# Breast cancer epidemiology: calibrating simulations via optimization

Michael C. Ferris
(joint work with Geng Deng, Dennis G. Fryback, Vipat Kuruchittham)

We investigate the use of optimization and data mining techniques for calibrating the input parameters to a discrete event simulation code. In the context of a breast-cancer epidemiology model we show how a hierarchical classifier can accurately predict those parameters that ensure the simulation replicates benchmark data within 95% confidence intervals. We formulate an optimization model that evaluates solutions based on an integer valued score function. The scores are determined from a simulation run (and are therefore subject to stochastic variations), and are expensive to calculate.

The Wisconsin Breast Cancer Epidemiology Simulation uses detailed individual-woman level discrete event simulation of four processes (breast cancer natural history, detection, treatment and non-breast cancer mortality among US women) to replicate breast cancer incidence rates according to the Surveillance, Epidemiology, and End Results (SEER) Program data from 1975 to 2000. Incidence rates are calculated for four different stages of tumor growth, namely in-situ, localized, regional and distant; these correspond to increasing size and/or progression of the disease. Each run involves the simulation of 3 million women, and takes approximately 8 minutes to execute on a 1GHz Pentium machine with 1Gb of RAM.

The four simulated processes overlap in very complex ways, and thus it is very difficult to formulate analytical models of their interactions. However, each of them can be modeled by simulation; these models need to take into account the increase in efficiency of screening processes that has occurred since 1975, the changes in non-screen detection due to increased awareness of the disease and a variety of other changes during that time. The simulations are grounded in mathematical and statistical models that are formulated using a parameterization. For example, the natural history process in the simulation can be modeled using a Gompertzian growth model that is parameterized by a mean and variance that is typically unknown exactly, but for which a range of reasonable values can be estimated. The overall simulation facilitates interaction between the various components, but it is extremely difficult to determine values for the parameters that ensure the simulation replicates known data patterns across the time period studied. In all there are 37 of these parameters, most of which interact with each other and are constrained by linear relationships. Further details can be found in [1, 3].

A score is calculated that measures how well the simulation output replicates an estimate of the incidence curves in each of the four growth stages. Using SEER and Wisconsin Cancer Reporting System (WCRS) data, we generate an envelope that captures the variation in the data that might naturally be expected in a population of the size we simulated. For the 26 years in consideration, the four growth stages give a total of 104 points, each of which is tested to see if it lies in the envelope. The number of points outside the envelope is summed to give the score (0 is ideal). While it could be argued that distance to the envelope might be a better measure, such calculations are scale dependent and were not investigated. Unfortunately, the score function also depends on the "history" of breast cancer incidence and mortality that is generated in the simulation based on a random seed value $\omega$. We will adopt the notation $f_\omega(v)$ where $v$ represents the vector of parameters, and $\omega$ indexes the replication. While we are interested in the distribution (over $\omega$) of $f_\omega(v)$, we will focus here on the problem:

$$\min_v \max_\omega f_\omega(v).$$

The purpose of this study is to determine parameter values $v$ that generate small values for the scoring function. Prior to the work described here, acceptance sampling had been used to fit the parameters. Essentially, the simulation was run tens of thousands of times with randomly chosen inputs to determine a set of good values. With over 450,000 simulations, only 363 were found that had a score no more than 10. That is, for a single replication $\omega$, 363 vectors $v$ had $f_\omega(v) \leq 10$.

Our first goal was to generate many more vectors $v$ with scores no more than 10. To do this, we attempted to use the given scoring function data to generate a classifier that quickly predicts whether a given vector $v$ is in

$$L(\lambda) = \{v | f_\omega(v) \leq \lambda\}, \text{for a fixed replication } \omega.$$

We typically use $\lambda = 5$ to indicate good fit and $\lambda = 10$ for acceptable parameter choices. Our approach is as follows:

- Split the data into a training (90%) and testing (10%) set.
- Given the training set, generate a (hierarchical) classifier that predicts membership of $L(\lambda)$. Validate this classifier on the testing set.
- Generate 100,000 potential values for $v$, uniformly at random.
- For those vectors $v$ that the classifier predicts are in $L(\lambda)$, evaluate $f_\omega(v)$ via simulation.

