Alex Graber

Andrew Armstrong

OPR 624

D-Optimality with Level Balance Constraints

**Introduction:**

A pharmaceutical market research firm uses simulated patient treatment as a method to understand physician demand in specific treatment areas. In this method, a limited universe of patients is designed in order to represent as much of the actual disease area patient universe as possible. Patients are defined by multiple attributes (age, gender, BMI, etc.), and each attribute may have multiple levels (male/female, etc.). The breakdown of attribute levels is provided by a specified distribution to be achieved in the simulated universe. These simulated patients are then treated, where a given treatment (yes/no) can be related back to the patient design.

Patient design in this manner is a specialized choice methodology somewhat analogous to conjoint. In both conjoint and patient simulation, respondents are forced to make a decision based on a stimulus that is composed of multiple attributes and levels (Kuhfeld, Huber, & Zwerina, 1996). When the number of attributes and levels grow beyond a small set, presenting the full design (full factorial) becomes a challenge due to both the number of combinations required and the amount of burden placed on the respondent. Fractional factorial designs, then, seek to allow the research to eke as much data out of the analysis as possible but use a much more limited subset of stimuli.

Much research has been done on the topic of identifying efficient experimental designs (Hauser & Rao, 2002). The current standard seems to be *D-error* – roughly, the geometric mean of the eigenvalues of the covariance matrix (*D-efficiency* is the inverse of *D-error*) (Kuhfeld, Huber, & Zwerina, 1996). Thus, the goal of an efficient design is to minimize *D-error* (thus maximizing *D-efficiency*). It has been shown that *D-efficient* designs satisfy four principles: orthogonality, level balance, minimal overlap, and utility balance:

“*Orthogonality* is satisfied when the levels of each attribute vary independently of one another. *Level balance* is satisfied when the levels of each attribute appear with equal frequency. *Minimal overlap* is satisfied when the alternatives within each choice set have nonoverlapping attribute levels. *Utility balance* is satisfied when the utilities of alternatives within choice sets are the same” (Kuhfeld, Huber, & Zwerina, 1996).

The standard method to identify an efficient design is to use one of any variant of the Fedorov Algorithm which, given a starting design, recursively makes exchange(s) that reduce *D-error* until some convergence criteria is met. This method is susceptible to local minima; it may be necessary to run multiple iterations of the Fedorov Algorithm with different random starting designs to find the most efficient design (Kuhfeld, Huber, & Zwerina, 1996).

**Initial Problem Description:**

Contrary to standard *d-optimal* conjoint designs, the design for patient simulation inherently contains *d-error* as a result of violating the principle of level balance. Since the goal is for the simulated patient universe to map to the actual patient universe, the researcher may need to control for the distribution of levels within each attribute. Additionally, certain attributes and levels may have required interactions (i.e., a patient must be female to be pregnant).

As a toy problem, let us consider a patient universe in which patients are defined by:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | Distribution |
| **Age** | Youth | Adult | Elderly | 25/50/25 |
| **Gender** | Male | Female |  | 50/50 |
| **BMI** | Underweight | Normal | Obese | 25/25/50 |

Expanding out all possibilities into the entire candidate set, this would be 3\*3\*2\*3 = 54 unique patient profiles. Given respondent time is expensive, and high respondent burden decreases quality of results, we seek to reduce time-in-survey by creating a fractional-factorial design of 8 unique patient profiles. As we want to extract as much data from the exercise as possible, the 8-profile fractional-factorial design must be as efficient as possible.

Practically speaking, the number of attributes is limited to no more than 25, each with at most 5 levels due to the complexity of the simulation, limited respondent pool, and limited number of experiments possible per respondent. Thus, at most, the candidate set contains 5^25 (approx. 3x1017) possibilities – and will generally be significantly smaller as not all 25 attributes are used and most contain fewer than 5 levels. However, the worst-case scenario requires approximately 2x1010 gigabytes to merely store the candidate set. The combinatorics problem explains why stochastic search algorithms such as simulated annealing or genetic algorithms are frequently used instead of an exhaustive search against a complete candidate set.

