Optimizing Model Combinations

Jacob Barhak
Austin, TX
Jacob.barak@gmail.com

Aaron Garrett
Jacksonville State University
Jacksonville, Alabama
agarrett@jsu.edu

W. Andrew Pruett
University of Mississippi Medical Center
Jackson, Mississippi
wpruett@umc.edu

ABSTRACT

Consider a situation where multiple models are available to solve a certain modeling problem. Their performance varies on new data since they were developed based on different datasets by different modelers. One modeling approach is to select the most fitting model to some reference data through competition. Another approach, recently popular in challenges, is to combine the models into an ensemble model. Using combination, each model has a potential to influence the combined model and is superior to plain competition when possible since knowledge is accumulated rather than selected.

There are several challenges in combination. One is determining the importance of each model. This challenge increases in difficulty where models are simulated using random methods such as Monte-Carlo. These simulations typically have accuracy related to time by square root order. So efficiency with respect to time and computing power required is a factor.

This paper will discuss theoretical methods for combining models using very simple examples. Some implementations are discussed with reference to code examples. Advantages and drawbacks of the different techniques are discussed to allow modelers choose the preferred combination approach. Implications on accuracy and computing power and use of parallelization techniques are discussed.

A disease modeling application using one of the techniques is briefly discussed.

ABOUT THE AUTHORS

Jacob Barhak specializes in chronic disease modeling with emphasis on using computational technological solutions. The Reference Model for disease progression was self developed by Dr. Barhak as a freelancer. He is the developer of the Micro Simulation Tool (MIST). Dr. Barhak has diverse international background in engineering and computing science. For additional information please visit http://sites.google.com/site/jacobbarhak/

Aaron Garrett is an Assistant Professor in the Department of Mathematical Computing and Information Sciences, Jacksonville State University in Alabama. His interests include evolutionary computation and machine learning. He is the developer of INSPYRED, a software library that includes biologically-inspired computation and encompasses a broad range of algorithms including evolutionary computation, swarm intelligence, and neural networks. For additional information please visit http://mcis.jsu.edu/faculty/agarrett/

W. Andrew Pruett is a mathematician trained in cardiovascular, renal, and endocrinologic physiology with a broad knowledge base in numerical analysis and algorithms. His work at University of Mississippi Medical Center involves instruction and work on a model of calcium homeostasis for HumMod. For additional information visit: https://www.umc.edu/Education/Schools/Medicine/Basic_Science/Physiology_and_Biophysics/Pruett-Drew.aspx

Optimizing Model Combinations

Jacob Barhak
Austin, TX
Jacob.barak@gmail.com

Aaron Garrett
Jacksonville State University
Jacksonville, Alabama
agarrett@jsu.edu

W. Andrew Pruett
University of Mississippi Medical Center
Jackson, Mississippi
wpruett@umc.edu

INTRODUCTION

The incentive for this work came from the disease modeling domain where different disease modelers try to predict disease progression through different models. Each of these models describes phenomena observed in different data sets that are typically disjoint since modeling teams rarely share their data. Even within the same dataset, there are sometimes multiple models describing the data in different points in time (Clarke and Gray 2004, Hayes and Leal 2013). Assuming that each model faithfully described the data it models, models should be similar in performance in theory. However, in practice this is not the case. The Mount Hood challenge where diabetes modelers compare and contrast their models is an example showing how different models report different predictions on similar inputs (Palmer and the Mount Hood 5 Modeling Group 2013, The Mount Hood 4 Modeling Group 2007).

The differences arise from several reasons including: 1) Models were initially constructed using different data sets. 2) Modelers taking different assumptions. 3) Even with similar data modelers rely on different aspects of data such as emphasis on treatments or inclusions of some biomarkers that other teams do not model. 4) Models have different structures such a Markov models or micro-simulation models which may add random noise.

While the obvious scientific goal is to create a model that mechanistically explains multiple datasets simultaneously, this goal may not be attainable for some problems, or it might not be economically feasible to construct or analyze a solution. There may be value in obtaining models that reproduce a system's response to many different protocols without necessarily reflecting all of the underlying biochemistry or physiology.

Considering those factors, decision makers face a difficult decision when choosing a model since they often view a model as a black box and are not aware of all the details modelers are aware of, just like customers are not always familiar with engineering work behind the product they buy in a retail shop. However, unlike retail products, it is possible to merge and unify models together to fit a specific modeling question. In fact this has been practiced in several modeling competitions. The Netflix challenge (Bell and Bennett 2009) is a famous example where models have been merged from several modeling teams to better predict movie preferences. Following that challenge it is quite common to merge modeling teams in other challenges. The Heritage Health Prize challenge (Heritage Health Prize web site n.d.) is another example where the challenge design has included the option of merging modeling teams in anticipation of models to be merged to better predict health outcomes. These are examples of challenges where the combination was of human teams, yet once model code exists, the process of combining models can be automated. Many times these are called Ensemble Models and are quite popular with implementations available for them in Machine learning libraries (Doig 2015). The idea of Ensemble models is not new, those have been actively and successfully used for weather forecast from the 90s (Ensemble forecasting Wiki), yet this idea has not been employed by other modeling fields, and is especially missing in the disease modeling field where models are still compared in competition rather than merged.