Since the classifier is cheap to evaluate, this process facilitates a more efficient exploration of the parameter space. Clearly, instead of using a single replication $\omega$, we could instead replace $f_\omega(v)$ by $\max_{\omega\in\Omega} f_\omega(v)$ where $\Omega = \{\omega_1, \ldots, \omega_m\}$ for some $m > 1$. In fact this was carried out. The difficulty is that we require replication data (for our experiments we choose $m = 10$) and we update the definition of $L(\lambda)$ appropriately. However, the process we follow is identical to that outlined here.

In our setting, $v$ has dimension 37. Using expert advice, we only allowed 9 dimensions to change; the other 28 values were fixed to the feasible values that have highest frequency of occurrence over the "positive" samples. For example, if $v_{37}$ can take possible values from $[\phi_1, \phi_2, \ldots, \phi_n]$, then we set the value of $v_{37}$ to be $\arg\min_{i=1}^n \frac{P_i}{W_i}$, where $P_i$ and $W_i$ are the number of appearances of $\phi_i$ in the positive and whole sample set. This is similar to using a naive Bayesian classifier to determine which value has the highest likelihood to be "positive". Our experiments showed this choice of values outperformed even the values that experts deemed appropriate for these 28 values; a posteriori analysis confirmed their superiority.

We generated a hierarchical classifier. The key difficulty in generating a classifier is the fact that we have a vast majority of "negative" data points (i.e. vectors

$v \notin L(\lambda)$). By successively projecting our training data into two dimensional slices, we identified two pairs of planes (meanGamma/varGamma and onsetProp/lag) in which only "negative" data points in our training set were present outside a small band of values. The top level of the classifier labels points outside these bands as "negative". For The remaining points (within the bands, the positive and negative points are intermingled) are classified using the following procedure.

Given a particular training set $A$, a variety of support vector machine classifiers can be generated by solving an optimization problem for values $u$ and $\gamma$, and utilizing the kernel classifier [7]

$$K(v', A')u - \gamma \leq 0$$

to imply that a new point $v$ is "negative", where $K$ is a given kernel function. We used the following kernels: linear, polynomial degree 2, polynomial degree 3, and Gaussian. We also used the C45 decision tree classifier and $k$-nearest neighbor classifier with $k = 5$. All of these classifiers are publicly available [6, 9].

Furthermore, the one-sided sampling approach [4] was used to generate a number of different training sets; the sampling approach iteratively removes "negative" points in a rigorously defined manner, and we stop this process when there are approximately 500 "negative" points remaining (there are around 300 "positive" points in each training set). The resulting classifier is evaluated on the testing set using the measures

$$TP = \frac{\#\text{ correctly classified positives}}{\text{total }\#\text{ of positives}} \text{ and } TN = \frac{\#\text{ correctly classified negatives}}{\text{total }\#\text{ of negatives}}$$

(Note that cross validation accuracy is inappropriate to use in this setting since it can be made large by classifying all points as "negative" due to the imbalanced nature of the data.) Classifiers are discarded if the value of $TP$ is less than 0.9 (typically $TN$ is around 0.4). This value was chosen to guarantee the probability of removing positive points in error is small. We also generate training sets via resampling with replacement.

For a uniform sample of 100,000 potential values of $v$, the naive banding classifier removes all but 8640. Each of the above classifiers was used successively to determine if the point $v$ was "negative" (and hence removed from consideration) and if not $v$ was passed onto the next classifier. This process was repeated until the number of points being removed decreased to zero. At that stage there were 788 points that were hypothesized to be "positive". These 788 points were tested using simulation, and 65% were found to be "positive". This is a significant improvement over the random sampling scheme.

A further sequence of classifiers was determined from a training set generated using the above sampling schemes, but where we adjusted the number of "negative" points in the training set so that the resulting values for $TP$ and $TN$ were approximately 0.6 and 0.7. These additional classifiers have a larger chance of removing "positive" samples in error, but reduce the number of remaining points in our sample much more quickly. For our example set the remaining 788 points was reduced to 220 points. Evaluating these remaining points via simulation, 195 were found to be in $L(10)$. Thus, with very high success rate (89%), our classifier is able to predict values of $v$ that have a low score $f_\omega(v)$.

We employed the classifier technique above to generate a large number of samples in $L(30)$. 1558 samples were selected for further evaluations with replications. In this, we included the original 363 positive samples, another 195 positive samples selected by the classifiers and 1000 negative samples from the original data set. The 1000 negative samples were chosen such that their original scores were less than or equal to 30. For the other 28 parameters of these samples, we fixed them using the 'optimal setting' we calculated. Each parameter sample was evaluated 10 times for the real function $f_{\omega_i}(v), i = 1, 2, \ldots, 10$. Since we used a different setting for the other 28 parameters, the new results were a little different from the first replication, but the scores were generally better. We found that 310 out of the 1558 samples had maximum score less than 10.