**Model Definition:**

Our goal is to maximize the weighted *d-optimality* of the design matrix, penalized for missing distributions and impossible variable interactions, and subject to the distributions of each attribute’s levels and interactions, where each attribute’s level is represented by a binary variable.

*Objective Function* (Wanida Limmun, 2012):

where N is the number of observations, δ are vectors of relaxation variables, and X is the design matrix:

With decision variables *A, B, G* representing attributes:

Age: ,

i.e. the age group classification for each patient *i*.

Gender: ,

i.e. the gender classification for each patient *i*.

BMI: ,

i.e. the BMI classification for each patient *i*.

For easier constraint formulation, we can use the Dantzig-Wolfe reformulation to rewrite our integer variables where the capital letter represents the binary variable series replacing an integer variable, and the lowercase letter represents the integer set of levels permissible for the given attribute:

*Subject to*:

Age group proportions:

Gender proportions:

BMI group proportions:

Binary constraints:

Interaction slacks: While not specified in the toy problem, it is entirely possible that we may have interactions specified in the design (i.e., men cannot be pregnant). In these cases, the slacks are the count of the impossible interactions. We will penalize these interaction slacks twice because they are more costly to the design than a missed distribution.

**Theory:**

For our discrete-choice design, the information matrix of an *n*-point design is

, where X is an *n* x *p* design matrix.

Using as variance estimator, where xi represents a row. See (Labadi, 2015; Triefenback, 2008) for more details regarding optimality theory.

To perform a sequential switch, a ‘delta function’ is defined that allows a less expensive update to the objective function value through the determinant of the information matrix as well as a variance estimator for the swap (Triefenback, 2008):

In order to update our objective function at each iteration, we use the value 1+Δ as the ratio between the new and old objective function value. This allows us to pick out row swaps at each iteration that maximize the increase in the objective function. However, we must alter this ratio if we want to penalize the slacks on our proportions in our objective function, while picking out rows that both maximize the increase in the objective function minimize this penalty:

Therefore, we can define a new update criterion:

This criterion allows us to figure out the row swap that maximizes our objective function, given that the slacks are penalized. It also allows us to terminate the algorithm as the improvement Δp converges to zero, i.e. the marginal improvement of another swap becomes trivial.

**Modified Fedorov Algorithm:**

We have implemented a modified Fedorov Algorithm (Labadi, 2015; Triefenback, 2008) that considers the slack of distribution constraints (step 4) when performing the iterative state search:

1. Calculate the *candidate set*, the set of all theoretically possible combinations. Because of the possibility of explosive growth with combinatorics, this will not always be feasible.
2. Generate an initial *n*-point design (an arbitrary design with a nonsingular information matrix) that generally obeys distribution constraints
3. Compute M, MT , and the determinant of M
4. Perform an exhaustive search across the design matrix X and the entire candidate set, using the delta function and (xi,xj) to identify the pair of points that maximally improve *D-optimality*, penalizing the slack from the distribution constraints. Perform the swap.  
   If efficiency metric is sufficiently close to optimal (or improvement from variance estimator is sufficiently small), stop. If the iteration limit is reached, stop.
5. Set and return to step 3

**Parallelized Modified Fedorov Algorithm:**

We attempted to then parallelize the modified Fedorov Algorithm as it has ‘embarrassingly parallel’ tasks in the exhaustive search. In step 4 above, it should be possible to calculate (xi,xj) in parallel. Our R implementation fails, seemingly due to a bug in the doParallel or foreach libraries that prevent passing a reference class object to the parallel environment. While the parallel infrastructure may require too much overhead to outperform the non-parallel version for the toy problem (especially given that we must store (xi,xj) for each pair and sort the final list), we believe that as the problem size grows, the effects of parallelizing would show significant runtime improvements.

