oapackage Documentation

Pieter Eendebak, Alan Vazquez

Nov 01, 2022

CONTENTS:

The Orthogonal Array package contains functionality to generate and analyse orthogonal arrays, optimal designs and conference designs.

CHAPTER

ONE

INTRODUCTION

Orthogonal arrays, optimal designs and conference designs are important tools for the design of experiments [EN95] [HSS99] [WH09]. The Orthogonal Array package contains functionality to generate and analyse these types of designs. To generate the arrays and designs, the package uses the exhaustive enumeration algorithm of [SEN10] and the optimization algorithm of [ES17]. To analyze the arrays and designs, the package includes a wide variety of relevant statistical and combinatorial criteria. A large collection of orthogonal arrays, optimal designs and conference designs generated with the package are available in the Orthogonal Array package website [Een18].

1.1 Example usage

The Orthogonal Array package can be used to generate and manipulate arrays and designs. Additionally, it can calculate some of their statistical properties. The following example shows how to generate an orthogonal array with 8 runs and 2 factors, and calculate three relevant statistical properties:

Calculate D-efficiency

```
>>> import oapackage
>>> array=oapackage.exampleArray(0) # define an orthogonal array
>>> array.showarray()
array:
  0   0
  0   0
  0   1
  0   1
  1   0
  1   0
  1   1
  1   1
>>> D = array.Defficiency() # calculate the D-efficiency for estimating the interaction␣
↪effects model
>>> array_rank = array.rank() # calculate the rank of the design
>>> print('D-efficiency %f, rank %d' % (D, array_rank) )
D-efficiency 1.000000, rank 2
>>> gwlp = array.GWLP() # calculate the generalized word length pattern
>>> print('Generalized wordlength pattern: %s' % (gwlp,) )
Generalized wordlength pattern: (1.0, 0.0, 0.0)
```

The statistical properties of the arrays and designs are introduced in *Properties of designs*.

1.2 Interfaces

The Orthogonal Array package has interfaces in C++ and Python for generating, manipulating and analyzing all the types of arrays and designs. In this documentation, you will find references to both the Python and the C++ interface. The package also includes several command line tools.

For the generation of optimal designs [ES17], the Orthogonal Array package has also a Matlab interface; see the documentation README.Matlab.md.

1.3 License

The code is available under a BSD style license; see the file LICENSE for details. If you use this code or any of the results, please cite this program as follows:

- OApackage: A Python package for generation and analysis of orthogonal arrays, optimal designs and conference designs, P.T. Eendebak, A.R. Vazquez, Journal of Open Source Software, 2019
- *Complete Enumeration of Pure-Level and Mixed-Level Orthogonal Arrays*, E.D. Schoen, P.T. Eendebak, M.V.M. Nguyen, Volume 18, Issue 2, pages 123-140, 2010.

1.4 Acknowledgements

The code and ideas for this package have been contributed by Eric Schoen, Ruben Snepvangers, Vincent Brouerius van Nidek, Alan Roberto Vazquez and Pieter Thijs Eendebak.

1.5 Installation

The packge is continously tested on Linux and Windows. The Python interface is available from the Python Package Index. The package can be installed from the command line using pip:

```
$ pip install OApackage
```

The source code for the package is available on https://github.com/eendebakpt/oapackage. The command line tools use a cmake build system. From the command line, type the following:

```
$ mkdir -p build
$ cd build
$ cmake ..
$ make
$ make install
```

This creates the command line utilities and a C++ library.

To compile the Python interface use

```
$ python setup.py build
$ python setup.py install --user
```

The Python interface requires Numpy [TheScipycommunity12], Matplotlib [Hun07] and Swig. The code has been tested with Python 3.6, 3.7 and 3.8.

The R interface to the optimal design functionality of the package is available from CRAN. For the Matlab and Octave interface to the optimal design functionality see the file README.Matlab.md.

1.6 Related sites of orthogonal arrays

There are several related sites available online which include collections of orthogonal arrays. For instance, the website of Neil Sloane [Slo14], the website of Hongquan Xu [Xu18], the SAS website managed by Warren Kuhfeld [Kuh18], and the R package _DoE.base_ [GAX18] include lists and surveys of attractive orthogonal arrays gathered from different sources.

CHAPTER

TWO

EXAMPLE NOTEBOOKS

This section contains several examples for generating, manipulating and analyzing arrays and designs using the Orthogonal Arrays package. The examples are shown using Jupyter notebooks, which can be found on github https://github.com/eendebakpt/oapackage/tree/master/docs/examples.

2.1 Example script for Python interface to Orthogonal Array package

Pieter Eendebak pieter.eendebak@gmail.com

[1]:
```
import numpy as np
import oapackage

print("oapackage version: %s" % oapackage.version())
```

```
oapackage version: 2.6.0
```

Load an example array.

[2]:
```
array = oapackage.exampleArray(0)
array.showarray()
```

```
array:
  0   0
  0   0
  0   1
  0   1
  1   0
  1   0
  1   1
  1   1
```

Calculate properties of the array such as the D-efficiency for the main effects model, the generalized word length pattern and the rank.

[3]:
```
print("D-efficiency %f, rank %d" % (array.Defficiency(), array.rank()))
print("Generalized wordlength pattern: %s" % str(array.GWLP()))
```

```
D-efficiency 1.000000, rank 2
Generalized wordlength pattern: (1.0, 0.0, 0.0)
```

Calculate the generalized word length pattern for another example array.

[11]:
```
array = oapackage.exampleArray(11)
print("Generalized wordlength pattern: %s" % oapackage.oahelper.gwlp2str(array.GWLP()))
```

```
Generalized wordlength pattern: 1.00,0.02273,0.03926,0.5434,0.8244,2.217,1.043,0.126,0.
↪002066
```

2.1.1 Indexing

The `array_link` object can be indexed as a normal array.

[12]:
```
array[0:5, 2:3]
```

[12]:
```
array_link: 5, 1
```

[14]:
```
array[0:5, 2:4].showarray()
```

```
array:
  1   1
  0   1
  1   0
  0   0
  1   1
```

[15]:
```
print(array[0, 2])
```

```
1
```

2.1.2 Numpy

We can convert between Numpy arrays and `array_link` objects. Note that an `array_link` is always integer valued.

[19]:
```
X = (4 * np.random.rand(20, 10)).astype(int)
array = oapackage.array_link(X)
array.showarraycompact()
```

```
2003200102
0322120001
3030110033
0311131120
2200100200
2112333121
3110300211
3302313313
2331031302
1212232322
3003330322
0312311331
3222200313
2001123302
2222111302
0020322112
0101223211
```

(continues on next page)

(continued from previous page)

```
0223300100
1232030100
0320032113
```

Convert from `array_link` back to Numpy array.

[20]:
```
X2 = np.array(array)
```

2.2 Example arrays

The package contains several example arrays and designs. They can be retrieved using the method `exampleArray`.

[4]:
```
import oapackage
al = oapackage.exampleArray(-1, 1)
```

```
exampleArray 0: array in OA(8,2, 2^2)
exampleArray 1: array 3 in OA(16, 2, 2^5)
exampleArray 2: array 6 in OA(16, 2, 2^6)
exampleArray 3: array ? in OA(32, 3, 2^7)
exampleArray 4: array 4 in OA(16, 2, 2^7)
exampleArray 5: array 0 in OA(24, 2, 4 3 2^a)
exampleArray 6: array in OA(4, 2, 2^a)
exampleArray 7: array 0 in OA(4, 2, 2^a)?
exampleArray 8: array in OA(40, 3, 2^7)
exampleArray 9: array in A(40, 2^7), D-optimal
exampleArray 10: array in OA(9, 3^3)
exampleArray 11: D-optimal array in OA(44, 2^8)
exampleArray 12: even-odd array OA(64, 2^13)
exampleArray 13: array in OA(25, 2^5)
exampleArray 14: design in D(28, 2^5), D-efficiency is low
exampleArray 15: design in D(56, 2^10), D-efficiency is low
exampleArray 16: array in OA(32, 2, 2^5)
exampleArray 17: unique array in OA(64, 4, 2^7)
exampleArray 18: conference matrix of size 16, 7
exampleArray 19: conference matrix of size 4, 3
exampleArray 20: first LMC-0 double conference matrix in DC(24,3)
exampleArray 21: second LMC-0 double conference matrix in DC(16,4)
exampleArray 22: LMC-0 double conference matrix in DC(32,4)
exampleArray 23: LMC-0 double conference matrix in DC(32,6)
exampleArray 24: design in OA(64, 3, 2^16) (even-odd)
exampleArray 25: design in OA(64, 3, 2^16) (even-odd)
exampleArray 26: design in OA(64, 3, 2^16) (even-odd)
exampleArray 27: design in OA(64, 3, 2^16) (even-odd)
exampleArray 28: conference design in C(4, 3) in LMC0 form
exampleArray 29: conference design in C(4, 3)
exampleArray 30: conference design in C(8,4)
exampleArray 31: conference design in C(8,4)
exampleArray 32: first double conference design in DC(18,4)
exampleArray 33: second double conference design in DC(18,4)
exampleArray 34: third double conference design in DC(18,4)
exampleArray 35: first double conference design in DC(20,4)
```

(continues on next page)

(continued from previous page)

```
exampleArray 36: second double conference design in DC(20,4)
exampleArray 37: third double conference design in DC(20,4)
exampleArray 38: LMC0 conference design in C(30,3)
exampleArray 39: first LMC0 conference design in C(8,6)
exampleArray 40: first conference design in C(14, 5)
exampleArray 41: second conference design in C(14, 5)
exampleArray 42: third conference design in C(14, 5)
exampleArray 43: 2x2 array with zeros and a singe value -1
exampleArray 44: D-optimal strength 3 ortogonal array in OA(40,3, 2^7)
exampleArray 45: first conference design in C(20,8)
exampleArray 46: second conference design in C(20,8)
exampleArray 47: third conference design in C(20,8)
exampleArray 48: last conference design in C(20,8)
exampleArray 49: array 4347 C(20,8)
exampleArray 50: array 4506 C(20,8)
exampleArray 51: first array in C(12,4)
exampleArray 52: second array in C(12,4)
exampleArray 53: third array in C(12,4)
exampleArray 54: root array in OA(6,2,3^1 2^1)
exampleArray: no example array with index 55 exists
```

Select an example array and show it.

[3]:
```
al=oapackage.exampleArray(2, 1)
al.showarray()
```

```
exampleArray 2: array 6 in OA(16, 2, 2^6)
array:
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   1   1   1
  0   0   0   1   1   1
  0   1   1   0   0   1
  0   1   1   0   1   0
  0   1   1   1   0   1
  0   1   1   1   1   0
  1   0   1   0   0   1
  1   0   1   0   1   1
  1   0   1   1   0   0
  1   0   1   1   1   0
  1   1   0   0   1   0
  1   1   0   0   1   1
  1   1   0   1   0   0
  1   1   0   1   0   1
```

2.3 Example to write and read files with arrays from disk

The package can write arrays and designs to a custom file format. There is a text based format (specified by `oapackage.ATEXT`) and a binary format (specified by `oapackage.ABINARY`). See the online documentation for details on the file formats.

[1]:
```
import oapackage
import tempfile
```

Create a list of two example arrays.

[2]:
```
lst = [oapackage.exampleArray(32), oapackage.exampleArray(33)]
print(lst)
```

```
[array_link: 18, 4, array_link: 18, 4]
```

Write the two arrays to a file on disk.

[3]:
```
filename = tempfile.mktemp(".oa")
_ = oapackage.writearrayfile(filename, lst, oapackage.ATEXT)
```

Display information about the file written.

[4]:
```
oapackage.oainfo(filename)
```

```
file C:\Users\EENDEB~1\AppData\Local\Temp\tmpxc4dnsl6.oa: 18 rows, 4 columns, 2 arrays,␣
↪mode text, nbits 0
```

The arrays can be read from disk again using the command `readarrayfile`.

[6]:
```
lst2 = oapackage.readarrayfile(filename)
print(lst2)
```

```
(array_link: 18, 4, array_link: 18, 4)
```

The first array in the list that was read from disk is:

[7]:
```
lst2[0].showarray()
```

```
array:
  0   0   0   0
  0   0   0   0
  1   1   1   1
  1   1   1   1
  1   1  -1  -1
  1   1  -1  -1
  1  -1   1  -1
  1  -1   1  -1
  1  -1  -1   1
  1  -1  -1   1
 -1   1   1  -1
 -1   1   1  -1
 -1   1  -1   1
 -1   1  -1   1
 -1  -1   1   1
 -1  -1   1   1
```

(continues on next page)

(continued from previous page)

```
 -1  -1  -1  -1
 -1  -1  -1  -1
```

2.4 Enumerate orthogonal arrays

The Orthogonal Array package can completely enumerate all orthogonal arrays of a specified class. In this notebook, we enumerate specific classes of three-level orthogonal arrays and mixel-level orthogonal arrays.

First, we specify the class of three-level orthogonal arrays to enumerate. For example, we consider three-level orthogonal arrays of strength 2 with 27 runs and 8 factors.

[1]:
```
import oapackage
run_size = 27
strength=2
number_of_factors=8
factor_levels = 3
arrayclass=oapackage.arraydata_t(factor_levels, run_size, strength, number_of_factors)
print(arrayclass)
```

```
arrayclass: N 27, k 8, strength 2, s {3,3,3,3,3,3,3,3}, order 0
```

Second, we create the root array as the starting point of our enumeration.

[2]:
```
ll2=[arrayclass.create_root()]
ll2[0].showarraycompact()
```

```
00
00
00
01
01
01
02
02
02
10
10
10
11
11
11
12
12
12
20
20
20
21
21
21
22
```

(continues on next page)

(continued from previous page)

```
22
22
```

Third, extend the root array. It is also possible to extend a list of arrays.

[3]:
```
list3columns = oapackage.extend_arraylist(ll2, arrayclass)
print('extended to %d arrays with 3 columns' % len(list3columns))
list4columns = oapackage.extend_arraylist(list3columns, arrayclass)
print('extended to %d arrays with 4 columns' % len(list4columns))
```

```
extended to 9 arrays with 3 columns
extended to 711 arrays with 4 columns
```

It is also possible to extend selected arrays from a list.

[4]:
```
ll = oapackage.extend_arraylist(list4columns[0:8], arrayclass)
print('extended first 2 arrays to %d arrays with 5 columns' % len(ll))
```

```
extended first 2 arrays to 189 arrays with 5 columns
```

By adding one column at a time we can enumerate all three-level orthogonal arrays of strength 2 with 27 runs and 8 factors. The total computation time for this would be a couple of hours.

2.4.1 Mixed-level orthogonal arrays

The package can also enumerate mixed-level orthogonal arrays. For instance, consider enumerating all 16-run strength-2 orthogonal arrays with one four-level factor and nine two-level factors.

[9]:
```
run_size = 16
strength=2
number_of_factors = 10
factor_levels=[4,2,2,2,2,2,2,2,2,2]
arrayclass=oapackage.arraydata_t(factor_levels, run_size, strength, number_of_factors)
print(arrayclass)
```

```
arrayclass: N 16, k 10, strength 2, s {4,2,2,2,2,2,2,2,2,2}, order 0
```

Create the root array as the starting point of our enumeration.

[10]:
```
al=arrayclass.create_root()
al.showarraycompact()
```

```
00
00
01
01
10
10
11
11
20
20
21
21
```

(continues on next page)

(continued from previous page)

```
30
30
31
31
```

For these arrays, we can extend a single array or lists of arrays.

[14]:
```
array_list=[arrayclass.create_root()]
array_list_3columns=oapackage.extend_arraylist(array_list, arrayclass)
array_list_4columns=oapackage.extend_arraylist(array_list_3columns, arrayclass)
print('extended to %d arrays with 3 columns' % len(array_list_3columns))
print('extended to %d arrays with 4 columns' % len(array_list_4columns))
```

```
extended to 3 arrays with 3 columns
extended to 10 arrays with 4 columns
```

Finally, enumerate all 16-run strength-2 orthogonal arrays with one four-level factor and nine two-level factors.

[15]:
```
arrays = array_list_4columns
for extension_column in range(5, number_of_factors+1):
    extensions=oapackage.extend_arraylist(arrays, arrayclass)
    print('extended to %d arrays with %d columns' % (len(extensions), extension_column))
    arrays=extensions
```

```
extended to 28 arrays with 5 columns
extended to 65 arrays with 6 columns
extended to 110 arrays with 7 columns
extended to 123 arrays with 8 columns
extended to 110 arrays with 9 columns
extended to 72 arrays with 10 columns
```

2.4.2 Notes

- The numbers of isomorphism classes for various types of classes can be found at the webpage series of orthogonal arrays.
- For larger number of arrays the command line tools are more convenient and more memory efficient.

2.5 Generate orthogonal arrays with high D-efficiency

This notebook contains example code from the article Two-level designs to estimate all main effects and two-factor interactions by Eendebak, P. T. and Schoen, E. D. This example shows how to generate orthogonal arrays with a high D-efficiency in a reasonable amount of time (< 1 minute). For more results and details, see the paper.

Generate a D-optimal orthogonal array of strength 2 with 32 runs and 7 factors.

[12]:
```
import numpy as np
import matplotlib.pyplot as plt
import oapackage
%matplotlib inline
```

[2]:
```python
run_size = 32
number_of_factors=7
factor_levels=2
strength=2
nkeep=24 # Number of designs to keep at each stage

arrayclass=oapackage.arraydata_t(factor_levels, run_size, strength, number_of_factors)
print('In this example we generate orthogonal arrays in the class: %s' % arrayclass)
```

```
In this example we generate orthogonal arrays in the class: arrayclass: N 32, k 7,␣
↪strength 2, s {2,2,2,2,2,2,2}, order 0
```

First, generate orthogonal arrays with the function `extend_arraylist`. Next, keep the arrays with the best D-efficiency.

[7]:
```python
arraylist=[arrayclass.create_root()]

#%% Extend arrays and filter based on D-efficiency
options=oapackage.OAextend()
options.setAlgorithmAuto(arrayclass)

for extension_column in range(strength+1, number_of_factors+1):
    print('extend %d arrays with %d columns with a single column' % (len(arraylist),␣
↪arraylist[0].n_columns) )
    arraylist_extensions = oapackage.extend_arraylist(arraylist, arrayclass, options)

    # Select the best arrays based on the D-efficiency
    dd = np.array([a.Defficiency() for a in arraylist_extensions])
    ind = np.argsort(dd)[::-1][0:nkeep]
    selection = [ arraylist_extensions[ii] for ii in ind]
    dd=dd[ind]
    print('  generated %d arrays, selected %d arrays with D-efficiency %.4f to %.4f' %␣
↪(len(arraylist_extensions), len(ind), dd.min(), dd.max() ) )

    arraylist = selection
```

```
extend 1 arrays with 2 columns with a single column
  generated 5 arrays, selected 5 arrays with D-efficiency 0.0000 to 1.0000
extend 5 arrays with 3 columns with a single column
  generated 19 arrays, selected 19 arrays with D-efficiency 0.0000 to 1.0000
extend 19 arrays with 4 columns with a single column
  generated 491 arrays, selected 24 arrays with D-efficiency 0.9183 to 1.0000
extend 24 arrays with 5 columns with a single column
  generated 2475 arrays, selected 24 arrays with D-efficiency 0.9196 to 1.0000
extend 24 arrays with 6 columns with a single column
  generated 94 arrays, selected 24 arrays with D-efficiency 0.7844 to 0.8360
```

Show the best array from the list of D-optimal orthogonal arrays.

[20]:
```python
selected_array = selection[0]
print('Generated a design in OA(%d, %d, 2^%d) with D-efficiency %.4f' % (selected_array.
↪n_rows, arrayclass.strength, selected_array.n_columns, dd[0] ) )
print('The array is (in transposed form):\n')
selected_array.transposed().showarraycompact()
```

```
Generated a design in OA(32, 2, 2^7) with D-efficiency 0.8360
The array is (in transposed form):

00000000000000001111111111111111
00000000111111110000000011111111
00000111000111110001111100000111
00011001011001110110011100011001
00101010101010111010101100101010
01001011010110011101000101101001
01110010001101010111001010100011
```

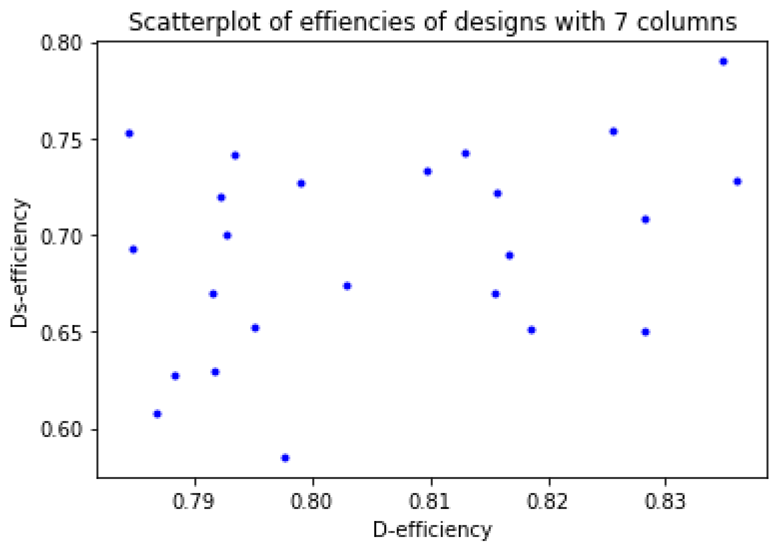
We calculate the D-, D_s- and D_1-efficiencies.

```
[21]: efficiencies = np.array([array.Defficiencies() for array in arraylist])
print(efficiencies)
```

```
[[0.8360354  0.72763273 1.        ]
 [0.83481532 0.78995185 1.        ]
 [0.82829522 0.70849985 1.        ]
 [0.82829522 0.65037401 1.        ]
 [0.82545286 0.75411444 1.        ]
 [0.8185915  0.65157148 1.        ]
 [0.81676643 0.68965719 1.        ]
 [0.81559279 0.72218391 1.        ]
 [0.81557799 0.66966068 1.        ]
 [0.81302642 0.74299714 1.        ]
 [0.80973102 0.73354611 1.        ]
 [0.80286424 0.67455544 1.        ]
 [0.79907785 0.72697978 1.        ]
 [0.79762174 0.58515515 1.        ]
 [0.79514152 0.65199837 1.        ]
 [0.79348419 0.74167976 1.        ]
 [0.7928208  0.70038649 1.        ]
 [0.79225237 0.71961467 1.        ]
 [0.79171981 0.62994172 1.        ]
 [0.79160277 0.66978711 1.        ]
 [0.7883109  0.62733236 1.        ]
 [0.78683679 0.60799604 1.        ]
 [0.78477264 0.69310129 1.        ]
 [0.7843891  0.7533355  1.        ]]
```

Visualize the D-efficiencies using a scatter plot.

```
[19]: plt.plot(efficiencies[:,0], efficiencies[:,1], '.b')
plt.title('Scatterplot of effiencies of designs with %d columns' % arraylist[0].n_
↪columns)
plt.xlabel('D-efficiency')
_=plt.ylabel('Ds-efficiency')
```



nbsphinx-code-borderwhite

2.6 Generate D-efficient designs

This notebook contains example code from the article Two-level designs to estimate all main effects and two-factor interactions by Eendebak, P. T. and Schoen, E. D. This example shows how to generate D-efficient designs with a user-specified optimization function.

[1]:
```
import numpy as np
import oapackage
import oapackage.Doptim

%matplotlib inline
```

Define the class of designs to generate.

[2]:
```
run_size = 40
number_of_factors = 7
factor_levels = 2
strength = 0
arrayclass = oapackage.arraydata_t(factor_levels, run_size, strength, number_of_factors)
print("We generate D-efficient designs with %d rows and %d columns\n" % (run_size,␣
↪number_of_factors))
```

```
We generate D-efficient designs with 40 rows and 7 columns
```

Generate a single D-efficient design using $\alpha = (1, 2, 0)$ as the parameters for the optimization function. For details on this parameter and its corresponding optimization function, see Two-Level Designs to Estimate All Main Effects and Two-Factor Interactions.

[3]:
```
alpha = [1, 2, 0]
scores, design_efficiencies, designs, ngenerated = oapackage.Doptim.Doptimize(
    arrayclass, nrestarts=30, optimfunc=alpha, selectpareto=True
)
```

```
Doptim: optimization class 40.2-2-2-2-2-2-2
Doptimize: iteration 0/30
Doptimize: iteration 29/30
Doptim: done (11 arrays, 0.4 [s])
```

[4]:
```
print("\nGenerated %d designs, the efficiencies for these designs are:" % len(designs))
for ii, d in enumerate(designs):
    dd = d.Defficiencies()
    print("array %d: D-efficiency %.4f, Ds-efficiency %.4f" % (ii, dd[0], dd[1]))

D = [d.Defficiency() for d in designs]
best = np.argmax(D)
print("\nThe design with the highest D-efficiency (%.4f) is:\n" % D[best])

designs[best].transposed().showarraycompact()
```

```

Generated 11 designs, the efficiencies for these designs are:
array 0: D-efficiency 0.8815, Ds-efficiency 0.9807
array 1: D-efficiency 0.9076, Ds-efficiency 0.9628
array 2: D-efficiency 0.8670, Ds-efficiency 0.9827
array 3: D-efficiency 0.8945, Ds-efficiency 0.9669
array 4: D-efficiency 0.9027, Ds-efficiency 0.9540
array 5: D-efficiency 0.8851, Ds-efficiency 0.9549
array 6: D-efficiency 0.8737, Ds-efficiency 0.9581
array 7: D-efficiency 0.9036, Ds-efficiency 0.9400
array 8: D-efficiency 0.8614, Ds-efficiency 0.9595
array 9: D-efficiency 0.8897, Ds-efficiency 0.9418
array 10: D-efficiency 0.9046, Ds-efficiency 0.9203

The design with the highest D-efficiency (0.9076) is:

0010011001011001010101011101101100011101
0011010111100101111100010111001100100010
1011011111001010100010111001000010011010
0101100001100111110011010001001100111110
0011100010001101001011101101110100010001
0011110000011011101001110011101010101000
1000110011101000101110001001101111001101
```

Optimizing with a different optimization target leads to different D-efficient designs. Below we compare the sets of designs generated with optimization target [1,0,0] and [1,2,0].

