Optimal Semi Split Plot Designs with R

Sebastian Hoffmeister and Andrea Geistanger

October 17, 2018

Contents

Abstract

This paper introduces Semi Split-Plot designs. They are a new class of experimental designs amd supports factors where only a reduced number of factor settings can be applied inside of one block. An algorithm to generate optimal Semi Split-Plot designs is preseneted. A tutorial for the r-package **rospd** that implements the algorithm is given. Semi Split-Plot designs are compared to completly randomized and split-plot designs in terms of balance, aliasing and predictive quality.

1 Introduction

Split-plot designs are a very well known quantity in the field of Design of Experiments (DoE). First introduced by Fisher [1], today they are a standard tool in many applications of DoE and well supported by most statistical software packages. Split-plot designs are the answer to the problem of experimental factors that are hard to change. A factor is hard to change when it is very expensive or time-intensive to change the factor's settings from one level to another.

Naturally, split-plot designs have been the first thought of the authors when working with a sample preperation robot in the pharmaceutical application of optimizing an assay. Sample preperation robots allow to perform many experiments completly automatically. One restriction is to not use too many different solvents in the process. There is only limited space to place the bottles with solvents on the robot. In the given application the experimentors are able to place four different bottles of solvents on the robot.

Whenever a fifth solvent is supposed to be used, the robot has to be stopped and the operator has to manually exchange the bottles of solvents. While this sounds like a typical hard-to-change factor (**HTC**) there is a major difference: For traditional hard to change factors we try to minimize the number of changes in factor levels. Instead of minimizing the overall number of changes, in this scenario we want

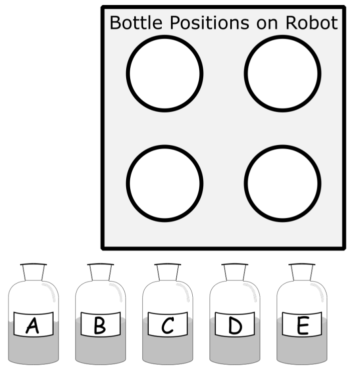

Figure 1: Sample Preperation Robot Problem

to build blocks of experiments in which only a subset of all possible factor levels are used. The problem could be interpreted as generating a blocked design that includes a restriction for each block to use not more than four different solvents.

This paper discusses the problem in more detail. An algorithm to generate optimal designs for this problem is proposed and some resulting designs are evaluated in comparison to completly randomized designs and split-plot designs.

The experienced DoE-expert might skip chapter 2 which gives a short description of split-plot designs and the typical way of analyzing them. Chapter 3 describes the sample preperation robot scenario in greater detail. In chapter 4 we propose an algorithm to generate optimal designs for the described problem. This algorithm is implemented in the rospd-package for the R-software. A short tutorial on how to use the package follows in chapter 5. A comparison of multiple design classes in terms of aliasing and predictive quality is done in chapter 6 before we conclude with a summary and an overview of future work (chapter 7).

2 Restricted Randomization in Split-Plot-Designs

When applying design of experiments (DoE) in the real world it is often difficult to follow one of the core concepts of DoE: Randomization. In a completely randomized design (**CRD**), treatments (combinations of factor levels) are performed in complete random order to avoid bias in the analysis due to uncontrolled factors. In many applications following this idea is a problem. Often there are restrictions to the experimental setup that make a complete randomized design extremely expensive or inconvenient at best.

Kowalski [2] discusses a example for this problem. When trying to optimize the quality of a printing process there are three factors to look at:

- **Blanket Type:** Two possibles types of blankets for the printing press.
- **Cylinder Gap:** The distance between the two cylinders of the printing press.
- **Press Speed:** The speed of the printing press.

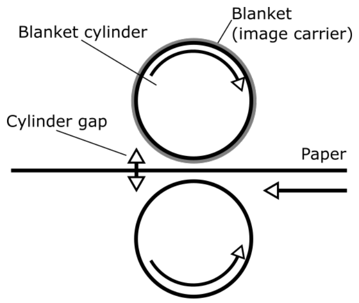

Figure 2: Printing Press Example

It is fairly simple to change the **Cyliner Gap** or **Press Speed** from one experiment to the next one, as these factors can be reset during the printing process. Exchanging the **Blanket Type** though is much more work. It requires the printing press to be stopped before the old blanket can be removed and the new one can be put on the press. Thus a completely randomized design, like the following one, is very time intensive as it involves exchanging the blanket quite often.

The reader should be aware that the operator is expected to exchange the blanket after each run even

Blanket Type	Cylinder Gap	Press Speed
2	low	high
2	high	low
2	low	low
1	low	high
2	low	low
1	high	low
1	low	low
2	high	high

Table 1: Printing Press Example as CRD

if consecutive runs are using the same type of blanket. The rationale behind that is, that ordinary least squares regression - that is typically used to analyze DoE-data - assumes independence of observations. Using the same blanket for multiple runs would violate this assumption as the experiments done with the same blanket would be correlated with each other.

Fisher [1] first wrote about that problem and gave a solution with **split-plot designs**. A split-plot design allows restricted randomization. For the current example this means that it is no longer necessary to exchange the blanket for each experiment. Instead the same blanket can be used inside of each **whole plot**.

WholePlot	BlanketType	CylinderGap	Speed
1	1	high	low
1	1	low	low
2	2	high	high
2	2	high	low
3	2	low	low
3	2	low	low
4	2	high	low
4	2	low	high

Table 2: Printing Press Example as Split Plot Design

In this example the factor **blanket type** would be called a **hard-to-change**-factor (HTC). It is inconvenient to reset this factor for each experiment and thus a restriction to the randomiztion of the experiment is imposed. The restriction limits the number of resets of the hard-to-change factor to a practical amount.