A specific sub group of models of interest in this paper are statistical micro-simulation models that generate results using Monte Carlo micro-simulation. These models have recently gained popularity within disease modeling. In such a simulation random numbers are fed as inputs to determine model results and simulations are repeated many times to generate output. Those models take time to compute and include noise in their output. These elements make combinations harder.

For the sake of simplicity this paper will consider models as assumptions coded as mathematical formulas and will follow a very simple example through various ways of combination. The methods to combine models will be studied for strengths and weaknesses. At the end of the paper we will briefly discuss an implementation of the preferred method for disease modeling purposes.

MODEL COMBINATION METHODS

There are several combination methods depending on the data we have. We will separate linear model combination from aggregate model combination and provide different solution methods for both formulations. We will start with the very simple linear case.

Very Simple Linear Model Combination

Assume that a phenomenon observed is generated by a function: r. Our target in modeling is to deduce that function from observations. We typically have several observations: $r_i = r(p_i)$ at input points p_i.
Let us assume that several modelers have provided models that mimic the behavior of $r(p)$, let us denote those possible candidates as: $f_j(p)$. Our goal is to find combination operators composed of parameters $t_j \in T$ such that when applied to $f_j(p)$ we imitate $r(p)$ as best we can. In other words our goal is to find t_j that minimizes the following fitness function:

$$s(t_j, f_j, r_i, p_i) = \sum_{i,j}(t_j \odot f_j(p_i) + e_{ij} - r_i)^2 \qquad (1)$$

Where e_{ij} is an unknown error associated with sampling $r(p)$ or simulating $f_j(p)$. Note that there are no guarantees on the functions being mathematically nice and models are typically more complicated and may not even fit into this simplified notation. However, this simplified notation will help us proceed. And we will simplify this even further by giving a simple example using polynomial functions.
Consider that $r(p) = 0.1 + 0.2p + 0.3p$ generated a set of points r_i for different p_i values sampled randomly from the support domain[0,1]. Three modelers provided us with 3 base functions that model the same phenomenon. $f_1(p)$=1 ; $f_2(p) = p$; $f_3(p) = p^2$. Our goal is to find coefficients t_j such that the linear combination of base functions imitates the observations, i.e. $r(p) = t_1 f_1(p) + t_2 f_2(p) + t_3 f_3(p)$. In this simple example the solution is obvious: $t_1 = 0.1; t_2 = 0.2; t_3 = 0.3$ since those are the multipliers of the base functions that construct $r(p)$.
Note that r and p may well be vector valued, and that the function, inputs, and outputs may be underdetermined or overdetermined.
In the above very simple solution we can resolve this problem using ordinary least squares regression.

Solution by Regression/Least Squares

The simplest and fastest solution for this problem would be solving an over constrained equations set. Here is the formulation:

$$\begin{gathered} t_1 f_1(p_1) + t_2 f_2(p_1) + t_3 f_3(p_1) = r(p_1) \\ t_1 f_1(p_2) + t_2 f_2(p_2) + t_3 f_3(p_2) = r(p_2) \\ \vdots \\ t_1 f_1(p_n) + t_2 f_2(p_n) + t_3 f_3(p_n) = r(p_n) \end{gathered} \qquad (2)$$

On the left side the values $f_j(p_i)$ represents known coefficients of the matrix $\boldsymbol{A}$ - after evaluation. On the right side the values $r(p_i)$ are our ground truth we collect in vector $\boldsymbol{b}$. The unknown coefficient vector $\boldsymbol{t} = [t_1 \quad t_2 \quad t_3]^T$ can be extracted by using the following expression:

$$\boldsymbol{t} = (\boldsymbol{A^T A})^{-1} \boldsymbol{A^T b} \qquad (3)$$

And although there are multiple ways to solve this over-constrained equation set, those methods are synonymous to linear regression and ordinary least squares.
In this specific simple example using regression makes sense and is a generally good solution. Regression has the advantage of having been used extensively and has many implementations.
However, it has drawbacks considering the larger picture:
1) Regression will assume the problem is linear and finds a single solution whereas linearity is not guaranteed in the general case and multiple local minima are possible. Moreover it assumed that the transformation is linear like in our example whereas there can be more complicated transformations may exist.

2) Adding constraints to the problem is not trivial. For example demanding that the coefficients values would be positive and sum to 1 to indicate the sum of base models should stay is not trivial.
However, this problem was too simplified compared to some population modeling problems and just a gateway towards a more complicated problem where we have aggregate solutions.