[5]:
```
scores0, design_efficiencies0, designs0, _ = oapackage.Doptim.Doptimize(
    arrayclass, nrestarts=30, optimfunc=[1, 0, 0], selectpareto=True
)
```

```
Doptim: optimization class 40.2-2-2-2-2-2-2
Doptimize: iteration 0/30
Doptimize: iteration 29/30
Doptim: done (13 arrays, 0.4 [s])
```

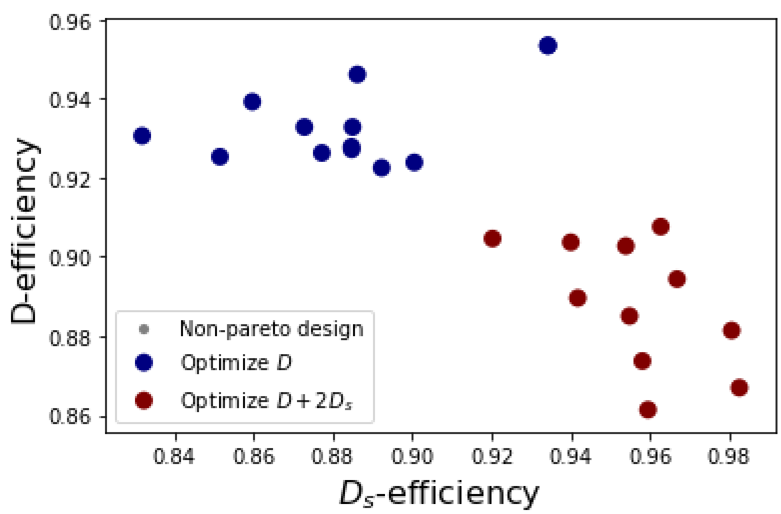
[7]:
```
def combineEfficiencyData(lst):
```

(continues on next page)

(continued from previous page)

```
    data = np.zeros((0, 4))

    for jj, dds in enumerate(lst):
        dds_index = np.hstack((dds, jj * np.ones((len(dds), 1))))
        data = np.vstack((data, dds_index))
    return data


design_efficiencies_combined = combineEfficiencyData([design_efficiencies0, design_
↪efficiencies])
plot_handles = oapackage.generateDscatter(
    design_efficiencies_combined, ndata=3, lbls=["Optimize $D$", "Optimize $D+2D_s$"],␣
↪verbose=0
)
```

```
Pareto: 23 optimal values, 24 objects
```



nbsphinx-code-borderwhite

```
<Figure size 432x288 with 0 Axes>
```

[ ]:

2.7 Example script to use Nauty from Python

Nauty can be used to reduce any graph into a normal form. In this notebook, we show how to use the Nauty functionality from Python.

[1]:
```
import numpy as np
import oapackage
```

Define a function to invert a permutation.

[2]:
```
def inverse_permutation(perm):
    inverse = [0] * len(perm)
    for i, p in enumerate(perm):
```

(continues on next page)

(continued from previous page)

```
        inverse[p] = i
    return inverse
```

We define a graph with 5 nodes. The graph is defined by the incidence matrix of size 5×5 and a coloring with two colors.

[3]:
```
graph = np.zeros((5, 5), dtype=int)
graph[0, 1] = graph[0, 2] = graph[0, 3] = graph[1, 3] = 1
graph = np.maximum(graph, graph.T)  # make array symmetric
colors = [0, 0, 0, 1, 1]
```

Reduce the graph to normal form using Nauty.

[4]:
```
help(oapackage.reduceGraphNauty)
```

```
Help on function reduceGraphNauty in module oalib:

reduceGraphNauty(G, colors=None, verbose=1)
    Return vertex transformation reducing array to normal form

    The reduction is calculated using `Nauty <http://users.cecs.anu.edu.au/~bdm/nauty/>`_

    Args:
        G (numpy array or array_link) :   the graph in incidence matrix form
        colors (list or None): an optional vertex coloring
    Returns:
        v: relabelling of the vertices
```

[4]:
```
def reduce(graph, colors):
    tr = oapackage.reduceGraphNauty(graph, colors=colors, verbose=0)
    tri = inverse_permutation(tr)

    graph_reduced = oapackage.transformGraphMatrix(graph, tri)
    colors_reduced = [colors[idx] for idx in tr]
    return graph_reduced, colors_reduced, tr


graph_reduced, colors_reduced, tr = reduce(graph, colors)

print("input graph: ")
print(graph)

print("normal form reduction: %s" % (tr,))
print("reduced graph: ")
print(graph_reduced)
print("colors reduced: %s" % (colors_reduced,))
```

```
input graph:
[[0 1 1 1 0]
 [1 0 0 1 0]
 [1 0 0 0 0]
 [1 1 0 0 0]
```

(continues on next page)

(continued from previous page)

```
 [0 0 0 0 0]]
normal form reduction: (1, 2, 0, 4, 3)
reduced graph:
[[0 0 1 0 1]
 [0 0 1 0 0]
 [1 1 0 0 1]
 [0 0 0 0 0]
 [1 0 1 0 0]]
colors reduced: [0, 0, 0, 1, 1]
```

Apply a random permutation to the graph and reduce the graph again.

[5]:
```
perm = np.random.permutation(5)
iperm = inverse_permutation(perm)
print("random permutation: %s" % (perm,))
graph2 = graph[perm, :][:, perm]
colors2 = [colors[idx] for idx in perm]
```

```
random permutation: [3 2 0 4 1]
```

Show the transformed matrix and color vector.

[8]:
```
print(graph2)
print(colors2)
```

```
[[0 0 0 0 0]
 [0 0 0 1 1]
 [0 0 0 0 1]
 [0 1 0 0 1]
 [0 1 1 1 0]]
[1, 1, 0, 0, 0]
```

[6]:
```
graph2_reduced, colors2_reduced, tr2 = reduce(graph2, colors2)

print("input graph: ")
print(graph2)

print("tr2: %s" % (tr2,))
print("reduced graph: ")
print(graph2_reduced)

print("colors2_reduced: %s" % (colors2_reduced,))
```

```
input graph:
[[0 0 1 0 1]
 [0 0 1 0 0]
 [1 1 0 0 1]
 [0 0 0 0 0]
 [1 0 1 0 0]]
tr2: (4, 1, 2, 3, 0)
reduced graph:
[[0 0 1 0 1]
 [0 0 1 0 0]
 [1 1 0 0 1]
```

(continues on next page)

(continued from previous page)

```
 [0 0 0 0 0]
 [1 0 1 0 0]]
colors2_reduced: [0, 0, 0, 1, 1]
```

Check that the two reduced graphs are equal.

```
[9]: if np.all(graph_reduced == graph2_reduced):
         print("reduced arrays are equal!")
     if np.all(colors_reduced == colors2_reduced):
         print("reduced colors are equal!")
```

```
reduced arrays are equal!
reduced colors are equal!
```

2.8 Analyse isomorphisms of a set of orthogonal arrays with N=56

In this example, we show how to identify isomorphic arrays from a list of two-level strength-2 orthogonal arrays with 56 runs and 28 factors.

```
[1]: import numpy as np
     import oapackage
     import oapackage.graphtools
     from oapackage.graphtools import selectIsomorphismClasses
```

Read the arrays from a file and determine their class.

```
[2]: sols=oapackage.readarrayfile('OAN56K28.oa')
     arrayclass=oapackage.arraylink2arraydata(sols[0])
     print(arrayclass)
     print('loaded %d arrays' % len(sols))
```

```
arrayclass: N 56, k 28, strength 2, s {2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
↪2,2,2}, order 0
loaded 9 arrays
```

Determine the unique isomorphism classes using Nauty.

```
[3]: b,mm=selectIsomorphismClasses(sols, verbose=0)

     print('from %d arrays selected %d unique isomorphism classes' % (len(mm),np.unique(b).
     ↪size))
     print('indices: %s' % str(b) )
```

```
from 9 arrays selected 6 unique isomorphism classes
indices: [0 4 5 3 1 5 3 2 4]
```

Verify that the first two arrays are indeed non-isomorphic:

```
[4]: jj=np.abs(sols[0].Jcharacteristics(4))
     n0, _=np.histogram(jj, [0,8,16,24])
     print(n0)
     jj=np.abs(sols[1].Jcharacteristics(4))
```

(continues on next page)

(continued from previous page)

```
n1, _=np.histogram(jj, [0,8,16,24])
print(n1)
```

```
[    0 18018  2457]
[    0 18032  2436]
```

Since the first two arrays have different J-characteristics (see the section Statistical properties of orthogonal arrays for details), they are non-isomorphic. For the isomorphic arrays, it is possible to obtain the array transformation to make the arrays identical using the function `reduceConferenceTransformation`.

2.9 Generation and analysis of conference designs

In this notebook, we show how to generate conference designs and calculate properties of these designs. For details on conference designs and their properties, see Properties of conference designs and A Classification Criterion for Definitive Screening Designs, Schoen et al., 2018 and [Enumeration and Classification of Definitive Screening Designs] (in preparation).

Load required libraries and define the class of conference designs to enumerate.

[3]:
```
import oapackage
conference_class=oapackage.conference_t(12, 6, 0)
print(conference_class)
```

```
conference class: number of rows 12, number of columns 6
```

Define the root array and extend the lists of conference designs.

[4]:
```
conference_designs=[[conference_class.create_root_three_columns()]]

for ii, ncols in enumerate(range(4, 8)):
    arrays = oapackage.extend_conference (conference_designs[ii], conference_class,␣
↪verbose=0)
    conference_designs.append(arrays)
    print('extension resulted in %d designs with %d columns' % (len(arrays), ncols))
```

```
extension resulted in 9 designs with 4 columns
extension resulted in 42 designs with 5 columns
extension resulted in 123 designs with 6 columns
extension resulted in 184 designs with 7 columns
```

2.9.1 Calculate properties of conference designs

Here, we show how to calculate relevant properties of conference designs. Select a 12-run 7-factor conference design generated previously.

[5]:
```
design = conference_designs[4][0]
design.showarray()
```

```
array:
  0   1   1   1   1   1   1
  1   0  -1  -1  -1  -1  -1
  1   1   0  -1  -1   1   1
```

(continues on next page)

(continued from previous page)

```
  1   1   1   0   1  -1  -1
  1   1   1  -1   0  -1   1
  1   1  -1   1   1   0  -1
  1   1  -1   1  -1   1   0
  1  -1   1   1  -1   1  -1
  1  -1   1   1  -1  -1   1
  1  -1   1  -1   1   1  -1
  1  -1  -1   1   1  -1   1
  1  -1  -1  -1   1   1   1
```

A sensible criterion to evaluate conference designs is the so-called F_4-vector (Schoen et al., 2019). We can compute the F_4-vector of a conference design as follows.

```
[6]: design.FvaluesConference(4)
```

```
[6]: (0, 25, 10)
```

Schoen et al., 2019 showed that conference designs are good building blocks for definitive screening designs (Xiao et al. 2012). The Orthogonal Array package can construct a definitive screening design from a conference design.

```
[9]: dsd = oapackage.conference2DSD(design)
     dsd.showarray()
```

```
array:
  0   1   1   1   1   1   1
  1   0  -1  -1  -1  -1  -1
  1   1   0  -1  -1   1   1
  1   1   1   0   1  -1  -1
  1   1   1  -1   0  -1   1
  1   1  -1   1   1   0  -1
  1   1  -1   1  -1   1   0
  1  -1   1   1  -1   1  -1
  1  -1   1   1  -1  -1   1
  1  -1   1  -1   1   1  -1
  1  -1  -1   1   1  -1   1
  1  -1  -1  -1   1   1   1
  0  -1  -1  -1  -1  -1  -1
 -1   0   1   1   1   1   1
 -1  -1   0   1   1  -1  -1
 -1  -1  -1   0  -1   1   1
 -1  -1  -1   1   0   1  -1
 -1  -1   1  -1  -1   0   1
 -1  -1   1  -1   1  -1   0
 -1   1  -1  -1   1  -1   1
 -1   1  -1  -1   1   1  -1
 -1   1  -1   1  -1  -1   1
 -1   1   1  -1  -1   1  -1
 -1   1   1   1  -1  -1  -1
  0   0   0   0   0   0   0
```

For the resulting definitive screening design, the Orthogonal Array package can compute some statistical properties based on projections into a smaller number of factors.

[7]:
```
PEC4, PIC4, PPC4 = oapackage.conference.conferenceProjectionStatistics(design,␣
↪ncolumns=4, verbose=1)
```

```
conferenceProjectionStatistics: projection to 4 columns: PEC 0.286 PIC 3.111 PPC 0.458
```

2.10 Example code for delete-one-factor projections

Any orthogonal array can be reduced to delete-one-factor projection form using `reduceDOPform`. The method is described in the article A canonical form for non-regular arrays based on generalized wordlength pattern values of delete-one-factor projections by Eendebak, P. T. In this notebook, we reduce an example array to its delete-one-factor projection form.

Load required libraries and select an example orthogonal array with 16 runs and 7 factors.

[4]:
```
import oapackage

al = oapackage.exampleArray(4)
al = oapackage.reduceDOPform(al)
al.showarray()
```

```
array:
  0   0   0   0   0   0   0
  0   0   0   0   0   0   1
  0   0   0   1   1   1   0
  0   0   0   1   1   1   1
  0   1   1   0   0   1   0
  0   1   1   0   0   1   1
  0   1   1   1   1   0   0
  0   1   1   1   1   0   1
  1   0   1   0   1   0   0
  1   0   1   0   1   1   1
  1   0   1   1   0   0   0
  1   0   1   1   0   1   1
  1   1   0   0   1   0   1
  1   1   0   0   1   1   0
  1   1   0   1   0   0   1
  1   1   0   1   0   1   0
```

A key property of the delete-of-factor projection form is that the generalized word length patterns (GWLPs) of the projections are ordered. For details on the GWLP, see Statistical properties of orthogonal arrays.

[5]:
```
print("GWLP %s" % str(al.GWLP()))
for ii in range(0, al.n_columns):
    bl = al.deleteColumn(ii)
    print("Delete column %d: GWLP %s" % (ii, str(bl.GWLP())))
```

```
GWLP (1.0, 0.0, 0.0, 3.5, 2.5, 0.5, 0.5, 0.0)
Delete column 0: GWLP (1.0, 0.0, 0.0, 1.5, 1.0, 0.5, 0.0)
Delete column 1: GWLP (1.0, 0.0, 0.0, 1.75, 0.75, 0.25, 0.25)
Delete column 2: GWLP (1.0, 0.0, 0.0, 1.75, 0.75, 0.25, 0.25)
Delete column 3: GWLP (1.0, 0.0, 0.0, 2.0, 1.0, 0.0, 0.0)
Delete column 4: GWLP (1.0, 0.0, 0.0, 2.0, 1.0, 0.0, 0.0)
```

(continues on next page)

(continued from previous page)

```
Delete column 5: GWLP (1.0, 0.0, 0.0, 2.0, 1.0, 0.0, 0.0)
Delete column 6: GWLP (1.0, 0.0, 0.0, 3.0, 2.0, 0.0, 0.0)
```

The symmetry group of the projection GWLPs can be calculated. This symmetry group determines how fast an array can be reduced to normal form.

```
[6]: dof_values = oapackage.projectionDOFvalues(al)
sg = oapackage.symmetry_group(dof_values, False)
sg.show(1)
```

```
symmetry group: 7 elements, 4 subgroups: 1 2 3 1
```

It is also possible to reduce mixed-level arrays to their delete-of-factor projection forms. We now show an example involving an orthogonal array with 24 runs, one four-level factor, one three-level factor and three two-level factors.

```
[8]: al = oapackage.exampleArray(5)
al.showarray()
dopgwlp = oapackage.projectionGWLPs(al)
print("delete-one-factor GWLP values %s" % ([x.raw_values() for x in dopgwlp],))
```

```
array:
  0   0   0   0   0
  0   0   0   0   0
  0   1   0   0   0
  0   1   1   1   1
  0   2   1   1   1
  0   2   1   1   1
  1   0   0   1   1
  1   0   0   1   1
  1   1   0   0   1
  1   1   1   1   0
  1   2   1   0   0
  1   2   1   0   0
  2   0   1   0   1
  2   0   1   0   1
  2   1   0   1   0
  2   1   1   0   1
  2   2   0   1   0
  2   2   0   1   0
  3   0   1   1   0
  3   0   1   1   0
  3   1   0   1   1
  3   1   1   0   0
  3   2   0   0   1
  3   2   0   0   1
delete-one-factor GWLP values [(1.0, 0.0, 0.0, 0.0, 0.6666666666666666), (1.0, 0.0, 0.0,␣
↪2.1111111111111111, 0.0), (1.0, 0.0, 0.0, 1.8888888888888888, 0.4444444444444444), (1.0,
↪ 0.0, 0.0, 2.3333333333333335, 0.0), (1.0, 0.0, 0.0, 1.8888888888888888, 0.
↪4444444444444444)]
```

The delete-of-factor values for mixel-level arrays consists of the factor level of the deleted column and the GWLP of the projection.

[9]:
```
oapackage.projectionGWLPs
dopgwp = oapackage.projectionGWLPs(al)
arrayclass = oapackage.arraylink2arraydata(al)
dofvalues = oapackage.projectionDOFvalues(al)
factor_levels = arrayclass.factor_levels()

for column, dof_value in enumerate(dofvalues):
    print("column %d: factor level %s" % (column, factor_levels[column]))
    print("   delete-of-factor value: %s" % (list(dof_value.raw_values()),))
```

```
column 0: factor level 4
   delete-of-factor value: [-4.0, 1.0, 0.0, 0.0, 0.0, 0.6666666666666666]
column 1: factor level 3
   delete-of-factor value: [-3.0, 1.0, 0.0, 0.0, 2.111111111111111, 0.0]
column 2: factor level 2
   delete-of-factor value: [-2.0, 1.0, 0.0, 0.0, 1.8888888888888888, 0.4444444444444444]
column 3: factor level 2
   delete-of-factor value: [-2.0, 1.0, 0.0, 0.0, 2.3333333333333335, 0.0]
column 4: factor level 2
   delete-of-factor value: [-2.0, 1.0, 0.0, 0.0, 1.8888888888888888, 0.4444444444444444]
```

Show the reduced array.

[10]:
```
reduced_array = al.reduceDOP()
al.showarray()
```

```
array:
  0   0   0   0   0
  0   0   0   0   0
  0   1   0   0   0
  0   1   1   1   1
  0   2   1   1   1
  0   2   1   1   1
  1   0   0   1   1
  1   0   0   1   1
  1   1   0   0   1
  1   1   1   1   0
  1   2   1   0   0
  1   2   1   0   0
  2   0   1   0   1
  2   0   1   0   1
  2   1   0   1   0
  2   1   1   0   1
  2   2   0   1   0
  2   2   0   1   0
  3   0   1   1   0
  3   0   1   1   0
  3   1   0   1   1
  3   1   1   0   0
  3   2   0   0   1
  3   2   0   0   1
```

2.11 Minimal number of runs for an orthognal array

In this example we calculate the minimum number of runs required for an orthogonal array. For more details, see Schoen et al.

[6]:
```
import itertools
import numpy as np


def minimum_number_of_runs(factor_levels, strength):
    """Calculate the minimum number of runs for an orthogonal array

    The minimum number of runs is based on the strength conditions. IWhether a design␣
↪actually exists
    Args:
        factor_levels: Factor levels of the design
        strength: Strength of the array
    Returns:
        Minimum number of runs

    """
    runs = [np.prod(tt) for tt in itertools.combinations(factor_levels, strength)]
    N = np.lcm.reduce(runs)
    return N
```

We run the method on several examples.

[7]:
```
strength = 3
factor_levels = [2, 3, 3, 4, 5]
N = minimum_number_of_runs(factor_levels, strength)

print(f"for a design of strength {strength} and factor levels {factor_levels} we require␣
↪(a multiple of) {N} runs")
```

```
for a design of strength 3 and factor levels [2, 3, 3, 4, 5] we require (a multiple of)␣
↪360 runs
```

[12]:
```
strength = 2
factor_levels = [3, 2, 2, 2, 2]
N = minimum_number_of_runs(factor_levels, strength)

print(f"for a design of strength {strength} and factor levels {factor_levels} we require␣
↪(a multiple of) {N} runs")
```

```
for a design of strength 2 and factor levels [3, 2, 2, 2, 2] we require (a multiple of)␣
↪12 runs
```

[ ]:

2.12 Example of GWLP calculation for mixed-level designs

This notebook contains example code to compute the type-specific generalized word-length pattern (GWLP) for mixed-level designs, more specifically for regular four-and-two-level designs.

In four-and-two-level designs, the four-level factors are constructed us the grouping scheme of Wu & Zhang (1989) where the levels of a four-level factor A are based on the levels of three two-level factors a_1, a_2 and a_3, called the pseudo-factors, where $I = a_1 a_2 a_3$:

$$\begin{array}{ccccc} a_1 & a_2 & a_3 & & A \\ 1 & 1 & 0 & \rightarrow & 0 \\ 1 & 0 & 1 & \rightarrow & 1 \\ 0 & 1 & 1 & \rightarrow & 2 \\ 0 & 0 & 0 & \rightarrow & 3 \end{array}$$

The two-level factors can either be main factors or be entirely aliased with a combination of the main factors. Such aliased factors are called added factors. If one of the main factors, used in an added factor, is also used as pseudo-factor in a four-level factor, then the added factor has type I. If it is used as pseudo-factor in two distinct four-level factor, it has type II, etc . . ..

[1]:
```
import numpy as np

import oapackage
```

Create a $4^1 2^7$ four-and-two-level regular design in 32 runs with 1 four-level factor and 7 two-level factors.

[2]:
```
array = oapackage.exampleArray(56, 1)
array = array.selectFirstColumns(7)
arrayclass = oapackage.arraylink2arraydata(array)
array.showarraycompact()
```

```
exampleArray 56: design in OA(32, 42^{18})
0000000
0100110
0101010
0001100
0110011
0010101
0011001
0111111
1010100
1110010
1111110
1011000
1100111
1000001
1001101
1101011
2101101
2001011
2000111
2100001
2011110
2111000
2110100
```

(continues on next page)

(continued from previous page)

```
2010010
3111001
3011111
3010011
3110101
3001010
3101100
3100000
3000110
```

The GWLP for mixed-level designs is computed using the adapted forms of equations (7) and (8) in Xu and Wu (2001).

2.12.1 Distance distribution

First, the distance distribution is first computed for all combinations of (i, j) with $i \in \{0, 1\}$ and $j = 0, \ldots, 7$ and $i \neq j$. In this example with a $4^1 2^7$ design, that is a 2×8 matrix B, where $B_{i,j}$ is the number of rows that have $(1 - i)$ different elements if their four-level parts and $(7 - j)$ different elements in their two-level parts, divided by the number of runs.

```
[3]: Dm = oapackage.distance_distribution_mixed(array, 0)
D = np.array(Dm).astype(int)
print(f"distance distribution for mixed-level design\n{D}")
```

```
distance distribution for mixed-level design
[[1 0 1 4 1 0 1]
 [0 2 6 8 6 2 0]]
```

This is verified since the sum of the matrix is 32 which is equal to $\binom{32}{2}32^{-1}$.

2.12.2 McWilliams Transforms

Now, the second step in the computation of the GWLP, is to use the McWilliams Transforms to obtain the GWLP from the distance distribution matrix. The McWilliams Transform will create another 2×8 matrix B', obtained using the following formula

$$B'_{j_1,j_2} = N^{-1} \sum_{i_1=0}^{m} \sum_{i_2=0}^{n} B_{i_1,i_2} P_{j_1}(i_1; 1, 4) P_{j_2}(i_2; 7, 2)$$

where $P_j(x, n, s)$ is the Krawtchouck polynomials for a total of n factors with s levels.

```
[4]: N = array.n_rows
factor_levels_for_groups = arrayclass.factor_levels_column_groups()
Bprime = oapackage.macwilliams_transform_mixed(Dm, N, factor_levels_for_groups,
↪verbose=0)
print(f"MacWilliams transform:")
print(np.array(Bprime).astype(int))
```

```
MacWilliams transform:
[[1. 0. 0. 0. 1. 0. 0.]
 [0. 0. 2. 0. 4. 0. 0.]]
```

This matrix is equivalent to the type specific generalized word-length pattern, where the row index indicates the type of words (0 to 1) and the column index indicates the length of the words (0 to 7).

The generalized word-length pattern (A) can be obtained by summing the rows of the B' matrix anti-diagonally. That is:

$$A_j(D) = \sum_{i'+j'=j} B'_{i',j'}$$

```
[4]: gwlp_mixed = oapackage.GWLPmixed(array, 0)
     print(f"GWLP: {gwlp_mixed}")
```

```
GWLP: (1.0, 0.0, 0.0, 2.0, 1.0, 4.0, 0.0, 0.0)
```

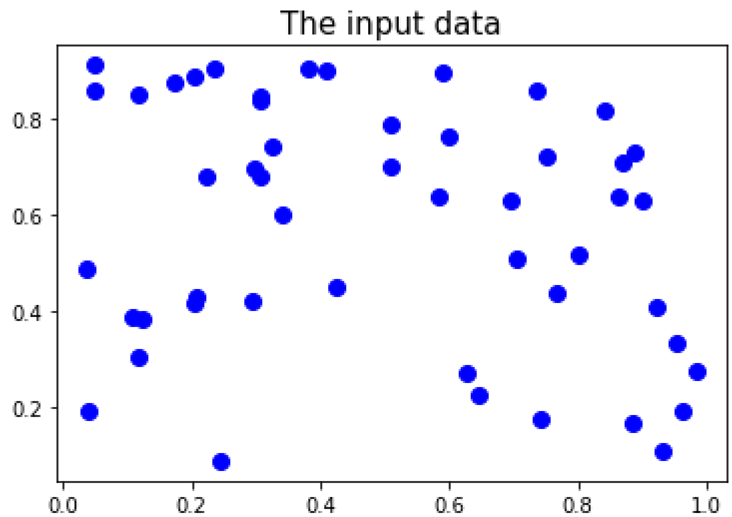
2.13 Calculate a Pareto optimal set

Pareto optimality (or multi-objective optimization) allows one to search for optimal solutions for an optimization problem with multiple objectives. The Pareto class in the Orthogonal Array package allows one to calculate the Pareto optimal elements (called the Pareto frontier).

```
[1]: import numpy as np
     import matplotlib.pyplot as plt
     %matplotlib inline

     import oapackage
```

First, define a dataset of 50 random vectors. The vectors have length 2, so there are 2 objectives to be optimized.

```
[2]: datapoints=np.random.rand(2, 50)

     for ii in range(0, datapoints.shape[1]):
         w=datapoints[:,ii]
         fac=.6+.4*np.linalg.norm(w)
         datapoints[:,ii]=(1/fac)*w

     h=plt.plot(datapoints[0,:], datapoints[1,:], '.b', markersize=16, label='Non Pareto-
     ↪optimal')
     _=plt.title('The input data', fontsize=15)
     plt.xlabel('Objective 1', fontsize=16)
     plt.ylabel('Objective 2', fontsize=16)
```



nbsphinx-code-borderwhite

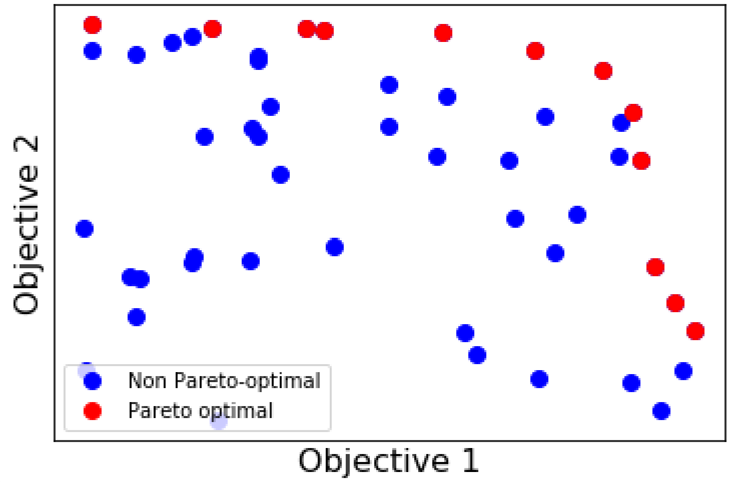
Create a structure (`ParetoDoubleLong`) to keep track of the data.