In contrast to the hard-to-change-factor, all other factors - that can be reset for each experiment without a lot of effort - are called **easy-to-change**-factors (ETC). A split-plot design is still randomized inside of the whole plots so that the easy-to-change-factors are changing as much as possible.

There are many forms of split-plot designs and many examples of different types of hard-to-change factors. Jones et al. [3], [4] and Kowalski [2] give a good overview of this topic.

2.1 Design Model

The **design model** of a split-plot design is different to the traditional design models of completely randomized designs (CRD). For the analysis of CRDs an ordinary least squares regression is applicable:

$$Y = X\beta + \epsilon$$

where:

- Y is a $n \times 1$-vector containing the measured responses. Here n represents the number of runs in the design.
- β is a $p \times 1$-vector containing the model parameters. Here p is the number of parameters in the model including the intercept.
- X is the $n \times p$-model matrix, where each column represents one model effect and each of the n rows represent one run of the design.
- ϵ is a $n \times 1$-vector representing the residuals of the model. The residuals are assumed to be identically, independently and normally distributed with mean 0 and standard deviation σ.

As mentioned before there is a problem with the assumption of independence of residuals when using a split-plot design. The design is set up so that some factor settings are not changed for a group of experiments. Thus all experiments in one whole plot are correlated with each other. This correlation arrises because the experimentator is missing the variability that is introduced to the process by changing the hard-to-change-factor. To account for the correlation structure in the analysis, split-plot designs are usually analyzed by using mixed models of the following structure:

$$Y = X\beta + Z\gamma + \epsilon$$

Here Y, X and β are the same like before.

- Z is a $n \times b$-matrix representing the whole plot structure of the design. b is the number of hard-to-change-factors.
- γ is a $b \times 1$ vector containing the random effects of the whole plots. Their estimators represent the variability between different whole plots. Is is assumed that $\gamma \sim N(0, \sigma_w^2)$.

The variance-covariance matrix of the response vector Y can be written as (Jones [3]):

$$V = \sigma^2 I_n + \sigma_w^2 ZZ'$$

V is a block diagonal matrix, where each block represents one whole plot. σ^2 is the residual variance, while σ_w^2 represents the variance between whole plots. A more in depth description of the analysis of split-plot designs is given at Jones [3] and Næs [5].

2.2 Design Setup

The design of an experiment goes hand in hand with the design model. Thus the modification of the design model has some implication for the setup of a split-plot design. While some programs still provide split-plot-variants for classical full- and fractional-factorial designs (Jones [3]), optimal designs have proven their value with a much higher flexibility (Jones & Goos [6], [4]). When planning optimal split-plot designs there are two major factors to be considered:

1. How many whole-plots should be chosen?
2. How to calculate the chosen optimality criterion considering the different type of analysis that is done for split-plot designs?

The number of whole plots defines how often the experimentor needs to reset the condition of the corresponding hard-to-change-factor. Thus this decision is often made with an economical background. At the same time the number of whole plots is very relevant for the estimation of the random-effects part of the model and for the power of the hard-to-change-factor effects (see JMP Help [7]). The minimal number of whole plots is $l + 1$ where l is the number of levels of the hard-to-change-factor. Using only l whole plots would make it impossible to differentiate between the effect of the hard-to-change-factor and the whole plot-effect (which is the variability that comes from resetting the condition of the hard-to-change factor).

The second aspect that is relevant when working with optimal split-plot-designs are the necessary adjustments to optimality criteria. As most optimality criteria depend on the chosen design model it is necessary to adjust them to the new design model including the random effects. Jones and Goos [6], [8] as well as Hooks et al. [9] give an overview of how to calculate different optimality criteria in the presence of random effects:

- **D-Optimality:** $D(X) = |X'V^{-1}X|$ with X being the design matrix and V being the variance-covariance matrix of the response as it was defined above. V depends on Z which is the whole-plot-structutre of the design and on the ratio of σ^2 and σ_w^2.
- **I-Optimality:** $I(X) =$ average prediction variance $= 2^{-N} tr[(X'V^{-1}X)^{-1}B]$ where $B = \int_{x\in[-1,+1]^N} f(x)f'(x)dx$ is called the moments matrix for the experimental region $\chi = [-1, +1]^N$. N is the number of factors.
- **A-Optimality:** $A(X) = trace(X'V^{-1}X)$.

3 Less Restrictive Randomization

Split-plot designs using HTC- and ETC-factors are not always flexible enough to represent all real world problems. The following example comes from an assay development. A crucial part of many assays is the sample preparation. Sample preperation cleans the sample (blood serum, plasma or urine for example) from undesired components like fats or proteins. Getting rid of those parts of the sample helps in terms of reproducibility of measurements and lowers the limit of quantification. For optimizing the sample preparation workflows laboratories often use robots. These robots allow to test a lot of different workflows in a reasonable time. This way it is possible to test hundreds of different workflows over night.

Typical DoEs for finding a optimal sample preparation workflow might include factors like **pH**-values, **incubation times**, **incubation temperatures**, usage of different **types of solutions**, etc. The authors where facing a problem with the different **types of solutions** in a workflow optimization DoE. As part of the optimization 6 different types of solvents shall be testet. These solvents are stored in bottles and need to be placed on the robot. The robot will automatically pick the right solution for any given experiment. But only 4 different bottles can be stored on the robot due to space limitations.

