A Simple "Black Box" Method for Obtaining Aggregate Models from More Detailed Models: Applied to Unopposed Military Mobility Modeling

Stephen A. Kukolich
Richard Schaffer
Dan Speicher
Tom Schady
Lockheed Martin Information Systems,
Advanced Simulation Center
37 North Ave., Burlington, MA 01803
781-505-9500
skukolich@lads.is.lmco.com
schaffer@lads.is.lmco.com
dspeiche@lads.is.lmco.com
tschady@lads.is.lmco.com

Keywords:
Analysis, Aggregate, Mobility

ABSTRACT: *We present a straightforward method for obtaining an aggregate model consistent with an existing higher resolution model. The technique involves fitting the results of tests using the higher resolution model. This is a "black box" technique as it is not necessary to look inside the higher resolution model to see how it works. The method only requires access to externally visible outcomes. As such, it is especially valuable when the desired aggregate model must incorporate the complex effects resulting from combinations of simpler detailed models. Such complex effects may be difficult or impossible to capture through more direct analytic techniques.*

The technique represents a straightforward application of standard least squares fit methods. While the user of the technique must still propose the form of the aggregate formulae, the technique determines which parameters in the formulae are most significant. By determining which parts of a model are the most important, this method demonstrates which parts deserve more developer attention. Many modeling situations could benefit from such an approach.

In applying this technique to unopposed military movement modeling, a script was constructed to pick random start and stop goal points on the terrain. By avoiding human bias in the choice of sample problems, we hope to draw general conclusions from the results. Results include probabilistic weights according to estimated likelihood of occurrence in a general simulation. Alternative sample distributions are possible, including those which concentrate more sample points where the effects on results are more extreme, rapidly varying, or difficult to accurately model. Choosing appropriate sample spaces is one of the most difficult tasks, especially when testing the models requires more complex situations.

1. Introduction

JSIMS and other near-future military simulation projects may require the synthetic natural environment to service multi-resolution simulation. This means a heterogeneous mix of platform and aggregate representations forces may exist in the same simulated battle-space simultaneously. It would be very disturbing if, for example, drastically changing the resolution at which the forces are being simulated at were to change where those forces could get to on the simulated terrain. This is just one of many related consistency issues that must be addressed in trying to deal with multi-resolution simulation. Multi-resolution simulation is needed in simulating ground forces because the available computer resources cannot support simulating the required sized forces by representing all of them at the platform (e.g. tank) level.

The simple approach described here attempts to tackle some of these consistency problems by deriving the

aggregate component models statistically from the more detailed models. This should build in some amount of consistency at the component model level, though by itself this certainly does not guarantee consistency at the simulation output level, which might be described by who won the battle. The specific area studied here is unopposed cross-country military movement. However the techniques described here are quite generic, and should easily generalize to other component models and to deriving higher-level aggregate models from less aggregated models.

Section 2 describes all of the theory behind these fit algorithms, including the pseudo-inverse tricks needed to solve potential numerical instabilities arising from linear dependence and from our ranking algorithm.

Section 3 describes an application of these techniques to extract an aggregate mobility model from a platform mobility model, and unit behaviors driving that model.

Section 4 concludes this paper.

2. "Black Box" Fit Algorithms

2.1 Introduction

This section will give the basic mathematics for the linear fit techniques, and the algorithm for ranking fit expressions in order of importance. Towards the end of this section, a matrix pseudo-inverse technique is discussed, along with an extra modification required to give numerical stability in the context of the ranking process.

This section starts by describing an example problem, that of developing an efficient formula describing how fast a military ground unit can travel over simulated terrain. By construction, such a formula must have some statistical consistency with the original model, and is thus a candidate for use as an aggregate model consistent with the original simulation.

Let $route_i$ be the i^{th} route, where the "route" includes everything that should describe a specific experimental condition, including: the spatial path over a database, the unit type and formation(s), and the environmental conditions (e.g. has rained, is foggy etc.). Let T_i be a measured travel time for a given $route_i$.