[3]:
```
pareto=oapackage.ParetoDoubleLong()

for ii in range(0, datapoints.shape[1]):
    w=oapackage.doubleVector( (datapoints[0,ii], datapoints[1,ii]))
    pareto.addvalue(w, ii)

pareto.show(verbose=1)
```

```
Pareto: 12 optimal values, 12 objects
```

Plot the results.

[4]:
```
lst=pareto.allindices() # the indices of the Pareto optimal designs

optimal_datapoints=datapoints[:,lst]

h=plt.plot(datapoints[0,:], datapoints[1,:], '.b', markersize=16, label='Non Pareto-
↪optimal')
hp=plt.plot(optimal_datapoints[0,:], optimal_datapoints[1,:], '.r', markersize=16, label=
↪'Pareto optimal')
plt.xlabel('Objective 1', fontsize=16)
plt.ylabel('Objective 2', fontsize=16)
plt.xticks([])
plt.yticks([])
_=plt.legend(loc=3, numpoints=1)
```



nbsphinx-code-borderwhite

2.14 Isomorphism reduction for conference designs

In this example, we show how to test if two conference designs are isomorphic. We consider conference designs with 10 runs and 3 factors, and calculate a reduction to their normal form. Using the reduction, we determine if the two designs are isomorphic.

[1]:
```
import oapackage
import numpy as np

A = np.array(
    [0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, -1, 1, 1, -1, 1, -1, 1, 1, -1, 1, 1, -1, -
↪1, 1, -1, -1]
).reshape(10, 3)
B = np.array(
    [0, 1, 1, 1, 0, -1, 1, 1, 1, 1, 1, 1, 1, 1, -1, 1, 1, -1, 1, -1, 0, 1, -1, 1, 1, -1,␣
↪1, 1, -1, -1]
).reshape(10, 3)
array1 = oapackage.makearraylink(A)
array2 = oapackage.makearraylink(B)

array1.showarray()
array2.showarray()
```

```
array:
  0   1   1
  1   0   1
  1   1   0
  1   1   1
  1   1  -1
  1   1  -1
  1  -1   1
  1  -1   1
  1  -1  -1
  1  -1  -1
array:
```

(continues on next page)

(continued from previous page)

```
0   1   1
1   0  -1
1   1   1
1   1   1
1   1  -1
1   1  -1
1  -1   0
1  -1   1
1  -1   1
1  -1  -1
```

We calculate the normal forms of the conference designs using the function `reduceConference` or `reduceConferenceTransformation`. The result of the former is the reduced design, while the result of the latter is an object describing the tranformation to normal form. The normal form is calculated using Nauty.

[2]:
```
help(oapackage.reduceConference)
```

```
Help on function reduceConference in module oalib:

reduceConference(arg1, verbose=0)
    reduceConference(array_link arg1, int verbose=0) -> array_link
    reduceConference(array_link arg1) -> array_link



    Reduce conference matrix to normal form using Nauty

    See also: reduceConferenceTransformation
```

[3]:
```
T1 = oapackage.reduceConferenceTransformation(array1, verbose=1)
T2 = oapackage.reduceConferenceTransformation(array2, verbose=1)
T1.show()
```

```
reduceConferenceTransformation: reduce design with 10 rows, 3 columns
reduceConferenceTransformation: reduce design with 10 rows, 3 columns
row permutation: {2,0,1,9,7,8,5,6,3,4}
  row flips: {1,1,-1,1,1,1,1,1,1,1}
column permutation: {0,1,2}
  col flips: {1,-1,-1}
```

We can check whether the designs are isomorphic by comparing the normal forms.

[4]:
```
design_equal = T1.apply(array1) == T2.apply(array2)
print("designs isomorphic? %s" % design_equal)
```

```
designs isomorphic? 1
```

The designs are isomorphic. So, it is possible to calculate a reduction of the second design into the first design.

[5]:
```
TT = T1.inverse() * T2
TT.show()
```

```
row permutation: {0,1,8,9,6,7,2,4,5,3}
  row flips: {-1,1,1,1,1,1,1,1,1,1}
column permutation: {0,1,2}
  col flips: {1,-1,-1}
```

[6]:
```
r1 = T1.apply(array1)
r1.showarray()
r2 = T2.apply(array2)
r2.showarray()
```

```
array:
  1   0  -1
  1  -1   0
  0   1   1
  1   1   1
  1   1   1
  1   1  -1
  1   1  -1
  1  -1   1
  1  -1   1
  1  -1  -1
array:
  1   0  -1
  1  -1   0
  0   1   1
  1   1   1
  1   1   1
  1   1  -1
  1   1  -1
  1  -1   1
  1  -1   1
  1  -1  -1
```

Calculate some properties of a conference design; see the section Properties of conference designs for details.

[7]:
```
print("array 1: is_conference() %d" % array1.is_conference())
print("array 1: J2-characteristics %s" % (oapackage.Jcharacteristics_conference(array1,␣
↪number_of_columns=2),))
```

```
array 1: is_conference() 1
array 1: J2-characteristics (0, 0, 0)
```

[8]:
```
help(oapackage.Jcharacteristics_conference)
```

```
Help on function Jcharacteristics_conference in module oalib:

Jcharacteristics_conference(array, number_of_columns, verbose=0)
    Jcharacteristics_conference(array_link array, int number_of_columns, int verbose=0) -
↪> intVector
    Jcharacteristics_conference(array_link array, int number_of_columns) -> intVector
```

CHAPTER

THREE

DATA REPRESENTATION

All designs handled by the OApackage are integer valued. The designs (whether these are orthogonal arrays, optimal designs or conferences designs) are stored in an `array_link()` object. The definitions of orthogonal arrays, optimal designs and conference designs are included in the section *Definitions of arrays and designs*.

3.1 Data structures

The package contains several data structures. Here, we describe the main structures and their use.

`array_link()`
: The structure containing an orthogonal array is called the `array_link()` structure. Lists of arrays are stored in the `arraylist_t()` object, which is implemented as a `std::deque` container.

`arrayfile_t()`
: This object allows for reading and writing of arrays to disk.

`arraydata_t()`
: The structure describing a certain class of orthogonal arrays or optimal designs.

`conference_t()`
: The structure describing a certain class of conference designs.

`array_transformation_t()`
: The structure describing a transformation of an orthogonal array, which includes row and column permutations, as well as permutations to the symbols in one or more columns.

`conference_transformation_t()`
: The structure describing a transformation of conference design or double conference design, which includes row and column permutations, as well as sign switches to the elements in one or more rows and columns.

3.2 Representing arrays

The structure containing an orthogonal array is called the *`array_link`* structure. It consists of a specified number of rows and columns, the data (integer valued) and an index. In the Python interface, the `array_link()` object can be indexed just as normal arrays.

It is also possible to convert to a Numpy array. The `array_link` object implements the Python array interface, so most operations from packages such as Numpy work on the `array_link()` object.

Array representation and indexing in Python

```
>>> import oapackage; import numpy as np
>>> al=oapackage.exampleArray(0)
>>> al.showarray()
array:
  0   0
  0   0
  0   1
  0   1
  1   0
  1   0
  1   1
  1   1

>>> al[2,1]
1
>>> X=np.array(al)
>>> X
array([[0, 0],
       [0, 0],
       [0, 1],
       [0, 1],
       [1, 0],
       [1, 0],
       [1, 1],
       [1, 1]], dtype=int16)
```

The C++ class is *array_link*.

3.3 Classes of arrays

The *arraydata_t* object represents data about a class of orthogonal arrays, e.g. the class $\mathrm{OA}(N; t; s^k)$; see *Definitions of arrays and designs*. The *conference_t* object represents data about a class of conference designs.

3.4 Array transformations

Transformations of (orthogonal) arrays consist of row, column and level permutations. A transformation is represented by the *array_transformation_t* object.

For a given transformation, the column permutations are applied first, then the level permutations and finally the row permutations. The level and column permutations are not commutative.

The conference design transformations also allow for row sign switches and are described by the class *conference_transformation_t*.

3.5 Reading and writing arrays

Reading and writing arrays to disk can be done with the `arrayfile_t` class.

Write an array or a list of arrays to disk

```
>>> import oapackage
>>> list_of_arrays = [oapackage.exampleArray(24), oapackage.exampleArray(25)]
>>> _ = oapackage.writearrayfile('test.oa', list_of_arrays)
>>> oapackage.oainfo('test.oa')
file test.oa: 64 rows, 16 columns, 2 arrays, mode text, nbits 0
>>> al=oapackage.exampleArray()
>>> af=oapackage.arrayfile_t('test.oa', al.n_rows, al.n_columns)
>>> af.append_array(al)
>>> print(af)
file test.oa: 8 rows, 2 columns, 1 arrays, mode text, nbits 8
>>> af.closefile()
```

The arrays can be written in text or binary format. For more details on the file format, see the section *File formats*.

The Python interface is `oalib.arrayfile_t()` and the C++ interface is `arrayfile_t`.

3.6 File formats

The Orthogonal Array package stores arrays in a custom file format. There is a text format which is easily readable by humans and a binary format which is faster to process and memory efficient.

3.6.1 Plain text array files

Arrays are stored in plain text files with extension `.oa`. The first line contains the number of columns, the number of rows and the number of arrays (or -1 if the number of arrays is not specified). Then, for each array, a single line with the index of the array, followed by N lines containing the array.

A typical example of a text file is the following:

```
5 8 1
1
0 0 0 0 0
0 0 0 1 1
0 1 1 0 0
0 1 1 1 1
1 0 1 0 1
1 0 1 1 0
1 1 0 0 1
1 1 0 1 0
-1
```

This file contains exactly 1 array with 8 rows and 5 columns.

3.6.2 Binary array files

Every binary file starts with a header, which has the following format:

```
[INT32] 65 (magic identifier)
[INT32] b: Format: number of bits per number. Currently supported are 1 and 8
[INT32] N: number of rows
[INT32] k: kumber of columns
[INT32] Number of arrays (can be -1 if unknown)
[INT32] Binary format number: 1001: normal, 1002: binary diff, 1003: binary diff zero
[INT32] Reserved integer
[INT32] Reserved integer
```

The format of the remainder of the binary file depends on the binary format specified. For the normal binary format, the format is as follows. For each array, the number is specified in the header:

```
[INT32] Index
[Nxk elements] The elements contain b bits
```

If the number of bits per number is 1 (e.g. a 2-level array), then the data is padded with zeros to a multiple of 64 bits. The data of the array is stored in column-major order. The binary file format allows for random access reading and writing. The `binary diff` and `binary diff zero` formats are special formats.

A binary array file can be compressed using gzip. Most tools in the Orthogonal Array package can read these compressed files transparently. Writing to compressed array files is not supported at the moment.

3.6.3 Data files

The analysis tool (`oaanalyse`) writes data to disk in binary format. The format consists of a binary header:

```
[FLOAT64] Magic number 30397995;
[FLOAT64] Magic number 12224883;
[FLOAT64] nc: Number of rows
[FLOAT64] nr: Number of columns
```

After the header there follow `nc*nr [FLOAT64]` values.

3.6.4 MD5 sums

To check data integrity on disk, the packages includes functions to generate MD5 sums of designs.

Calculate md5 sum of a design

```
>>> import oapackage; al=oapackage.exampleArray(0)
>>> al.md5()
'6454c492239a8e01e3c01a864583abf2'
```

The C++ functions are *array_link::md5()* and *md5()*.

3.7 Command line interface

Several command line tools are included in the Orthogonal Array package. For each tool, help can be obtained from the command line by using the switch -h. The tools include the following:

oainfo
This program reads Orthogonal Array package data files and reports the contents of the files. For example:

```
$ oainfo result-8.2-2-2-2.oa
Orthogonal Array package 1.8.7
oainfo: reading 1 file(s)
file result-8.2-2-2.oa: 8 rows, 3 columns, 2 arrays, mode text, nbits 0
$
```

oacat
Shows the contents of a file with orthogonal arrays for a data file.

oacheck
Checks or reduces an array to canonical form.

oaextendsingle
Extends a set of arrays in LMC form with one or more columns.

oacat
Shows the contents of an array file or data file.

Usage: `oacat [OPTIONS] [FILES]`

oajoin
Reads one or more files from disk and join all the array files into a single list.

oasplit
Takes a single array file as input and splits the arrays into a specified number of output files.

oapareto
Calculates the set of Pareto optimal arrays in a file with arrays.

oaanalyse
Calculates various statistical properties of arrays in a file. The properties are described in section *Properties of designs*.

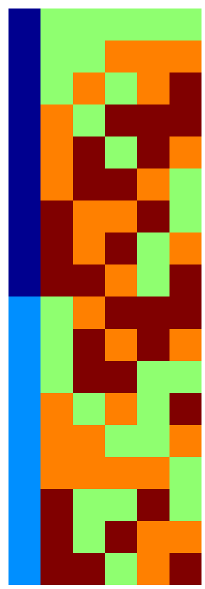


Fig. 1: Orthogonal array in $\mathrm{OA}(18, 23^a, 2)$.

CHAPTER

FOUR

GENERATION OF DESIGNS

The Orthogonal Array package can be used to generate several classes of arrays and designs. A collection of arrays and designs generated by the package is available on the website http://www.pietereendebak.nl/oapackage/index.html.

4.1 Generation of orthogonal arrays

A list of arrays in LMC form (i.e., lexicographically minimum in columns) can be extended to a list of arrays in LMC form with one additional column. Details about the algorithm are described in [SEN10].

The main function for array extension is the function *extend_arraylist()*. The arguments for this function are the list of arrays to extend, a specification of the class of arrays in *arraydata_t* and the options *OAextend* for the algorithm.

An example of a session that extends an array is:

```
>>> import oapackage
>>> nrows=8; ncols=3;
>>> arrayclass=oapackage.arraydata_t(2, nrows, 2, ncols)
>>> root_array=arrayclass.create_root()
>>> root_array.showarraycompact()
00
00
01
01
10
10
11
11
>>> array_list=oapackage.extend_array(root_array, arrayclass)
>>> print('found %d extensions of the root array' % len(array_list))
found 2 extensions of the root array
```

A more detailed example is included in *Enumerate orthogonal arrays*.

4.2 Conference designs

A conference design is an $N \times k$ matrix with entries 0, -1, +1 such that i) in each column the symbol 0 occurs exactly one time and ii) all columns are orthogonal to each other. For details on conference designs, see the section *Properties of conference designs* and [SEG19]. An example of a session to generate conference designs is the following:

Generate conference designs with 8 rows

```
>>> import oapackage
>>> conference_class=oapackage.conference_t(8, 6, 0)
>>> array = conference_class.create_root_three_columns()
>>> array.showarray()
array:
  0   1   1
  1   0  -1
  1   1   0
  1   1   1
  1   1  -1
  1  -1   1
  1  -1   1
  1  -1  -1
>>> l4=oapackage.extend_conference ([array], conference_class, verbose=0)
>>> l5=oapackage.extend_conference ( l4, conference_class, verbose=0)
>>> l6=oapackage.extend_conference ( l5, conference_class, verbose=0)
>>> print('number of non-isomorphic conference designs with 6 columns: %d' % len(l6) )
number of non-isomorphic conference designs with 6 columns: 11
```

An example notebook with more functionality is included in *Generation and analysis of conference designs*. The full interface for conference designs is available in the *Interface for conference designs*.

The main functions to extend conference and double conference designs are `extend_conference()` and `extend_double_conference()`, respectively. The low-level functions for generating candidate extension columns of conference and double conference designs are `generateConferenceExtensions()` and `generateDoubleConferenceExtensions()`, respectively.

The conference designs are generated in *LMC0* form.

4.3 Calculation of D-efficient designs

D-efficient designs (sometimes called D-optimal designs) can be calculated with the function `oapackage.Doptim.Doptimize()`. This function uses a coordinate-exchange algorithm to generate designs with good properties for the D-efficiency. With the coordinate-exchange algorithm, the following target function T is optimized:

$$T = \alpha_1 D_{\text{eff}} + \alpha_2 D_{s,\text{eff}} + \alpha_3 D_{1,\text{eff}}$$

Here, α is a weight vector specified by the user. Details on the D_{eff}, $D_{s,\text{eff}}$ and $D_{1,\text{eff}}$ can be found in the section *Optimality criteria for D-efficient designs*.

A Python script to generate D-efficient designs with 40 runs and 7 factors is shown below.

Example of Doptimize usage

```
>>> N=40; s=2; k=7;
>>> arrayclass=oapackage.arraydata_t(s, N, 0, k)
>>> print('We generate optimal designs with: %s' % arrayclass)
We generate optimal designs with: arrayclass: N 40, k 7, strength 0, s {2,2,2,2,2,2,2},␣
↪order 0
>>> alpha=[1,2,0]
>>> scores, dds, designs, ngenerated = oapackage.Doptimize(arrayclass, nrestarts=40,␣
↪optimfunc=alpha, selectpareto=True, verbose=0)
Doptimize: iteration 0/40
Doptimize: iteration 39/40
>>> print('Generated %d designs, the best D-efficiency is %.4f' % (len(designs), dds[:,
↪0].max() ))
Generated 10 designs, the best D-efficiency is 0.9198
```

The parameters of the `Doptimize()` function are documented in the code.

To calculate the D-, D_s- and D_1-efficiencies of the designs, we can use the method *Defficiencies()*. For details on these efficiencies, see the section *Optimality criteria for D-efficient designs* and [ES17].

In [ES17], it is shown that one can optimize a linear combination of the D-efficiency and D_s-efficiency to generate a rich set of good compromise designs. From the generated designs, the optimal ones according to Pareto optimality can be selected.

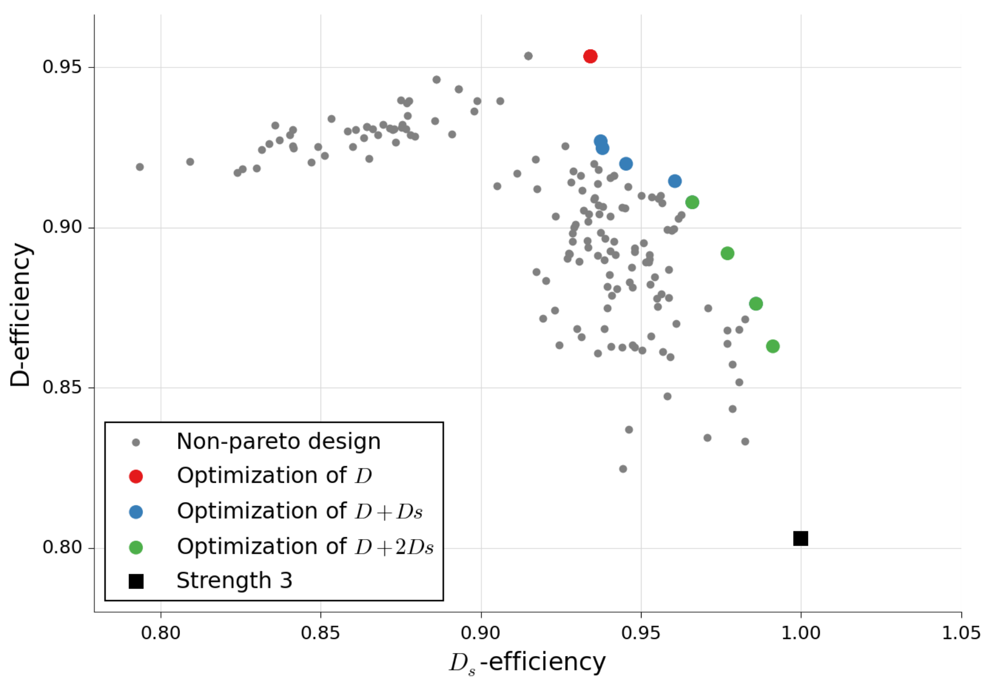


Fig. 1: Scatterplot for the D-efficiency and D_s-efficiency for generated designs in $\mathrm{OA}(40; 2; 2^7)$. The Pareto optimal designs are colored, while the non-Pareto optimal designs are grey. For reference the strength-3 orthogonal array with highest D-efficiency is also included in the plot.

4.4 Even-odd arrays

The even-odd arrays are a special class of orthognal arrays with at least one of the odd J_k-characteristics unequal to zero. More information on this class of designs will appear later.

CHAPTER

FIVE

NORMAL FORM OF ARRAYS

The Orthogonal Array package contains functions to reduce arrays and designs to canonical form with respect to some ordering. The default ordering for orthogonal arrays is the lexicographic ordering in columns [SEN10]. The default ordering for conference designs is the LMC0 ordering [SEG19]. Alternative orderings include the delete-one-factor projection ordering introduced in [Een13] or the even-odd ordering. For a given ordering of a set of arrays, the minimal element of all arrays in an isomorphism class defines a unique representative of that isomorphism class.

Specialized packages such as Nauty [McK81], [MP13] can also reduce arrays to their canonical form using state-of-the-art, graph-isomorphism methods. However, these methods do not take into account the special structure of the arrays and so, they cannot be tailored to create normal forms of a specific form.

5.1 Reduction to LMC normal form

The Orthogonal Array package implements theory and methods from the article Complete enumeration of pure-level and mixed-level orthogonal arrays, Schoen et al. to reduce orthogonal arrays to their LMC normal form. The C++ function to perform the reduction is *reduceLMCform()*. An example on how to use this function is shown below.

```
>>> import oapackage
>>> oapackage.set_srand(1)
>>> array = oapackage.exampleArray(1, 0).selectFirstColumns(3)
>>> array = array.randomperm()
>>> print('input array:'); array.transposed().showarraycompact()
input array:
1100010111010001
0101100110100011
0000001111101101
>>> reduced_array = oapackage.reduceLMCform(array)
>>> print('reduced array:'); reduced_array.transposed().showarraycompact()
reduced array:
0000000011111111
0000111100001111
0001011101110001
```

It is also possible to check whether an array is in normal form with the *LMCcheck()* method:

```
>>> import oapackage
>>> array = oapackage.exampleArray(1)
>>> lmc_type = oapackage.LMCcheck(array)
>>> if lmc_type == oapackage.LMC_MORE:
...     print('array is in minimal form')
```

(continues on next page)

(continued from previous page)

```
... elif lmc_type == oapackage.LMC_LESS:
...     print('array is not in minimal form')
array is in minimal form
```

5.2 Reduction to delete-one-factor projection form

The article A canonical form for non-regular arrays based on generalized word length pattern values of delete-one-factor projections [Een13] describes a canonical form of an orthogonal array based on delete-one-factor projections. The C++ interface to delete-one-factor projection form is *reduceDOPform()*. The reduction method works well for large arrays with a large variation in the projection values.

An example on how to use this reduction is shown in *Example code for delete-one-factor projections*, which can be found in the example notebooks section.

5.3 Reduction using graph isomorphisms

The function `reduceOAnauty()` reduces an orthogonal array to Nauty canonical form. To reduce general graphs to Nauty canonical form, the Orthogonal Array package includes the function `reduceGraphNauty()`.

Reduce a design to normal form using Nauty

```
>>> oapackage.set_srand(1)
>>> al = oapackage.exampleArray(0).randomperm()
>>> al.showarray()
array:
  0   0
  0   1
  1   1
  0   1
  1   0
  0   0
  1   0
  1   1
>>> transformation=oapackage.reduceOAnauty(al, 0)
>>> transformation.show()
array transformation: N 8
column permutation: {0,1}
level perms:
{0,1}
{0,1}
row permutation: {0,5,1,3,4,6,2,7}
>>> alr=transformation.apply(al)
>>> alr.showarray()
array:
  0   0
  0   0
  0   1
  0   1
```

(continues on next page)

(continued from previous page)

```
  1   0
  1   0
  1   1
  1   1
```

5.4 Normal forms for conference designs

For conference designs, a convenient normal form is the LMC0 ordering (sometimes also called L0 ordering) [SEG19].

LMC0 ordering

The LMC0 ordering for conference designs is defined in three steps:

Definition LMC0 i: ***Order of elements***
The LMC0 order of the factor levels -1, 0 and +1 is 0 < +1 < -1.

Definition LMC0 ii: ***Order of columns***
A column a is smaller than a column b according to LMC0 ordering, notated as a < b, if either of the following conditions hold:

1. If we replace the values -1 by +1 in both columns, then the first element where the columns differ is smaller in a than in b according to Definition 1.
2. The zeros in column a are in the same position as the zeros in column b, and the first element where the columns differ is smaller in a than in b according to Definition i.

Definition LMC0 iii: ***Order of designs***
Conference design A is smaller than conference design B according to LMC0 ordering, notated as A < B, if the first column where the designs differ is smaller in A than in B.

The definition implies that the ordering of designs is column-by-column and that the position of zeros in the columns is dominant over the values +1, -1. To check whether a design is in LMC0 form, we can use `LMC0check()`:

Conference design in normal form

```
>>> array = oapackage.exampleArray(53,1)
exampleArray 53: third array in C(12,4)
>>> array.showarray()
array:
  0   1   1   1
  1   0  -1   1
  1   1   0  -1
  1   1   1  -1
  1   1   1  -1
  1   1  -1   1
  1   1  -1   1
  1  -1   1   0
  1  -1   1   1
  1  -1   1   1
  1  -1  -1  -1
```

(continues on next page)

(continued from previous page)

```
  1  -1  -1  -1
>>> oapackage.LMC0check(array) == oapackage.LMC_LESS
True
```

CHAPTER

SIX

PROPERTIES OF DESIGNS

This section shows the structural and statistical properties of the orthogonal arrays, conference designs and D-efficient designs generated by the Orthogonal Array package. The properties of the arrays and designs are calculated using the *array_link* object or functions from the package.