It is possible to work with 4 different solutions and have a completely randomized design - even though one might argue that using always the same bottles of solutions is a violation of the assumption of independence. The latter is ignored for the moment as we can very well assume that the product quality of the solutions is very stable. The robot can work with four solutions completely independently. When a fifth solution is used in the DoE, it becomes more complicated. Now the robot has to be stopped and the operator will exchange one or more bottles before the robot is able to continue it's work.

This is very inconvenient because it makes it impossible to perform experiments over night without having an operator supervising the robot at all time. A completley randomized DOE might lead to a workflow like in table 3.

Especially for larger DOEs using a completly randomized design would be very inconvenient. It removes most of the benefits that come from automated experimentation as a lot of user interaction with the robot is required. Treating the factor **type of solution** as a HTC-factor is an alternative (see table 4).

Run	Solution	Run	Solution
1	A	10	D
2	B	11	E
3	A	12	A
4	C	13	B
5	D	Change Bottles	
Change Bottles		14	C
6	E	15	D
7	A	16	D
8	B	17	C
9	C	18	A
Change Bottles		...	...

Table 3: Completely Randomized Design

Whole Plot	Solution	Whole Plot	Solution
1	A	3	D
1	A	3	D
1	A	3	D
1	A	3	D
Change Bottles		Change Bottles	
2	C	4	B
2	C	4	B
2	C	4	B
2	C	4	B
Change Bottles		...	...

Table 4: Split Plot Design

Following this approach we can reduce the work for the operator but there are two problems:

1. **Randomization:** Split-plot designs are always a nod to practicality but all split-plot designs restrict the randomization of the DoE. This makes split-plots vulnerable to effects due to non-controlled factors. For the given use case it would be possible to do better in terms of randomization compared to a split-plot design.

2. **Correlation structure:** One advantage of split-plot-designs is that a correlation between experiments inside of one whole plot is accepted. For the given use case this structure does not really fit the problem. Rather than stopping the robot after each whole plot it would be much

more desireable to stop the robot after the fourth whole plot is finished. Thus the correlation structure is different than it would be assumed by the split-plot-design-setup.

A better structured design for the given problem would look like this:

Whole Plot	Solution	Whole Plot	Solution
1	A	3	A
1	B	3	B
1	C	3	C
1	D	3	E
Change Bottles		Change Bottles	
2	E	4	F
2	F	4	E
2	A	4	C
2	B	4	A
Change Bottles		...	...

Table 5: Semi Split Plot Design

All runs are grouped similarly like in a split-plot-design. The restriction to randomization is different though. In a split-plot-design there is at least one hard-to-change-factor that is hold constant in each whole plot. Now there is one factor - we will call it **semi-hard-to-change** (SHTC) - that is not fixed to one level inside of each whole plot. Instead the factor levels can change from one run to the next. But rather than using any of the possible factor levels, the randomization only allows to use four out of the six possible factor levels in each group.

It would not be wrong to think about this setup as a design using random blocks. Each random block contains experiments that can be performed without operator interaction. But here we introduce an additional restriction that ensures that only a given number of different settings of the SHTC-factor are used inside of each block.

4 An Algorithm to Generate Optimal Semi-Split-Plot Designs

There is a wide variety of algorithms to produce optimal designs. The Fedorov algorithm (Miller and Nguyen [10]) is probably one of the more popular ones. JMP as an example for a commercial DoE software uses an coordinate-exchange algorithm (Meyer and Nachtsheim [11]). The R-package rospd uses a modified version of Jone's and Goos' [6] candidate-set-free algorithm to create (D-) optimal semi-split-plot designs. The algorithm and the modifications are described in the following.

4.1 Prerequisites

To use the algorithm the following information has to be provided:

- **Factors:** The algorithm can handle continuous or categorical factors that are easy-, hard- or semi-hard-to-change.
- **Whole-Plot-Structure:** If there are any hard- or semi-hard-to-change-factors (SHTC) a matrix representing the whole plot structure needs to be defined. This matrix defines the whole plots of the design and therfore when and how often each HTC- or SHTC-factor can be changed.
- **SHTC-Group-Size:** This integer defines how many different settings of a SHTC-factor can be used in one whole plot.
- **Number of runs:** The number of experiments done for the DoE.
- **Constraints:** Possible constraints to the factors space can be defined.
- **Optimality Criterion:** The algorithm works with any optimality criterion that expresses the quality of a design as a number.

4.2 Initial Random Design

The algorithm starts by generating a random initial design table. The design table is a data frame containing one combination of factor settings for each run of the design. The initial design uses random factor levels for each cell in the data table. It respects all restrictions like restrictions to the randomization and constraints to the factor space. This includes a possible SHTC-structure as well.

For the given example the initial random design might look like table 6.

solvent	pH	time	solvent	pH	time
A	12	10	B	3	10
A	12	15	B	12	10
A	7.50	20	B	7.50	10
A	7.50	10	B	12	15
A	7.50	10	B	12	10
D	3	15	C	3	20
D	3	20	C	12	20
D	12	15	C	7.50	15
D	7.50	20	C	12	15
D	12	15	C	7.50	15

Table 6: Initial Random Design

4.3 Updating the design

The algorithm starts with a random initial design and updates cells of that design table sequentially to improve the optimality of the design. Therefore an iteration matrix (table 7) is generated. The iteration matrix specifies in which order the elements of the design are updated.

solvent	pH	time	solvent	pH	time
1	2	7	23	24	29
1	3	8	23	25	30
1	4	9	23	26	31
1	5	10	23	27	32
1	6	11	23	28	33
12	13	18	34	35	40
12	14	19	34	36	41
12	15	20	34	37	42
12	16	21	34	38	43
12	17	22	34	39	44

Table 7: Iteration Matrix

Each index number in the iteration matrix represents one update step. In the first step all cells of the design table are updated for which the iteration matrix equals 1. In the current example this is the first whole plot of the factor solvent. The second step updates the first row of the factor pH, the second step updates the second row of the factor pH and so on.