Next, let us assume that we can compute various terms for a given route, and that we are going to fit for the coefficients for those terms in a total travel time fit formula. Thus the assumed formula is:

$$\hat{T}(route_i; \vec{a}) = \sum_j a_j X_j(route_i)$$

where the sum over j is over all fit terms, $X_j(route_i)$ is any reasonable formula applied to the $route_i$ information, and a_j is an as-yet-unknown constant coefficient which will be computed by the subsequent fit procedure. Thus, each j^{th} fit term is a product of $X_j(route_i)$ and each a_j. The resultant function $\hat{T}(route_i; \vec{a})$ can compute an estimated travel time for any given route. The argument "$\vec{a}$" is displayed here to show explicitly that the $T(\ )$ function depends on the set of a_j fit parameters; this argument might be dropped in subsequent formulae.

Example $X_j(route_i)$ functions on a route are:

```
distance / commanded_speed ,
sum_absolute_turn_angles ,
number_of_bridges_crossed ,
net_slope_upwards*distance ,
net_slope_downwards*distance ,
average_abs_slope*distance .
```

2.2 Basic least squares fits for linear coefficients

This section gives a derivation of the formulae for obtaining the unknown fit coefficients, $\vec{a}$ above, from some experimental data. An equivalent derivation can be found in all good experimental analysis books, e.g. Bevington [1], and is only given here because it is short, and helps to make this paper more complete as an isolated document.

After running many experiments, the experimental data can be put in the form of "data points", where each single "data point" is a set of values like:

$$T_i, X_1(route_i), X_2(route_i) \ldots .$$

These data points could easily be stored in a data file, as one line of ASCII numbers per data point. This allows a computer program to effect the fit process described in this paper, independent of where the data comes from, or what the data means.

To begin the derivation of the fit formulae, a c^2 error measure is defined. This c^2 error measure is effectively a distance between the set of data points $\{T_i\}$ and the function $T(\ )$, which the desired $T(\ )$ function should minimize:

$$c^2 \equiv \sum_i \left(\frac{T_i - T(route_i; a_j)}{s_i} \right)^2 .$$

The $1/s_i^2$ values are effectively weights, W_i, that include a pre-experiment estimate of the relative statistical errors in each of the T_i data values. If the deviations arise from a sum of apparently random and frequent fluctuations (possibly due to our simplified model not correctly modeling the whole system), a good guess for the uncertainties in the T_i values might be $s_i^2 = T_i$. If deviations tend to be consistently high or low for a given route, then $s_i = T_i$ would be a better approximation.

The "method of maximum likelihood", says that the best fitting, i.e. most probable, set of a_j coefficients are those which minimize the c^2. Elementary calculus says that if c^2 is at a minimum with respect to variations in an a_k parameter, then the partial derivative of c^2 with respect to that a_k parameter must equal zero. (As a notational convenience, we use the index *k* instead of index *j* for the remainder of the section - it indexes the same set of parameters.)

$$0 = \partial c^2 / \partial a_k \Rightarrow$$

$$0 = \sum_i \left(\frac{1}{s_i^2} \right) \left(T_i - T(route_i; a_j) \right) \frac{\partial T(route_i; a_j)}{\partial a_k}$$

where we have used the chain rule and the definition of $T(\ )$ to obtain:

$$\frac{\partial T(route_i; a_j)}{\partial a_k} = \frac{\partial T(route_i; a_j)}{\partial a_k}$$

Because of the assumed linear dependence of *T* on the unknown a_j parameters, these *N* equations formed by taking partial derivatives with respect to each particular a_k, will lead to *N* linear equations in the *N* unknown a_j parameters:

$$\sum_j A_{kj} a_j = b_k ,$$

where

$$A_{kj} = \sum_i \left(1/s_i^2 \right) * X_k(route_i) * X_j(route_i),$$

$$b_k = \sum_i \left(1/s_i^2 \right) * X_k(route_i) * T_i,$$

and where A_{kj} are the elements of a square matrix *A*, and b_k are the elements of a column vector *b*, and both are easily computed from the input data points.