6.1 Definitions of arrays and designs

Before introducing the structural and statistical properties, we define orthogonal arrays, conference designs and D-efficient designs:

An orthogonal array (OA) of strength t, N runs and n factors at s levels is an $N \times n$ array of symbols $0, \ldots, (s-1)$, such that for every subset of t columns, every t-tuple occurs equally often [Rao47]. The set of all strength-t OAs with N runs and n factors at s levels is denoted by $\mathrm{OA}(N; t; s^n)$. If $s = 2$, the OA is called a two-level OA and the set of all strength-t two-level OAs with N runs and n factors is denoted as $\mathrm{OA}(N; t; 2^n)$.

For N even, a conference design [SEG19] C is an $N \times n$ array which satisfies $C^T C = (n-1)I_n$, with $C_{ii} = 0$ and $C_{ij} \in \{-1, 1\}$, for $i \neq j$ and $i, j = 1, \ldots, n$. A $N \times N$ conference design E such that $EE^T = (n-1)I_n$ is called a conference matrix; see [EN95], [CD06] and [XLB12].

A D-optimal design [DAT07] (X) is an $N \times n$ array which maximizes the D-efficiency, defined as $(\det(X_M^T X_M)^{1/p})/N$, for a given $N \times p$ model matrix X_M (for details see *Model matrices*). The Orthogonal Array package uses a coordinate-exchange algorithm to generate designs that optimize the D-efficiency. Since there is no guarantee that the resulting designs have the largest possible D-efficiency, we refer to them as D-efficient designs in this documentation. An orthogonal array is called D-optimal orthogonal array if it provides the largest D-efficiency among all comparable orthogonal arrays.

6.2 Structural properties of an array

The OApackage can calculate the rank of an array, defined as the maximum number of linearly independent column or row vectors in the array. The rank of an array is useful for several other functions in the package. For two-level arrays, the OApackage can also check if the arrays are foldover arrays. A two-level array is called a foldover array if half of its runs are mirror images of the other half, in the sense that the factor levels are changed from 0 to 1 and from 1 to 0.

For example, to calculate the rank of a two-level orthogonal array and determine whether the array is a foldover array, one can use the methods *array_link::rank()* and *array_link::foldover()*:

Calculate rank of array and test for foldover

```
>>> array = oapackage.exampleArray(1) # Select an example two-level orthogonal array
>>> array.showarray() # Show the two-level orthogonal array
array:
  0   0   0   0   0
  0   0   0   0   0
  0   0   0   1   1
  0   0   1   0   1
  0   1   0   1   0
  0   1   1   0   0
  0   1   1   1   1
  0   1   1   1   1
  1   0   0   1   1
  1   0   1   0   1
  1   0   1   1   0
  1   0   1   1   0
  1   1   0   0   1
  1   1   0   0   1
  1   1   0   1   0
  1   1   1   0   0
>>> print(array.rank()) # Calculate the rank of the array
5
>>> print(array.foldover()) # Determine if the array is foldover
False
```

Other structural properties such as whether an array involves two levels or is symetric can be found in the documentation of *array_link*, which shows the full set of methods available.

6.3 Model matrices

For orthogonal arrays and conference designs, the OApackage can calculate different model matrices. The model matrices available depend on the type of array and design.

Model matrices for two-level orthogonal arrays

For two-level orthogonal arrays, the levels of the array are first coded according to the map $0 \to -1$ and $1 \to +1$. The coded matrix is referred to as the design matrix. The main effect contrast vectors are given by the columns in the design matrix. The contrast vectors associated to the two-factor interactions are calculated by taking products between two different columns in the design matrix. The model matrix consists of the intercept column (i.e. a columns of ones) and the contrast vectors associated to the main effects and, optionally, the two-factor interactions.

Model matrices for conference designs

The model matrix for a conference design consists of the intercept column (i.e. a columns of ones) and the contrast vectors associated to the main effects and, optionally, the second-order effects (two-factor interactions and quadratic effects). The main effect contrast vectors are given by the columns in the conference design. The contrast vectors associated to the second-order effects are calculated by taking products between two columns in the conference design.

Model matrices for mixed-level orthogonal arrays

For mixed-level orthogonal arrays, the main effect contrast vectors are defined by the Helmert contrasts. The contrast vectors associated to the two-factor interactions are calculated by taking products between two different columns in the matrix containing the Helmert contrasts of the array; see Model matrices for mixed-level orthogonal arrays for details. The model matrix consists of the intercept column (i.e. a columns of ones) and the contrast vectors associated to the main effects and, optionally, the two-factor interactions.

An example on how to generate an interaction model matrix for a two-level orthogonal array is shown below.

Calculate interaction effects model matrix

```
>>> array=oapackage.exampleArray(0,1)
exampleArray 0: array in OA(8,2, 2^2)
>>> array.showarray()
array:
  0   0
  0   0
  0   1
  0   1
  1   0
  1   0
  1   1
  1   1
>>> M=oapackage.array2modelmatrix(array, 'i')
>>> print(M)
[[ 1. -1. -1.  1.]
 [ 1. -1. -1.  1.]
 [ 1. -1.  1. -1.]
 [ 1. -1.  1. -1.]
 [ 1.  1. -1. -1.]
 [ 1.  1. -1. -1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
```

6.4 Statistical properties of orthogonal arrays

Orthogonal arrays are commonly evaluated in terms of their generalized wordlength pattern [XW01] (GWLP). Two-level OAs are also commonly evaluated in terms of their J_k-characteristics and F-vectors [DT99]. The OApackage can calculate all these statistical criteria: *array_link::GWLP()*, *array_link::Fvalues()*, *array_link::Jcharacteristics()*.

The following example shows how to calculate the GWLP, F_k-values and J_k-characteristics from an *array_link* object:

Calculate GWLP and F-values

```
>>> al=oapackage.exampleArray(1,1) # Select an example array
exampleArray 1: array 3 in OA(16, 2, 2^5)
>>> gwlp = al.GWLP() # Calculate its generalized word length pattern
>>> print('GWLP: %s'% str(gwlp) )
GWLP: (1.0, 0.0, 0.0, 1.0, 1.0, 0.0)
>>> print('F3-value: %s' % str(al.Fvalues(3))) # Calculate the F_3 values
F3-value: (4, 6)
>>> print('F4-value: %s' % str(al.Fvalues(4))) # Calculate the F_3 values
F4-value: (1, 4)
>>> print('J3-characteristics: %s' % str(al.Jcharacteristics(3))) # Calculate the J_3-
↪characteristics
J3-characteristics: (-8, -8, 0, 0, 0, -8, 0, -8, 0, 0)
```

We now briefly mention some technical details of the J_k-characteristics, the F_k-values and the GWLP.

J_k-characteristics

To calculate J_k-characteristics of a two-level OA, the OApackage codes the levels of the array as -1 and $+1$. To this end, the package uses the mapping $0 \to -1$ and $1 \to +1$. Let D be an $N \times n$ with coded levels -1 and $+1$. For $S = \{l_1, \ldots, l_k\}$, a subset of k different factors of $D = (d_{il})$, define

$$j_k(S; D) = \sum_{i=1}^{N} d_{il_1} \cdots d_{il_k}.$$

The $|j_k(S; D)|$ values are called the J_k-characteristics, which necessarily equal $N - 4q$ [DT02], where $q \leq N/4$ is a non-negative integer.

F_k-values

The F_k-vector collects the frequencies of all the J_k-characteristics. More specifically, the vector $F_k = (f_{k1}, \ldots, f_{kv})$, where $v = N/4$ and f_{ku} denotes the frequency of the J_k-characteristics which are equal to $4(v + 1 - u)$. When calculating an F_k-vector, the OApackage shows only the vector $(f_{k1}, \ldots, f_{kv})$, whose elements are referred to as the F_k-values.

Generalized word length pattern

Consider an OA, D, of strength t with N runs and n factors at s levels. Let X_0 be a column of ones, X_1 the matrix involving the contrast vectors associated with the main effects, and X_j the matrix involving the contrast vectors associated with the j-factor interactions, $j \geq 2$. We assume that the column vectors in X_1 are normalized so that they have the same length $\sqrt{N}$. For $j = 0, \ldots, n$, let

$$A_j(D) = N^{-2} 1_N^T X_j X_j^T 1_N,$$

where 1_N denotes the $N \times 1$ column of ones. The value of $A_j(D)$ is invariant to the choice of the orthonormal contrasts used; see [XW01] for details. The vector $(A_0(D), \ldots, A_n(D))$ is called the generalized word length pattern (GWLP). To increase the speed of the computations for the GWLP, the OApackage uses the distance distribution and the MacWilliams identities as in [XW01] and [Xu09].

6.5 Optimality criteria for D-efficient designs

In [ES17], D-efficient designs for the model including the intercept, all main effects and all two-factor interactions are generated. The OApackage provides functionality to compute the optimality criteria used to generate the D-efficient designs in [ES17]. Moreover, the package can calculate the well-known A- and E-optimality criteria from the literature on Optimal Experimental Design [DAT07]. The functions to perform the calculations are *array_link::Defficiency()*, *array_link::DsEfficiency()*, *array_link::Aefficiency()*, *array_link::Eefficiency()*.

The following example shows how to calculate the D-, D_s-, A- and E-efficiency for a design that permits the estimation of the interaction model.

Calculate optimality criteria for D-efficient designs

```
# Select an array that can estimate the interaction model
>>> al = oapackage.exampleArray(11, 1)
exampleArray 11: D-optimal array in OA(44, 2^8)
>>> print('D-efficiency: %.4f' % al.Defficiency())
D-efficiency: 0.8879
>>> print('Ds-efficiency (Eendebak and Schoen, 2017): %.4f' % al.DsEfficiency())
Ds-efficiency (Eendebak and Schoen, 2017): 0.8059
>>> print('A-efficiency for the interaction model: %.4f' % al.Aefficiency())
A-efficiency for the interaction model: 0.7906
>>> print('E-efficiency for the interaction model: %.4f' % al.Eefficiency())
E-efficiency for the interaction model: 0.3602
```

Calculation of D-, A- and E-efficiency

Let X be again the $N \times p$ interaction model matrix (see section *Model matrices*) consisting of a column of ones and the contrast vectors associated to the main and two-factor interactions of n factors, where $p = 1 + n + (n)(n-1)/2$. The D-, A- and E-efficiency are calculated using the eigenvalues of the singular-value decomposition (SVD) of X. To calculate the rank of a matrix, the lower-upper (LU) decomposition, as implemented in the Eigen package [GJ+10], is used.

Let $\lambda_1, \ldots, \lambda_p$ be the eigenvalues of the SVD of X. The OApackage calculates the D-, A- and E-efficiency of a design D as follows:

$$D_{\text{eff}}(D) = (\prod_j \lambda_j)^{1/p}/N,$$

$$A_{\text{eff}}(D) = N(\sum_j \lambda_j^{-1})/m,$$

$$E_{\text{eff}}(D) = \min_j \lambda_j.$$

D_s-efficiency and D_1-efficiency

In [ES17], the D_s-efficiency is used to assess the joint precision of the main effects in the interaction model. Let the interaction model matrix X be split into X_1, containing the contrast vectors associated with the main effects only, and X_{02}, containing the intercept column and the contrast vectors associated to the two-factor interactions. The D_s-criterion of a design D is defined as

$$D_{s,\text{crit}}(D) = \det(X^T X)/\det(X_{02}^T X_{02}),$$

where X_{02} is necessarily of full rank. Similar to the calculations of the D-efficiency, the OApackage calculates the D_s-criterion using the eigenvalues of the SVD of the matrices X and X_{01}. Finally, the package calculates the D_s-efficiency of D as $D_{s,\text{eff}}(A) = D_{s,\text{crit}}(A)^{1/m}$, where m is the number of factors.

In a similar way the D_1-efficiency of a design A with n factors and model matrix of intercept and main effects X_{01}, is defined as

$$D_{1,\text{eff}}(A) = (\det((X_{01})^T (X_{01}))^{1/(n+1)}$$

6.6 Projection capacities

Other relevant statistical criteria to evaluate a two-level design with N runs and k factors include the so-called projection estimation capacity (PEC) and projection information capacity (PIC) [LST07]. These criteria focus on the projections of the two-level design onto a smaller number of factors. More specifically, the PEC and PIC summarize the performance of all the N-run subdesigns with $l \leq k$ factors in terms of the capacity to estimate the interaction model and the D-efficiency for this model, respectively.

The PEC and PIC are based on the so-called PEC and PIC sequences, which are formally defined as follows. Let PEC_l denote the proportion of N-run l-factor subdesigns that permit the estimation of the interaction model in l factors, that is, the model including the intercept, all l main effects and all $l(l-1)/2$ two-factor interactions. The PEC sequence is the vector $(PEC_1, PEC_2, \ldots, PEC_k)$. Now, let PIC_l denote the average D-efficiency for the interaction model in l factors accross all N-run l-factor subdesigns. The PIC sequence is the vector $(PIC_1, PIC_2, \ldots, PIC_k)$. The OApackage can calculate the PEC and PIC sequences of two-level designs with `PECsequence()` and `PICsequence()`, respectively.

The following example shows how to compute the PEC and PIC sequences of a two-level orthogonal array using the OApackage.

Calculate the PEC and PIC sequences

```
>>> al=oapackage.exampleArray(1,1)
exampleArray 1: array 3 in OA(16, 2, 2^5)
>>> PEC = al.PECsequence()
>>> print('PEC sequence: %s'% ','.join(['%.2f' % x for x in PEC]) )
PEC sequence: 1.00,1.00,1.00,0.80,0.00
>>> PIC = al.PICsequence()
>>> print('PIC sequence: %s'% ','.join(['%.2f' % x for x in PIC]) )
PIC sequence: 1.00,1.00,0.95,0.66,0.00
```

6.7 Properties of conference designs

In [SEG19], it is shown that the F_4 vector is useful for classifying definitive screening designs [XLB12] that are generated by folding over a conference design. To calculate the F_4 vector, we first need to compute the J_4-characteristics of the conference design. The calculations for the J_k-characteristics of conference designs are similar as for orthogonal arrays; see *Statistical properties of orthogonal arrays*. Consider a definitive screening design constructed from an N-run conference design with at least four factors. The F_4 vector of this design collects the frequencies of the J_4-characteristics of $2N8\lambda$ for $\lambda = 1, \ldots, N/4$ when N is a multiple of 4, or $\lambda = 1, \ldots, (N2)/4$ when N is an odd multiple of 2.

Note: the J_4-values of a definitive screening design generated by folding over a conference design are twice the value of the J_4-values of the conference design. The F_4 vector of a conference design and the corresponding definitive screening design are equal.

Calculate the F4 vector for a conference design

```
>>> import oapackage
>>> array=oapackage.exampleArray(47,1)
exampleArray 47: third conference design in C(20,8)
>>> F4=array.FvaluesConference(4)
>>> print(F4)
(0, 2, 4, 51, 13)
>>> definitive_screening_design = oapackage.conference2DSD(array)
```

The individual J_k-characteristics can be calculated with the method *Jcharacteristics_conference()*. For conference designs, we can calculate the projection statistics using `conferenceProjectionStatistics()`.

Calculate projection statistics for conference designs

```
>>> array = oapackage.exampleArray(46, 1)
exampleArray 46: second conference design in C(20,8)
>>> pec, pic, ppc = oapackage.conference.conferenceProjectionStatistics(array)
>>> print('Projection estimation capacity for 4 columns: %.3f'  % pec)
Projection estimation capacity for 4 columns: 0.986
>>> J3 = oapackage.Jcharacteristics_conference(array, number_of_columns = 3)
```

CHAPTER

SEVEN

C++ LIBRARY

The full documentation for the C++ library can be build from the source code using doxygen. For convenience we provide links the main functions on this page.

7.1 Interface for D-optimal designs

Contains functions to generate optimal designs.

For more information see "Two-Level Designs to Estimate All Main Effects and Two-Factor Interactions", P.T. Eendebak and E.D. Schoen, 2017

Enums

enum **coordinate_exchange_method_t**

Different methods for the optimization. The default method DOPTIM_SWAP is a coordinate-exchange algorithms.

Values:

enumerator **DOPTIM_UPDATE**

replace a random element with a random value

enumerator **DOPTIM_SWAP**

swap two elements at random

enumerator **DOPTIM_FLIP**

randomly flip an element between 0 and 1

enumerator **DOPTIM_AUTOMATIC**

automatically select one of the methods

enumerator **DOPTIM_NONE**

perform no optimization

Functions

double **scoreD**(const std::vector<double> efficiencies, const std::vector<double> weights)

Calculate score from a set of efficiencies

The score is the weighted sum of the efficiencies.

Parameters

- **efficiencies** – Vector with calculated efficiencies
- **weights** – Weights for the efficiencies

Returns
Weighted sum of the efficiencies

DoptimReturn **Doptimize**(const *arraydata_t* &arrayclass, int nrestarts, const std::vector<double> alpha, int verbose, *coordinate_exchange_method_t* method = DOPTIM_AUTOMATIC, int niter = 300000, double maxtime = 100000, int nabort = 5000)

Generates optimal designs for the specified class of designs

The method uses a coordinate-exchange algorithm to optimze a target function defined by the optimziation paramaters. The optimization is performed multiple times to prevent finding a design in a local minmum of the target function.

The method is described in more detail in "Two-Level Designs to Estimate All Main Effects and Two-Factor Interactions", Eendebak et al., 2015, Technometrics, https://doi.org/10.1080/00401706.2016.1142903.

Parameters

- **arrayclass** – Class of designs to optimize
- **nrestarts** – Number of restarts to perform
- **alpha** – Optimization parameters. The target function is alpha_1 D + alpha_2 D_s + alpha D_1
- **verbose** – Verbosity level
- **method** – Method for optimization algorithm
- **niter** – Maximum number of iterations for each restart
- **maxtime** – Maximum calculation time. If this time is exceeded, the function is aborted
- **nabort** – Maximum number of iterations when no improvement is found

Returns
A structure with the generated optimal designs

DoptimReturn **DoptimizeMixed**(const *arraylist_t* &sols, const *arraydata_t* &arrayclass, const std::vector<double> alpha, int verbose = 1, int nabort = -1)

Function to generate optimal designs with mixed optimization approach

This function is beta code. See Doptimize for detauls of the parameters.

array_link **optimDeff**(const *array_link* &array, const *arraydata_t* &arrayclass, std::vector<double> alpha, int verbose = 1, *coordinate_exchange_method_t* optimmethod = DOPTIM_AUTOMATIC, int niter = 100000, int nabort = 0)

Optimize a design according to the optimization function specified.

Arguments:

Parameters

- **array** – Array to be optimized
- **arrayclass** – Structure describing the design class
- **alpha** – 3x1 array with optimization parameters
- **verbose** – Verbosity level
- **optimmethod** – Optimization method to use
- **niter** – Number of iterations
- **nabort** – Number of iterations after which to abort when no improvements are found

Returns
Optimized designs

struct **DoptimReturn**

#include <Deff.h> Structure containing results of the Doptimize function

Public Members

std::vector<std::vector<double>> **dds**

calculated efficiencies for the generated designs

arraylist_t **designs**

designs generated

int **nrestarts**

number of restarts performed

int **_nimproved**

7.2 Interface for array properties

Contains functions to calculate properties of arrays.

Author: Pieter Eendebak pieter.eendebak@gmail.com Copyright: See LICENSE.txt file that comes with this distribution

Defines

GWLPvalue

Typedefs

typedef mvalue_t<double> **DOFvalue**

delete-one-factor projection value

Enums

enum **model_matrix_t**

Values:

enumerator **MODEL_CONSTANT**

only the intercept

enumerator **MODEL_MAIN**

intercept and main effects

enumerator **MODEL_INTERACTION**

intercept, main effects and second order interactions

enumerator **MODEL_SECONDORDER**

intercept, main effects and second order effects(interactions and quadratic effects)

enumerator **MODEL_INVALID**

invalid model

enum **paretomethod_t**

Values:

enumerator **PARETOFUNCTION_DEFAULT**

enumerator **PARETOFUNCTION_J5**

Functions

void **DAEefficiencyWithSVD**(const Eigen::MatrixXd &secondorder_interaction_matrix, double &Deff, double &vif, double &Eeff, int &rank, int verbose)

Calculate D-efficiency and VIF-efficiency and E-efficiency values using SVD.

int **array2rank_Deff_Beff**(const *array_link* &al, std::vector<double> *ret = 0, int verbose = 0)

Calculate the rank of the second order interaction matrix of an orthogonal array

The model is the intercept, main effects and interaction effects The rank, D-efficiency, VIF-efficiency and E-efficiency are appended to the second argument

The return vector is filled with the rank, Defficiency, VIF efficiency and Eefficiency double **Defficiency**(const *array_link* &orthogonal_array, int verbose = 0)

Calculate D-efficiency for a 2-level array using symmetric eigenvalue decomposition.

std::vector<double> **Defficiencies**(const *array_link* &array, const *arraydata_t* &arrayclass, int verbose = 0, int addDs0 = 0)

Calculate efficiencies for an array

Parameters

- **array** – Array to use in calculation
- **arrayclass** – Specification of the array class
- **verbose** – Verbosity level
- **addDs0** – If True, then add the Ds0-efficiency to the output

Returns

Vector with the calculate D-efficiency, the main effect robustness (or Ds-optimality) and D1-efficiency for an orthogonal array

double **VIFefficiency**(const *array_link* &orthogonal_array, int verbose = 0)

Calculate VIF-efficiency of matrix.

double **Aefficiency**(const *array_link* &orthogonal_array, int verbose = 0)

Calculate A-efficiency of matrix.

double **Eefficiency**(const *array_link* &orthogonal_array, int verbose = 0)

Calculate E-efficiency of matrix (1 over the VIF-efficiency)

std::vector<double> **Aefficiencies**(const *array_link* &orthogonal_array, int verbose = 0)

calculate various A-efficiencies

std::vector<double> **projDeff**(const *array_link* &array, int number_of_factors, int verbose = 0)

Calculate D-efficiencies for all projection designs

Parameters

- **array** – Design to calculate D-efficiencies for
- **number_of_factors** – Number of factors into which to project
- **verbose** – Verbosity level

Returns

Vector with calculated D-efficiencies

std::vector<double> **PECsequence**(const *array_link* &array, int verbose = 0)

Calculate the projection estimation capacity sequence for a design

The PECk of a design is the fraction of estimable second-order models in k factors. The vector (PEC1, PEC2, . . . ,) is called the projection estimation capacity sequence. See "Ranking Non-regular Designs", J.L. Loeppky, 2004.

Parameters

- **array** – Input array
- **verbose** – Verbosity level

Returns

Vector with the caculated PEC sequence

std::vector<double> **PICsequence**(const *array_link* &array, int verbose = 0)

Calculate the projection information capacity sequence for a design.

The PICk of a design is the average D-efficiency of estimable second-order models in k factors. The vector (PIC1, PIC2, . . . ,) is called the PIC sequence.

Parameters

- **array** – Input array
- **verbose** – Verbosity level

Returns

Vector with the caculated PIC sequence

ndarray<double> **macwilliams_transform_mixed**(const *ndarray*<double> &B, int N, const std::vector<int> &factor_levels_for_groups, int verbose = 0)

Calculate MacWilliams transform.

Calculate MacWilliams transform for mixed level data.

Parameters

- **B** – Input array
- **N** –
- **verbose** – Verbosity level
- **factor_levels_for_groups** – Factor levels for the groups
- **B** – Input array
- **N** – Number of runs
- **factor_levels_for_groups** – Factor levels
- **verbose** – Verbosity level

Returns

MacWilliams transform of the input array

Returns

Transform if the input array

std::vector<double> **distance_distribution**(const *array_link* &array)

Return the distance distribution of a design

The distance distribution is described in "Generalized minimum aberration for asymmetrical fractional factorial designs", Wu and Xu, 2001

Parameters

al – Array for which to calculate the distribution

Returns

Distance distribution

std::vector<int> **distance_distribution_shape**(const *arraydata_t* arrayclass)

Return shape of distance distribution for mixed level design

Parameters

arrayclass – Specification of the array class

Returns
Shape of the distance distribution

ndarray<double> **distance_distribution_mixed**(const *array_link* &array, int verbose = 0)

Return the distance distribution of a mixed-level design

The distance distribution is described in "Generalized minimum aberration for asymmetrical fractional factorial designs", Wu and Xu, 2001. For mixed-level designs more details can be found in "A canonical form for non-regular arrays based on generalized wordlength pattern values of delete-one-factor projections", Eendebak, 2014.

Parameters

- **al** – Array for which to calculate the distribution
- **verbose** – Verbosity level

Returns
Distance distribution

void **distance_distribution_mixed_inplace**(const *array_link* &al, *ndarray*<double> &B, int verbose = 0)

template<class **Type**>
std::vector<double> **macwilliams_transform**(std::vector<*Type*> B, int N, int s)

Calculate MacWilliams transform.

std::vector<int> **Jcharacteristics**(const *array_link* &array, int number_of_columns = 4, int verbose = 0)

Calculate Jk-characteristics of a matrix

The calculated Jk-values are signed.

Parameters

- **array** – Array to calculate Jk-characteristics for
- **number_of_columns** – Number of columns
- **verbose** – Verbosity level

Returns
Vector with calculated Jk-characteristics

std::vector<double> **GWLP**(const *array_link* &array, int verbose = 0, int truncate = 1)

Calculate GWLP (generalized wordlength pattern)

The method used for calculation is from Xu and Wu (2001), "Generalized minimum aberration for asymmetrical

fractional factorial desings". For non-symmetric arrays see "Algorithmic Construction of Efficient Fractional Factorial Designs With Large Run

Sizes", Xu, Technometrics, 2009.

A more detailed description of the generalized wordlength pattern can also be found in the documentation at https://oapackage.readthedocs.io/.

Parameters

- **array** – Array to calculate the GWLP value for
- **verbose** – Verbosity level
- **truncate** – If True then round values near zero to solve double precision errors

Returns
Vector with calculated generalized wordlength pattern

std::vector<double> **GWLPmixed**(const *array_link* &array, int verbose = 0, int truncate = 1)

Calculate GWLP (generalized wordlength pattern) for mixed-level arrays.

The method used for calculation is from "Algorithmic Construction of Efficient Fractional Factorial Designs With Large Run

Sizes", Xu, Technometrics, 2009.