Aggregate Model Combination

In some cases, and specifically in population modeling such as disease modeling of clinical trials, our observations are aggregate. Meaning that for each point p_i we do not know $r(p_i)$, instead we observe some statistics about those results $g(\{r(p_i)\})$. The aggregation function g is typically some sort of statistical function such as mean, standard deviation or other statistics aggregating the observed individual sightings p_i. Therefore the ground truth we can rely on is the value of the aggregate function, e.g. the published observed outcome of the clinical trial cohort rather than the individual outcomes of each patient.
Assuming that modelers provided us with good models that describe the same phenomena then we can still write a fitness function, although more complicated:

$$s(t_j, f_j, r_i, p_i) = \sum_j \; g((t_j \odot \{f_j(p_i\,) + e_{ij}\}) - \; g(\{r(p_i)\}))^2 \text{ (4)}$$

Let us write this problem using our polynomial example assuming that g is the mean and assuming number of samples is the same we get the following expression:

$$g(\{r(p_i)\} = \sum_i 0.1 + 0.2p_i + 0.3p_i^2 \qquad (5)$$

And therefore the entire fitness function while accumulating noise and accounting for linearity of the transformation would be:

$$s(t_j, f_j, r_i, p_i) = (t_1 \sum_i 1 + t_2 \sum_i p_i + t_3 \sum_i p_i^2 + \; \sum_i e_i - \; \sum_i 0.1 + 0.2p_i + 0.3p_i^2))^2 \qquad (6)$$

Note, however, that we now have several solutions to the problem since multiple combinations of t_j exist that will minimize that expression. In fact any combination of t_j that falls on the hyperplane
$t_1 \sum_i 1 + t_2 \sum_i p_i + t_3 \sum_i p_i^2 + \; \sum_i e_i - \; \sum_i 0.1 + 0.2p_i + 0.3p_i^2 = 0$ will minimize the fitness function.
And the true solution for the problem lies on this hyperplane. If we have additional constraints on t_j we can add them to reduce the solution space, yet any of these solutions will improve our fitness to the observed aggregate data. Recall that the assumption is that each one of the models f_j is considered to represent the phenomena we observe. Therefore a combination of those can be considered as a valid model under certain conditions. For example the average of results of two models seems a reasonable compromise. The formulation in this example is just a weighted average. And our goal here is to find weights that will improve fitness to other known aggregate data. Initially we care less if there are multiple solutions as long as we are getting closer to our absolute truth by improving the fitness. However, if we have a guess we want to end up relatively close to it and therefore an iterative optimization method would be preferred.

Iterative Solution by Gradient Descent:

We can use an iterative gradient method to find a solution for the model combination problem. Those are local optimization methods where the solution starts with a guess and then improved each step. This solution is built upon the assumption that even with the error, the fitness function gradient $s(t_j, f_j, r_i, p_i)$ can be computed for any given t_j. For illustration purposes think of the fitness function s as a topographic map where t_j represent a position on the map. The basic idea of the algorithm is to figure out the steepest slope at each point and take a step towards that direction. This is why the algorithm is sometimes referred to as steepest descent. Ideally we reach the bottom of the valley close to our starting guess.
Formally the gradient descent algorithm is as follows:

1. Start with initial guess t_j
2. Approximate the gradient $\nabla s(t_j, f_j, r_i, p_i) = [s(t_j + h\delta_{jk}, f_j, r_i, p_i) - s(t_j, f_j, r_i, p_i)/h]$ where k represents the dimension of the gradient member and h represents the step size we are taking to approximate the derivative in that dimension and δ_{jk} is Kronecker delta.
3. Take a step of size d towards the negative gradient direction, $t_j \leftarrow \; t_j - d\nabla s(t_j, f_j, r_i, p_i)/\|\nabla s(t_j, f_j, r_i, p_i)\|$
4. Loop back to step 2 unless the stop criteria is reached.

For the gradient we can use the approximated gradient using first order forward derivative approximation will look like:

$$\nabla s(t_j, f_j, r_i, p_i) = \sum_i \frac{1}{h} \begin{bmatrix} s(t_1 + h, t_{2,}, t_3, f_j, r_i) - s(t_1, t_{2,}, t_3, f_j, r_i) \\ s(t_1, t_{2,} + h, t_3, f_j, r_i) - s(t_1, t_{2,}, t_3, f_j, r_i) \\ s(t_1, t_{2,}, t_3 + h, f_j, r_i) - s(t_1, t_{2,}, t_3, f_j, r_i) \end{bmatrix} \quad (7)$$

Note that the gradient components can be computed in parallel. This is especially important when considering models that compute for a long time and use High Performance Computing (HPC) elements.
This algorithm is also suitable for non linear functions. Moreover it allows adding constraints such as bounds for t_j in between algorithm steps as can be seen in the accompanying code (ModelCombiner on GitHub Online).
However, this algorithm is not perfect for the following reasons:

1. The algorithm is slower than the regression counterpart since it requires computation of each dimension during each step.
2. It requires fiddling around with several parameters controlling step size and stopping criteria. These are not always easy to determine and it may slow down convergence. It is especially hard to determine when the fitness function has noise included since the noise may affect the gradient. There are variations and improvements to this basic gradient based algorithm that avoid such issues, yet they typically involve some sort of added computation.
3. The algorithm is greedy and strives for local minima rather than computing the global optimum. Note that this can be seen as an advantage in some cases such as finding the best model combination closest to a certain model.