Inverting the *A* matrix can yield the desired coefficients:

$$a_j = \sum_k \left(A^{-1} \right)_{jk} b_k .$$

These conclude the derivation of the a_j formulae.

A few extra notes are in order concerning the inverse matrix, A^{-1}.

First, the matrix *A* might not be directly invertible using floating point arithmetic, e.g. by Gaussian elimination, because it might have a (nearly) zero determinant. A useable inverse matrix can still be found in the form of a pseudo inverse, which is described in a later section.

Also, the diagonal elements of A^{-1} matrix gives estimates in the uncertainties of how well the a_j have been determined from the given data,

$$s_{a_j} = \sqrt{\left(A^{-1} \right)_{jj}} .$$

These uncertainties are a result of having imperfect knowledge of the assumed underlying statistical distribution, because we only have a finite number of random data samples from that statistical distribution. These uncertainties would go to zero, given an infinite number of samples.

However, nearly zero uncertainties does not imply zero error, because a proper discussion of errors would necessarily include a discussion of how well the simplified model (and statistical distribution assumptions made in the fit process) describe the

original system. Such an extensive discussion will not be started here.

This section has given a derivation of the formulae for obtaining the unknown a_j coefficients from a set of data points.

2.3 Search to rank importance of fit terms

The technique works as follows: the user supplies the guesses for expressions to use in the fit, i.e. $\{X_j\}$, and the fit tool outputs the fit coefficients $\{a_j\}$, and describes which terms were most important in fitting to the data. Here is described a simple technique to rank the X_j terms in descending order of importance.

This technique begins with an empty ordered "kept" set of X_j terms (or correspondingly a_j coefficients). At each major iteration, it will search for the best remaining X_j (or a_j) to move from the "unused" set to the "kept" set of terms (or coefficients). The best X_j to move is the X_j that will give the greatest reduction in the c^2 error measure.

"Keeping" an X_j term is equivalent to allowing its corresponding a_j coefficient to be non-zero.

For example, suppose there are three fit terms X_1, X_2, X_3, which when combined with *T*, label all of the columns in a given data set. Furthermore, the 3 by 3 matrix *A* and length 3 vector b are formed from this data set as described above.

The first major iteration would look for the term which does the best, when it is the only term used in the fit. So for each term, the appropriate submatrix of *A* is kept, and the appropriate subvector of b is kept, and the corresponding $Aa = b$ equation is solved (using pseudo inverse methods described later) for the corresponding *a* coefficient, e.g. $A_{22}a_2 = b_2$.

To evaluate which fit is best, we need to be able to recompute c^2 for each of the subsets of the *a* vector being tried. The formula for c^2 can be recast as:

$$c^2 = a^{\mathrm{T}}\mathrm{A}a - 2a^{\mathrm{T}}b + c,$$

where a is the column vector containing the a_j coefficients, and where matrix *A* and vector b are defined above, and where scalar *c* is given by:

$$c = \sum_i \left(1/s_i^2\right) * \mathrm{T}_i * \mathrm{T}_i .$$

When only some of the a_j coefficients are currently in the "kept" category, this c^2 formula can still be used by setting the other "unkept" coefficients to zero. Alternatively, one can substitute the appropriate submatrix of *A* for the whole matrix *A*, and similarly for b into the above c^2 formula. Note that in either case we never have to recompute an *A* submatrix or b subvector from the original data points, just use portions of the whole *A* matrix and whole b vector, which are computed just once. The difference in speed obtained by not recomputing from the original data could be important if there are many thousands of data points, and many X_j terms being tried.