Parameters

- **array** – Array to calculate the GWLP value for
- **verbose** – Verbosity level
- **truncate** – If True then round values near zero to solve double precision errors

Returns
Vector with calculated generalized wordlength pattern

std::vector<GWLPvalue> **projectionGWLPs**(const *array_link* &al)

calculate delete-one-factor GWLP (generalized wordlength pattern) projections

std::vector<GWLPvalue> **sortGWLP**(std::vector<GWLPvalue>)

sort a list of GWLP values and return the sorted list

double **CL2discrepancy**(const *array_link* &array)

Calculate centered L2-discrepancy of a design

The method is from "A connection between uniformity and aberration in regular fractions of two-level factorials", Fang and Mukerjee, 2000

array_link **array2secondorder**(const *array_link* &array)

Calculate second order interaction model for 2-level array

Parameters
array – Array to calculate second order interaction model from

Returns
Array interaction effects

array_link **array2xf**(const *array_link* &array)

calculate second order interaction model for 2-level array

Parameters
array – Array to calculate second order interaction model from

Returns
Array with intercept, main effects and interaction effects

array_link **conference_design2modelmatrix**(const *array_link* &conference_design, const char *mode, int verbose = 0)

Calculate model matrix for a conference design

Parameters

- **conference_design** – Conference design
- **mode** – Can be 'm' for main effects, 'i' for interaction effects or 'q' for quadratic effects

- **verbose** – Verbosity level

Returns
Calculated model matrix

Eigen::MatrixXd **array2modelmatrix**(const *array_link* &array, const char *mode, int verbose = 0)

Convert orthogonal array or conference design to model matrix

The model matrix consists of the intercept, main effects and (optionally) the interaction effects and quadratic effects. The order in the interaction effects is (c1, c2)=(0,0), (1,0), (2,0), (2,1), … with c2<c1 for columns c1, c2. The size of the model matrix calculated by this function is given by *array2modelmatrix_sizes*.

For conference designs the method *conference_design2modelmatrix* is used. For orthogonal array the calculated is performed with *array2eigenModelMatrixMixed*.

Parameters

- **array** – Orthogonal array or conference design
- **mode** – Type of model matrix to calculate. Can be 'm' for main effects, 'i' for interaction effects or 'q' for quadratic effects
- **verbose** – Verbosity level

Returns
Calculated model matrix

std::vector<int> **array2modelmatrix_sizes**(const *array_link* &array)

Return the sizes of the model matrices calculated

Parameters
array – Orthogonal array or conference designs

Returns
List with the sizes of the model matrix for: only intercept; intercept, main; intercept, main, and iteraction terms, intercept, main and full second order

Eigen::MatrixXd **array2xfeigen**(const *array_link* &array)

calculate second order interaction model for 2-level array

Parameters
array – Array to calculate second order interaction model from

Returns
Array with intercept, main effects and interaction effects

int **arrayrankFullPivQR**(const *array_link* &al, double threshold = -1)

return rank of an array based on Eigen::FullPivHouseholderQR

int **arrayrankColPivQR**(const *array_link* &al, double threshold = -1)

return rank of an array based on Eigen::ColPivHouseholderQR

int **arrayrankFullPivLU**(const *array_link* &al, double threshold = -1)

return rank of an array based on Eigen::FullPivLU

int **arrayrankSVD**(const *array_link* &al, double threshold = -1)

return rank of an array based on Eigen::JacobiSVD

int **arrayrank**(const *array_link* &array)

calculate the rank of an array

int **arrayrankInfo**(const Eigen::MatrixXd&, int verbose = 1)

Return rank of an array. Information about the different methods for rank calculation is printed to stdout.

int **arrayrankInfo**(const *array_link* &array, int verbose = 1)

Return rank of an array. Information about the different methods for rank calculation is printed to stdout.

Eigen::MatrixXd **arraylink2eigen**(const *array_link* &array)

convert *array_link* to Eigen matrix

double **conditionNumber**(const *array_link* &matrix)

Return the condition number of a matrix.

void **calculateParetoEvenOdd**(const std::vector<std::string> infiles, const char *outfile, int verbose = 1, arrayfilemode_t afmode = ABINARY, int nrows = -1, int ncols = -1, *paretomethod_t* paretomethod = PARETOFUNCTION_DEFAULT)

Calculate the *Pareto* optimal arrays from a list of array files

Pareto optimality is calculated according to (rank; A3,A4; F4)

Pareto<mvalue_t<long>, long> **parsePareto**(const *arraylist_t* &arraylist, int verbose, *paretomethod_t* paretomethod = PARETOFUNCTION_DEFAULT)

mvalue_t<long> **A3A4**(const *array_link* &al)

calculate A3 and A4 value for array

Parameters

al – Array for which to calculate A3 and A4

Returns

Object with A3 and A4

inline mvalue_t<long> **F4**(const *array_link* &al, int verbose = 1)

calculate F4 value for 2-level array

template<class **IndexType**>
Pareto<mvalue_t<long>, *IndexType*>::pValue **calculateArrayParetoRankFA**(const *array_link* &array, int verbose)

Calculate properties of an array and create a *Pareto* element

The values calculated are:

1) Rank (higher is better) 2) A3, A4 (lower is better) 3) F4 (lower is better, sum of elements is constant)

Valid for 2-level arrays of strength at least 3

template<class **IndexType**>
void **addJmax**(const *array_link* &al, typename *Pareto*<mvalue_t<long>, *IndexType*>::pValue &p, int verbose = 1)

add Jmax criterium to *Pareto* set

template<class **IndexType**>
Pareto<mvalue_t<long>, *IndexType*>::pValue **calculateArrayParetoJ5**(const *array_link* &al, int verbose)

Calculate *Pareto* element with J5 criterium.

template<class **IndexType**>
inline void **parseArrayPareto**(const *array_link* &array, *IndexType* i, *Pareto*<mvalue_t<long>, *IndexType*> &pset, int verbose)

Add array to list of *Pareto* optimal arrays

The values to be optimized are:

1) Rank (higher is better) 2) A3, A4 (lower is better) 3) F4 (lower is better, sum of elements is constant)

inline double **Cvalue2Dvalue**(double Cvalue, int number_of_columns)

convert C value to D-efficiency value

inline double **Dvalue2Cvalue**(double Defficiency, int number_of_columns)

convert D-efficiency value to C value

template<class **Type**>

class **ndarray**

#include <arrayproperties.h> Class representing an n-dimensional array

The data is stored in a flat array. The dimensions are stored in a vector `dims`.

Public Functions

inline **ndarray**(const std::vector<int> dims)

Class represensing an n-dimensional array.

Parameters

dims – Dimension of the array

inline **ndarray**(const *ndarray*<*Type*> &rhs)

Copy constructor Copies the internal data

inline **~ndarray**()

inline void **initialize**(const *Type* value)

Initialize array with specified value.

inline int **sizeof_type**() const

Return size of ndarray template type.

inline bool **type_is_floating_point**() const

Return True is the data type is of floating point type.

inline void **info**() const

inline std::string **idxstring**(int linear_idx) const

Convert linear index to string representing the index.

inline long **totalsize**() const

size of the array (product of all dimensions)

inline void **show**() const

print the array to stdout

inline void **linear2idx**(int ndx, int *nidx = 0) const

convert a linear index to normal indices

inline void **linear2idx**(int ndx, std::vector<int> &nidx) const

convert a linear index to normal indices

inline int **getlinearidx**(int *idx) const

From an n-dimensional index return the linear index in the data.

inline void ***data_pointer**() const

Return pointer to data.

inline void **setconstant**(*Type* val)

set all values of the array to specified value

inline void **set**(int *idx, *Type* val)

set value at position

inline void **setlinear**(int idx, *Type* val)

set value using linear index

inline *Type* **getlinear**(int idx) const

get value using linear index

inline *Type* **get**(int *idx) const

get value using n-dimensional index

Public Members

Type ***data**

std::vector<int> **dims**

int **k**

dimensions of the array

int **n**

std::vector<int> **cumdims**

std::vector<int> **cumprod**

Private Functions

inline void **initialize_internal_structures**(const std::vector<int> dimsx)

Initialize internal structures

Data pointer is created, but not set with data

Parameters

dimsx – Dimensions of the array

class **rankStructure**

#include <arrayproperties.h> Structure to efficiently calculate the rank of the second order interaction matrix of many arrays

The efficiency is obtained if the arrays share a common subarray. The theory is described in "Efficient rank calculation for matrices with a common submatrix", Eendebak, 2016

Public Types

typedef Eigen::FullPivHouseholderQR<Eigen::MatrixXd> **EigenDecomp**

Public Functions

inline **rankStructure**(const *array_link* &al, int nsub = 3, int verbose = 0)
constructor

inline **rankStructure**(int nsub = 3, int id = -1)
constructor

void **info**() const
print information about the rank structure

void **updateStructure**(const *array_link* &al)
update the structure cache with a new array

int **rankdirect**(const Eigen::MatrixXd &array) const
calculate the rank of an array directly, uses special threshold

int **rankxfdirect**(const *array_link* &array) const
calculate the rank of the second order interaction matrix of an array directly

int **rankxf**(const *array_link* &array)
calculate the rank of the second order interaction matrix of an array using the cache system

Public Members

array_link **alsub**

int **r**

int **verbose**
verbosity level

int **ks**
number of columns of subarray in cache

int **nsub**
number of columns to subtract from array when updating cache

int **id**
used for debugging

Private Members

EigenDecomp **decomp**
decomposition of subarray

Eigen::MatrixXd **Qi**

long **ncalc**
internal structure

long **nupdate**

7.3 Interface for array tools

Contains the *array_link* class and related classes.

This file contains method and classes to work with (orthogonal) arrays.

Author: Pieter Eendebak pieter.eendebak@gmail.com Copyright: See LICENSE.txt file that comes with this distribution

Defines

MPI_ARRAY_T

Typedefs

typedef Eigen::MatrixXd **MatrixFloat**

typedef Eigen::ArrayXd **ArrayFloat**

typedef Eigen::VectorXd **VectorFloat**

typedef double **eigenFloat**

typedef short int **array_t**
data type for elements of orthogonal arrays

typedef const short int **carray_t**
constant version of array_t

typedef short int **rowindex_t**

typedef int **colindex_t**

type used for row indexing

typedef const int **const_colindex_t**

type used for column indexing

typedef *array_t* ***array_p**

pointer to array

constant version of type used for column indexing

typedef *carray_t* ***carray_p**

pointer to constant array

typedef *rowindex_t* ***rowperm_t**

typedef *colindex_t* ***colperm_t**

type of row permutation

typedef *array_t* ***levelperm_t**

type of column permutation

typedef int **vindex_t**

type of level permutation

typedef signed char **conf_t**

data type for elements of conference designs

typedef std::vector<*conf_t*> **conference_column**

data type for column of a conference design

typedef std::vector<*conference_column*> **conference_column_list**

list of columns of conference designs

typedef std::deque<*array_link*> **arraylist_t**

container with arrays

Enums

enum **ordering_t**

Values:

enumerator **ORDER_LEX**

lexicograph minimal by columns ordering

enumerator **ORDER_J5**

J5 based ordering.

Functions

void **throw_runtime_exception**(const std::string exception_message)

void **eigenInfo**(const *MatrixFloat* m, const char *str = "eigen", int verbose = 1)

Print information about an Eigen matrix

Parameters

- **m** – Matrix about which to print information
- **str** – String to prepend in output
- **verbose** – Verbosity level

void **print_eigen_matrix**(const *MatrixFloat* matrix)

Print Eigen matrix to stdout

void **eigen2numpyHelper**(double *pymat1, int n, const *MatrixFloat* &m)

int **sizeof_array_t**()

return size in bytes of array_t type

int **sizeof_double**()

return size in bytes of double type

inline std::vector<int> **possible_F_values**(int N, int strength)

possible values for J-values of 2-level design

bool **file_exists**(const std::string filename)

return true if the specified file exists

bool **file_exists**(const char *filename)

return true if the specified file exists

bool **oa_file_exists**(const char *filename)

return true if the specified oa file exists

bool **oa_file_exists**(const std::string filename)

return true if the specified oa file exists

arraydata_t ***readConfigFile**(const char *file)

Read array configuration from file.

inline void **copy_array**(const *array_t* *src, *array_t* *const dst, const int nrows, const int ncols)

Make a copy of an array.

inline int **destroy_array**(*array_t* *array)

Delete an array.

Parameters

array –

Returns

static inline *array_t* ***create_array***(const int nrows, const int ncols)

Create an array.

Parameters

- **nrows** – Number of rows
- **ncols** – Number of columns

Returns

inline *array_t* ***create_array***(const *arraydata_t* *ad)

Create an array from an *arraydata_t* structure.

inline *array_t* ***clone_array***(const *array_t* *const array, const *rowindex_t* nrows, const *colindex_t* ncols)

Clone an array.

int **compareLMC**(const *array_link* &lhs, const *array_link* &rhs)

Return -1 if the first array is smaller in LMC ordering than the second array, 0 if equal and 1 otherwise

array_link **exampleArray**(int idx = 0, int verbose = 0)

Return example array

Parameters

- **idx** – Index of example array to return
- **verbose** – If True, then print information about the array to stdout

std::vector<int> **Jcharacteristics_conference**(const *array_link* &array, int number_of_columns, int verbose = 0)

Calculate Jk-characteristics for a conference design

Parameters

- **array** – Conference design
- **number_of_columns** – Specifies the number of columns to use
- **verbose** – Verbosity level

Returns

A vector of calculated inner products between all combinations of k columns.

array_link **hstack**(const *array_link* &array1, const *array_link* &array2)

concatenate 2 arrays in vertical direction

concatenate 2 arrays in horizontal direction

array_link **hstack**(const *array_link* &array, const *conference_column* &column)

concatenate array and conference_column

array_link **hstacklastcol**(const *array_link* &A, const *array_link* &B)

concatenate the last column of array B to array A

conference_column **vstack**(const *conference_column* &column_top, const *conference_column* &column_bottom)

concatenate two columns

void **perform_column_permutation**(const *array_link* source, *array_link* &target, const std::vector<int> perm)

perform column permutation for an array

void **perform_row_permutation**(const *array_link* source, *array_link* &target, const std::vector<int> perm)

perform row permutation for an array

arraydata_t **arraylink2arraydata**(const *array_link* &array, int extracols = 0, int strength = 2)

create *arraydata_t* structure from array

Parameters

- **array** – Array to use as input specifiction for array class
- **extracols** – Number of extra columns to add to the number of columns of the array
- **strength** – Strength to set in the array class. If -1, then use the strength of the array

arraylist_t **addConstant**(const *arraylist_t* &lst, int value)

add a constant value to all arrays in a list

std::vector<int> **getJcounts**(*arraylist_t* *arraylist, int N, int k, int verbose = 1)

Return number of arrays with j_{2n+1}=0 for number_of_arrays<m

void **create_root**(*array_t* *array, const *arraydata_t* *arrayclass)

set first columns of an array to root form

void **create_root**(const *arraydata_t* *arrayclass, *arraylist_t* &solutions)

Creates the root of an orthogonal array. The root is appended to the list of arrays.

int **array_diff**(*carray_p* A, *carray_p* B, const *rowindex_t* r, const *colindex_t* c, *rowindex_t* &rpos, *colindex_t* &cpos)

Compare 2 arrays and return position of first difference.

inline void **fastJupdate**(const *array_t* *array, *rowindex_t* N, const int J, const *colindex_t* *column_indices, *array_t* *tmp)

helper function to calculate J-values

int **jvalue**(const *array_link* &array, const int J, const int *column_indices)

Calculate J-value for a 2-level array

int **jvaluefast**(const *array_t* *array, *rowindex_t* N, const int J, const *colindex_t* *column_indices)

Calculate J-value for a column combination of a 2-level array

We assume the array has values 0 and 1. No boundary checks are performed.

std::vector<*jstruct_t*> **analyseArrays**(const *arraylist_t* &arraylist, const int verbose, const int jj = 4)

Analyse a list of arrays.

void **showArrayList**(const *arraylist_t* &lst)

print a list of arrays to stdout

long **nArrays**(const char *fname)

return number of arrays in an array file

void **arrayfileinfo**(const char *filename, int &number_of_arrays, int &number_of_rows, int &number_of_columns)

return information about file with arrays

Parameters

- **filename** – Filename of array file
- **number_of_arrays** – Variable is set with number of arrays
- **number_of_rows** – Variable is set with number of rows
- **number_of_columns** – Variable is set with number of columns

arraylist_t **readarrayfile**(const char *fname, int verbose = 1, int *setcols = 0)

Read all arrays in a file

Parameters

- **fname** – Filename to read from
- **verbose** – Verbosity level
- **setcols** – Pointer to return number of columns from array file

Returns

List of arrays

int **readarrayfile**(const char *filename, *arraylist_t* *arraylist, int verbose = 1, int *setcols = 0, int *setrows = 0, int *setbits = 0)

Read all arrays in a file and append then to an array list

Parameters

- **filename** – Filename to read from
- **arraylist** – Pointer to list of arrays
- **verbose** – Verbosity level
- **setcols** – Reference that is set with the number of columns from the file
- **setrows** – Reference that is set with the number of rows from the file
- **setbits** – Reference that is set with the number of bits from the file

Returns

int **writearrayfile**(const char *filename, const *arraylist_t* &arraylist, *arrayfile*::*arrayfilemode_t* mode = *arrayfile*::ATEXT, int nrows = *NRAUTO*, int ncols = *NRAUTO*)

Write a list of arrays to file on disk

Parameters

- **filename** – Filename to use
- **arraylist** – List of arrays to write
- **mode** – Mode for the file with designs
- **nrows** – If the list of arrays is empty, use this number of rows for the design file
- **ncols** – If the list of arrays is empty, use this number of rows for the design file

Returns

Value zero if succesfull

int **writearrayfile**(const char *filename, const *array_link* &array, *arrayfile*::*arrayfilemode_t* mode = *arrayfile*::ATEXT)

Write a single array to file.

int **append_arrayfile**(const char *filename, const *array_link* array)

Append a single array to an array file. creates a new file if no file exists.

void **selectArrays**(const std::string filename, std::vector<int> &idx, *arraylist_t* &fl, int verbose = 0)

Make a selection of arrays from binary array file, append to list.

array_link **selectArrays**(std::string filename, int index)

Select a single array from a file.

arraylist_t **selectArrays**(const *arraylist_t* &input_list, std::vector<int> &idx)

Make a selection of arrays.

arraylist_t **selectArrays**(const *arraylist_t* &input_list, std::vector<long> &idx)

Make a selection of arrays.

void **selectArrays**(const *arraylist_t* &input_list, std::vector<int> &idx, *arraylist_t* &output_list)

Make a selection of arrays, append to list.

void **selectArrays**(const *arraylist_t* &input_list, std::vector<long> &idx, *arraylist_t* &output_list)

Make a selection of arrays, append to list.

template<class **Container**, class **IntType**>
void **keepElements**(*Container* &al, std::vector<*IntType*> &idx)

From a container keep all elements with specified indices.

template<class **Container**, class **IntType**>
void **removeElements**(*Container* &al, std::vector<*IntType*> &idx)

From a container remove all elements with specified indices.

template<class **MType**>
void **selectArraysMask**(const *arraylist_t* &al, std::vector<*MType*> &mask, *arraylist_t* &rl)

Make a selection of arrays from a list, append to list.

template<class **IndexType**>
void **appendArrays**(const *arraylist_t* &al, const typename std::vector<*IndexType*> &idx, *arraylist_t* &lst)

Append selection of arrays to existing list.

void **appendArrays**(const *arraylist_t* &arrays_to_append, *arraylist_t* &dst)

Append set of arrays to existing list.

template<class **atype**>
void **write_array_format**(const *atype* *array, const int nrows, const int ncols, int width = 3)

Write a formatted array

template<class **atype**>
void **write_array_format**(FILE *fid, const *atype* *array, const int nrows, const int ncols)

Write an array to a file pointer.

template<class **atype**>
void **write_array_latex**(std::ostream &ss, const *atype* *array, const int nrows, const int ncols)

write an array in latex style

void **convert_array_file**(std::string input_filename, std::string output_filename, *arrayfile*::*arrayfilemode_t* output_format, int verbose = 0)

Convert a file with arrays to a different format

bool **readbinheader**(FILE *fid, int &nr, int &nc)

Read header for binary data file. Return true if valid header file

The header consists of 4 integers: 2 magic numbers, then the number of rows and columns

void **writebinheader**(FILE *fid, int number_rows, int number_columns)

Write header for binary data file.

template<class **Type**>

void **vector2doublebinfile**(const std::string fname, std::vector<*Type*> vals, int writeheader = 1)

Write a vector of numeric elements to binary file as double values.

void **vectorvector2binfile**(const std::string fname, const std::vector<std::vector<double>> vals, int writeheader, int na)

Write a vector of vector elements to binary file.

MatrixFloat **array2eigenX1**(const *array_link* &array, int intercept = 1)

Convert 2-level array to main effects in Eigen format

Parameters

- **array** – Array to convert
- **intercept** – If True, then include the intercept

Returns

The main effects model

MatrixFloat **array2eigenX2**(const *array_link* &array)

Convert 2-level array to second order interaction matrix in Eigen format

The intercept and main effects are not included.

Parameters

array – Array to convert

Returns

The second order interaction model

MatrixFloat **array2eigenModelMatrix**(const *array_link* &array)

Convert 2-level array to second order interaction model matrix (intercept, main effects, interaction effects)

Parameters

array – Design of which to calculate the model matrix

Returns

Eigen matrix with the model matrix

std::pair<*MatrixFloat*, *MatrixFloat*> **array2eigenModelMatrixMixed**(const *array_link* &array, int verbose = 1)

Create first and second order model matrix for mixed-level orthogonal array

For 2-level arrays a direct calculation is used. For mixel-level arrays Helmert contrasts are used.

Parameters

- **array** – Input array
- **verbose** – Verbosity level

Returns

Pair with main effects and two-factor interaction model

std::vector<int> **numberModelParams**(const *array_link* &array, int order = -1)

Calculate number of parameters in the model matrix

A list of integers is returned, with the number of columns in:

- The intercept (always 1)
- The main effects

- The interaction effects (second order interaction terms without quadratics)
- The quadratic effects

Parameters

- **array** – Orthogonal array or conference design
- **order** – Not used any more

Returns
List of sizes

int **arrayInFile**(const *array_link* &array, const char *array_file, int verbose = 1)

return index of specified array in a file. returns -1 if array is not found

Parameters

- **array** – Array to find
- **array_file** – Location if file with arrays
- **verbose** – Verbosity level

Returns
Position of array in list

int **arrayInList**(const *array_link* &array, const *arraylist_t* &arrays, int verbose = 1)

return index of specified array in a list. returns -1 if array is not found

Parameters

- **array** – Array to find
- **arrays** – List of arrays
- **verbose** – Verbosity level

Returns
Position of array in list

Variables

const int **NRAUTO** = 0

struct **arraydata_t**

#include <arraytools.h> Specifies a class of arrays.

The specification includes the number of rows, number of columns, factor levels and strength.

Public Functions

arraydata_t()

Specifies a class of orthogonal arrays

The specification includes the number of rows, number of columns, factor levels and strength.

An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, …, s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

arraydata_t(*array_t* s, *rowindex_t* N, *colindex_t* strength, *colindex_t* ncols)

Specifies a class of orthogonal arrays

The specification includes the number of rows, number of columns, factor levels and strength.

An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, …, s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

Parameters

- **s** – Factor levels
- **N** – Number of rows
- **strength** – Strength for class
- **ncols** – Number of columns for the class

arraydata_t(const std::vector<int> s, *rowindex_t* N, *colindex_t* strength, *colindex_t* ncols)

Specifies a class of orthogonal arrays

The specification includes the number of rows, number of columns, factor levels and strength.

An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, …, s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

Parameters

- **s** – Factor levels
- **N** – Number of rows
- **strength** – Strength for class
- **ncols** – Number of columns for the class

arraydata_t(const *array_t* *s_, *rowindex_t* N, *colindex_t* strength, *colindex_t* ncols)

Specifies a class of orthogonal arrays

The specification includes the number of rows, number of columns, factor levels and strength.

An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, …, s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

arraydata_t(const *arraydata_t* &adp)

Specifies a class of orthogonal arrays

The specification includes the number of rows, number of columns, factor levels and strength.

An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, …, s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

arraydata_t(const *arraydata_t* *adp, *colindex_t* newncols)

Specifies a class of orthogonal arrays

The specification includes the number of rows, number of columns, factor levels and strength.

An orthogonal array of strength t, N runs, k factors (columns) and factor levels s[i] is an N times k array with symbols 0, 1, …, s[i]-1 in column i such that for every t columns every t-tuple of elements occurs equally often.

~arraydata_t()

arraydata_t &**operator=**(const *arraydata_t* &ad2)

int **operator==**(const *arraydata_t* &ad2)

bool **ismixed**() const

return true if the class represents mixed-level arrays

bool **is2level**() const

return true if the class represents a 2-level array

array_link **randomarray**(int strength = 0, int ncols = -1) const

return random array from the class. this operation is only valid for strength 0 or 1

void **writeConfigFile**(const char *filename) const

Write file with specification of orthognal array class.

Parameters

filename – Filename to write to

std::string **idstr**() const

return string with class representation

std::string **idstrseriesfull**() const

return string with class representation. series of level is expended

std::string **fullidstr**(int series = 0) const

return string with class representation

std::string **latexstr**(int cmd = 0, int series = 0) const

return latex string describing the class

inline *arraydata_t* **reduceColumns**(int k)

std::string **showstr**() const

Return string used for displaying the class.

void **show**(int verbose = 1) const

void **complete_arraydata**()

Calculate derived data such as the index and column groups from a design.

void **lmc_overflow_check**() const

check whether the LMC calculation will overflow

void **complete_arraydata_fixlast**()

void **complete_arraydata_splitn**(int ns)

void **set_colgroups**(const std::vector<int> splits)

void **set_colgroups**(const symmetry_group &sg)

set column group equal to that of a symmetry group

std::vector<int> **get_column_groups_sizes**() const

return sizes of the column groups

void **show_colgroups**() const

show column groups in the array class

void **calculate_oa_index**(*colindex_t* strength)

calculate the index of the orthogonal arrays in this class

array_link **create_root**(int n_columns = -1, int fill_value = 0) const

return the root array for the class

int **getfactorlevel**(int idx) const

return the factor level for the specified column return -1 if the column index is invalid

inline std::vector<int> **getS**() const

return factor levels

std::vector<int> **factor_levels**() const

return factor levels

std::vector<int> **factor_levels_column_groups**() const

return factor levels for the column groups

void **reset_strength**(*colindex_t* strength)

Reset strength of arraydata.

Parameters

strength – The strength to reset the structure to

colindex_t **get_col_group**(const *colindex_t* col) const

Return index of the column group for a column.

bool **is_factor_levels_sorted**() const

Return True if the factor levels are sorted from large to small.