Note, however, that other algorithms can optimize this aggregate model combination as explained hereafter.

Solution by Evolutionary Computation:

Evolutionary computation (DeJong and Spears 1993, Spears et al. 1993, Fogel 1994, Fogel 2000) has been shown to be a very effective stochastic optimization technique (Bäck et al. 1997, Michalewicz and Fogel 2004). Essentially, an evolutionary computation (EC) attempts to mimic the biological process of evolution to solve a given problem (DeJong 2006).
Evolutionary computations operate on potential solutions to a given problem. These potential solutions are called individuals. The quality of a particular individual is referred to as its fitness, which is used as a measure of survivability (DeJong 2006). Most evolutionary computations maintain a set of individuals (referred to as a population). During each generation, or cycle, of the evolutionary computation, individuals from the population are selected for modification, modified in some way using evolutionary operators (typically some type of recombination and/or mutation) to produce new solutions, and then some set of existing solutions is allowed to continue to the next generation (Fogel 2000). Viewed in this way, evolutionary computation essentially performs a parallel, or beam, search across the landscape defined by the fitness measure (Russell and Norvig 2000, Spears et al. 1993). A beam search is simply a search algorithm that maintains k states, rather than just one state, at each iteration.
EC algorithms are the most flexible and costly solution considering our aggregate model combination problem. This group of algorithms has powerful optimization capabilities that are generally unconstrained by mathematical niceness. The algorithms are heuristic in design and make very little assumptions on the problem at hand. EC includes many types of algorithms including evolutionary algorithms, genetic algorithms, and even simulated annealing. Those algorithms typically require:

1. The definition of a fitness function – in our case we have $s(t_j, f_j, r_i, p_i)$
2. Definition of variation function – similar to a step function yet can combine multiple solutions typically combining some random element. Variation can be defined in mutation where the solution changes itself, and, in some algorithms in the family, crossover can be defined where two solutions are merged.
3. Definition of selection operator – to figure out which of the solutions is considered best
4. Definition of a terminator – to figure out when to stop the algorithm

The Inpyred library (Garrett 2015 software, Garrett 2016 software documentation) implements this family of algorithms and allows users to define the above elements along with other more advanced options.
Those algorithms, can find a global minimum for the problem presented to them and they allow minimizing very complex functions where other methods may fail. More importantly they can potentially accommodate more advanced transformations that combine models. With such algorithms the transformation $\boldsymbol{T}$ can be much more than a linear combination with coefficients t_j. It is possible to choose a combination function that involves all sorts of

building blocks that can be combined together with the candidate models to form the combined function. Such an approach was executed in the past in (Schmidt and Lipson 2009). Model parameters that we denote as p_i as can be potentially used in the combination. For example a disease model that combines models that use the age parameter can potentially use age as a factor in the combining the different disease models using EC.

Despite these potential capabilities, EC has drawbacks:

1. They are much more time consuming than other solutions since they evaluate the functions many more times for many candidate solutions. This is significant if simulation takes a long time.
2. They typically rely on randomness, which makes figuring convergence harder

Considering the simple aggregate model combination problem, these algorithms have little benefit especially since there are multiple solutions. However, these should not be dismissed due to their capability of handling very complex functions and model transformation.

IMPLEMENTATION

The three solution approaches were implemented considering the simple problem and aggregate model combination problem. We provided source code for it so others can reproduce our results. The simple problem was chosen on purpose to be easy to follow. The source code is provided in (Barhak & Garrett 2016).
The regression example solves the very simple linear problem combination while gradient descent and Simulated Annealing (SA) were used to solve the aggregate model combination problem. We chose SA to represent EC techniques since it is one of the simplest in this family.
We added some noise to the system and randomly generated the individual sample points p_i to handle variations.

Results and Analysis:

Here are some outputs provided by the code. Basically we are trying to see how well solutions cope with this noise. Note that we are particularly interested in Monte Carlo noise that is reduced at the inverse square root of number of repetitions of simulation, so noise level is kept constant while running simulations with different number of sample points. We also tested the algorithm for two types of support: 1) fixed – where the set of sample points p_i is fixed and evenly spaced. 2) Random – where the points are fixed amongst all function evaluations yet are randomly spaced in the support interval. Figure 1 shows the sensitivity analysis of the simple model combination problem using regression while Figure 2 shows the sensitivity analysis of the aggregate model combination problem using gradient descent and EC.

The sensitivity analysis clearly shows that Monte Carlo noise is significant in affecting results in all methods as expected. Some minor improvement may be noticed in some cases when population size increases. This may be attributed to the fact that Monte Carlo error diminishes in the inverse square root of number of samples. Random support seems to have a negative effect on gradient descent in some cases. However, the type of support seems to be less influential on the results with EC indicating that a nice distribution of the support vector of p_i does not have a significant advantage over a random sample in EC. This issue with gradient descent can disappear when repeating the same simulation again to reduce negative Monte Carlo error effects. This is also probably why EC optimization seems less sensitive to lower noise levels, simulations are repeated there many times already. If sufficient computing power is available users may wish to consider EC technique rather than the greedy gradient descent. Nevertheless, due to speed and due to the local greedy property we chose gradient descent for our application.