The iteration matrix is structured in a way to first update all cells in the first whole plot. The first whole plot is defined by the factor that is hardest to change - that is the factor with the smallest number of changes. Inside of each whole plot the algorithm updates each cell beginning with HTC- and SHTC-factors ordered by the number of changes from few changes to many changes. ETC-factors are updated last.

The updating works differently depending on the type of the factor:

- **ETC-Factors** are updated one cell at a time. Each possible level for that factor is considered. For continuous factors the user has to specify how many intervals between minimum and maximum factor level will be considered. If replacing the current value in the design table with any of the other possible values improves the optimality of the design, the best value is used.
- **HTC-Factors:** are updated in groups. Each possible value is considered. Instead of updating just one cell, all cells of the current whole plot in the design table are replaced simultaneously with the same value. The value that grants the best optimality for the design is used.
- **SHTC-Factors** are updated in groups defined by the whole plots. Other than for HTC-factors the values for each cell in one whole plot can be different. All possible combinations are tested and the best one in terms of the chosen optimality is used.

Updating a SHTC-factor is laborious. Looking at the first whole plot of the current example there are many possible factor settings. The restriction of the design is to not use more than four of the six possible solvents in one whole plot. There are $\frac{m!}{(m-k)!} = \frac{6!}{(6-4)!} = 15$ possible combinations to pick four solvents out of the six.

Here the parameters m and k are:

- m: the overall number of levels of the SHTC-factor
- k: the number of different levels that can be used in one whole plot.

For each of the 15 sets of solvents there are $m^{n_w} = 4^5 = 1024$ possible combinations with n_w being the size of the whole plot. Thus it requires calculating the optimality of $15 * 1024 = 15360$ designs to determine the best settings for the SHTC-factor **solvent** just for the first whole plot.

A faster but less comprehensive way of updating SHTC-factors was implemented as well. Here all elements of the SHTC-factor in one whole plot are updated sequentially. This is the same approach that is used for ETC-factors with the exception that not all possible factor levels are used but rather a subset defined by the number of possible settings inside of one whole plot. The sequential updating is done for all possible subsets of factor levels out of all factor levels of the SHTC-factor.

By default the algorithm will use one random start and update each element of the design table up to 25 times. At the end of each step the algorithm checks if any cell of the desing was updated. If no changes where made the algorithm converged and stops.

5 Using the rospd-Package

The rospd-package (**R** **O**ptimal **S**plit **P**lot **D**esigns) is an implementation of the previously described algorithm. This chapter shows how to use the package function to create optimal semi-split-plot designs. The core part of the package are the two classes *doeFactor* and *doeDesign*. The *doeDesign* class collects all information required to generate an optimal design. This includes a list of *doeFactors* that should be investigated in the DoE.

5.1 The doeFactor class

Objects of class *doeFactor* represent factors in the DoE. *doeFactor* is a S4 class with the following slots:

- **name:** The name of the factor as a character.
- **type:** Either *"continuous"* or *"categorical"*. Continuous factors are numeric factors that could theoretically take any real number in a given range. Categorical factors use either character or numeric values. They are interpreted as nominal variables.
- **levels:** For continuous factors the user specifies the range of the factor as a numeric vector containing minimum and maximum. For categorical factors a vector with all possible factor levels - either numeric or characters - is required.
- **number.levels:** The number of levels is only required for continuous factors. This integer represents how many different levels are used during the design generation. E.g. for $number.levels = 3$ and $levels = c(100, 300)$ the factor levels that are used during the design updating are 100, 200 and 300. For $number.levels = 5$ and $levels = c(100, 300)$ the possible factor levels are 100, 150, 200, 250 and 300. The number of levels have a strong impact on the runtime of the algorithm.
- **changes:** This defines if a factor is *"easy"*-, *"hard"*- or *"semi.hard"*-to-change.
- **semi.htc.group.size:** This is only required for SHTC-factors. This integer defines how many different factor levels can be used inside of one whole plot.

The factors of the current example can be declared the following way:

```
# pH - an ETC continuous factor
phFactor <- new("doeFactor",
                name="pH",
                type="continuous",
                levels = c(3,12),
                number.levels = as.integer(2),
                changes="easy"
                )

# time - an ETC continuous factor
timeFactor <- new("doeFactor",
                name="time",
                type="continuous",
                levels = c(10,20),
                number.levels = as.integer(2),
                changes="easy"
                )

# solvent - a SHTC categorical factor
solventFactor <- new("doeFactor",
                     name="solvent",
                     type="categorical",
                     levels=c("A", "B", "C",
                        "D", "E", "F"),
                     changes="semi.hard",
                     semi.htc.group.size=
                      as.integer(4)
                     )
```

5.2 The doeDesign class

The *doeDesign*-class is the heart of rospd. It stores all information required to generate an optimal design and will contain the final design table afterwards as well. This way any *doeDesign*-object can serve as a documentation of how the design was created. *doeDesign* is a S4-class. As it uses a lot of different slots only the required ones will be discussed here.