Public Members

rowindex_t **N**

number of runs

colindex_t **ncols**

total number of columns (factors) in the design

colindex_t **strength**

strength of the design

array_t ***s***

pointer to factor levels of the array

ordering_t **order**

Ordering used for arrays.

colindex_t **ncolgroups**

number of groups of columns with the same number of levels

colindex_t ***colgroupindex**

specifies for each column the index of the column group

colindex_t ***colgroupsize**

specifies for each column the size of the column group

int **oaindex**

index of the array

struct **array_link**

#include <arraytools.h>

Public Functions

array_link()

A class representing an integer valued array

array_link(*rowindex_t* nrows, *colindex_t* ncols, int index)

A class representing an integer valued array

The array is intialized with zeros.

Parameters

- **nrows** – Number of rows
- **ncols** – Number of columns
- **index** – Number to keep track of lists of designs

array_link(*rowindex_t* nrows, *colindex_t* ncols, int index, *carray_t* *data)

A class representing an integer valued array

Initialize with data from a pointer.

array_link(const *array_link*&)

A class representing an integer valued array

Initialize with data from another *array_link* object.

array_link(Eigen::MatrixXd &eigen_matrix)

A class representing an integer valued array

Initialize with data from an Eigen matrix.

array_link(const *array_link* &array, const std::vector<int> &column_permutation)

A class representing an integer valued array

The array is initialized by permuting the columns of another array

Parameters

- **array** – Source to copy from
- **column_permutation** – The permuntation to apply

array_link(const *array_t* *array, *rowindex_t* nrows, *colindex_t* ncols, int index = 0)

A class representing an integer valued array

array_link(const *array_t* *array, *rowindex_t* nrows, *colindex_t* ncolsorig, *colindex_t* ncols, int index)

A class representing an integer valued array

array_link(const std::vector<int> &values, *rowindex_t* nrows, *colindex_t* ncols, int index = 0)

A class representing an integer valued array

The array is initialized by copying the values from a vector.

~array_link()

array_link **clone**() const

void **showarray**() const

print array to stdout

std::string **showarrayString**() const

print array to string

void **showarraycompact**() const

print array to stdout in compact format (no whitespace between elemenents)

void **showproperties**() const

print array properties to stdout

bool **is2level**() const

return true if the array is a 2-level array (e.g. only contains values 0 and 1)

bool **is_mixed_level**() const

return true is the array is a mixel-level array

bool **is_orthogonal_array**() const

return true is the array is array with values in 0, 1, . . . , for each column

bool **is_conference**() const

return true if the array is a +1, 0, -1 valued array

bool **is_conference**(int number_of_zeros) const

return true if the array is a +1, 0, -1 valued array, with specified number of zeros in each column

bool **isSymmetric**() const

return true if the array is symmetric

void **makeSymmetric**()

make the array symmetric by copying the upper-right to the lower-left *array_link* **deleteColumn**(int index) const

return array with selected column removed

array_link **selectFirstRows**(int nrows) const

return array with first number_of_arrays rows

array_link **selectFirstColumns**(int ncolumns) const

return array with first number_of_arrays columns selected

array_link **selectLastColumns**(int ncolumns) const

return array with last number_of_arrays columns selected

array_link **selectColumns**(const std::vector<int> c) const

select columns from an array

array_link **selectColumns**(int c) const

select single column from an array

inline void **setColumn**(int c, const std::vector<int> v)

set a column of the array to the given vector

inline void **setColumn**(int c, const std::vector<signed char> v)

set a column of the array to the given vector

array_link **transposed**() const

return transposed array

double **Defficiency**() const

calculate D-efficiency

double **DsEfficiency**(int verbose = 0) const

calculate main effect robustness (or Ds-optimality)

std::vector<double> **Defficiencies**(int verbose = 0, int addDs0 = 0) const

calculate D-efficiency, calculate main effect robustness (or Ds-optimality) and D1-efficiency for an orthogonal array

double **VIFefficiency**() const

double **Aefficiency**() const

calculate A-efficiency

double **Eefficiency**() const

calculate E-efficiency

std::vector<int> **Fvalues**(int number_of_columns) const

Calculate F-values of a 2-level matrix.

This assumes the strength is at least 3. Otherwise use the *jstruct_t* object

std::vector<int> **FvaluesConference**(int number_of_columns) const

Calculate F-values of a conference design

Parameters

number_of_columns – Number of columns to use

Returns

The Fk vector with k the number of columns specified std::vector<int> **Jcharacteristics**(int jj = 4) const

Calculate the Jk-characteristics of the matrix (the values are signed)

Parameters

`jj` – Number of columns to use

Returns

Vector with calculated Jk values

std::vector<double> **PECsequence**(int verbose = 0) const

Calculate the projective estimation capacity sequence.

std::vector<double> **PICsequence**(int verbose = 0) const

Calculate the projective information capacity sequence.

int **rank**() const

calculate rank of array

std::vector<double> **GWLP**(int truncate = 1, int verbose = 0) const

Calculate generalized wordlength pattern

See also:

GWLP

int **strength**() const

calculate strength of an array

bool **foldover**() const

return true if the array is a foldover array

array_t **min**() const

array_t **max**() const

double **CL2discrepancy**() const

Calculate centered L2 discrepancy

The method is from "A connection between uniformity and aberration in regular fractions of two-level factorials", Fang and Mukerjee, 2000

array_link **randomperm**() const

apply a random permutation of rows, columns and levels of an orthogonal array

array_link **randomcolperm**() const

apply a random permutation of columns of an orthogonal array

array_link **randomrowperm**() const

apply a random permutation of rows of an orthogonal array

MatrixFloat **getModelMatrix**(int order, int intercept = 1, int verbose = 0) const

Caculate model matrix of an orthogonal array

This function uses *array2eigenModelMatrixMixed* for the calculation.

Parameters

- **order** – For 0 return only the intercept; for 1 return intercept and main effects; for 2 return intercept, main effects and interaction effects.
- **intercept** – If 1, then include the intercept in the output.
- **verbose** – Verbosity level

Returns
Calculated model matrix

array_link &**operator=**(const *array_link* &rhs)

array_link &**deepcopy**(const *array_link* &rhs)

array_link &**shallowcopy**(const *array_link* &rhs)

int **operator==**(const *array_link* &rhs) const

Return True if both arrays are equal.

Parameters
rhs – Array to compare to

Returns
1 if arrays are equal. 0 otherwise. Returns 0 if arrays have different sizes

int **operator!=**(const *array_link* &rhs) const

int **operator<**(const *array_link* &rhs) const

int **operator>**(const *array_link* &rhs) const

int **equalsize**(const *array_link* &rhs) const

return true of two array have the same dimensions

array_link **operator+**(const *array_link*&) const

elementwise addition

array_link **operator+**(*array_t* value) const

elementwise addition

array_link **operator-**(const *array_link*&) const

array_link **operator-**(*array_t* value) const

array_link **operator***(const *array_link* &rhs) const

elementwise multiplication

array_link **operator***(*array_t* value) const

array_link **operator*=**(*array_t* value)

array_link **operator+=**(*array_t* value)

array_link **operator-=**(*array_t* value)

inline const *array_t* &**atfast**(const *rowindex_t* r, const *colindex_t* c) const

get element from array, no error checking, inline version

inline *array_t* &**atfast**(const *rowindex_t* r, const *colindex_t* c)

get element from array, no error checking, inline version

array_t **_at**(const *rowindex_t*, const *colindex_t*) const
get element at specified position, no bounds checking

array_t **_at**(const int index) const
get element at specified position, no bounds checking

array_t **at**(const *rowindex_t*, const *colindex_t*) const
get element at specified position

array_t **at**(const int index) const
get element at specified position

array_t &**at**(const *rowindex_t*, const *colindex_t*)
get element at specified position

void **setconstant**(*array_t* value)
set all elements in the array to a value

void **setvalue**(int row, int col, int value)
set value of an array

void **setvalue**(int row, int col, double value)
set value of an array

void **_setvalue**(int row, int col, int value)
set value of an array, no bounds checking!

void **negateRow**(*rowindex_t* row)
multiply a row by -1

void **show**() const
print information about array

std::string **showstr**() const
return string describing the array

std::string **md5**() const
return md5 sum of array representation (as represented with 32bit int datatype in memory)

bool **columnEqual**(int column_index, const *array_link* &rhs, int column_index_rhs) const
return true if two columns are equal

int **firstColumnDifference**(const *array_link* &A) const
return index of first different column

bool **firstDiff**(const *array_link* &A, int &r, int &c, int verbose = 1) const
Calculate row and column index of first difference between two arrays

The difference is according to the column-major ordering.

void **create_root**(const *arraydata_t* &arrayclass, int fill_value = 0)
create root in arraylink

double **nonzero_fraction**() const
return fraction of nonzero elements in array

void **clear**()
fill array with zeros

inline void **getarraydata**(int *pymat1, int n)

template<class **numtype**>
inline void **setarraydata**(const *numtype* *tmp, int n)

internal function

template<class **numtype**>
inline void **setarraydata_transposed**(const *numtype* *input_data, int n)

internal function

inline void **setarraydata**(std::vector<int> tmp, int n)

special method for SWIG interface

template<class **numtype**>
inline void **setarraydata**(std::vector<*numtype*> tmp, int n)

internal function

void **setcolumn**(int target_column, const *array_link* &source_array, int source_column = 0) const

set column to values

void **init**(*rowindex_t* r, *colindex_t* c)

symmetry_group **row_symmetry_group**() const

return the row_symmetry group of an array

array_link **reduceLMC**() const

return the LMC form of the array

array_link **reduceDOP**() const

return the delete-one-factor-projection form of the array

MatrixFloat **getEigenMatrix**() const

return the array as an Eigen matrix

int **columnGreater**(int c1, const *array_link* &rhs, int rhs_column) const

return true of specified column is smaller than column in another array

void **debug**() const

Public Members

rowindex_t **n_rows**

Number of rows in array.

colindex_t **n_columns**

Number of columns in array.

int **index**

Index number.

array_t ***array**

Pointer to array data.

Public Static Attributes

static const int **INDEX_NONE** = 0

static const int **INDEX_ERROR** = -1

static const int **INDEX_DEFAULT** = 0

Private Functions

bool **equal_size**(const *array_link* &array) const
    return true if both arrays have the same size

bool **_valid_index**(const *rowindex_t* r, const *colindex_t* c) const

bool **_valid_index**(int index) const

Friends

friend std::ostream &**operator<<**(std::ostream&, const *array_link* &A)
    print an array to output stream

class **jstructbase_t**
    #include <arraytools.h> struct to hold data of an array, e.g. J-characteristic. Abstract base class

    Subclassed by *jstructconference_t*

Public Functions

int **maxJ**() const
    calculate maximum J value

inline std::vector<int> **Jvalues**() const
    calculate possible values in F vector

std::vector<int> **calculateF**() const
    Calculate histogram of J values

```
\return Histogram of J values

The histogram bins are given by the values of @ref Jvalues.
```

virtual void **calc**(const *array_link* &array) = 0
    Calculate the J-values for a given array.

void **show**()
    Show contents of structure.

void **showdata**(int verbose = 1)

std::string **showstr**()

inline int **allzero**()

return 1 if all vals are zero

Public Members

std::vector<int> **values**

calculated J-characteristics

std::vector<int> **jvalues**

std::map<int, int> **jvalue2index**

map from Jk-value to index in the jvalues variable

int **jj**

number of columns

struct **symmdata**

#include <arraytools.h> structure containing data related to symmetries of arrays

Public Functions

symmdata(const *array_link* &al, int minlen = 1)

inline void **show**(int verbose = 1) const

inline std::vector<int> **checkIdx**(int col = -1) const

list with indices set to check for symmetry reductions

Public Members

array_link **rowvalue**

array_link **orig**

array_link **ft**

class **jstruct_t**

#include <arraytools.h> struct to hold data of an array, e.g. J-characteristic, rank

See papers: Minimum G2-aberration properties of two-level foldover designs, Butler, 2004 Design Selection and Classification for Hadamard Matrices Using Generalized Minimum Aberration Criteria, Deng and Tang

Public Functions

jstruct_t()

Create an object to calculate J-characteristics.

jstruct_t(const *array_link* &al, int jj = 4)

Create an object to calculate J-characteristics.

jstruct_t(const int N, const int K, const int jj = 4)

Create an object to calculate J-characteristics.

jstruct_t(const *jstruct_t* &js)

Create an object to calculate J-characteristics.

~jstruct_t()

jstruct_t &**operator=**(const *jstruct_t* &rhs)

int **maxJ**() const

calculate maximum J value

int **number_J_values**(int strength) const

Calculate the number of possible J values that can occur for the given strength.

std::vector<int> **Fval**(int strength = 3) const

Calculate possible values in F vector

Parameters

strength – Strength to use

Returns

Vector with possible Jk values (ordered from high to low)

std::vector<int> **calculateF**(int strength = 3) const

calculate histogram of J values for a 2-level array

void **calculateAberration**()

Calculate aberration value

This is equal to the sum of the squares of all Jk values, divided by the number of rows squared.

The calculated abberation is stored in the variable abberation.

void **show**() const

Show contents of structure.

void **showdata**()

std::string **showstr**()

int **allzero**() const

return 1 if all J values are zero, otherwise return 0

void **calcj5**(const *array_link* &al)

calculate J-characteristics of a 2-level array, special function for jj=5

Public Members

int **N**
number of rows in array

int **k**
number of columns in array

int **jj**
J-characteristic that is calculated.

int **nc**
number of column combinations possible

std::vector<int> **values**
contains calculated J-values

double **abberration**
calculated abberation

Private Functions

void **init**(int N, int k, int jj)
init data structures

void **calc**(const *array_link* &al)
calculate J-characteristics of a 2-level array

void **calcj4**(const *array_link* &al)
calculate J-characteristics of a 2-level array, special function for jj=4

class **jstructconference_t** : public *jstructbase_t*
#include <arraytools.h> Calculate J-characteristics of conference designs

Public Functions

inline **jstructconference_t**(int N, int jj = 4)
Create structure to calculate J-characteristics of conference designs

Parameters

- **N** – Number of rows
- **jj** – Number of columns to use for the Jk-characteristics

inline **jstructconference_t**(const *array_link* &array, int jj = 4)
Calculate J-characteristics of a conference design

Parameters

- **array** – Array to calculate the J-characteristics for

- **jj** – Number of columns to use for the Jk-characteristics

Private Functions

void **calcJvalues**(int N, int jj)

virtual void **calc**(const *array_link* &al)
    Calculate the J-values for a given array.

class **array_transformation_t**

#include <arraytools.h> Contains a transformation of an array.

Contains an array transformation. The transformation consists of column, row and level permutations. The level and column permutations are not commutative (since the level permutations are tied to a particular column). We apply the column permutations first.

Public Functions

array_transformation_t(const *arraydata_t* *arrayclass)

array_transformation_t(const *arraydata_t* &arrayclass)

array_transformation_t()

array_transformation_t(const *array_transformation_t* &transformation)
    copy constructor

array_transformation_t &**operator=**(const *array_transformation_t* &at)
    assignment operator

~array_transformation_t()

void **show**() const
    show the array transformation

bool **isIdentity**() const
    return true if the transformation is equal to the identity

array_transformation_t **inverse**() const
    return the inverse transformation

void **reset**()
    return the transformation to the identity transformation

void **randomize**()
    initialize to a random transformation

void **randomizecolperm**()
    initialize with a random column permutation

void **randomizerowperm**()
    initialize with a random row permutation

array_link **apply**(const *array_link* &array) const
    apply transformation to an *array_link* object

int **operator==**(const *array_transformation_t* &t2) const
: Comparison operator.

array_transformation_t **operator***(const *array_transformation_t* b) const
: composition operator. the transformations are applied from the left

void **apply**(*array_t* *sourcetarget) const
: apply transformation to an array (inplace)

void **apply**(const *array_t* *source, *array_t* *target) const
: apply transformation to an array

void **print_transformed**(*carray_t* *source) const
: apply transformation and show resulting array

void **show**(std::ostream &out) const

std::vector<int> **rowperm**() const
: return the row permutation of the transformation

std::vector<int> **colperm**() const
: return the column permutation of the transformation

std::vector<int> **lvlperm**(int c) const
: return the level permutations of the transformation

void **setrowperm**(std::vector<int> row_permutation)
: set the row permutation of the transformation

void **setcolperm**(std::vector<int> column_permutation)
: set the column permutation of the transformation

void **setlevelperm**(int column_index, std::vector<int> lvl_permutation)
: set the level permutation of the transformation

Public Members

rowperm_t **rperm**
: row permutation

colperm_t **cperm**
: column permutation

levelperm_t ***lperms**
: level permutations

const *arraydata_t* ***ad**
: type of array

Private Functions

void **allocate_data_structures**()
initialize permutation structures

void **free_data_structures**()
free permutation structures and *arraydata_t* structure

class **conference_transformation_t**
#include <arraytools.h> Contains a transformation of a conference matrix.

Contains an array transformation. The transformation consists of column permutations, row permutations and sign switches for both the rows and columns.

The sign switches and the permutations are not commutative. We apply the permutations first and then the sign flips.

Public Functions

conference_transformation_t()

conference_transformation_t(int nrows, int ncols)
default constructor

conference_transformation_t(const *array_link* &al)

conference_transformation_t(const *conference_transformation_t* &T)

void **show**(int verbose = 1) const
show the array transformation

bool **isIdentity**() const
return true if the transformation is equal to the identity

conference_transformation_t **inverse**() const
return the inverse transformation

void **reset**()
return the transformation to the identity transformation

void **randomize**()
initialize to a random transformation

void **randomizecolperm**()
initialize with a random column permutation

void **randomizerowperm**()
initialize with a random row permutation

void **randomizecolflips**()
initialize with random col switches

void **randomizerowflips**()
initialize with random row switches

array_link **apply**(const *array_link* &al) const
apply transformation to an *array_link* object

int **operator==**(const *conference_transformation_t* &rhs) const

conference_transformation_t **operator***(const *conference_transformation_t* &rhs) const

composition operator. the transformations are applied from the left

E.g. (T1*T2)(x) = T1(T2(x))

inline void **setrowperm**(std::vector<int> rp)

inline void **setcolperm**(std::vector<int> cp)

Public Members

std::vector<int> **rperm**

row permutation of the transformation

std::vector<int> **cperm**

column permutation of the transformation

std::vector<int> **cswitch**

sign flips for the columns

std::vector<int> **rswitch**

sign flips for the rows

int **nrows**

number of rows

int **ncols**

number of columns

Private Functions

void **init**(int nr, int nc)

struct **arraywriter_t**

#include <arraytools.h> structure to write arrays to disk, thread safe

Public Functions

inline **arraywriter_t**()

inline **~arraywriter_t**()

inline void **flush**()

flush all output files

inline void **writeArray**(const *array_link* &A)

write a single array to disk

inline void **writeArray**(const *arraylist_t* &lst)

write a list of arrays to disk

inline void **initArrayFiles**(const *arraydata_t* &ad, int kstart, const std::string prefix, arrayfilemode_t mode = ABINARY_DIFF)

initialize the result files

inline int **nArraysWritten**() const

return the total number arrays written to disk

inline void **closeafiles**()

Public Members

std::vector<arrayfile_t*> **afiles**

Pointers to different data files.

Since depth_extend is a depth first approach we need to store arrays with a different number of columns

bool **writearrays**

only write arrays if this variable is true

int **nwritten**

number of arrays written to disk

int **verbose**

verbosity level

namespace **arrayfile**

Enums

enum **arrayfilemode_t**

file format mode

Values:

enumerator **ATEXT**

text based format

enumerator **ALATEX**

write arrays to a text file in a format that can be parsed by LaTeX

enumerator **ABINARY**

binary format

enumerator **ABINARY_DIFF**

binary format storing differences of arrays enumerator **ABINARY_DIFFZERO**

binary format storing differences of arrays and zero offsets

enumerator **AERROR**

enumerator **A_AUTOMATIC**

automatically determine the format

enumerator **A_AUTOMATIC_BINARY**

automatically determine the format (but binary)

enum **afilerw_t**

file mode for array file

Values:

enumerator **READ**

enumerator **WRITE**

enumerator **READWRITE**

struct **arrayfile_t**

#include <arraytools.h> Structure for reading or writing a file with arrays.

The format of the file is determined by the `arrayfilemode_t` The format described in detail in the documentation of the OApackage https://oapackage.readthedocs.io/en/latest/.

Public Functions

arrayfile_t()

Structure for reading or writing a file with arrays

arrayfile_t(const std::string filename, int verbose = 1)

Structure for reading or writing a file with arrays

Parameters
- **filename** – File to open for reading
- **verbose** – Verbosity level

arrayfile_t(const std::string filename, int nrows, int ncols, int narrays = -1, *arrayfilemode_t* mode = ATEXT, int number_of_bits = 8)

Structure for reading or writing a file with arrays

Open new array file for writing

Parameters
- **filename** – File to open
- **nrows** – Number of rows
- **ncols** – Number of columns
- **narrays** – Specify a number of arrays, or -1 to add dynamically
- **mode** – File mode

- **number_of_bits** – Number of bits to use for storage. For 2-level arrays only 1 bit is needed

~arrayfile_t()

destructor function, closes all filehandles

void **createfile**(const std::string filename, int nrows, int ncols, int narrays = -1, *arrayfilemode_t* m = ATEXT, int number_of_bits = 8)

Open a new file for writing and (if opened) close the current file.

void **closefile**()

close the array file

int **isopen**() const

return true if file is open

int **seek**(int pos)

seek to specified array position

int **read_array**(*array_link* &a)

read array and return index

array_link **readnext**()

read next array from the file

arraylist_t **readarrays**(int nmax = *NARRAYS_MAX*, int verbose = 1)

read set of array from the file

void **flush**()

flush any open file pointer

bool **isbinary**() const

return true if the file has binary format

int **append_arrays**(const *arraylist_t* &arrays, int startidx = -1)

append list of arrays to the file

void **append_array**(const *array_link* &a, int specialindex = -1)

append a single array to the file

void **add_comment**(const std::string &comment)

Add a comment to an array file (only available in text mode)

int **swigcheck**() const

return True if code is wrapped by SWIG

std::string **showstr**() const

return string describing the object

inline size_t **pos**() const

return current position in file

inline bool **hasrandomaccess**() const

return true of the file format has random access mode

void **updatenumbers**()

int **read_array**(*array_t* *array, const int nrows, const int ncols)

read array and return index

void **finisharrayfile**()

void **setVerbose**(int v)

set verbosity level

int **getnbits**()

Public Members

std::string **filename**

location of file on disk

int **iscompressed**

True of the file is compressed with gzip.

int **nrows**

number of rows of the arrays

int **ncols**

number of columns of the arrays

int **nbits**

number of bits used when storing an array

arrayfilemode_t **mode**

file mode, can be ATEXT or ABINARY, ABINARY_DIFF, ABINARY_DIFFZERO

afilerw_t **rwmode**

file opened for reading or writing

int **narrays**

number of arrays in the file

int **narraycounter**

FILE ***nfid**

int **gzfid**

pointer to compressed file

int **verbose**

verbosity level

Public Static Functions

static *arrayfile*::*arrayfilemode_t* **parseModeString**(const std::string format)
parse string to determine the file mode

static inline int **arrayNbits**(const *arraydata_t* &ad)
return number of bits necessary to store an array

static inline int **arrayNbits**(const *array_link* &A)
return number of bits necessary to store an array

Public Static Attributes

static const int **NARRAYS_MAX** = 2 * 1000 * 1000 * 1000
maximum number of arrays in structure

Protected Functions

void **writeheader**()

void **read_array_binary**(*array_t* *array, const int nrows, const int ncols)
Read a binary array from a file.

Private Functions

int **headersize**() const
return header size for binary format array

int **barraysize**() const
return size of bit array

size_t **afwrite**(void *ptr, size_t t, size_t n)
wrapper function for fwrite or gzwrite

size_t **afread**(void *ptr, size_t sz, size_t cnt)
wrapper function for fread or gzread

int **read_array_binary_zero**(*array_link* &a)

void **write_array_binary**(*carray_t* *array, const int nrows, const int ncols)

void **write_array_binary**(const *array_link* &A)

void **write_array_binary_diff**(const *array_link* &A)
Write an array in binary diff mode to a file

We only write the section of columns of the array that differs from the previous array.

void **write_array_binary_diffzero**(const *array_link* &A)
Write an array in binary diffzero mode

Private Members

array_link **diffarray**

7.4 Interface for conference designs

Contains functionality to generate and analyse conference designs. For more information see:

- https://en.wikipedia.org/wiki/Conference_matrix
- "A Classification Criterion for Definitive Screening Designs", Schoen et al., The Annals of Statistics, 2019

Author: Pieter Eendebak pieter.eendebak@gmail.com Copyright: See LICENSE.txt file that comes with this distribution

Typedefs

typedef *CandidateGeneratorConference* **CandidateGenerator**

Functions

void **print_column**(const *conference_column* &column, const char *msg = 0)

print a candidate extension

void **showCandidates**(const std::vector<*conference_column*> &column_candidates)

Show a list of candidate extensions

Parameters

column_candidates – List of candidates to show

array_link **conference2DSD**(const *array_link* &conference_design, bool add_zeros = 1)

Convert conference design to definitive screening design

The DSD is created by appending the negated design to the conference design and then appending a row of zeros.

Parameters

- **conference_design** – Array with the conference design
- **add_zeros** – If True, then append a row of zeros

Returns

The DSD generated from the conference design

array_link **reduceConference**(const *array_link*&, int verbose = 0)

Reduce conference matrix to normal form using Nauty

See also:

reduceConferenceTransformation

conference_transformation_t **reduceConferenceTransformation**(const *array_link* &conference_design, int verbose)

Reduce conference matrix to normal form using Nauty

The design is converted to a graph representation. The graph is then reduced using Nauty to normal form and the resulting graph translated back to a conference design.

Parameters

- **conference_design** – Design to be reduced to normal form
- **verbose** – Verbosity level

Returns

A transformation that converts the input design to normal form

conference_extend_t **extend_conference_matrix**(const *array_link* &design, const *conference_t* &conference_type, int extcol, int verbose = 1, int maxzpos = -1)

Extend a single conference design with candidate columns

Parameters

- **design** – Conference design
- **conference_type** – Type specification for the conference designs
- **extcol** – Index of column to generate extensions for
- **verbose** – Verbosity level
- **maxzpos** – Maximum position of zero in specified design

Returns

Structure with information about the possible extensions

arraylist_t **extend_conference**(const *arraylist_t* &lst, const *conference_t* conference_type, int verbose, int select_isomorphism_classes = 0)

Extend a list of conference designs with a single column.

The list of conference designs is extended by adding to each design the candidate extentions generated by *CandidateGenerator*.