Also, it is usually beneficial to work with a "reduced" c^2

$$c_r^2 = c^2 / N_{\mathrm{data}},$$

where N_{data}, is the number of data points in the given data set, indexed by *i* in the above formulae. The reason this is frequently better is that if one were to have multiple sets of data, each of a different length, the reduced c_r^2 values still may be comparable between the data sets, where c^2 would not be.

Getting back to the example, we might find that

$$c_{r,02}^2 = a_{22} * A_{22} * a_{22} - 2 * a_{22} * b_{22} + c,$$

$$\chi_{r,02}^2 = 10.2,\ \chi_{r,01}^2 = 15.6,\ \chi_{r,03}^2 = 23.1$$

This would mean that X_2 is the best next term to move into the permanently "kept" set.

The second major iteration, on separate steps, would try adding X_1, then X_3 to the "kept" set, which is currently $\{a_2\}$, to see which was best. "Trying" X_3 involves solving $Aa = b$ for *a*, and finding c_r^2, using the currently "kept" set of $\{a_2, a_3\}$. Explicitly, the

submatrices in $Aa = b$ when $\{a_2, a_3\}$ is being used would look like:

$$\begin{pmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} b_2 \\ b_3 \end{pmatrix}.$$

Again, either X_1 or X_3 would be added to the permanently "kept" set, based on which try gave the lowest c_r^2.

The final results include the final a_j coefficients, a ranking of the X_j terms given in descending order of importance, and the descending values of c_r^2 resulting from each major iteration. The degree of reduction in c_r^2 with each major iteration tells how important the newly added X_j term is. If many terms are given to try, it is highly probably that the last few terms will contribute very little to reducing the error measure.

In theory, c_r^2 should be reduced with the addition of each newly kept term. The X_j terms are all vectors in the N_{data}-dimensional space, and in theory the *T*-vector resulting from each fit is the nearest projected point of the original T_i vector onto the subspace spanned by the kept X_j vectors. This vector space is measured using W_i weight values as a distance metric. Augmenting the subspace with another X_j vector cannot increase the distance between the original point and its nearest projection onto the subspace. Therefore, c_r^2 should never increase with the addition of each new X_j term to the kept set. If c_r^2 does increase with any major step it means that there are some numerical round-off problems involved. The following pseudo-inverse section corrects all of the numerical difficulties that we came across.

Various alternatives to above described ranking technique have been considered, including using linear correlation coefficients, and directly investigating the pseudo-inverse eigenvectors described below. However, we have found the above ranking technique to be the most useful.

In addition to solving for the a_j coefficients which give a best fit, we have given a technique for ranking the X_j terms in descending order of importance to the fit process.

2.4 A stable pseudo-inverse of *A* is required

Linear dependence and problems associated with floating point numerical round-off are serious issues in getting reasonable results from the above fit process. Therefore, in addition to using double precision arithmetic, a pseudo-inverse technique with a carefully controlled eigenvalue threshold $l_{\text{threshold}}$ is required.

When computing A^{-1} (the matrix inverse of *A*), it is a good idea to use a Singular Value Decomposition (SVD) inverse technique instead of Gaussian elimination. SVD will give a reasonable inverse even when *A* is singular - which could happen if there is an accidental linear dependence between the X_j terms. Instead of using a standard SVD algorithm (see "Numerical Recipes" [2]) on matrix *A*, we take advantage of the fact that *A* is real and symmetric, and use eigenvector-eigenvalue decomposition methods. The latter methods work equally well on real, symmetric matrices, and use more widely understood mathematics. So, to compute an SVD-based pseudo-inverse of matrix *A*, eigen-decomposition is applied to *A* (implemented using routines from "Numerical Recipes" converted to double precision):

$$A = S\Lambda S^{\mathrm{T}}$$

where Λ has diagonal entries that are the eigenvalues of *A*, $\Lambda_{ll} = l_l$, and where the off-diagonal entries of Λ are all zero, and where *S* is the matrix whose columns are the corresponding eigenvectors of *A*.