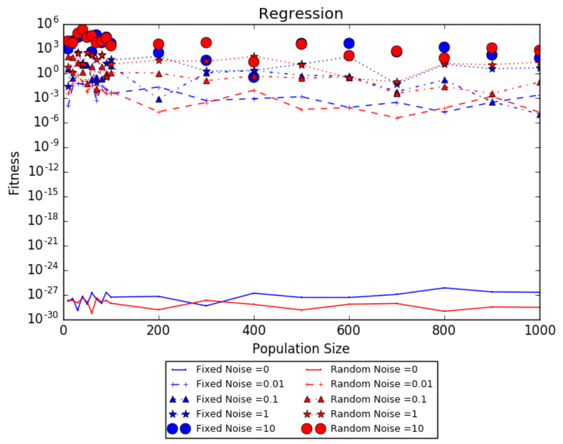

Figure 1. Simple problem solution with regression sensitivity analysis to change in support types (color), noise levels (marker size), and population sizes (x axis).

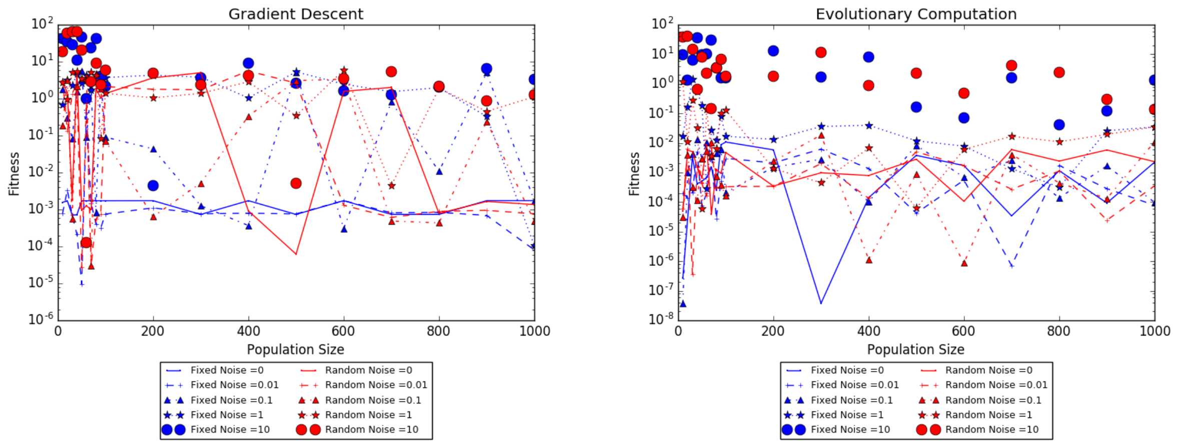

Figure 2. Aggregate problem solution with gradient descent (left) and evolutionary computation (right). Sensitivity analysis to change in support types (color), noise levels (marker size), and population sizes (x axis).

APPLICATION

The motivation for this work started with a disease modeling application in mind. The Reference Model for Disease progression (Barhak2015a, Barhak 2015b, Barhak 2014, Barhak and Leff 2013) is a validation model composed of multiple modeling components and assumptions. It uses HPC to make models compete against published aggregate data available in the literature through clinical trial reports. So far the league of models approach was used. The fitness function was displayed like a score board of a league trying to figure out which model variation leads the league or behaves best for a certain population cohort. However, the number of components in the model grew to the point where computational power required for calculating all possible model combinations became unreasonable.
Therefore in an attempt to reduce computation costs and increase accuracy, it was decided to merge models rather than make them compete by searching continuous parameter space of combinations. The Reference Model is more complicated than the examples provided previously. However, it does share the Monte Carlo component and based on aggregate data. The model combination implementation chosen was gradient descent.
The gradient components are computed all in parallel using the same mechanism that allowed the models to compete as a league in the past using HPC. Nevertheless even with parallelization the computation time is long. It takes several minutes to calculate outcomes of one model variation per population cohort and currently the model consists of 96 population cohorts. Therefore parallelization is very helpful. However, the iterative nature of the optimization

algorithm requires repeating the computation in a manner than is not parallelizable. So assuming access to a large set of computers, each iteration will still take several minutes. However, with the 16 core cluster used for these results computation, each iteration took about a day and a half. The simulations were stopped after 10 iteration which were roughly two weeks on the calendar.
The fitness of the best model and mean of fitness of all models in the iteration was recorded and the results show clear improvement as seen in figure 3. Note that the best model for each iteration was chosen from multiple scenarios calculated in parallel in each gradient descent iteration, each such parallel scenario includes the perturbed model to calculate gradient using finite differences. Since Monte Carlo variation adds noise, it is better to look at the mean model fitness for all models in the iteration as a measure of algorithm improvement. For example, fitness of the best model did not improve in iteration 3, however, the results indicate that there was improvement on average considering perturbation in all dimensions and all model variations calculated in parallel in that iteration.

