, URL `http://CRAN.R-project.org/package=subselect`.

Park MY, Hastie T (2007). ***glmpath**: $L_1$-Regularization Path for Generalized Linear Models and Cox Proportional Hazards Model.* R package version 0.94, URL `http://CRAN.R-project.org/package=glmpath`.

Prevost M, Calcagno V, Renoult L, Heppleston AC, Debruille JB (2010). "Psychological Traits of Healthy Participants Affect Specific Independent Components of the N400 Potential." Unpublished manuscript.

R Development Core Team (2010). *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/`.

Ripley BD (2003). "Selecting Amongst Large Classes of Models." Lecture, URL `http://www.stats.ox.ac.uk/~ripley/Nelder80.pdf`.

Trevino V, Falciani F (2006). "**GALGO**: An R Package for Multivariate Variable Selection Using Genetic Algorithms." *Bioinformatics*, **22**, 1154–1156.

Urbanek S (2009). ***rJava**: Low-Level R to Java Interface.* R package version 0.8-1, URL `http://CRAN.R-project.org/package=rJava`.

Venables WN, Ripley BD (1997). *Modern Applied Statistics with S-PLUS.* 3rd edition. Springer-Verlag, New York.

Wickham H (2009). ***meifly**: Interactive Model Exploration Using **GGobi**.* R package version 0.1.1, URL `http://CRAN.R-project.org/package=meifly`.

Yang WX (2004). "An Improved Genetic Algorithm Adopting Immigration Operator." *Intelligent Data Analysis*, **8**, 385–401.

**Affiliation:**

Vincent Calcagno
McGill University

Redpath Museum. c/o Biology Dept.
1205 av Docteur-Penfield
Montreal, QC, H3A 1B1, Canada
E-mail: vincent.calcagno@mcgill.ca

*Journal of Statistical Software* http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 34, Issue 12 *Submitted:* 2009-06-19
May 2010 *Accepted:* 2010-01-29