The scores varied from replication to replication. We intended to solve the problem of minimizing $g(v)$, where $g(v)$ is some combination of the multiple score values. We either let $g(v) = \max_{i=1}^{n} f_{\omega_i}(v)$ or let $g(v) = \frac{\sum_{i=1}^{n} f_{\omega_i}(v)}{n}$ to serve this purpose. Since the lowest possible score is 0, minimizing $g(v) = \max_{i=1}^{n} f_{\omega_i}(v)$ is equivalent to reducing the upper bound. Thus, minimizing $g(v)$ also controls the distribution of function values. Our results showed that there were only small differences among the different choices of $g(v)$. In subsequent work, we primarily used the objective function $g(v) = \max_{i=1}^{n} f_{\omega_i}(v)$.

Given the 10 replications of each sample, we used the DACE toolbox [5] to fit a kriging model $\hat{g}(v)$ to the data, which we considered as a surrogate function [2] for our objective function $g(x)$. The DACE toolbox is an interpolation tool for data approximation. It takes the input of sites $(v_1, v_2, v_3, \ldots, v_n)$ and function evaluations at the sites $(g(v_1), g(v_2), g(v_3), \ldots, g(v_n))$, and predicts the approximated values at unseen sites. It works as an interpolation tool because the predictions over the original sites exactly match the function evaluations.

We used the Nelder-Mead simplex method (a derivative free method) to optimize the surrogate function and generated several local minimizers based on different trial starting points. It is worth noting that: the first 5 parameters can be regarded as continuous variables. The following 4 parameters: 'aggr4Node', 'aggr5Node', 'lag', 'LMPRegress' can only take on limited values. Actually, we found a total of 108 combinations for these 4 parameters. We implemented a simple branching technique when searching for the minimizers of the surrogate function. The min problem was split into 108 small sub-problems over the 5 continuous variables, where in each sub-problem the 4 parameters are fixed to one setting. The solution of the minimization problem is the best solution of the 108 sub-problems.

These local minimizers were evaluated by simulation. To improve our results further, we updated the surrogate function with the simulation results of the local minimizers and repeated the optimization. The parameter values found using this process outperform all previous values found. Our best parameter generated a score distribution with a mode of 2. Furthermore, expert analysis of various output curves generated from the simulation results with the best set of parameter values confirms the quality of this solution. All the results showed that the surrogate function using the DACE toolbox performs well.

Our next step was to determine an approximation of the level set $L(\lambda)$ of g(v). The application experts term these level sets "islands" since in general they are comprised of a union of disconnected sets. We defined each connected island with 3 key elements: a base point, a grid size vector and a set of representing points. A base point is a sample point that is known to be in the island. The grid size vector defines the mesh size in all directions. The representing points are from the mesh grid that pervades the whole island.

The islands constructed in this way have the following properties:

1. The distance between two islands is defined as the minimum distance between any two points from the representing point sets ($\min_{i,j} d(s_i, s_j)$, where $s_i, s_j$ are from the two point sets respectively).
2. To decide if a point is in the island or not, we can first snap it to the nearest mesh grid point, and see whether this point is one of the representing points or not.
3. Island size is determined by both the grid size vector and the size of the representing point set.
4. The goodness of approximation to a level set $L(\lambda)$ strongly depends on the grid size vector. A smaller grid size is required to guarantee a good approximation.

We followed the steps below to approximate a level set from a known base point and a mesh grid size vector:

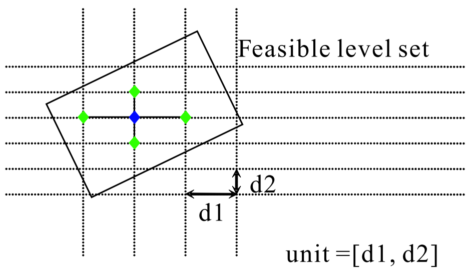

FIGURE 1. Starting from a base point in the island, search for its neighbors and include all feasible neighbors to make a larger set. The neighbors are points that are adjacent in the mesh grid.