To reduce the effect of Monte Carlo error, some of the model simulations were repeated 25 times. The models chosen were the best model from each of the 10 iterations and the 10 best overall perturbations from the optimization run. The models were executed in two scenarios of population generation, with parameter correlation and without correlation. The results showed that all models selected were better that the original initial guess model. Fitness scores of these models are shown in figure 4.

The Reference Model is constantly being improved by adding more knowledge and models to it. Future publications will discuss those results in details. Yet this new model combination capability is a significant change in the technology of the model. The Reference Model so far was able to answer the question: what is the best model from a given set of models and assumptions considering a certain query asked about given data. With the implementation of model combination, it is now possible to figure out how to better compose the best model to answer a query about clinical data composed of the building blocks given to it while those building blocks can include models and assumptions.

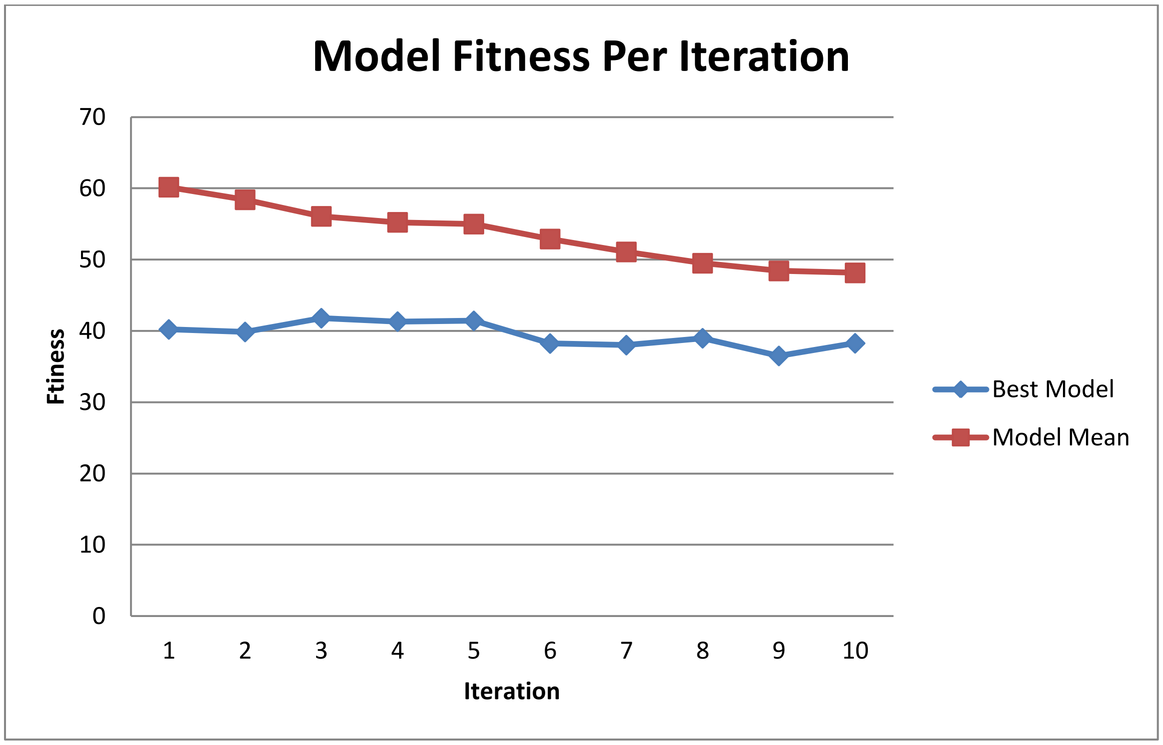

Figure 3. Model fitness improvement per optimization iteration. Best model fitness and mean of model fitness of 92 model variations calculated in the same iterations are shown.

