A Constraint Programming Approach to Microplate Layout Design

Andreína Francisco and Ola Spjuth

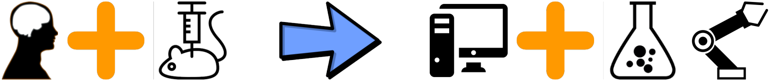
Pharmaceutical Bioinformatics Research Group
Department of Pharmaceutical Biosciences
Uppsala University

Sweden

September 7, 2020

Our Lab's Research Focus

How can we accelerate drug discovery using AI, automation, and intelligent design of experiments?



We want to:

- Predict safety concerns
- Explain drug mechanisms
- Screen for new drugs
- ...

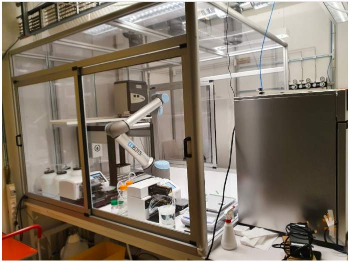
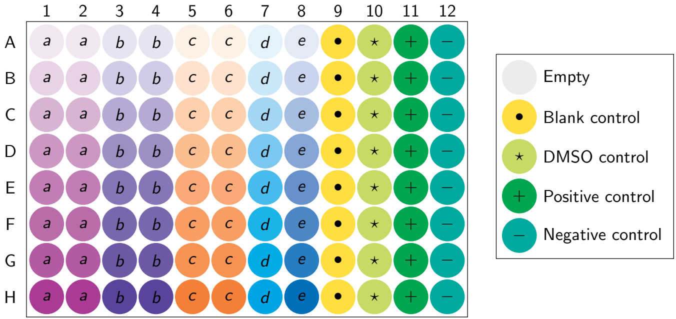


Plate Layout Design In An Ideal World

Everything is perfectly organized!



Letters = compounds

Color intensities = concentrations

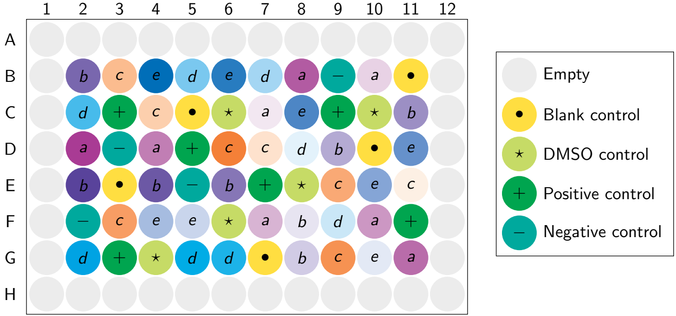
Plate Layout Design In The Real World

- The outermost rows and columns suffer from **edge effect**
- Instruments are imperfect
- Plate cleaning and handling is imperfect
- Time makes a difference: not everything is piped at the same time
- Limited resources: replication vs time vs money vs availability
- ...

Constraints

- Leave outermost rows and columns empty.
- A replica (a compound in all concentrations) must appear on the same plate.
- Different replicas should go on different plates (if there are more plates than number of replicas)
  - ⇒ If there aren't enough plates, spread them as much as possible.
- Extra empty wells should be located near the border
  - ⇒ Opinions vary. Some advocate for clustering them on the last plate.
- Strike a balance between what is near the center of the plate and what is near the boarders
  - ⇒ Currently under discussions and testing!

The Real Plate Layout Design Problem



Input Parameters

- Types and amounts of controls (32 blanks, 16 positives, . . .)
- Number of compounds
- Number of concentrations
- Plate size (96-well, 384-well, . . .)
- Number of replicas
- ...

The Variables

Pre-calculated quantities:

```
%% Number of wells needed.
int: total_wells = (compounds*replicates*concentrations) + sum(controls);

%% Number of plates needed.  Note that plates might not be full
int: numplates = ceil(total_wells/((numcols-2)*(numrows-2)));
```

Variables to model the problem:

```
%% Plates (the solution)
array [Plates,Rows,Columns] of var 0..(experiments+num_controls): plates;

%% Redundant variables (can also be modelled as sets
array [1..experiments] of var Plates: compound_location;
```

(Parts of) The Model

```
%% Edge effect: Leave first and last rows of every plate empty
constraint forall(i in Plates, j in {1,numrows}, k in Columns)(plates[i,j,k] = 0);
```

```
%% Edge effect: Leave first and last columns of every plate empty
constraint forall(i in Plates, j in Rows, k in {1,numcols})(plates[i,j,k] = 0);
```

```
%% Exactly the total number of controls
constraint count_eq(array1d(1..numplates*numcols*numrows, plates), 0, emptywells);
```

```
%% Exactly the total number of empty wells
constraint global_cardinality(array1d(1..numplates*numcols*numrows, plates),[
    experiments+i | i in 1..num_controls],controls);
```

```
%% A compound with all concentrations must appear on a single plate
constraint forall(l in 1..experiments where (l mod concentrations = 1))(all_equal([
    compound_location[i] | i in l..l+concentrations-1]));
```

```
%% Channelling constraint: an experiment appears on a given plate
constraint forall(l in 1..experiments, i in Plates)(compound_location[l] == i <->
    count_eq([array1d(1..numplates*numcols*numrows,plates)[p]
    | p in (i-1)*numcols*numrows+1..i*numcols*numrows],l,1));
```

Ongoing Work

- **Estimating robustness of a particular design**
- Including optional (or alternative) constraints needed by other labs
- Manuscript under preparation. The MiniZinc model, together with some sample data, will be available on GitHub soon!

Questions?

Thank you for listening!