- **factors:** A list of all factors that are used in the DoE as objects of class *doeFactor*. This list can contain one or more items.
- **whole.plot.structure:** The whole plot structure is a data.frame representing when changes can be made for all HTC- and SHTC-factor. This data.frame needs to contain one column for each HTC- and SHTC-factor using the factor names as column names. The whole plots of each factor are represented by index numbers starting at 1 for each factor.
- **number.runs:** The number of experiments as *integer*.
- **design.model:** The design model as a *formula*. The formula is supposed to contain only the fixed effects part of the model as the random effects part is already defined by the **whole.plot.structure**.
- **optimality.function:** a *function* that can be used to calculate the optimality of a given design. The algorithm will use the *doeDesign*-object as the only argument of the function. Making use of the slots **design.matrix**, **design.model** and **variance.ratio** of the class **doeDesign** should allow to calculate most optimality criteria. The rospd package provides a predefined function to calculate D-Optimality. The function is a wrapper of the C-implementation of the skpr-package [12].
- **variance.ratio:** The expected ratio of σ^2 (residual variance) and σ_w^2 (between whole plot variance). This is required for the calculation of some optimality criteria. By default a variance ratio of 1 is used.

There is a function **GenerateNewDoEDesign** that should be used to initialize a new *doeDesign*-object. A *doeDesign*-object for the current example might look like this:

```
doeSpecification <- GenerateNewDoeDesign(
    factors = list(
     solventFactor,
     phFactor,
     timeFactor),
    whole.plot.structure =
     data.frame(solvent=
     rep(1:6, each=10)),
    number.runs = as.integer(60),
    design.model = ~solvent+pH+time,
    optimality.function = DOptimality
)
```

5.3 Generating Optimal Designs

The function **GenerateOptimalDesign** generates an optimal design for the given specification. The function uses the following arguments:

- **doeSpec:** The doe specification as an object of class *doeDesign*.

- **random.start:** This argument controls how many different random starting designs are used. The default setting is 1. The function will return a list of one doeDesign for each random start. The designs in this list will be sorted from best to worst in terms of their optimality.

- **max.iter:** The maximum number of iterations. One iterations updates all cells of the design table once.

The following code generate a D-optimal semi split-plot design.

```
optimalDesign <-
  GenerateOptimalDesign(doeSpecification)
```

Table 8 shows a subset of the generated design.

6 Design Evaluation

This section evaluates the quality of the generated design. For that purpose we will compare the semi-split-plot design with three alternative designs. Further more multiple semi-split-plot designs with different numbers of whole plots are compared to each other.

6.1 Sample Preparation Robot Example

The previously generated semi-split-plot design will be compared to three alternative solutions:

- a **completely randomized design** (CRD)

- a standard **split-plot** design using 12 whole plots each of size 5.

- a standard **split-plot** design using 8 whole plot. To end up with 60 runs, 4 of the whole plots use size 7 and 4 of the whole plots are of size 8.

The rationale here is to have the CRD as the best case scenario. As the CRD does not include any restrictions on randomization and assumes completely independent runs it will always provide the best statistical properties. As discuss this design would not be chosen in the real world.

The split-plot design with 12 whole plots is not that realistic as well, as it involves much more interactions of the operator with the machine than we would like to do - twelve instead of only 6 for the semi-split-plot design. A split-plot design with eight whole plots is more realistic but on the lower end in terms of the number of whole plots. This limits the precision for the whole-plot-to-whole-plot-variance estimator.

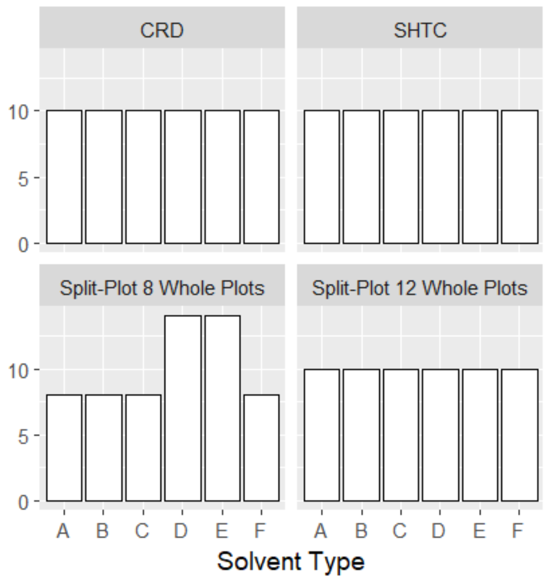

Figure 3: Frequencies of Solvent Types

Figure 3 shows that the CRD, the semi-split-plot and the split-plot design using 12 whole plots are prefectly balanced. For the split-plot design using eight whole plots it is not possible to achieve balance for the factor solvent.

The aliasing heatmaps in figure 5 show that there are only minor differences when comparing the designs by their inter-factor correlations. Blue cells in the figure represent low correlations while red cells

Whole Plot	Solvent	pH	Time	Whole Plot	Solvent	pH	Time
1	B	12	20	5	D	12	20
1	B	3	10	5	D	3	10
1	F	3	20	5	E	3	10
1	C	12	10	5	E	12	10
1	C	3	20	5	E	12	20
1	C	12	20	6	A	12	20
1	D	3	10	6	A	3	10
1	F	12	10	6	A	3	20
1	D	12	10	6	B	3	20
1	F	3	20	6	B	3	10
2	E	12	20	6	B	12	10
2	A	12	20	6	F	12	20
2	A	12	10	6	E	3	20
2	E	3	20	6	E	12	10
2	B	3	20	6	F	12	10