The extension algorithm tries to generate designs in LMC0 normal form and prunes any designs that are not in LMC0 form.

Parameters

- **lst** – List of conference designs
- **conference_type** – Type specification for the conference designs
- **verbose** – Verbosity level
- **select_isomorphism_classes** – If True then select only a single design for each isomorphism class specified by the conference type.

Returns

List of generated conference designs

arraylist_t **extend_conference_plain**(const *arraylist_t* &lst, const *conference_t* conference_type, int verbose, int select_isomorphism_classes = 0)

Extend a list of conference designs with a single column, plain version without caching

Research function.

arraylist_t **extend_conference_restricted**(const *arraylist_t* &lst, const *conference_t* conference_type, int verbose)

Extend a list of conference designs with a single column

Research function.

arraylist_t **extend_double_conference**(const *arraylist_t* &lst, const *conference_t* conference_type, int verbose)

Extend a list of double conference matrices with an additional column

The list of designs is extended by adding each design with the candidate extentions generated by *CandidateGeneratorDouble*.

Parameters

- **lst** – List of double conference designs
- **conference_type** – Type specification for the double conference designs
- **verbose** – Verbosity level

Returns

List of generated double conference designs

arraylist_t **selectConferenceIsomorpismClasses**(const *arraylist_t* &list, int verbose, *matrix_isomorphism_t* itype = CONFERENCE_ISOMORPHISM)

Select representatives for the isomorphism classes of a list of conference arrays

The method uses Nauty for reduction to normal form and selection of isomorphism classes.

Parameters

- **list** – List of designs
- **verbose** – Verbosity level
- **itype** – Specification of the type of isomorphism to use

Returns

Selected isomorphism classes

std::vector<int> **selectConferenceIsomorpismIndices**(const *arraylist_t* &lst, int verbose, *matrix_isomorphism_t* itype = CONFERENCE_ISOMORPHISM)

select representatives for the isomorphism classes of a list of conference arrays, return indices of classes

arraylist_t **selectLMC0doubleconference**(const *arraylist_t* &list, int verbose, const *conference_t* &ctype)

Select double conference designs in LMC0 form

Parameters

- **list** – List of double conference designs
- **verbose** – Verbosity level
- **ctype** – Specifiation of the class of designs

Returns

List with only the designs in the input list that are in LMC0 normal form.

arraylist_t **selectLMC0**(const *arraylist_t* &list, int verbose, const *conference_t* &ctype)

Select conference designs in LMC0 form

Parameters

- **list** – List of conference designs
- **verbose** – Verbosity level
- **ctype** – Specification of the class of designs

Returns

List with only the designs in the input list that are in LMC0 normal form.

std::vector<*conference_column*> **generateConferenceExtensions**(const *array_link* &array, const *conference_t* &conference_type, int zero_index, int verbose = 1, int filter_symmetry = 1, int filterj2 = 1)

Generate candidate extensions for a conference design

Parameters

- **array** – Design to be extended
- **conference_type** – Class of conference designs
- **zero_index** – Index of zero in candidate column
- **verbose** – Verbosity level
- **filter_symmetry** – If True, filter based on symmetry
- **filterj2** – If True, filter based on J2 values

Returns

List of generated extensions

std::vector<*conference_column*> **generateConferenceRestrictedExtensions**(const *array_link* &array, const *conference_t* &conference_type, int zero_index, int verbose = 1, int filter_symmetry = 1, int filterip = 1)

Generate candidate extensions for restricted isomorphism classes

std::vector<*conference_column*> **generateDoubleConferenceExtensions**(const *array_link* &array, const *conference_t* &conference_type, int verbose = 1, int filter_symmetry = 1, int filterip = 1, int filterJ3 = 0, int filter_symmetry_inline = 1)

generate extensions for double conference matrices in LMC0 form

std::vector<*conference_column*> **generateSingleConferenceExtensions**(const *array_link* &array, const *conference_t* &conference_type, int zero_index, int verbose, int filter_symmetry, int filterj2, int filterj3, int filter_symmetry_inline = 0)

generate extensions for conference matrices in LMC0 form

The the method assumes that the input array is in LMC0 format. In particular, the element at row 0 and column 0 of the design should be a zero.

Parameters

- **array** – Design to generate extensions for
- **conference_type** – Specification of the type of designs
- **zero_index** – Passed to checkZeroPosition to determine whether a zero in the extension column is in a valid position
- **verbose** – Verbosity level
- **filter_symmetry** – If True than reject extensions which are not minimal according to the row symmetry group of the specified design
- **filterj2** – If True than reject extensions which do not satisfy the J2 criterea
- **filterj3** – If True than reject extensions which do not satisfy the J3 criterea
- **filter_symmetry_inline** – Quick rejection of extensions which do not satisfy the symmetry criterion.

Returns
List of extensions of the design

int **maxz**(const *array_link* &al, int column_index = -1)

return max position of zero in array, returns -1 if no zero is found

The parameter k specifies the column to search in. For k=-1 all columns are searched.

bool **compareLMC0**(const *array_link* &array_first, const *array_link* &array_second)

Return true if the first array is smaller in LMC-0 ordering than the second array

arraylist_t **sortLMC0**(const *arraylist_t* &arrays)

sort list of conference designs according to LMC0 ordering

lmc_t **LMC0checkDC**(const *array_link* &al, int verbose = 0)

lmc_t **LMC0check**(const *array_link* &array, int verbose = 0)

bool **isConferenceFoldover**(const *array_link* &array, int verbose = 0)

return true if the design is a foldover array

std::vector<int> **double_conference_foldover_permutation**(const *array_link* &double_conference)

For a double conference design return a row permutation to a single conference design

If the design is not a foldover design then the first element of the returned permutation is -1.

Parameters
double_conference – A double conference design

Returns
Permutation such that the top block of the resulting design forms a single conference design

int **minz**(const *array_link* &al, int column_index)

return minimal position of zero in specified column of a design

class **conference_t**

#include <conference.h> Structure representing the type of conference designs.

Public Types

enum **conference_type**

Type of conference design.

Values:

enumerator **CONFERENCE_NORMAL**

normal conference design

enumerator **CONFERENCE_DIAGONAL**

conference design with zeros only on diagonal

enumerator **DCONFERENCE**

double conference design

Public Functions

conference_t()

Structure representing the type of conference designs

conference_t(int N, int k, int j1zero)

Structure representing the type of conference designs

Parameters

- **N** – Number of rows
- **k** – Number of columns
- **j1zero** – If True then require the J1-characteristics to be zero

conference_t(const *conference_t* &rhs)

Structure representing the type of conference designs

std::string **idstr**() const

array_link **create_root**() const

create the unique representative of the 2 column conference design in LMC0 form

array_link **create_root_three_columns**() const

create the unique representative of the 3 column conference design in LMC0 form

arraylist_t **createDoubleConferenceRootArrays**() const

create the root arrays with 1 column for the double conference matrices

arraylist_t **createRootArrays**() const

return the list of root arrays for the class of conference designs

inline std::string **__repr__**() const

return string representation of the object

Public Members

rowindex_t **N**

number of runs

colindex_t **ncols**

total number of columns (factors) in the design

conference_type **ctype**

defines the type of designs

matrix_isomorphism_t **itype**

defines the isomorphism type

bool **j1zero**

if true then J1 values should be zero

bool **j3zero**

if true then J3 values should be zero

class **CandidateGeneratorBase**

#include <conference.h> Class to generate candidate extensions with caching

We assume that the designs to be extended are run ordered, so that the caching has maximal effect.

The key idea used is that any valid extension of a design A with k columns is a permutation of a valid extension of the design B obtained by taking the first l < k columns of A. The permutations that are allowed are called the symmetry inflations. All the j2 checks performed for the extension of B do not have to be repeated for the permutations of this extension.

Subclassed by *CandidateGeneratorConference*, *CandidateGeneratorDouble*

Public Functions

CandidateGeneratorBase(const *array_link* &al, const *conference_t* &ct)

void **showCandidates**(int verbose = 1) const

Show the candidate extensions for each column

conference_column_list **candidates**(int k)

return all candidates for the kth column

Public Members

conference_t **ct**

type of designs to generate

int **verbose**

verbosity level

mutable *array_link* **al**

last array analyzed

mutable int **last_valid**

index of last valid column

Protected Functions

inline int **startColumn**(const *array_link* &alx, int verbose = 0) const

Find the starting column for the extension of a design

The static variable START_COL is the number of columns for which is the starting point if the cache is empty. Therefore for a design with initial columns the same, START_COL+1 is the first number of columns with valid entries.

For startcol k the elements in candidate_list[k] are valid, e.g. we can start with extensions valid for index k-1

Protected Attributes

mutable std::vector<*conference_column_list*> **candidate_list**

list of candidate extensions. the elements of candidate_list[k] correspond to columns with index k-1

Protected Static Attributes

static const int **START_COL** = 2

class **CandidateGeneratorConference** : public *CandidateGeneratorBase*

#include <conference.h> Class to generate conference candidate extensions.

Public Functions

CandidateGeneratorConference(const *array_link* &al, const *conference_t* &ct)

const std::vector<*conference_column*> &**generateCandidates**(const *array_link* &al) const
Generate a list of candidate extensions for the specified design.

std::vector<*conference_column*> **generateCandidatesZero**(const *array_link* &al, int kz) const
generate all candidate extensions with a zero at the specified position

class **CandidateGeneratorDouble** : public *CandidateGeneratorBase*

#include <conference.h> Class to generate double conference candidate extensions with caching.

Public Functions

CandidateGeneratorDouble(const *array_link* &al, const *conference_t* &ct)

const std::vector<*conference_column*> &**generateCandidates**(const *array_link* &al) const
Generate a list of candidate extensions for the specified design

This method uses symmetry inflation, assumes j1=0 and j2=0. Optimal performance is achieved when the arrays to be extended have identical first columns.

struct **conference_extend_t**

#include <conference.h> Helper structure containing extensions of conference designs

Public Functions

inline *conference_column* **combine**(int i, int j) const
list of candidate extensions

inline size_t **nExtensions**() const

inline *arraylist_t* **getarrays**(const *array_link* al) const
return the set of extension arrays

Public Members

std::vector<*conference_column*> **first**

std::vector<*conference_column*> **second**
list of first block candidate extensions

std::vector<*conference_column*> **extensions**
list of first block candidate extensions

class **DconferenceFilter**

#include <conference.h> class to filter single or double conference designs

Public Functions

DconferenceFilter(const *array_link* &_als, int filter_symmetry, int filterj2_, int filterj3_ = 1)

void **show**() const
  print object to stdout

std::vector<*conference_column*> **filterList**(const std::vector<*conference_column*> &lst, int verbose = 0) const
  filter a list of columns using the filter method

std::vector<*conference_column*> **filterListJ2last**(const std::vector<*conference_column*> &column_list) const

std::vector<*conference_column*> **filterListZero**(const std::vector<*conference_column*> &lst) const
  filter a list of cperms using the filterZero method

bool **filter**(const *conference_column* &c) const
  return True if the extension satisfies all checks

bool **filterJpartial**(const *conference_column* &column, int maxrow) const
  Filter on partial column (only last col)

  Parameters
  - **column** – Extension column
  - **maxrow** – the number of rows that are valid

bool **filterJ**(const *conference_column* &column, int j2start = 0) const
  return True if the extension satisfies all J-characteristic checks

bool **filterJlast**(const *conference_column* &c, int j2start = 0) const
  return True if the extension satisfies all J-characteristic checks for the last columns

bool **filterReason**(const *conference_column* &column) const
  return True if the extension satisfies all checks. prints the reason for returning True or False to stdout

bool **filterJ3**(const *conference_column* &column) const
  return True if the candidate satisfies the J3 check

bool **filterJ3s**(const *conference_column* &column, int idxstart) const
  return True if the candidate satisfies the J3 check for specified pairs

bool **filterJ3inline**(const *conference_column* &column) const
  return True if the candidate satisfies the J3 check

bool **filterSymmetry**(const *conference_column* &column) const
  return True of the candidate satisfies the symmetry check

bool **filterJ2**(const *conference_column* &c) const
  return True of the candidate extension satisfies the J2 check

bool **filterJ2last**(const *conference_column* &c) const
  return True of the candidate extension satisfies the J2 check for the last column of the array checked against

bool **filterZero**(const *conference_column* &c) const
  return True of the candidate extension satisfies the zero check

  This means that the first entries of the extension do not contain a zero.

Public Members

array_link **als**

int **filtersymm**

filter based on symmetry

int **filterj2**

filter based on j2 value

int **filterj3**

filter based on j3 value

int **filterfirst**

filter only columns with first value >=0

int **filterzero**

filter based on first occurence of zero in a column

mutable long **ngood**

int **inline_row**

row at which infile filtering is performed

symmdata **sd**

Private Members

array_link **dtable**

table of J2 vectors for J3 filter

array_link **inline_dtable**

table of J2 vectors for inline J3 filter

std::vector<int> **check_indices**

indices to check for symmetry check

int **minzvalue**

used for filtering based on zero

7.5 Interface for even-odd designs

Contains functions to generate even-odd designs.

The generation is done by defining a special ordering in the set of designs. The primary ordering is based in the J5 value of 5-column designs, the secondary ordering is the regular LMC ordering.

Typedefs

typedef *Pareto*<mvalue_t<long>, *array_link*>::pValue (***pareto_cb**)(const *array_link*&, int)

  callback function for *Pareto* calculations

typedef *Pareto*<mvalue_t<long>, *array_link*>::pValue (***pareto_cb_cache**)(const *array_link*&, int, *rankStructure* &rs)

  callback function for *Pareto* calculations with cache

Enums

enum **depth_alg_t**

  Values:

  enumerator **DEPTH_DIRECT**

  enumerator **DEPTH_EXTENSIONS**

Functions

void **processDepth**(const *arraylist_t* &goodarrays, *depth_alg_t* depthalg, *depth_extend_t* &dextend, *depth_extend_sub_t* &dextendsublight, int extensioncol, int verbose = 0)

  Extend arrays using a depth-first or breadth-first approach

  Parameters

  - **goodarrays** – List of arrays to extend
  - **depthalg** – Extend using depth-first or breadth-first
  - **dextend** – Option structure for the extension
  - **dextendsublight** – Data structure for the extensions
  - **extensioncol** – Column to extend
  - **verbose** – Verbosity level

void **depth_extend_hybrid**(const *arraylist_t* &alist, *depth_extend_t* &dextend, int extcol, const *OAextend* &oaextendx, int verbose)

  depth-first extension of arrays. depending on the symmetry group of the array to be extended a direct method is used or a method with caching of candidate columns void **depth_extend_direct**(const *arraylist_t* &alist, *depth_extend_t* &dextend, int extcol, const *OAextend* &oaextendx, int verbose)

variation of depth_extend for arrays with large symmetry groups

void **depth_extend_array**(const *array_link* &al, *depth_extend_t* &dextend, const *arraydata_t* &adfull, int verbose, *depth_extensions_storage_t* *ds = 0, int = 0)

depth extend a single array

template<class **IndexType**>
inline *Pareto*<mvalue_t<long>, *IndexType*>::pValue **calculateArrayParetoJ5Cache**(const *array_link* &al, int verbose, *rankStructure* &rs)

void **addArraysToPareto**(*Pareto*<mvalue_t<long>, *array_link*> &pset, *pareto_cb* paretofunction, const *arraylist_t* &arraylist, int jj, int verbose)

add arrays to set of *Pareto* results

void **addArraysToPareto**(*Pareto*<mvalue_t<long>, *array_link*> &pset, *pareto_cb_cache* paretofunction, const *arraylist_t* &arraylist, int jj, int verbose)

add arrays to set of *Pareto* results

Jcounter **readStatisticsFile**(const char *numbersfile, int verbose)

read statistics object from disk

void **writeStatisticsFile**(const char *numbersfile, const *Jcounter* &jc, int verbose)

write statistics object to disk

Jcounter **calculateJstatistics**(const char *afile, int jj = 5, int verbose = 1)

calculate J-value statistics

int **compareJ54**(const *array_link* &lhs, const *array_link* &rhs)

Return -1 if the first array is smaller in J54 ordering than the second array, 0 if equal and 1 otherwise

struct **depth_path_t**

#include <evenodd.h> structure containing current position in search tree

Public Functions

inline **depth_path_t**()

inline void **updatePositionGEC**(int k, int goodextensioncols)

inline void **updatePosition**(int k, int c, int m, int extensioncols, int goodextensioncols)

inline void **show**(int depth, int maxentries = 8) const

inline void **init**(int ncols, int _depthstart = 9)

Public Members

std::vector<int> **ncurr**
vector with current position

std::vector<int> **nmax**
vector with target

std::vector<int> **necols**
number of extension columns

std::vector<int> **ngecols**
number of good extension columns

int **depthstart**

struct **counter_t**
#include <evenodd.h> structure to count and show number of arrays generated, the structure is thread safe

Public Functions

counter_t(int n)

void **addNfound**(int col, int num)

long **nArrays**() const

void **addNumberFound**(int n, int k)

void **clearNumberFound**()

void **addNumberFound**(const *counter_t* &de)

void **showcountscompact**() const
show information about the number of arrays found

void **showcounts**(const *arraydata_t* &ad) const
show information about the number of arrays found

void **showcounts**(const char *str, int first, int last) const
show information about the number of arrays found

Public Members

std::vector<int> **nfound**

struct **depth_extend_sub_t**

#include <evenodd.h> Helper structure for dynamic extension

In this structure we keep track of pointers to valid column extensions

Public Functions

inline **depth_extend_sub_t**(int nn = 0)

inline void **resize**(int nn)

inline size_t **n**() const

inline std::vector<int> **updateExtensionPointers**(int extcol)

arraylist_t **initialize**(const *arraylist_t* &alist, const *arraydata_t* &adf, const *OAextend* &oaextend)

initialize the new list of extension columns

inline *arraylist_t* **selectArraysZ**(const *arraylist_t* &alist) const

select the arrays with are LMC and hence need to be written to disk

inline *arraylist_t* **selectArraysXX**(const *array_link* &al, const *arraylist_t* &elist) const

inline void **info**() const

Public Members

std::vector<int> **lmctype**

std::vector<int> **lastcol**

last column changed in lmc check

std::vector<double> **strengthcheck**

std::vector<int> **valididx**

int **verbose**

struct **depth_extend_t**

#include <evenodd.h> Helper structure for dynamic extension.

This structure allows for writing the generated arrays to disk. It also contains functions to print progress of the extension.

Multiple copies of this class are made, but they all share the same *counter_t* and *arraywriter_t* object. Also t0 and tp are shared

Public Functions

inline **depth_extend_t**(const *arraydata_t* *ad_, double _logtime = 10000000, int _discardJ5 = -1)

inline **depth_extend_t**(const *depth_extend_t* &de)

inline **~depth_extend_t**()

inline void **show**()

inline void **setNarraysMax**(long n)

inline void **maxArrayCheck**()

inline void **showsearchpath**(int depth) const

inline bool **showprogress**(int showtime = 1, int depth = 0, int forcelog = 0)
show information about the progress of the loop

inline void **info**() const

inline void **setposition**(int k, int c, int m, int extensioncols = -1, int goodextensioncols = -1)
set the position in the dextend structure

inline void **setpositionGEC**(int k, int goodextensioncols)
set the position in the dextend structure

Public Members

int **verbose**

OAextend **oaextend**

const *arraydata_t* ***ad**

int **loglevelcol**

double **logtime**
print progress every x seconds

arraylist_t **extension_column_list**

int **writearrays**
if set to true write arrays to disk

int **discardJ5**

long **discardJ5number**
if true, then we discard the designs which have J5 maximal

arraywriter_t ***arraywriter**

counter_t ***counter**

Public Static Attributes

static double **t0**

static double **tp**

Private Members

long **narraysmax**

depth_path_t **searchpath**

struct **depth_extensions_storage_t**

#include <evenodd.h> Helper structure for the even-odd depth extension.

Public Functions

inline void **resize**(size_t s)

inline void **set**(int ai, const *arraylist_t* &goodarrays, const *arraylist_t* &extension_column_list, *depth_alg_t* depthalg, const *depth_extend_sub_t* &dextendsub)

Public Members

std::vector<*arraylist_t*> **columnextensionsList**

std::vector<*arraylist_t*> **goodarrayslist**

std::vector<*depth_alg_t*> **depthalglist**

std::vector<*depth_extend_sub_t*> **dextendsubList**

struct **jindex_t**

#include <evenodd.h> helper class for indexing statistics of designs

The index consists of the number of columns and the value for the J-characteristic

Public Functions

inline **jindex_t**(int colindex, int jvalue)

inline bool **operator<**(const *jindex_t* &rhs) const

inline std::string **toString**() const

Public Members

int **k**

number of columns

int **j**

J-value.

class **Jcounter**

#include <evenodd.h> object to hold counts of maximum J_k-values

Public Functions

inline **Jcounter**()

inline **Jcounter**(int N, int jj = 5, int k = -1)

bool **validData**()

bool **hasColumn**(int col) const

return true if specified column is in the data

inline bool **isOpen**() const

inline void **showPerformance**() const

long **narrays**() const

void **show**() const

show statistics of the object

int **maxCols**() const

long **getCount**(int k, int j) const

std::vector<long> **getTotalsJvalue**(int jval) const

std::vector<long> **getTotals**() const

void **showcompact**() const

show statistics of the object

Jcounter &**operator+=**(*Jcounter* &jc)

void **addArrays**(const *arraylist_t* &arraylist, int verbose = 0)

add list of arrays to object

void **addArray**(const *array_link* &al, int verbose = 0)
add single array to statistics object

Public Members

int **N**
number of rows

int **jj**

std::vector<int> **fvals**

std::map<*jindex_t*, long> **maxJcounts**

double **dt**
time needed for calculation

Private Functions

void **init**(int N, int jj, int k = -1)

7.6 Interface for extension of LMC designs

Contains functions to generate and extend orthogonal arrays.

Enums

enum **dfilter_t**
Values:

enumerator **DFILTER_NONE**
no filtering on D-efficiency

enumerator **DFILTER_BASIC**
filtering on D-efficiency

enumerator **DFILTER_MULTI**
filtering on D-efficiency with multi column prediction

enum **dcalc_mode**
Values:

enumerator **DCALC_ALWAYS**

always calculate efficiency

enumerator **DCALC_COND**

only calculate efficiency for LMC_LESS

Functions

arraylist_t **extend_arraylist**(const *arraylist_t* &array_list, *arraydata_t* &array_class, *OAextend* const &oaextend_options)

Extend a list of orthogonal arrays

See also:

extend_array(const array_link &, arraydata_t &, OAextend const &)

Parameters

- **array_list** – The list of arrays to be extended
- **array_class** – Class of arrays to generate
- **oaextend_options** – Parameters for the extension algorithm

Returns

List of all generated arrays

arraylist_t **extend_arraylist**(const *arraylist_t* &array_list, const *arraydata_t* &arrayclass)

Extend a list of arrays with default options

See also:

extend_array(const array_link &, arraydata_t &, OAextend const &)

int **extend_arraylist**(const *arraylist_t* &array_list, *arraydata_t* &array_class, *OAextend* const &oaextend_options, *colindex_t* extensioncol, *arraylist_t* &extensions)

Extend a list of orthogonal arrays

See also:

extend_array(const array_link &, arraydata_t &, OAextend const &)

Parameters

- **array_list** – The list of arrays to be extended
- **array_class** – Class of arrays to generate
- **oaextend_options** – Parameters for the extension algorithm
- **extensioncol** – Index of column to be added to the designs
- **extensions** – List to append generated designs to

Returns
List of all generated arrays

Returns
Number of candidate arrays generated

arraylist_t **extend_array**(const *array_link* &array, *arraydata_t* &array_class, *OAextend* const &oaextend)

Extend a single orthogonal array

Parameters

- **array** – The array to be extended
- **array_class** – Class of arrays to generate
- **oaextend** – Parameters for the extension algorithm

arraylist_t **extend_array**(const *array_link* &array, *arraydata_t* &arrayclass)

Extend a single orthogonal array with the default LMC algorithm

See also:

extend_array(const array_link &, arraydata_t &, OAextend const &)

int **extend_array**(const *array_link* &array, const *arraydata_t* *arrayclass, const *colindex_t* extension_column, *arraylist_t* &extensions, *OAextend* const &oaextend)

Extend an orthogonal array with a single column

See also:

extend_array(const array_link &, arraydata_t &, OAextend const &)

Parameters

- **array** – Array to extend
- **arrayclass** – Array data for the full array
- **extension_column** – Column to extend
- **extensions** – List to which generated valid extensions are added
- **oaextend** – Structure with options

Returns
Number of candidate extensions generated

arraylist_t **runExtendRoot**(*arraydata_t* arrayclass, int max_number_columns, int verbose = 0)

Run the LMC extension algorithm starting with the root array

See also:

extend_array(const array_link &, arraydata_t &, OAextend const &)

class **OAextend**

#include <extend.h> Options for the extend code.

class containing parameters of the extension and LMC algorithm

Public Types

enum **extendarray_mode_t**

Specification of how to use the generated extensions.

Values:

enumerator **APPENDEXTENSION**

append extension column to extension list

enumerator **APPENDFULL**

append full array to extension list

enumerator **STOREARRAY**

store extension to disk

enumerator **NONE**

do not store generated extensions

Public Functions

inline **OAextend**()

Options for the extension algorithm

inline **OAextend**(const *OAextend* &o)

Options for the extension algorithm

inline **OAextend**(*arraydata_t* &arrayclass)

Options for the extension algorithm

The algorithm is automatically determined from the specified arrayclass.

void **setAlgorithm**(*algorithm_t* algorithm, *arraydata_t* *ad = 0)

Set the algorithm to use for LMC checks.

void **setAlgorithmAuto**(*arraydata_t* *ad = 0)

Set the algorithm automatically.

inline *algorithm_t* **getAlgorithm**() const

Return algorithm used.

inline std::string **getAlgorithmName**() const

Return algorithm used (as string)

void **updateArraydata**(*arraydata_t* *arrayclass = 0) const

update the options structuer with the specified class of designs

void **info**(int verbose = 1) const

print configuration to stdout

std::string **__repr__**() const

Public Members

double **singleExtendTime**

time before printing progress of single extension, [seconds]

int **nLMC**

number of arrays LMC tested before printing progress of single extension

int **checkarrays**

perform LMC test after generation of array

int **check_maximal**

if true then return at once if a single extension has been found

int **use_row_symmetry**

adds a symmetry check to the extension algorithm based in symmetry of row permutations

int **init_column_previous**

init column with previous column in extension (if in the same column group)

extendarray_mode_t **extendarraymode**

determines how the extension arrays are stored

arrayfile_t **storefile**

j5structure_t **j5structure**

Public Static Functions

static inline *algorithm_t* **getPreferredAlgorithm**(const *arraydata_t* &arrayclass, int verbose = 0)

Return preferred extension algorithm

Parameters

- **arrayclass** – Class of designs to extend
- **verbose** – Verbosity level

Returns

Algorithm selected to be used for this class

Private Members

algorithm_t **algmode**

Algorithm mode.

struct **dextend_t**

#include <extend.h> Structure for dynamic extension of arrays based on D-efficiencies.