**Results**

Running the modified Fedorov Algorithm for our toy problem on a top-of-the-line computer requires approximately 25 seconds per 100 iterations. With lambda > 0, It becomes evident that oscillation is present, and the toy problem stops due to reaching the iteration limit, not due to convergence. Additionally, the *d-optimality* of the resulting design (35.3) actually makes the design worse compared to the initial, randomly-generated design (53.1).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initial design**   |  |  |  | | --- | --- | --- | | 1 | 1 | 0 | | 0 | 0 | 2 | | 1 | 0 | 1 | | 2 | 1 | 2 | | 1 | 0 | 0 | | 2 | 1 | 2 | | 0 | 0 | 1 | | 1 | 1 | 1 | | **Final design (lambda=1)**   |  |  |  | | --- | --- | --- | | 2 | 1 | 2 | | 2 | 1 | 2 | | 0 | 0 | 1 | | 1 | 1 | 1 | | 1 | 1 | 1 | | 1 | 0 | 0 | | 1 | 0 | 1 | | 1 | 0 | 0 | |

With lambda = 0, the algorithm converges in 10 iterations over 2 seconds and demonstrates a significant improvement in *d-optimality* (131.0) over the original random design (53.9).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initial design**   |  |  |  | | --- | --- | --- | | 1 | 1 | 0 | | 0 | 0 | 2 | | 1 | 0 | 1 | | 2 | 1 | 2 | | 1 | 0 | 0 | | 2 | 1 | 2 | | 0 | 0 | 1 | | 1 | 1 | 1 | | **Final design (lambda=0)**   |  |  |  | | --- | --- | --- | | 2 | 0 | 2 | | 0 | 1 | 2 | | 2 | 1 | 0 | | 2 | 0 | 2 | | 0 | 1 | 2 | | 2 | 1 | 0 | | 2 | 1 | 0 | | 2 | 0 | 2 | |

Both storage and runtime requirements limit the utility of this algorithm. For large problems, it is infeasible to store the entirety of the candidate set; even if storage were possible, exhaustively iterating across 3x1017 possibilities (as defined in our worst-case scenario) would require more time than any user is likely to be willing to spend.

**Genetic Algorithm**

Given the infeasibilities associated with running a discrete, exhaustive search on large candidate sets, we have also implemented a genetic algorithm to perform a stochastic search (Wanida Limmun, 2012):

1. Generate the initial herd of size *population* – a list of randomly generated designs
2. Calculate the *d-optimality* of each. Preserve some number of elites.
3. Breed random pairs of the non-elite stock (i.e., generate 2 new designs with random 50% from each parent).
4. Randomly mutate cells within the children.
5. Recombine the herd and assess the fitness (*d-optimality*) of each. Cull the poor performers, keeping only *population*.
6. Identify the most fit of the new generation; compare fitness to best from prior generation. If fitness increase is sufficiently large, increment the number of generations and go back to step 2. Otherwise stop. If the maximum number of generations is reached, stop.

Steps 2, 3, 4, and 5 are all ‘embarrassingly parallel’, so it is possible to implement a parallelized genetic algorithm to allow faster traversal of the search space or larger populations and more generations.

**Genetic Algorithm Results**

These algorithms outperform our implementations of the modified Fedorov Algorithms. With lambda=1, our genetic algorithm converges in 22 iterations taking 1.1 seconds to a solution with *d-optimality* 131 (significantly better than the Fedorov Algorithm, and close to the Fedorov Algorithm’s optimal solution with lambda=0).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initial design**   |  |  |  | | --- | --- | --- | | 1 | 1 | 0 | | 0 | 0 | 2 | | 1 | 0 | 1 | | 2 | 1 | 2 | | 1 | 0 | 0 | | 2 | 1 | 2 | | 0 | 0 | 1 | | 1 | 1 | 1 | | **Final design (lambda=1)**   |  |  |  | | --- | --- | --- | | 2 | 0 | 2 | | 2 | 1 | 0 | | 0 | 1 | 2 | | 2 | 0 | 2 | | 2 | 0 | 2 | | 0 | 1 | 2 | | 2 | 1 | 0 | | 0 | 1 | 2 | |