Table 8: Part of the Semi Split Plot Design

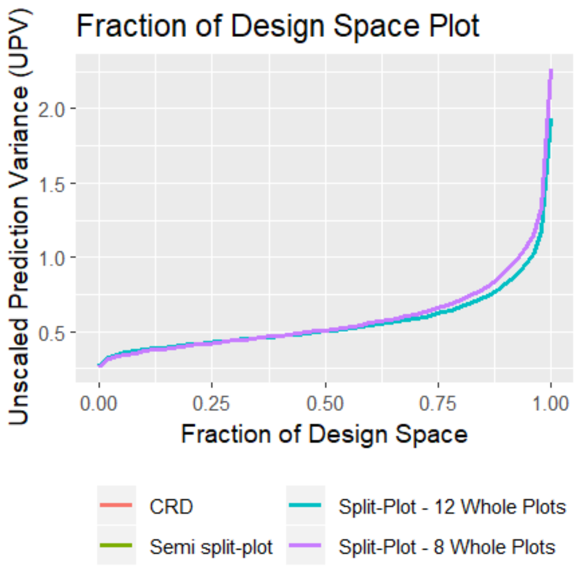

Figure 4: FDS Plot

are representing high correlations. All designs show some correlation in between the effects of the different solvents. The split-plot design with 8 whole plots shows a little less homogeneous correlation structure of the solvent effects. This is the result of imposing very strict restrictions to randomization.

The whole plot (**wp_solvent** in the graph) in case of the semi split-plot design has much lower correlation with the solvent effects compared to the traditional split-plot designs.

The FDS-plot in figure 4 shows that the traditional split-plot designs with 8 whole plots performs slightly worse than the other designs in terms of prediction variance. All other designs show equal prediction variance represented by the bottom line in the graph.

Overall the semi-split-plot design is mostly equal to the CRD in terms of balance, aliasing and predictive quality. When comparing it to the split-plot designs much lower correlations of whole plot and solvent effects can be observed. Considering the practical implications of the different designs the semi-split-plot is by far the prefered solution for the given problem.

6.2 Required Number of Whole Plots

The previous evaluation showed that split-plot designs with SHTC-factors are not worse in terms of statistical quality than CRDs or traditional split-plots with a sufficient number of whole plots. When dealing with HTC-factors the number of whole plots

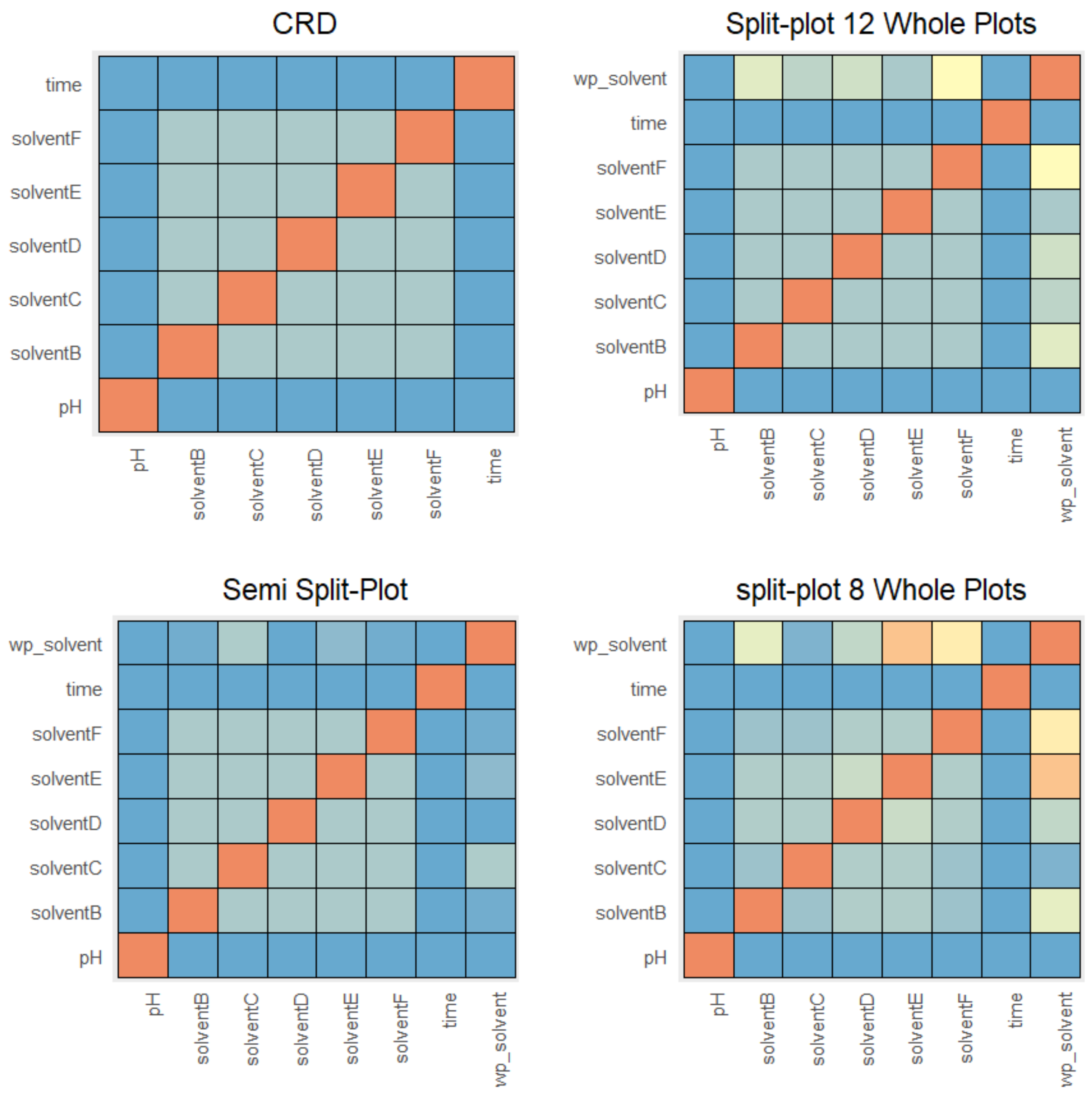

Figure 5: Correlations of Factors and Whole Plots

is often the critical parameter. As SHTC-factors allow much more flexibility compared to true HTC-factors we are able to reduce the required number of whole plots for the given example.