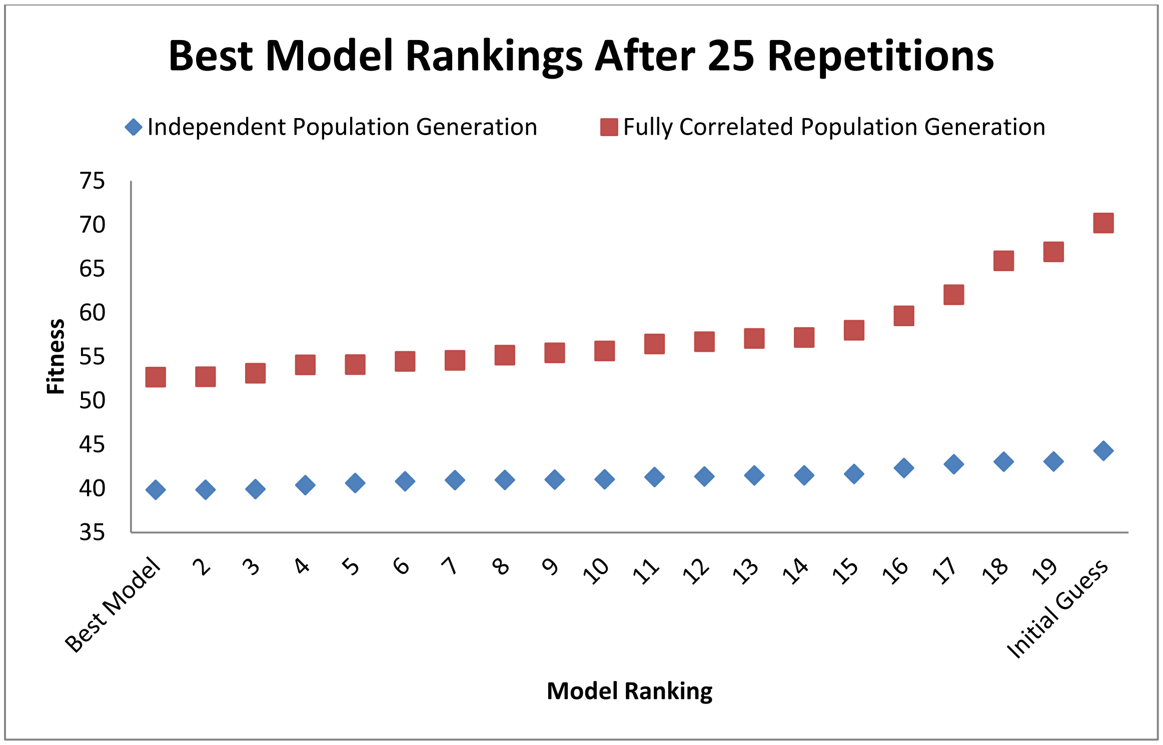

Figure 4. Model fitness for 19 best models selected from all optimizations after repeated 25 times to reduce Monte Carlo error compared to the initial guess model before optimization.

CONCLUSIONS

This paper discusses an approach to combine several models to improve results beyond the capability of each one model individually. It is aimed mainly at disease models where a lot of aggregate information from clinical studies is available yet such model combination technology is not in use. It was shown that both simple greedy gradient descent algorithm and more complex and time consuming evolutionary computation can theoretically improve models by combination.

Gradient descent was used for combining multiple disease models to demonstrate initial feasibility of this technique. However, it is important to note that the results also show that optimization may fail in the aggregate case. In which case, this just means no improvement over a model combination we already are using. Therefore it is suggested to start optimization with an initial at a point where all models have some contribution to the combined model. In that case, no model will be excluded in case of optimization failure. Also, it is beneficial to run many simulations in parallel to try to reduce Monte-Carlo errors. When running many simulations in parallel it is also possible to use competition and combination in parallel to create an improved "assumption engine" that gains the best of all worlds and can handle situations where combining models is not straightforward.

In any case, it is important to recall that with both competition and combination, model fitness depends on: 1) Query/Question asked, 2) Model assumptions, 3) Input data the model relies on. Nevertheless the more of these elements that exist, the better models we will get with a better "assumption engine". Better assumption engines will eventually reduce the modeling problem in some cases to a tradeoff between amount of computing power available and data available.

REPRODUCIBILITY INFORMATION

The results for this paper were calculated on Windows 7 machine using the Model Combiner tool that is available on (Barhak & Garrett 2016). Python 2.7.11 and Anaconda 2.4.1 (64-bit) with Inspyred 1.0 were used.
The Reference Model results were generated using MIST version (0,94,1,0) and model version 33, results are stored in the archive MIST_RefModel_2015_12_20_OPTIMIZE.zip and best repetition in the archive MIST_RefModel_2016_01_28_BEST_REPEAT_TraceBack.zip.

ACKNOWLEDGEMENTS

This paper used the Micro Simulation Tool (MIST) that is based on IEST. The IEST GPL disease modeling framework was initially supported by the Biostatistics and Economic Modeling Core of the MDRTC (P60DK020572) and by the Methods and Measurement Core of the MCDTR (P30DK092926), both funded by the National Institute of Diabetes and Digestive and Kidney Diseases. The modeling framework was initially defined as GPL and was funded by Chronic Disease Modeling for Clinical Research Innovations grant (R21DK075077) from the same institute. The Reference Model for Disease Progression and MIST framework were developed without financial support.