Next let us define the pseudo-inverse of *A* via a pseudo-inverse function applied to the eigenvalues:

$$\text{pseudo\_inverse}(\Lambda_{ll}) =$$

$$\begin{cases} 0 & |\Lambda_{ll}| < l_{\text{threshold}} \\ 1/\Lambda_{ll} & \text{otherwise} \end{cases}$$

Then the pseudo inverse of *A*, to be used in place of A^{-1} is:

$$\text{pseudo\_inverse}(A)_{jk} =$$

$$\sum_l S_{jl} * \text{pseudo\_inverse}(\Lambda_{ll}) * S_{kl}.$$

If some inverse eigenvalues are set to zero by the pseudo-inverse method, it means that some X_j terms

are linearly dependent on others (at least for the given set of experimental data points), and can probably be discarded.

A reasonable way to compute the threshold value $l_{\text{threshold}}$ is to multiply a small fraction (e.g. 10^{-9}) by the sum of the absolute values of the eigenvalues.

In the X_j ranking technique described above, it is vital that $l_{\text{threshold}}$ be computed just once from all of the eigenvalues of full matrix *A*. If it were to be recomputed from the current submatrix of *A* each time, anomalies in the form of increasing c_r^2 values can result.

The method of computing a pseudo-inverse matrix described above is very useful in the above fit techniques, for solving matrix equations of the form $Aa = b$ for the column vector of unknowns

$$a = \text{pseudo\_inverse}(A)b \ .$$

2.5 Summary

This section has described a straightforward linear-coefficient least squares technique which, when coupled with experimental outcomes, can be applied to developing a simpler model from a more detailed model. The user of this technique is responsible for guessing fit expressions to try, and providing experimental data in the proper form. However, the user need not worry about linear dependence amongst the guessed expressions, nor about which expressions are most likely, because the algorithms given here will reliably rank the expressions from most to least important. Results also detail measures of that importance in the form of improvements to c_r^2, and the desired fit coefficients.

3. Mobility Experiments

3.1 Experiment Design

This section explains the setup of our experiment to create an aggregate model using the black box technique. The scenario used was the unopposed movement of a platoon of tanks.

The high-resolution model that we selected was the SIMNET SAF tracked mobility model. The CCTT SAF mobility model might have been a more popular choice. However, at the start of these mobility experiments, the fidelity of the implementation of the CCTT mobility model in ModSAF had not been confirmed. The major drawback of the SIMNET tracked mobility model is the assumption that the speed going up slopes degrades linearly with the slope angle. Other than that, the SIMNET tracked mobility model does include the typical effects of speed and acceleration limits, which depend on soil type.

We decided to create an aggregate model that would predict the amount of time it would take a unit to travel a given distance. To do a linear fit to create the aggregate model, we needed to generate a large number of data points. Each data point consisted of various terms that described the physical attributes of the route the unit took and the time it took the unit to complete its trip.

To generate these data points, we performed the following steps. First, we ran a TCL script that had previously been tied into ModSAF. This script repeatedly created a unit in ModSAF at a random location, tasked the unit to move to another random location, and recorded the start point, the end point, and the travel time. We then ran a program called "rdata" over the data we had collected. The rdata program took each starting and ending point and calculated the values of the terms for that route. It worked by first using libroutemap, taken from ModSAF, to plot a course around all of the major obstacles in the route. It then examined the route segment by segment and used this information to calculate the values of the terms for that route.

At this point, we had a datafile with one line for each data point, where each data point corresponded to a single tasking of a unit by the TCL script. Each line of the datafile was of the following format:

time term1 term2 term3 term4 ...

Where the "time" was the time taken by the unit to travel the requested distance, as recorded by the TCL script, and each term was the value calculated for that term by the rdata program. The values of these terms are the X_j variables in the formula shown in section 2.1. This format was appropriate for feeding into the linear fitting program.