In traditional split-plot designs the required number of whole plots depends first of all on the number of HTC-factor levels. There need to be at least $l + 1$ wholeplots to be able to estimate the factor effects and the in between whole plot variation. With l being the number of levels of the HTC-factor.

For SHTC-factors this is much less of a problem, as changing SHTC-factor levels inside of the whole plots is possible. Thus the minimum number of whole plots is only depending on the overall number of factor levels and the number of factor levels that can be used inside of one whole plot. All factor levels have to be used at least once. At the same time the correlation of whole plots and all other factors should be minimized to be able to estimate the model effects with high precision and avoid aliasing of whole plot effects and SHTC-factor effects.

In the following we compare semi split-plot designs for the previous example with varying number of whole plots.

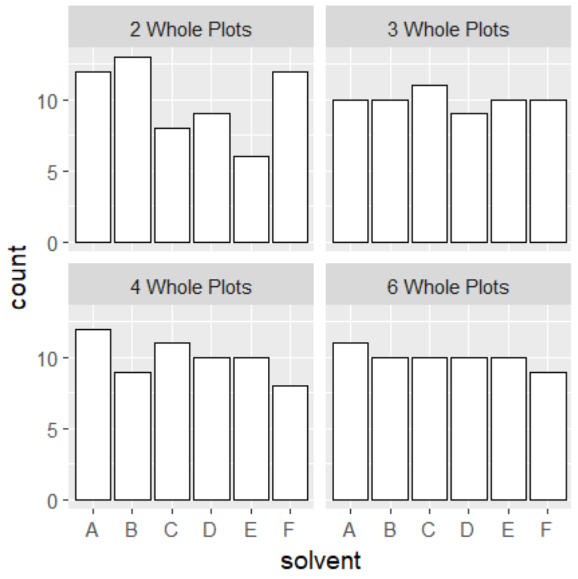

Figure 6: Distribution of Solvent Types for Varying Numbers of Whole Plots

Figure 6 shows that the algorithm is not able to find a solution granting perfect balance in terms of the SHTC-factor when using only two whole plots. This problem is less severe when using three whole plots instead of just two. That statement is only true for the given example. In general the minimum number of whole plots to achieve perfect balance and minimize correlation of whole plots and the SHTC-factor depends on the size of the whole plots and on the number of possible levels in each whole plot.

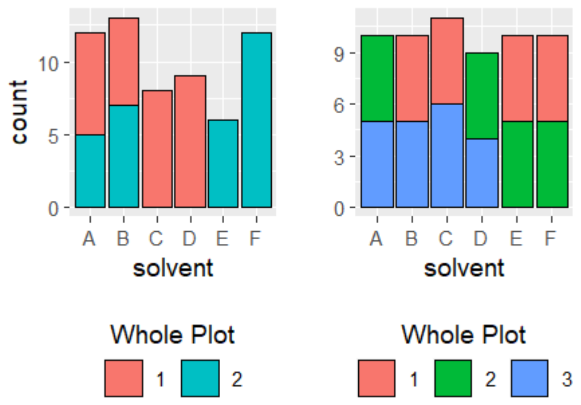

Figure 7: Distribution of Solvent Types for 2 and 3 Whole Plots

The design with two whole plots shows some correlation of solvent effects and the whole plot. This correlation is reduced with every additional whole plot introduced. Going from two whole plots to three whole plots reduces the correlation a lot (see figures 7 and 8). Adding more whole plots does not decrease the correlation by much. This is not a problem as the three whole plot design shows very low correlations anyways.

7 Conclusion and Future Work

We introduced the concept of semi-split-plot designs and semi-hard-to-change factors. An algorithm to generate optimal semi-split-plot designs was proposed. The implementation of this algorithm is available on Github `https://github.com/neuhier/rospd` and will be published on CRAN in the future. The current implementation is done mostly in R and