Public Functions

inline **dextend_t**()

inline void **resize**(int nn)

void **DefficiencyFilter**(double Dfinal, int k, int kfinal, double Lmax, int verbose = 1)

perform filtering using D-efficiency

std::vector<int> **filterArrays**(const *array_link* &al, const *arraylist_t* &earrays, *arraylist_t* &earraysout, std::vector<std::vector<double>> &edata, int verbose = 1)

filter the arrays based on values in filter

Public Members

std::vector<*lmc_t*> **lmctype**

results of minimal form calculations

std::vector<int> **lastcol**

last column changed in lmc check

std::vector<double> **Deff**

calculated efficiency values

std::vector<int> **filter**

indices of filtered arrays

dfilter_t **filtermode**

dcalc_mode **Dcheck**

int **directcheck**

perform immediate LMC check in extension

long **ntotal**

total number of arrays found

long **nlmc**

total number of arrays found in LMC form

long **n**

total number of arrays found passing all tests

double **DmaxDiscard**

long **nmaxrnktotal**

Public Static Attributes

static const int **NO_VALUE** = 0

7.7 Interface for graph tools

This file contains definitions and functions related to graphs and designs.

Author: Pieter Eendebak pieter.eendebak@gmail.com, (C) 2016

Copyright: See COPYING file that comes with this distribution

Enums

enum **matrix_isomorphism_t**

Isomorphism types for matrices

Isotopy: permute rows, columns and symbols Matrix isomorphism: permute rows and columns Conference isomorphism: permute rows, columns and to row and column negations (values in 0, +1, -1) Orthogonal array isomorphism: permutations of rows, columns and column symbol permutations

Values:

enumerator **ISOTOPY**

isotopy: permute rows, columns and symbols

enumerator **MATRIX_ISOMORPHISM**

permute rows and columns

enumerator **CONFERENCE_ISOMORPHISM**

permute rows, columns and to row and column negations (values in 0, +1, -1)

enumerator **OA_ISOMORPHISM**

permutations of rows, columns and column symbol permutations

Functions

array_link **transformGraph**(const *array_link* &graph, const std::vector<int> vertex_permutation, int verbose = 1)

Apply a vertex permutation to a graph.

array_transformation_t **reduceOAnauty**(const *array_link* &array, int verbose = 0)

Reduce an orthogonal array to Nauty minimal form. the array transformation is returned.

array_transformation_t **reduceOAnauty**(const *array_link* &array, int verbose, const *arraydata_t* &arrayclass)

Reduce an orthogonal array to Nauty minimal form. the array transformation is returned.

std::pair<*array_link*, std::vector<int>> **array2graph**(const *array_link* &array, int verbose = 1)

Convert orthogonal array to graph representation

The conversion method is as in Ryan and Bulutoglu. The resulting graph is bi-partite. The graph representation can be used for isomorphism testing.

std::pair<*array_link*, std::vector<int>> **array2graph**(const *array_link* &array, int verbose, const *arraydata_t* &arrayclass)

Convert orthogonal array to graph representation

The conversion method is as in Ryan and Bulutoglu. The resulting graph is bi-partite. The graph representation can be used for isomorphism testing.

array_transformation_t **oagraph2transformation**(const std::vector<int> &pp, const *arraydata_t* &arrayclass, int verbose = 1)

From a relabelling of the graph return the corresponding array transformation.

Variables

const *matrix_isomorphism_t* **CONFERENCE_RESTRICTED_ISOMORPHISM** = OA_ISOMORPHISM

isomorphism type for column and row permtations and column permutations

namespace **nauty**

Functions

std::vector<int> **reduceNauty**(const *array_link* &graph, std::vector<int> colors, int verbose = 0)

Reduce a colored graph to Nauty minimal form

The transformation returned is from the normal form to the specified graph.

Parameters

- **graph** – Graph in incidence matrix form
- **colors** – Colors of the graph nodes
- **verbose** – Verbosity level

Returns

Relabelling of the graph vertices

7.8 Interface for LMC normal forms

This file contains definitions and functions to perform minimal form tests and reductions.

Author: Pieter Eendebak pieter.eendebak@gmail.com

Copyright: See LICENSE file that comes with this distribution

Defines

ORDER_J5_SMALLER

ORDER_J5_GREATER

ORDER_J45_SMALLER

ORDER_J45_GREATER

stringify(name)

SDSMART

Typedefs

typedef unsigned int **rowsort_value_t**

typedef *rowindex_t* **rowsortlight_t**

typedef larray<*rowindex_t*> **rowpermtypelight**

typedef larray<*colindex_t*> **colpermtypelight**

typedef std::vector<int> **colpermtype**

typedef std::vector<*colpermtype*> **colpermset**

typedef std::tr1::shared_ptr<*symmdata*> **symmdataPointer**

typedef double **jj45_t**

Value representing the ordered combination of J5 and the 5 J4-values in the J54 ordering.

Enums

enum **lmc_t**

Possible results for the LMC check

Values:

enumerator **LMC_LESS**

Found a permutation which leads to a lexicographically smaller array.

enumerator **LMC_EQUAL**

Found a permutation which leads to a lexicographically equal array.

enumerator **LMC_MORE**

Found a permutation which leads to a lexicographically larger array.

enumerator **LMC_NONSENSE**

No valid result.

enum **algorithm_t**

different algorithms for minimal form check

Values:

enumerator **MODE_LMC**

LMC minimal form.

enumerator **MODE_J4**

LMC minimal form with J4 method.

enumerator **MODE_J5ORDER**

J5 minimal form.

enumerator **MODE_J5ORDERX**

J5 minimal form.

enumerator **MODE_INVALID**

enumerator **MODE_AUTOSELECT**

Automatically select the algorithm.

enumerator **MODE_LMC_SYMMETRY**

debugging method

enumerator **MODE_LMC_2LEVEL**

LMC minimal form, specialized for 2-level arrays.

enumerator **MODE_LMC_DEBUG**

debugging method

enumerator **MODE_J5ORDER_2LEVEL**

J5 minimal form for 2-level arrays.

enum **initcolumn_t**

method used for initialization of columns

Values:

enumerator **INITCOLUMN_ZERO**

Initialize column with zeros.

enumerator **INITCOLUMN_PREVIOUS**

Initialize column with values of previous column.

enumerator **INITCOLUMN_J5**

Initialize column with values based on J5 value.

enum **j5structure_t**

variations of the J45 structures

Values:

enumerator **J5_ORIGINAL**

Ordering based in J5 in succesive columns.

enumerator **J5_45**

Ordering based on J5 and the 5-tuple of J4 values. Also called the L5 ordering.

enum **REDUCTION_STATE**

variable indicating the state of the reduction process

Values:

enumerator **REDUCTION_INITIAL**

the reduction is equal to the initial

enumerator **REDUCTION_CHANGED**

the reduction was changed

enum **OA_MODE**

main mode for the LMC routine: test, reduce or reduce with initialization

Values:

enumerator **OA_TEST**

test for minimal form

enumerator **OA_REDUCE**

reduce to minimal form

enumerator **OA_REDUCE_PARTIAL**

reduce to partial minimal form

enum **INIT_STATE**

initial state for reduction algorithm

Values:

enumerator **INIT_STATE_INVALID**

enumerator **COPY**

copy from array argument

enumerator **INIT**

initialized by user

enumerator **SETROOT**

set initial state to root array

Functions

inline std::string **algorithm_t_list**()

Return string representation of available algorithm modes.

std::string **algnames**(*algorithm_t* m)

return name of the algorithm

static inline bool **operator<**(const rowsort_t &a, const rowsort_t &b)

Comparision operator for the rowsort_t structure.

static inline bool **operator>**(const rowsort_t &a, const rowsort_t &b)

Comparision operator for the rowsort_t structure.

void **apply_hadamard**(*array_link* &al, *colindex_t* hcolumn)

Apply Hadamard transformation to orthogonal array.

LMCreduction_helper_t ***acquire_LMCreduction_object**()

return static structure from dynamic global pool, return with releaseGlobalStatic

void **release_LMCreduction_object**(*LMCreduction_helper_t* *p)

void **clear_LMCreduction_pool**()

release all objects in the pool

bool **valid_ptr**(const *symmdataPointer* &sd)

Return true if the (smart) symmdataPointer pointer is allocated

template<class **Type**>
void **insert_if_not_at_end_of_vector**(std::vector<*Type*> &cp, const *Type* &value)

Append element to vector if the element the element is not at the end of vector.

bool **is_root_form**(const *array_link* &array, int strength)

Return True if the array is in root form

Parameters

- **array** – Array to check
- **strength** – Strength to use

Returns
True if the array is in root form for the specified strength

lmc_t **LMCreduction_train**(const *array_link* &al, const *arraydata_t* *ad, *LMCreduction_t* *reduction, const *OAextend* &oaextend)

helper function for LMC reduction

lmc_t **LMCcheck**(const *array_t* *array, const *arraydata_t* &ad, const *OAextend* &oaextend, *LMCreduction_t* &reduction)

Perform LMC check or reduction on an array.

lmc_t **LMCcheck**(const *array_link* &array, const *arraydata_t* &ad, const *OAextend* &oaextend, *LMCreduction_t* &reduction)

Perform LMC check or reduction on an array.

lmc_t **LMCcheck**(const *array_link* &array)

Perform LMC check on an orthogonal array

Parameters
array – Array to be checked for LMC minimal form

Returns
Result of the LMC check

lmc_t **LMCcheckOriginal**(const *array_link* &array)

Perform LMC check on a 2-level orthogonal array

The algorithm used is the original algorithm from "Complete enumeration of pure-level and mixed-level orthogonal arrays", Schoen et al, 2009

Parameters
array – Array to be checked for LMC minimal form

Returns
Result of the LMC check

void **reduceArraysGWLP**(const *arraylist_t* &input_arrays, *arraylist_t* &reduced_arrays, int verbose, int dopruning = 1, int strength = 2, int dolmc = 1)

reduce arrays to canonical form using delete-1-factor ordering

array_transformation_t **reductionDOP**(const *array_link* &array, int verbose = 0)

Caculate the transformation reducing an array to delete-on-factor normal

The normal form is described in "A canonical form for non-regular arrays based on generalized wordlength pattern values of delete-one-factor projections", Eendebak, 2014

Parameters

- **array** – Orthogonal array
- **verbose** – Verbosity level

Returns

The transformation that reduces the array to normal form

array_link **reduceDOPform**(const *array_link* &array, int verbose = 0)

Reduce an array to canonical form using delete-1-factor ordering

The normal form is described in "A canonical form for non-regular arrays based on generalized wordlength pattern values of delete-one-factor projections", Eendebak, 2014

Parameters

- **array** – Orthogonal array
- **verbose** – Verbosity level

Returns

The array transformed to normal form

void **selectUniqueArrays**(*arraylist_t* &input_arrays, *arraylist_t* &output_arrays, int verbose = 1)

select the unique arrays in a list, the original list is sorted in place. the unique arrays are append to the output list

std::vector<GWLPvalue> **projectionDOFvalues**(const *array_link* &array, int verbose = 0)

Calculate projection values for delete-of-factor algorithm

array_link **reduceLMCform**(const *array_link* &array)

reduce an array to canonical form using LMC ordering

std::vector<int> **LMCcheckLex**(*arraylist_t* const &list, *arraydata_t* const &ad, int verbose = 0)

Apply LMC check (original mode) to a list of arrays

lmc_t **LMCcheckLex**(*array_link* const &array, *arraydata_t* const &arrayclass)

Perform minimal form check with LMC ordering.

lmc_t **LMCcheckj4**(*array_link* const &array, *arraydata_t* const &arrayclass, *LMCreduction_t* &reduction, const *OAextend* &oaextend, int jj = 4)

Perform minimal form check with J4 ordering.

lmc_t **LMCcheckj5**(*array_link* const &array, *arraydata_t* const &arrayclass, *LMCreduction_t* &reduction, const *OAextend* &oaextend)

Perform minimal form check for J5 ordering (in the paper this is called the L5 ordering)

void **print_rowsort**(rowsort_t *rowsort, int N)

Print the contents of a rowsort structure.

Parameters

- **rowsort** – Pointer to rowsort structure
- **N** – Number of elements

void **print_column_rowsort**(const *array_t* *arraycol, rowsort_t *rowsort, int N)

Variables

const *algorithm_t* **MODE_ORIGINAL** = MODE_LMC

struct **LMCreduction_helper_t**

#include <lmc.h> Contains structures used by the LMC reduction or LMC check.

Part of the allocations is for structures that are constant and are re-used each time an LMC calculation is performed. Some other structures are temporary buffers that are written to all the time.

Public Functions

LMCreduction_helper_t()

~LMCreduction_helper_t()

inline void **show**(int verbose = 1) const

void **init**(const *arraydata_t* *adp)

void **freeall**()

int **update**(const *arraydata_t* *adp)

update structure with new design specification

int **needUpdate**(const *arraydata_t* *adp) const

void **init_root_stage**(*levelperm_t* *&lperm_p, *colperm_t* *&colperm_p, const *arraydata_t* *adp)

void **init_nonroot_stage**(*levelperm_t* *&lperm_p, *colperm_t* *&colperm_p, *colperm_t* *&localcolperm_p, *dyndata_t* **&dynd_p, int &dynd_p_nelem, *array_t* *&colbuffer, const *arraydata_t* *adp) const

inline void **init_rootrowperms**(int &totalperms, *rowperm_t* *&rootrowperms, *levelperm_t* *&lperm_p)

Static initialization of root row permutations.

inline void **init_rootrowperms_full**(int &totalperms, *rowperm_t* *&rootrowperms, *levelperm_t* *&lperm_p)

Static initialization of root row permutations (full group)

Public Members

int **LMC_non_root_init**

int **LMC_root_init**

int **LMC_reduce_root_rowperms_init**

arraydata_t ***ad**

int **LMC_root_rowperms_init**

int **nrootrowperms**

number of root row permutations

rowperm_t ***rootrowperms**

pointer to row permutations that leave the root unchanged

int **LMC_root_rowperms_init_full**

int **nrootrowperms_full**

rowperm_t ***rootrowperms_full**

array_t ***colbuffer**

dyndata_t ****dyndata_p**

buffer for a single column

colindex_t ****colperm_p**

dynamic data; row permutations

colindex_t ****localcolperm_p**

column permutations

array_transformation_t ***current_trans**

local column permutation

struct **LMCreduction_t**

#include <lmc.h> Class to describe an LMC reduction.

The most important variable is the transformation itself, contained in transformation. The state contains information about how the reduction was performed.

Public Functions

LMCreduction_t(const *LMCreduction_t* &at)

LMCreduction_t(const *arraydata_t* *arrayclass)

copy constructor

~LMCreduction_t()

LMCreduction_t &**operator=**(const *LMCreduction_t* &at)

inline *array_link* **getArray**() const

Assignment operator.

inline void **setArray**(const *array_link* al)

inline void **setArray**(const *array_t* *array, int nrows, int ncols)

inline void **updateSDpointer**(const *array_link* al, bool cache = false)

update the pointer to the symmetry data based on the specified array

inline void **releaseStatic**()

release internal *LMCreduction_helper_t* object

inline void **initStatic**()

acquire a reference to a *LMCreduction_helper_t* object

inline *LMCreduction_helper_t* &**getReferenceReductionHelper**()

return a reference to a object with LMC reduction data

void **reset**()

reset the reduction: clears the symmetries and sets the transformation to zero

void **show**(int verbose = 2) const

std::string **__repr__**() const

void **updateFromLoop**(const *arraydata_t* &ad, const *dyndata_t* &dynd, *levelperm_t* *lperms, const *array_t* *original)

called whenever we find a reduction

void **updateTransformation**(const *arraydata_t* &ad, const *dyndata_t* &dynd, *levelperm_t* *lperms, const *array_t* *original)

inline void **updateLastCol**(int col)

Public Members

array_t ***array**

array_transformation_t ***transformation**

pointer to transformation_t structure

OA_MODE **mode**

REDUCTION_STATE **state**

INIT_STATE **init_state**

int **maxdepth**

maximum depth for search tree

int **lastcol**

last column visited in algorithm

long **nred**
counter for number of reductions made

int **targetcol**

int **mincol**

int **nrows**

int **ncols**

LMCreduction_helper_t ***staticdata**

symmdataPointer **sd**

Private Functions

void **free**()

class **rowsorter_t**
#include <lmc.h> Structure to sort rows of arrays.

Public Functions

rowsorter_t(int number_of_rows)

~rowsorter_t()

Public Members

int **number_of_rows**

rowsort_t ***rowsort**

Private Functions

void **reset_rowsort**()

struct **dyndata_t**
#include <lmc.h> Contains dynamic data of an array.

The dynamic data are used in the inner loops of the LMC algorithm. In particular they keep track of the current row ordering and column permutation. By not applying these transformations to the array we can save calculation time.

We try to prevent copying the object, so it is re-used at different levels in the algorithm.

- N: static
  - col: changes at each column level
- rowsort: changes at each column level, used mainly in non-root stage
- colperm: changes at all levels

  See also:

  arraydata_t

Public Functions

dyndata_t(int N, int col = 0)

dyndata_t(const *dyndata_t* *dd)

dyndata_t(const *dyndata_t*&)

~dyndata_t()

dyndata_t &**operator=**(const *dyndata_t*&)

void **show**() const

void **reset**()

inline void **setColperm**(const *colperm_t* perm, int n)

inline void **setColperm**(const larray<*colindex_t*> &perm)

inline void **setColperm**(const std::vector<*colindex_t*> &perm)

void **getRowperm**(*rowpermtypelight* &rp) const
    get lightweight row permutation

void **getRowperm**(*rowperm_t* &rperm) const
    get row permutation

rowpermtypelight **getRowperm**() const
    return lightweight row permutation

colpermtypelight **getColperm**() const
    return column permutation

void **getColperm**(*colpermtypelight* &cp) const
    set column permutation

inline void **allocate_rowsortl**()
    allocate lightweight rowsort structure

inline void **deleterowsortl**()

inline void **initrowsortl**()
    initialize rowsortl from rowsort

inline void **rowsortl2rowsort**()
    copy rowsortl variable to rowsrt

void **copydata**(const *dyndata_t* &dd)

Public Members

colindex_t **col**
  active column

rowindex_t **N**
  number of rows

rowsort_t ***rowsort**
  ordering of rows

rowsortlight_t ***rowsortl**

colperm_t **colperm**
  current column permutation

Private Functions

void **initdata**(const *dyndata_t* &dd)

7.9 Interface for MD5 sums

Contains functions to calculate MD5 sums.

Functions

std::string **md5**(void *data, int number_of_bytes)
  calculate md5 sum of a data block in memory

std::string **md5**(const std::string &filename)
  calculate md5 sum of a file on disk

7.10 Interface for Pareto optimality

Class for calculation the *Pareto* optimal elements from a set of multi-valued objects.

Defines

myprintf

template<class **ValueType**, class **IndexType**>

struct **pareto_element**

#include <pareto.h> helper class for the *Pareto* class to hold elements

Public Types

typedef std::vector<*ValueType*> **pValue**

Public Functions

inline bool **dominates**(*pValue* v)

return true of the argument element dominates this value

inline bool **isdominated**(*pValue* v)

return true of the argument element is dominated by this value

inline bool **equal**(*pValue* v)

return true of the argument element is equal to this element

Public Members

pValue **value**

std::vector<*IndexType*> **indices**

template<class **ValueType**, class **IndexType**>

class **Pareto**

#include <pareto.h> Class to the calculate *Pareto* optimal elements.

The class is templated by the type of values to be compared and an index type. The index type is used to index the elements.

For elements added to the *Pareto* structure larger is better.

Public Types

typedef std::vector<*ValueType*> **pValue**

type for values of *Pareto* elements

typedef *pareto_element*<*ValueType*, *IndexType*> **pElement**

a pareto element consists of a pair (value, index)

Public Functions

inline **Pareto**()

Create an empty *Pareto* class.

inline **~Pareto**()

inline int **number**() const

return the total number of *Pareto* optimal values

inline int **numberindices**() const

return the total number *Pareto* optimal objects

inline std::string **__repr__**() const

inline void **show**(int verbose = 1)

show the current set of *Pareto* optimal elements

inline std::deque<*IndexType*> **allindicesdeque**() const

return all indices of the *Pareto* optimal elements as a std::deque

inline std::vector<*IndexType*> **allindices**() const

return all indices of the *Pareto* optimal elements

inline std::vector<*pValue*> **allvalues**() const

return the values of all *Pareto* optimal elements

inline bool **addvalue**(const *pValue* value, const *IndexType* idx)

add a new element

Public Members

int **verbose**

Verbosity level.

std::deque<*pareto_element*<*ValueType*, *IndexType*>> **elements**

contains a list of all *Pareto* optimal elements

Public Static Functions

static inline void **showvalue**(const *pValue* p)

show a *Pareto* element

CHAPTER

EIGHT

REFERENCES

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

TEN

LICENSE

Citation notice: if you use this work or any results of this work, please cite the work as (see the file README.md for details):

"OApackage: A Python package for generation and analysis of orthogonal arrays, optimal designs and conference designs", https://doi.org/10.21105/joss.01097

For more information contact (pieter.eendebak@gmail.com or eric.schoen@tno.nl).

This package also uses code from other works:

- Eigen (see http://eigen.tuxfamily.org/, MPL2 license)
- InfInt (by Sercan Tutar, MPL2 license)
- bitarray (by Isaac Turner, MIT license)
- md5.cpp (by RSA Data Security)
- msstdint.h (by Alexander Chemeris, see https://code.google.com/p/msinttypes/)
- oapackage._scanf (from https://github.com/joshburnett/scanf, MIT license)

BIBLIOGRAPHY

[CD06] C. J. Colbourn and J. H. Dinitz. *Handbook of Combinatorial Designs*. CRC Press, 2006.

[DT99] L.Y. Deng and B. Tang. Generalized resolution and minimum aberration criteria for plackett-burman and other nonregular factorial designs. *Statistica Sinica*, 9:1071–1082, 1999.

[DT02] L.Y. Deng and B. Tang. Design selection and classification for hadamard matrices using generalized minimum aberration criteria. *Technometrics*, 44:173–184, 2002.

[DAT07] A. N. Donev, A. C. Atkinson, and R. D. Tobias. *Optimum Experimental Designs, with SAS*. Oxford Statistical Science Series. Oxford University Press, United Kingdom, 2007. ISBN ISBN-13: 978-0-19-929659-0.

[Een13] P. T. Eendebak. A canonical form for non-regular arrays based on generalized worldlength pattern values of delete-one-factor projections. 2013. Technical report 2014007, University of Antwerp, Faculty of Applied Economics. URL: http://www.pietereendebak.nl/oapage/index.html.

[Een18] P. T. Eendebak. The Orthogonal Array package: results. 2018. URL: http://www.pietereendebak.nl/oapackage/.

[ES17] Pieter T. Eendebak and Eric D. Schoen. Two-level designs to estimate all main effects and two-factor interactions. *Technometrics*, 59(1):69–79, 2017. doi:10.1080/00401706.2016.1142903.

[EN95] Clemens Elster and Arnold Neumaier. Screening by conference designs. *Biometrika*, 82(3):589–602, 1995. URL: http://www.jstor.org/stable/2337536.

[GAX18] Ulrike Groemping, Boyko Amarov, and Hongquan Xu. Doe.base: full factorials, orthogonal arrays and base utilities for doe packages. 2018. URL: https://cran.r-project.org/package=DoE.base.

[GJ+10] Gaël Guennebaud, Benoît Jacob, and others. Eigen v3. http://eigen.tuxfamily.org, 2010.

[HSS99] A.S. Hedayat, N.J.A. Sloane, and J. Stufken. *Orthogonal arrays : theory and applications*. Springer, 1999. doi:10.1007/978-1-4612-1478-6.

[Hun07] J. D. Hunter. Matplotlib: a 2D graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[Kuh18] Warren Kuhfeld. Sas. 2018. URL: http://support.sas.com/techsup/technote/ts723.html.

[LST07] J. L. Loeppky, R. R. Sitter, and B. Tang. Nonregular designs with desirable projection properties. *Technometrics*, 49:454–467, 2007.

[McK81] B. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[MP13] Brendan D. McKay and Adolfo Piperno. Practical graph isomorphism, ii. *CoRR*, 2013.

[Rao47] C.R. Rao. Factorial experiments derivable from combinatorial arrangements of arrays. *Journal of the Royal Statistical Society Supplement*, 9:128–139, 1947.

[SEG19] Eric D. Schoen, Pieter T. Eendebak, and Peter Goos. A classification criterion for definitive screening designs. *Ann. Statist.*, 47(2):1179–1202, 04 2019. doi:10.1214/18-AOS1723.

[SEN10] Eric D. Schoen, Pieter T. Eendebak, and Man V. M. Nguyen. Complete enumeration of pure-level and mixed-level orthogonal arrays. *Journal of Combinatorial Designs*, 18(2):123–140, 2010. doi:10.1002/jcd.20236.

[Slo14] N.J.A. Sloane. A library of orthogonal arrays. 2014. URL: http://neilsloane.com/oadir/index.html.

[WH09] C.F.J. Wu and M.S. Hamada. *Experiments: Planning, Analysis and Optimization*. Wiley, 2009.

[XLB12] Lili Xiao, Dennis K. J. Lin, and Fengshan Bai. Constructing definitive screening designs using conference matrices. *Journal of Quality Technology*, 44(1):2–8, 2012. doi:10.1080/00224065.2012.11917877.

[Xu09] Hongquan Xu. Algorithmic construction of efficient fractional factorial designs with large run sizes. 2009.

[Xu18] Hongquan Xu. Doe.base: full factorials, orthogonal arrays and base utilities for doe packages. 2018. http://www.stat.ucla.edu/\protect\unhbox\voidb@x\penalty\@M\hqxu/nsf/.

[XW01] Hongquan Xu and C. F. J. Wu. Generalized minimum aberration for asymmetrical fractional factorial designs. *Annals of Statistics*, 29:1066–1077, 2001. doi:10.1214/aos/1013699993.

[TheScipycommunity12] The Scipy community. *NumPy Reference Guide*. SciPy.org, 2012. URL: http://docs.scipy.org/doc/numpy/reference/.

INDEX

A

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

V

W