We have shown how we created our initial data and then modified that data in preparation for doing a linear fit.

3.2 Choice of terms

This section lists the terms that we chose to use in the linear fit. It defines what each term is, and provides some insight into how and why we chose some of the terms we did, and problems that we encountered.

The ideas for the terms we chose to have the rdata program calculate for each data point came, in part, from previous experiments. In these previous experiments, we observed what aspects of the terrain and route had visible affects on a unit's travel time. Using these results, we arrived at a large set of potentially useful terms. Some preliminary runs of the linear fitting program enabled us to discard some of these terms, since their values correlated very poorly with the travel time. The following list shows the final set of terms that we determined were useful.

TERM	DEFINITION
num_turns	the number of turns in the route
tot_angles	the sum of the absolute value of the angles of all the turns in the route
ave_angles	the average angle of a turn
angles_0_45	the number of turns that were less then 45 degrees
angles_45_plus	the number of turns that were greater than or equal to 45 degrees
dist_3d	the length of the route
dist_2d	the flat length of the route; the Z-value of each point was ignored in this calculation
est_time_3d	estimated travel time, assuming a 3D route (dist_3d/commanded_speed)
est_time_2d	estimated travel time, assuming a flat route (dist_2d/commanded_speed)
steepest	the value of the steepest slope in the route
total_rise	the sum of the positive vertical changes in the route, in meters
ave_slope	the average slope, but only of the parts of the route with an uphill slope
sd_slope	the standard deviation of the slope on the positively sloped sections of the route
ave_slope_full	the average positive slope over the whole length of the route (total_rise/dist_2d)
sd_slope_full	the standard deviation of the positive slopes measured over the entire route
dist_0_15_2d	number of meters of the route that are positively sloped between 0 and 15 degrees, measured on a flat plane
dist_15_30_2d	number of meters of the route that are positively sloped between 15 and 30 degrees, measured on a flat plane
dist_30plus_2d	number of meters of the route that are positively sloped over 30 degrees, measured on a flat plane
dist_0_15_3d	number of meters of the route that are positively sloped between 0 and 15 degrees, measured in three dimensions
dist_15_30_3d	number of meters of the route that are positively sloped between 15 and 30 degrees, measured in three dimensions
dist_30plus_3d	number of meters of the route that are positively sloped over 30 degrees, measured in three dimensions
slope_dist_2d	ave_slope * dist_2d
slope_dist_3d	ave_slope * dist_3d
sd_slope_dist_2d	sd_slope * dist_2d
sd_slope_dist_3d	sd_slope * dist_3d
slope_sq_dist_2d	ave_slope * ave_slope * dist_2d
slope_sq_dist_3d	ave_slope * ave_slope * dist_3d

We had two-dimensional and three-dimensional versions of most of the terms because it was not immediately obvious to us which version would yield the best results. The last six terms are mathematical combinations of previous terms. We did this as an experiment, since by itself the average slope and the standard deviation of the slope do not directly correspond to the travel time. To illustrate, consider a route with a given average slope. Adding several hundred meters of flat terrain to the end of the route would not change the average slope, since our way of calculating the average slope only includes the positively sloped portions of the route. However, adding this extra flat difference would increase the travel time. Therefore, by multiplying in the distance

of the route, we hoped to improve the correlation with the travel time.

The choice of these terms took advantage of knowing that the ModSAF tracked model does not cause the speed to increase above the requested speed when travelling downhill. Therefore, downhill travel is treated as flat; our choice of terms reflects this, as we did not use terms that measured the downhill slopes.

Our next problem was that many of these terms are linearly dependent with respect to each other. This caused difficulties in determining which terms were needed to produce a good aggregate model, as the contribution of a highly correlated term may already be incorporated by another linearly dependent term. For example, the terms dist_2d and dist_3d are obviously very close to the same value, but it is not obvious which of these two terms is more important in calculating an accurate travel time. This problem was solved by having the linear fit program tolerate linear dependencies, and rank the terms in usefulness according to which term most reduced the chi-squared value at each part of the linear fit calculation.