REFERENCES

Bäck, T., U. Hammel, & Schwefel H.-P. (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation 1(1):3–17*. http://dx.doi.org/10.1109/4235.585888
Barhak J. (2014). The Reference Model for Disease Progression – Data Quality Control. Monterey CA. Paper retrieved from: http://dl.acm.org/citation.cfm?id=2685666   Presentation retrieved from: http://sites.google.com/site/jacobbarhak/home/SummerSim2014_Upload_2014_07_06.pptx
Barhak J. (2015a). The Reference Model uses Object Oriented Population Generation. *SummerSim 2015*. Chicago IL, USA. Paper retrieved from: http://dl.acm.org/citation.cfm?id=2874946   Presentation retrieved from: http://sites.google.com/site/jacobbarhak/home/SummerSim2015_Upload_2015_07_26.pptx
Barhak J. (2015b). The Reference Model for Disease Progression and Latest Developments in the MIST, *PyTexas 2015*. College Station, TX. Presentation retrieved from:: http://sites.google.com/site/jacobbarhak/home/PyTexas2015_Upload_2015_09_26.pptx Video retrieved from: https://www.youtube.com/watch?v=htGRRjia-QQ
Barhak J., & Garrett A. ModelCombiner (2016) [Software]. Available from https://github.com/Jacob-Barhak/ModelCombiner
Barhak J., & Leff H.S. (2013). Modeling a Chronic Disease Model and a Mental Health Model Using the Same Modeling Tools, *MODSIM World 2013*. Hampton, VA. Paper retrieved from: http://sites.google.com/site/jacobbarhak/home/MODSIM_World2013_Submitted_04Apr2013.pdf   , Presentation retrieved from: http://sites.google.com/site/jacobbarhak/home/MODSIM_World_Presented_2013_05_2.pptx
Bell R.M., Bennett J., Koren Y., & Volinsky C. (2009). The Million Dollar Programming Prize. *IEEE Spectrum*. Retrieved from http://spectrum.ieee.org/computing/software/the-million-dollar-programming-prize
Clarke P.M., Gray A.M., Briggs A., Farmer A.J., Fenn P., Stevens R.J., . . ., &UK Prospective Diabetes Study (UKDPS) Group (2004). A model to estimate the lifetime health outcomes of patients with type 2 diabetes: the United Kingdom Prospective Diabetes Study (UKPDS) Outcomes Model (UKPDS no. 68). *Diabetologia*, 47(10),1747-59. http://dx.doi.org/10.1007/s00125-004-1527-z
DeJong, K. A. (2006). *Evolutionary Computation: A Unified Approach*. MIT Press.
DeJong, K. A., & Spears W. (1993). On the state of evolutionary computation. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*. 618–623. San Mateo, CA: Morgan Kaufman.
Doig C. (Video file). Beginner's Guide to Machine Learning Competitions. Retrieved from *PyTexas 2016 YouTube Channel*: https://www.youtube.com/watch?v=ys2usamKyus
Ensemble forecasting (n.d.). Retrieved February 29,2016 from Wikipedia the Free Encyclopedia https://en.wikipedia.org/wiki/Ensemble_forecasting
Fogel, D. B. (1994). An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks*, 5(1),3–14. http://dx.doi.org/10.1109/72.265956

Fogel, D. B. (2000). What is evolutionary computation? *IEEE Spectrum*, 37(2), 26–32. *http://dx.doi.org/10.1109/6.819926*

Garrett A. Inspyred 1.0 Documentation: inspyred: Bio-inspired Algorithms in Python. [Software Documentation]. http://pythonhosted.org/inspyred/

Garrett A., Inspyred: Python library for bio-inspired computational intelligence (2015) [Software]. Available from https://github.com/aarongarrett/inspyred

Hayes A.J., Leal J., Gray A.M., Holman R.R., & Clarke P.M. (2013). UKPDS outcomes model 2: a new version of a model to simulate lifetime health outcomes of patients with type 2 diabetes mellitus using data from the 30 year United Kingdom Prospective Diabetes Study: UKPDS 82. *Diabetologia, 56(9), 1925-33*. http://dx.doi.org/10.1007/s00125-013-2940-y

Heritage Health Prize web site (n.d.). Retrieved February 29,2016 from http://www.heritagehealthprize.com/c/hhp

Michalewicz, Z., & Fogel D. B. (2004). *How to Solve It: Modern Heuristics*. Springer.

Palmer A.J., & The Mount Hood 5 Modeling Group (2013). Computer Modeling of Diabetes and Its Complications: A Report on the Fifth Mount Hood Challenge Meeting, *Value in Health*, 16(4), 670-685. http://dx.doi.org/10.1016/j.jval.2013.01.002

Russell, S., & Norvig P. (2002). *Artificial Intelligence*: A Modern Approach. Prentice Hall, 2nd edition.

Schmidt M., & Lipson H. (2009). Distilling free-form natural laws from experimental data, *science* 324 (5923), 81-85. http://dx.doi.org/10.1126/science.1165893

Spears, W. M., DeJong K. A., Bäck T., Fogel D. B., & deGaris H. (1993). An overview of evolutionary computation. In *Proceedings of the 1993 European Conference on Machine Learning*. Springer.

The Mount Hood 4 Modeling Group (2007). Computer Modeling of Diabetes and Its Complications, A report on the Fourth Mount Hood Challenge Meeting. *Diabetes Care*, (30), 1638–1646. http://dx.doi.org/10.2337/dc07-9919