With lambda=0, the genetic algorithm converged in an equivalent 22 iterations over 1.1 seconds to a solution with *d-optimality* 131, matching the Fedorov Algorithm’s performance in a fraction of the time.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initial design**   |  |  |  | | --- | --- | --- | | 1 | 1 | 0 | | 0 | 0 | 2 | | 1 | 0 | 1 | | 2 | 1 | 2 | | 1 | 0 | 0 | | 2 | 1 | 2 | | 0 | 0 | 1 | | 1 | 1 | 1 | | **Final design (lambda=0)**   |  |  |  | | --- | --- | --- | | 0 | 1 | 2 | | 2 | 0 | 2 | | 2 | 1 | 0 | | 2 | 1 | 0 | | 2 | 1 | 0 | | 0 | 1 | 2 | | 2 | 0 | 2 | | 0 | 1 | 2 | |

The parallelized genetic algorithm finds the same solutions but takes approximately 6 times longer due to the overhead of parallelization. With a larger problem, this overhead becomes negligible and parallelization becomes an asset.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Lambda** | **Time (sec)** | **Iterations** | **D-optimality** | **Improvement** |
| Fedorov | 1 | 20.1 | 100 | 35.3 | -17.8 |
| Genetic | 1 | 1.1 | 25 | 130.2 | 77.1 |
| Parallel Genetic | 1 | 6.6 | 24 | 130.2 | 77.1 |
| Fedorov | 0 | 2.0 | 10 | 131.0 | 76.3 |
| Genetic | 0 | 1.3 | 25 | 131.0 | 76.3 |
| Parallel Genetic | 0 | 7.3 | 28 | 131.0 | 76.3 |

**Full Problem Description**

As mentioned previously, the number of attributes in the full problem is generally limited to no more than 25, each with at most 5 levels due to the complexity of the simulation, limited respondent pool, and limited number of experiments possible per respondent. For a Type 2 Diabetes study, we might define patients with 16 attributes, each containing between 2-4 levels:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | Distribution |
| **Age** | Youth | Adult | Elderly |  | 25/50/25 |
| **Gender** | Male | Female |  |  | 50/50 |
| **Race** | Caucasian | African-American | Hispanic | Asian | 40/20/20/20 |
| **BMI** | Underweight | Normal | Obese |  | 25/40/40 |
| **Diabetes** | No | Yes |  |  | 50/50 |
| **Stroke** | No | Yes, Treated |  |  | 75/25 |
| **Heart** | No | Yes, Angina | Yes, MI |  | 50/25/25 |
| **LDL** | Low | Moderate | High |  | 25/50/25 |
| **BP** | Normal | Moderate | High |  | 25/50/25 |
| **A1C** | Normal | Moderate | High | Very High | 25/25/25/25 |
| **Renal** | Normal | Reduced | Impaired |  | 50/25/25 |
| **Creatinine** | Normal | Mild | Moderate |  | 50/25/25 |
| **UACR** | Normal | Microalbuminuria | Albuminuria |  | 33/33/33 |
| **Treatment** | None | ACE Inhibitor | Beta Blocker | CCB | 25/25/25/25 |
| **Heart History** | None | Immediate family | Extended family |  | 50/25/25 |
| **Smoker** | Non-smoker | Past | Current |  | 50/25/25 |

Among the attributes are a Type 2 Diabetes diagnosis (yes or no) and a given patient’s A1C levels (normal, moderate, high, or very high). As T2D diagnosis and A1C are linked, we have the following interaction constraints: (1) Patients with normal A1C *cannot* have T2D, and (2) Patients with very high A1C *must* have T2D.