We have explained and defined the terms we chose for the linear fitting. The difficulties we had in deciding which terms were better than others, caused us to include all the terms that seemed interesting. This led to the problem of linear dependence between terms, which we solved by making the linear fitting program more robust.

3.3 Results

This section explains how we made the aggregate model and gives the data that demonstrates the accuracy and robustness of the aggregate model.

Three sets of data were used in the creation of the aggregate model. The base set of data contained 1500 data points. These data points consisted of about 120 distinct start and end points. The multiple copies of each point were either the same route with a different speed or simple repeats. The various data points in this data set covered the full range of possible commanded speeds for the unit being tasked. The other two sets of data were comparison data sets. The first contained 300 data points, representing approximately 60 distinct start and end points. The second contained 150 data points, which consisted of approximately 30 distinct start and end points. The start and end points in the comparison data sets were completely different from the base data set.

We first ran the linear fitting program over the base data set. One of the outputs of this program is a coefficient for each term. These coefficients are the a_j terms in the formula from section 2.1. The following table shows another of the outputs of the program; the ranking of all the terms used in order of decreasing contribution to improving the c^2 value.

TERM	c^2	Δc^2
dist_3d	208.4903	+0.000000
dist_15_30_3d	163.5644	-44.925894
est_time_3d	150.9655	-12.598905
angles_0_45	140.4305	-10.535073
dist_2d	138.0646	-2.365841
est_time_2d	137.1702	-0.894415
dist_30plus_3d	136.2898	-0.880423
ave_angles	135.5885	-0.701245
angles_45_plus	133.9754	-1.613157
tot_angles	132.5012	-1.474155
ave_slope	132.2078	-0.293429
total_rise	131.8465	-0.361311
dist_0_15_3d	131.3915	-0.455013
ave_slope_full	130.9237	-0.467786
dist_0_15_2d	130.5069	-0.416741
dist_15_30_2d	130.1336	-0.373325
sd_slope	129.8662	-0.267457
slope_sq_dist_2d	129.1605	-0.705623
sd_slope_dist_2d	128.8372	-0.323383
slope_sq_dist_3d	128.4313	-0.405890
sd_slope_dist_3d	128.1129	-0.318392
steepest	128.0116	-0.101296
dist_30plus_2d	127.9714	-0.040193
slope_dist_2d	127.9468	-0.024575
slope_dist_3d	127.5861	-0.360710
sd_slope_full	127.4989	-0.087171
num_turns	127.4970	-0.001937

Note that only the first few terms actually make a noticeable contribution to improving the c^2 value.

These results, while interesting, do not actually give any sort of indication as to whether we have actually created a linear equation which could perform as a decent aggregate model. To determine how good our model will be, a checking program was run. This checking program reads each line of our data file, multiplies each coefficient generated by the linear fit program by the corresponding term's value, and adds all of these products together. As shown in section 2.1, this sum is the calculated travel time for this route.

The checking program compares the calculated time to the known time that was originally recorded back when we were collecting data. Running this checking program over our base data set gave the following results.

Average error was 0.242512
Worst error was 0.783882
Standard Deviation was 0.161821

The errors were calculated by dividing the difference between the calculated time and the actual time by the actual time. Therefore, on average, the aggregate model gives an answer that is only 24% off, with the worst error for the entire data set of 78%. However, since the aggregate model was derived from this set of data, we expect the error to be fairly low. Therefore, we also ran the checking program on the two comparison data sets. Since the positions in the comparison data sets are completely unrelated to the positions in the base data set, this is a good way to check the accuracy of the aggregate model.

Running the checking program over the first set of comparison data gave the following results.

