Proposed changes to speed function interface

Sam Rogers

Background

The `speed` package currently supports a variety of relatively simple experimental designs, including completely randomized designs, randomized complete block designs, Latin square designs. It can also handle some more complex heirarchical designs such as split-plot and strip-plot designs. Our current development focus is building support for more complex designs including nested cases, such as factorial designs and one-step multi-environment trial designs.

We are mindful of the fact that many users of the package are not R programmers, and so we want to keep the interface as simple as possible for people to use, especially for the more basic designs. However, we also want to provide a lot of flexibility for more advanced users. To this end, we are considering some changes to the interface for specifying the speed function, especially in the context of more complex designs.

Objectives for any changes to the speed interface, roughly in order of priority:

- Ensure that the interface is intuitive and easy to use
- Maintain simplicity for basic designs
- Ensure maximum flexibility for more complex designs
- Provide clear documentation and examples
- Provide good defaults for common use cases
- Allow for custom objective functions to be easily integrated
- Support for both formula and non-formula interfaces
- Minimize the learning curve for new users
- Consistency with other R packages and conventions
- Maintain backward compatibility where possible

Current Function Signature

The current function interface for `speed` is as follows:

```r
speed <- function(data,
                  swap,
                  swap_within = "1",
                  spatial_factors = ~ row + col,
                  grid_factors = list(dim1 = "row", dim2 = "col"),
                  iterations = 10000,
                  early_stop_iterations = 2000,
                  obj_function = objective_function,
                  quiet = FALSE,
                  seed = NULL,
                  ...)
```

This interface is relatively straightforward and works well for simple designs. Arguments are given as single values or simple formulas. For more complex designs, such as split-plot designs, the user currently specifies the design using named lists to match the different components. This allows for a lot of flexibility, because it can enable the user to specify arbitrary nesting structures, and to create designs where different objective functions or different numbers of iterations are applied to different levels of the design for example. However, this gets complicated and verbose as designs increase in complexity, and it is not very intuitive for new users. It may be that a user ends up with several named lists to specify the design.

The Challenge

The challenge is to find an interface that is both flexible enough to handle complex designs, while also being simple and intuitive for basic designs. We want to avoid a situation where the user has to learn a lot of new syntax or concepts to use the package effectively. We also want to ensure that the interface is consistent with other R packages and conventions, so that users can easily transfer their knowledge from one package to another.

Design Cases

At a conceptual level, there are three basic cases that we need to consider:

1. Single factor designs (e.g. completely randomised, randomised complete block, Latin square, balanced incomplete block, 2D blocking). These are relatively straightforward, and can be handled with the current interface.
2. Nested or Hierarchical designs with multiple factors (e.g. split-plot, strip-plot, MET). These are more complex, and require a more flexible interface to specify the different levels of nesting and the relationships between them. These designs are mostly supported in the current interface, but the syntax can get complicated and verbose.

3. Crossed designs (e.g. factorial designs with multiple factors that are crossed rather than nested so don't have the same concept of a heirarchy). These are also complex, and require a different approach to specifying the design. These designs are not currently supported in the package.

Case 2 and 3 are the main focus of the proposed changes to the interface.

Specific Challenges

- Some designs (e.g. split-plot, strip-plot) require multiple levels of nesting, where the higher level units all move together during swaps, while lower level units can move independently (within the constraints of the higher level units).
- Factorial designs that have multiple factors that are crossed, rather than nested,
- Some designs may require different objective functions, iterations and/or spatial or grid factors to be applied at different levels of the design.

Proposed Interface Options

We are considering two main options for the interface to better support complex designs while maintaining simplicity for basic designs. These options are not mutually exclusive, and we could potentially implement both. We are also open to other suggestions. The first option is basically an extension of the current interface, while the second option is a more radical change. A third option is a formula interface, which we are planning to implement regardless of which of the other two options we choose as an alternative interface for those who prefer it.

For simplicity in this discussion, we will assume a data frame `data` has been provided with the appropriate columns for the design factors, and the other arguments not related to the design specification (e.g., `iterations`, `quiet`, `seed`, etc.) remain unchanged.

1. Extended Current Interface

2. Nested List Interface

3. Formula Interface

We propose to introduce a formula interface for specifying the experimental design. This would allow users to specify the design using a formula syntax similar to that used in other R packages (e.g., `lm`, `lme4`). For example, a user could specify a randomized complete block design as `response ~ treatment + block`.

Questions for Testers

Please consider the proposed alternative interface versions and provide feedback on any of the following questions:

- Which signature feels most natural to use?
- Which would you be most likely to adopt in your own work?
- Are there specific features or arguments that must remain easy to access?
- Do you see any barriers for new users learning the function?
- Are there any specific use cases or designs that you would like to see supported?
- Do you have any other suggestions for improving the clarity and/or flexibility of the interface?