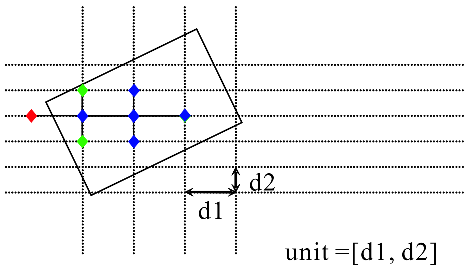

FIGURE 2. Choose a new added point in the point set, and do the similar steps as in 1. Find its neighbors and take the union of the new feasible neighbors with the set while discarding the infeasible neighbors.

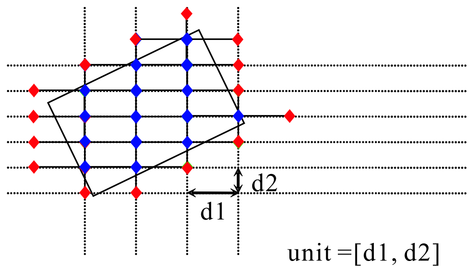

FIGURE 3. Finally, the procedures will result a large set of points, such that the set is bounded by adjacent infeasible neighbors. This is our representing set of an island that approximates the level set.

We have designed an algorithm to generate a series of disconnected islands:

1. Calculate a local minimizer for the surrogate function $\hat{g}(v)$. A good local minimizer is obtained by starting with a collection of initial points.
2. Expand from this point as a root point, and generate an island to approximate a level set $L(\lambda)$, following the steps above.
3. Calculate a new local minimizer that does not belong to any existing island. Repeat step 1 and 2.

In order to get an island of solutions with good quality, we set the level parameter $\lambda = 5$ on the surrogate function $\hat{g}(v)$. Thus, most of the islands have high probability to be within the level set $L(10)$ of the real function $g(v)$. Finally, we created 5 disconnected islands whose representing point size are 1033, 1235, 28, 42, 22, respectively. All of the islands had the same grid size vector: $[0.02, 0.02, 0.004, 0.02, 0.02, 0.01, 0.01, 0.5, 0.5]$.

While our procedure is somewhat ad-hoc, the following conclusions are evident:

- The classifier technique is cheap to use and predicts good parameter values very accurately without performing additional simulations.
- Imbalanced training data has a detrimental effect on classifier behavior. Ensuring the data is balanced in size is crucial before generating classifiers.
- The surrogate function generated by the DACE toolbox was not a smooth function. It has a lot of local minimizers and the global solution is hard to find.

We created 5 disconnect islands of perspective solutions. Future work will enhance biological understanding of the model parameters.

## References

[1] CISNET, http://cisnet.cancer.gov/profiles/.

[2] J.E. Dennis, A. Booker, P. Frank, D. Serafini, V. Torczon and M. Trosset, *A Rigorous Framework for Optimization of Expensive Functions by Surrogates*, Structural Optimization 17(1) (1999), 1–13.

[3] D.G. Fryback, N.K. Stout, M.A. Rosenberg, A. Trentham-Dietz, V. Kuruchittham and P.L. Remington, *The Wisconsin breast cancer epidemiology simulation model*, submitted to Journal of the National Cancer Institute Monographs, Jan. 2005.

[4] M. Kubat and S. Matwin, *Addressing the Curse of Imbalanced Data Sets: One Sided Sampling*, in the Proceedings of the Fourteenth International Conference on Machine Learning (1997), 179–186.

[5] S. N. Lophaven, H. B. Nielsen, J. Sndergaard, *DACE - A Matlab kriging toolbox, Informatics and Mathematical Modelling*, Technical University of Denmark, DTU, (2002).

[6] J. Ma, Y. Zhao and A. Stanley, *OSU SVM classifier Matlab Toolbox*, http://www.ece.osu.edu/∼maj/osu_svm/.

[7] O. L. Mangasarian, *Generalized Support Vector Machines*, in “Advances in Large Margin Classifiers”, A. Smola and P. Bartlett and B. Schölkopf and D. Schuurmans (eds.), MIT Press, Cambridge, MA, (2000), 135–146.

[8] J. A. Nelder and R. Mead, *A Simplex Method for Function Minimization*, Computer Journal 7, (1965), 308–313.

[9] J. Weston, A. Elisseeff, G. Bakir and F. Sinz, *The Spider, Matlab Machine Learning Toolbox*, http://www.kyb.tuebingen.mpg.de/bs/people/spider/.