Average error was 0.339922
Worst error was 0.641584
Standard Deviation was 0.113844

The second set of comparison data gave the following results.

Average error was 0.308581
Worst error was 0.655578
Standard Deviation was 0.191049

As expected, these errors were slightly worse than for the base data set, but they were still fairly small.

Once we had the terms ranked by contribution to chi-squared value, we reran the rdata program using only the terms that made the biggest contribution: dist_3d, dist_15_30_3d, est_time_3d, and angles_0_45. Running the linear fitting program on the base data set, followed by the checking program on each data set gave the following results for the base data set, the first comparison data set, and the second comparison data set, respectively.

Average error was 0.26152
Worst error was 0.773478
Standard Deviation was 0.158833

Average error was 0.335249
Worst error was 0.638632
Standard Deviation was 0.122243

Average error was 0.315842
Worst error was 0.764344
Standard Deviation was 0.177276

These results were all very close to our results when using all of the terms. However, we would expect that using a reduced set of terms will actually improve our aggregate model, since the non-contributing terms do not improve the accuracy of our results and are a potential source of noise.

To verify that the choice of terms was actually important, we reran all of the programs using the three worst terms: num_turns, sd_slope_full, and slope_dist_3d. This gave the following results.

Average error was 0.425289
Worst error was 1.00
Standard Deviation was 0.269396

Average error was 0.503153
Worst error was 0.984334
Standard Deviation was 0.254554

Average error was 0.482443
Worst error was 0.979472
Standard Deviation was 0.275635

This showed us that ordering the terms based on contribution to the c^2 value does provide a good method for picking which terms to use in our aggregate model.

The final task in verifying the accuracy of the aggregate model was to check our weighting term. As mentioned in section 2.2, we had to make an educated guess as to what value to use for s. The two values we tried were $s = \mathrm{T}_i$ and $s = \sqrt{\mathrm{T}_i}$. We tried running the fitting program with s set to both of these values, and got consistently better results by setting $s = \mathrm{T}_i$. To further confirm that this choice is the better one, we ran a program to compare the absolute values of the error between our known time and the calculated time to the actual time for each data point. The program found that the proportion between the two was nearly one. This means that the magnitude of the error varies very closely to size of the actual time, implying that T_i is a good value to use for s, when relatively weighting the data points.

We have succeeded in creating an aggregate model of the SIMNET mobility model that gives us around 30% error and requires only four terms calculated from the physical attributes of the route.

4. Summary and Future Work

We have presented the derivation of the black box technique for creating an aggregate model from a generic high fidelity model. We have also provided a specific sample use of this technique by creating an aggregate model from the SIMNET SAF tracked mobility model.

The aggregate model travel times deviated from the platform travel times by an average of 30%. It was also found that this technique is very effective at identifying the important terms.

We expect that the black box fit technique can easily be applied to other types of models. The user will need to create a tool for computing the terms for the linear fit, analogous to the rdata program we created; after this, the process should be very easy. The advantage of the black box technique is that no information is required about the underlying physics of the high fidelity model, making the creation of an aggregate model for a similar high fidelity model extremely simple.

It is likely that further improvements could be made to the aggregate model that we have demonstrated in this paper. Experimental data over longer route lengths would be valuable for developing mobility models for more aggregated units. More experimental data would also be useful in determining the linear fit terms more accurately.

There are other possible applications for this technique. One application could be to apply these fit techniques to determine the relative importance of different terrain features and resolutions to mobility models.

5. Acknowledgement

The research reported in this paper was sponsored by the Defense Advanced Research Projects Agency and the US Army Simulation, Training and Instrumentation Command. The work was performed as part of the Advanced Simulation Technology Thrust program under contract N61339-97-C-0033.

References

1. P. R. Bevington: Data Reduction and Error Analysis for the physical sciences, McGraw-Hill, New York, 1969.

2. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling: Numerical Recipes in C, Cambridge Univ., New York, 1988.