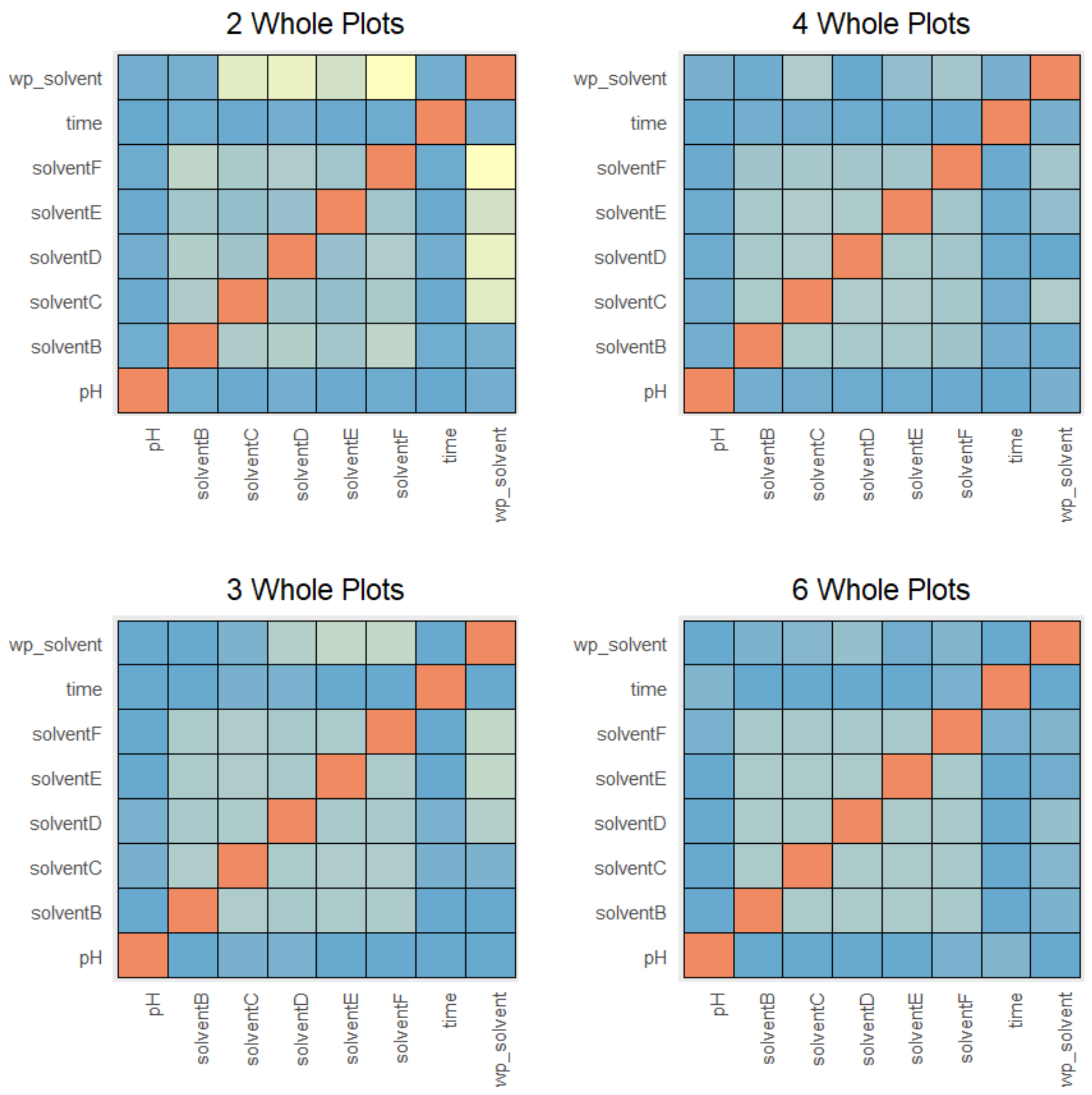

Figure 8: Degree of Aliasing for Different Numbers of Whole Plots

the performance could be improved by changing to a C-implementation.

The comparison of multiple design classes for the described use case showed that semi-split-plot designs can provide statistical qualities that are very comperable to the ones of CRDs. As CRDs are not feasible for this application the comparison to traditional split-plot designs is more relevant. The design evaluation showed that semi-split-plot designs can reduce the correlation of factors and whole plots. Thus they are preferable over split-plot designs with regard of statistical quality and feasibility.

References

[1] R. A. Fisher, *Statistical Methods for Research Workers*, pp. 66–70. New York, NY: Springer New York, 1992.

[2] S. M. Kowalski and K. J. Potcner, "How to recognize a split-plot experiment." `https://onlinecourses.science.psu.edu/stat503/sites/onlinecourses.science.psu.edu.stat503/files/lesson14/recognize_split_plot_experiment/index.pdf`, 11 2003. Accessed on 2018-06-29.

[3] B. Jones and C. J. Nachtsheim, "Split-plot designs: What, why, and how," *Journal of Quality Technology*, vol. 41, no. 4, pp. 340–361, 2009.

[4] B. Jones and P. Goos, *Optimal Design of Experiments*, pp. 277–282. Wiley-Blackwell, 2011.

[5] T. Næ s, A. Aastveit, and N. Sahni, "Analysis of split-plot designs: An overview and comparison of methods," vol. 23, pp. 801 – 820, 11 2007.

[6] B. Jones and P. Goos, "A candidate-set-free algorithm for generating d-optimal split-plot designs," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 56, no. 3, pp. 347–364, 2007.

[7] SAS, "Split plot designs with different numbers of whole plots." `https://www.jmp.com/support/help/14/split-plot-designs-with-different-numbers-of-who.shtml`, 06 2018. Accessed on 2018-06-25.

[8] B. Jones and P. Goos, "I-optimal versus D-optimal split-plot response surface designs," Working Papers 2012002, University of Antwerp, Faculty of Applied Economics, Jan. 2012.

[9] T. Hooks, D. Marx, S. Kachman, and J. Pedersen, "Optimality Criteria for Models with Random Effects," *Revista Colombiana de EstadÃ-stica*, vol. 32, pp. 17 – 31, 06 2009.

[10] A. J. Miller and N. Nam-Ky, "A fedorov exchange algorithm for d-optimal design.," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 43, no. 4, p. 669, 1994.

[11] R. K. Meyer and C. J. Nachtsheim, "The coordinate-exchange algorithm for constructing exact optimal experimental designs," *Technometrics*, vol. 37, no. 1, pp. 60–69.

[12] T. Morgan-Wall and G. Khoury, *skpr: Design of Experiments Suite: Generate and Evaluate Optimal Designs*, 2018. R package version 0.49.1.