THE EXTENDSIM OPTIMIZER

by Bob Diamond
President, Imagine That Inc.

Automated searching techniques can be used to find the best set of parameters for a simulation model. The term most often used to describe this kind of automated searching is optimization. Optimizers have the reputation of being hard to use and are often avoided when preparing model results. This paper shows both the use of the ExtendSim Evolutionary Optimizer and the design path that was taken to create it. Design choices were made with the object of reducing the learning and implementation barriers to using optimization within simulation models.

Brief Description of ExtendSim

The ExtendSim simulation application consists of a library-based simulation engine, a built-in development environment, and libraries of prebuilt blocks. Blocks are modeling constructs that contain a representative icon, system behavior, animation, a dialog interface, and online help. Simulation models are created by dropping blocks onto a worksheet, connecting them, and then entering model data. New libraries of blocks can be created by using the built-in development environment. Since libraries are open source, the user can easily create their own libraries by modifying existing blocks.

The communication of ExtendSim with blocks, and blocks with each other, consists of a repartee of messages. Some examples:

- When a user changes a block's dialog data, the block is messaged that its data has changed value. That block may, in turn, send messages to other collaborating blocks, communicating the fact that data has changed.
- In discrete event models, a queue will try to pull in available items. It will proceed to converse with connected blocks to get any items available.
- Blocks can act as agents by building, changing, and running models, and by controlling external applications.
- External applications can act as agents, controlling ExtendSim in the same way as blocks.

This message-sending architecture is extensible and is the basis for all computational and user interface development in ExtendSim, enabling blocks to initiate global interapplication consequences, even over networks.

The ExtendSim Optimizer exploits this collaborative architecture in its design.

What Does Optimization Mean?

Optimization describes the act of determining the ideal parameter values which will minimize or maximize an objective function within a simulation model in an automated way. The objective function typically represents a profit or cost. An Optimizer will run a model many times with different calculated parameter sets to search a solution space until conditions are satisfied that it has found a best set of parameters.

The Evolutionary Optimizer Block

The Evolutionary Optimizer block ships with all versions of ExtendSim. It can be placed in any discrete event, discrete rate, or continuous model to add optimization capability with minimum effort.

To use the Optimizer block in your model:

1 Drag the variables to be optimized to the block.
2 Enter a profit or cost equation to be optimized.
3 Add any local or global constraint equations.

The Optimizer Parameters tab in the Optimizer's dialog has additional choices:

- Buttons to set default choices for most models.
- Limits for number of samples and cases to run.
- The convergence percentage needed to finish.
- Choices to use the Mean or Median as an ex-pected value calculator, or the best of both.
- Antithetic sampling for sample reduction.
- Tail truncation in very noisy models.

Most users do not need to be concerned with anything except which Defaults button to use.

An Example Using the Evolutionary Optimizer

This section illustrates a very simple example model and the use of the Evolutionary Optimizer to find the best solution.

Problem description

Acme Egg Coloring, LLP, buys raw eggs, rapidly cooks and colors them, and ships them out.

- All finished eggs get sold to distributors.
- Any raw eggs left over at the end of the day must be destroyed according to the imaginary but strictly enforced Egg Coloring Act of 1777.
- Eggs are bought for $0.25 each and sold for $0.50 each.
- Eggs get processed one at a time according to a Lognormal distribution with a mean time of 5 minutes and a standard deviation of 1 minute.

THE EXTENDSIM OPTIMIZER

With these conditions, it would seem that Acme would want to buy only enough eggs to finish in one day with as few leftover eggs as possible, since those would have to be thrown away at a loss.

Model of Acme Egg Coloring, LLP

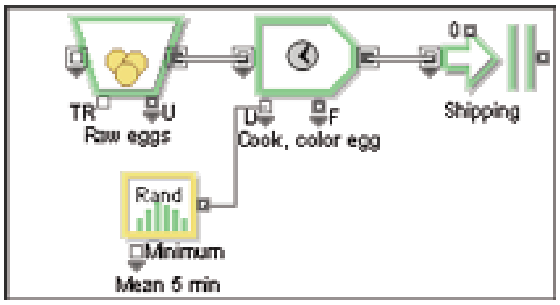


FIGURE 1. EXTENDSIM MODEL OF ACME EGG COLORING

The model consists of:

- A Resource block labeled "Raw eggs" that holds the raw eggs that were bought for that day.
- An Activity Delay block labeled "Cook, color egg" that pulls an egg from the Resource block and processes it.
- An Input Random Number block labeled "Mean 5 min" that calculates the processing time and is queried by the Activity block for each egg.
- An Exit block to ship the items out of the model, labeled "Shipping."