The candidate set for this problem has 30,233,088 different combinations. As before, we seek to reduce time-in-survey by creating a fractional-factorial design, this time consisting of 50 unique patient profiles. The 50-profile fractional-factorial design must be as efficient as possible.

**Model Definition:**

where N is the number of observations, δ are vectors of slack variables, and X is the design matrix consisting of *Aj* attributes:

With decision variables *A1…16* representing attributes:

|  |  |  |
| --- | --- | --- |
| Age |  |  |
| Gender |  |  |
| Race |  |  |
| BMI |  |  |
| Diabetes |  |  |
| Stroke |  |  |
| Heart |  |  |
| LDL |  |  |
| BP |  |  |
| A1C |  |  |
| Renal |  |  |
| Creatinine |  |  |
| UACR |  |  |
| Treatment |  |  |
| Heart History |  |  |
| Smoker |  |  |

Again using the Dantzig-Wolfe reformulation to rewrite our integer variables, *Z* represents the binary variable series replacing an integer variable, and *k* represents the integer set of levels permissible for the given attribute:

*Subject to*:

Age group proportions:

Gender proportions:

Race group proportions:

BMI group proportions:

Diabetes proportions:

Stroke proportions:

Heart group proportions:

LDL group proportions:

BP group proportions:

A1C group proportions:

Renal group proportions:

Creatine group proportions:

UACR group proportions:

Treatment group proportions:

Heart History group proportions:

Smoker group proportions:

Interaction Constraints:

**Real-World Performance**

The real-world problem is much too large for our implementation of the Fedorov Algorithm. With over 1 trillion swaps to calculate per iteration, the problem would take far too long to run to be useful. However, the larger problem space provides an opportunity to the improvement parallelization brings to our genetic algorithm. With lambda=1, the initial design has *d-optimality* 22.9. With lambda=0, the initial design has *­d-optimality* 59.9. Results are in the table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Lambda** | **Time (sec)** | **Iterations** | **D-optimality** | **Improvement** |
| Genetic | 1 | 19.2 | 44 | 47.1 | 24.2 |
| Parallel Genetic | 1 | 23.4 | 65 | 52.9 | 30.0 |
| Genetic | 0 | 13.4 | 31 | 77.7 | 17.8 |
| Parallel Genetic | 0 | 24.8 | 69 | 80.9 | 21.0 |

We see that the parallel genetic algorithm performed slower than the non-parallelized version but found improved designs. Additionally, these tests were run with population 100 and limited to 1000 generations. Tests with larger populations would likely begin to favor the parallel algorithm. Hypothetically, tests with more generations would also favor the parallel algorithm; however, we came nowhere near the 1000 generation limit in this test.

# Bibliography

Hauser, J., & Rao, V. (2002, September). Conjoint Analysis, Related Modeling, and Applications. In *IN MARKET RESEARCH AND MODELING: PROGRESS AND PROSPECTS: A TRIBUTE.* Kluwer Academic Publishers.

Kuhfeld, W., Huber, J., & Zwerina, K. (1996, September). *A General Method for Constructing Efficient Choice Designs.* Retrieved October 2018, from https://faculty.fuqua.duke.edu/~jch8/bio/Papers/Zwerina%20Kuhfeld%20Huber.pdf

Labadi, L. A. (2015, February). Some Refinements on Fedorov’s Algorithms for Constructing D-optimal Designs. *Brazilian Journal of Probability and Statistics, 29*, 53-70.

Triefenback, F. (2008). *Design of Experiments: The D-Optimal Approach and Its Implementation As a Computer Algorithm.* Umeå University, Department of Computing Science.

Warren F. Kuhfeld. (2001, January). *Multinomial Logit, Discrete Choice Modeling.* Retrieved October 2018, from https://www.stat.auckland.ac.nz/~balemi/Choice.pdf

**Appendix I**