The model is set to run for 8 hours and then stop. The calculation of how much profit Acme makes uses the following equation:

*Profit = (shippedEggs * 0.50) – (leftoverEggs * 0.25)* (1)

Any leftover eggs will really cut into Acme's revenue stream, enraging the stockholders. An option would be to run the model with different numbers of raw eggs until the optimum solution is found; however, it would be much easier to automate the process. Using the Evolutionary Optimizer block, the search for the optimum solution can occur without intervention.

Finding a solution using the Optimizer

In order for the Optimizer to search for a solution, it needs to know:

- Which variables to search.
- The allowable set of values for those variables.
- The objective function used to determine a good solution.

The objective function will tell the Optimizer how to compare cases and choose which results and corresponding parameter values are better.

Adding an Optimizer block to the model

First, add an Optimizer block to the model:

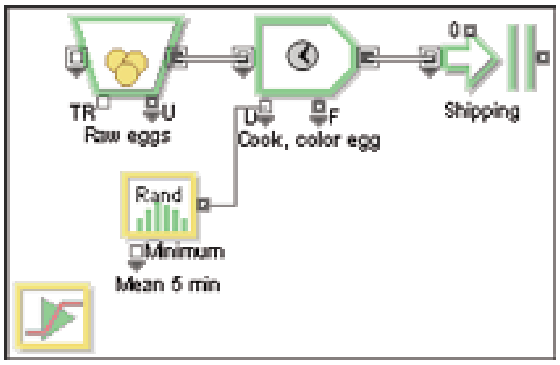


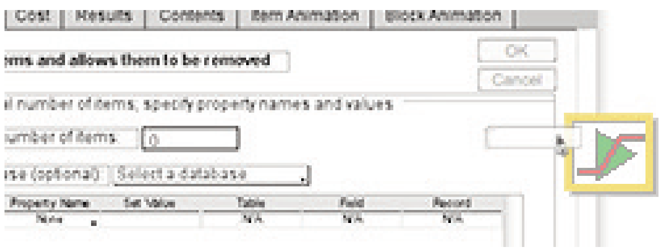
FIGURE 2. ADDING AN EVOLUTIONARY OPTIMIZER BLOCK

Now add the variables to be searched for and an objective function to the Optimizer.

Adding variables to the Optimizer

To add variables to the Optimizer, drag and drop them using the Clone tool. The Clone tool creates hot-linked duplicates of data when the user drags a dialog item from a dialog onto a document. Rather than having to open the dialog to enter the data or view results, clones can be used to enter values and report dialog values. By using the Clone tool to add variables into the Optimizer block, the user does not have to determine and enter variable names manually.

The variable to be optimized is the initial number of eggs in the Resource block. Figure 3 shows the Resource block's *Initial number* variable as it's being cloned from the Resource to the Optimizer.



ems and allows them to be removed

l number of items, specify property names and values

umber of items: 0

se (optional): Select a database

Property Name	Set Value	Table	Field	Record
None		N/A	N/A	N/A

FIGURE 3. CLONING INITIAL NUMBER OF EGGS FROM RESOURCE TO OPTIMIZER BLOCK

Next, open the Exit block to clone the variable representing the number of eggs shipped, so the Optimizer can calculate the revenue from this day's work.

When you drag each of these variables onto the Optimizer block, it will highlight to indicate that the cloning operation was successful.

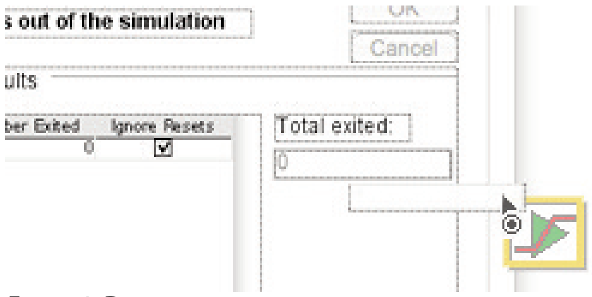
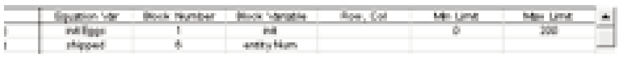


FIGURE 4. DRAGGING THE NUMBER OF EGGS SHIPPED LEFTWARD ONTO THE OPTIMIZER BLOCK

Adding the set of allowable values (ranges)

Opening the Optimizer block, you can see the Variables table (Figure 5) with the cloned parameter variables. Now enter the allowable range for the number of raw eggs to buy for that day. In this case, choose from 0 to 200 eggs, omitting any decimal point to keep these variables as integers. The 0 is obvious. The 200 is a guess based on the processing time per egg (about 5 minutes) and the time available for processing all eggs (480 minutes). That guess is doubled so as to not overly limit the range. Setting a range is how



	Equation Var	Block Number	Block Variable	Row, Col	Min Limit	Max Limit
	initEggs	1	init		0	200
	shipped	6	entryNum			

the solution space is limited. The closer the range is estimated, the faster it can converge to a solution.

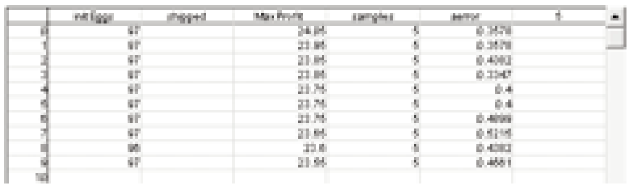
FIGURE 5. THE VARIABLES TABLE WITH *INITEGGS* AND *SHIPPED* VARIABLES ADDED

Adding the objective function

Now enter the objective function so the Optimizer can determine a good solution. This equation is the same as equation 1, but with the correct variable names:

*MaxProfit = shipped*0.50 - initEggs*0.25;* (2)

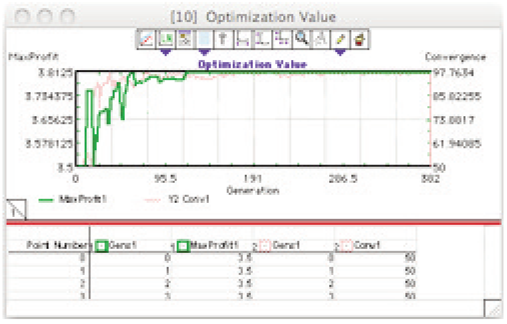
Results of the optimization



	initEggs	shipped	MaxProfit	samples	error	5
0	97		24.85	5	0.3578	
1	97		23.95	5	0.3578	
2	97		23.85	5	0.4082	
3	97		23.85	5	0.3347	
4	97		23.75	5	0.4	
5	97		23.75	5	0.4	
6	97		23.75	5	0.4899	
7	97		23.65	5	0.5215	
8	96		23.6	5	0.4082	
9	97		23.55	5	0.4661	
10						

After you run the simulation using the Run Optimization command, the Optimizer reports these results:

FIGURE 6. THE FINAL POPULATION OF RESULTS, BEST AT TOP



Point Number	Gens1	MaxProfit1	Gens1	Conv1
0	0	3.5	0	50
1	1	3.5	1	50
2	2	3.5	2	50
3	3	3.5	3	50

FIGURE 7. PLOT OF THE RUN SHOWING CONVERGENCE

Adding constraints

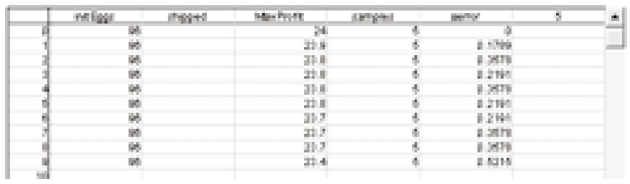
As you can see from the results, the optimum number of eggs bought should be 97. But you have to buy eggs by the dozen, so you need to add a constraint to the Optimizer, forcing it to buy eggs in groups of 12.

You can enter a constraint on the Constraints tab of the Optimizer that will take the initEggs variable and force it to be an integral dozen of eggs:

*newInitEggs = int(initEggs/12) * 12;* (3)

Constrained results

The new final population of results shows the best solutions with 96 eggs, equivalent to 8 dozen eggs.



	initEggs	shipped	MaxProfit	samples	error	5
0	96		24	5	0	
1	96		23.9	5	0.1789	
2	96		23.8	5	0.3578	
3	96		23.8	5	0.2191	
4	96		23.8	5	0.3578	
5	96		23.8	5	0.2191	
6	96		23.7	5	0.2191	
7	96		23.7	5	0.3578	
8	96		23.7	5	0.3578	
9	96		23.4	5	0.5215	
10						

1 FIGURE 8. CONSTRAINED RESULTS

You cannot buy 8 dozen eggs? Add a global constraint

Assume all egg vendors refuse to sell even numbered dozens of eggs due to the even more imaginary Odd Dozen Eggs Used in Egg Coloring Act of 1778. You have to buy an odd number of dozens!

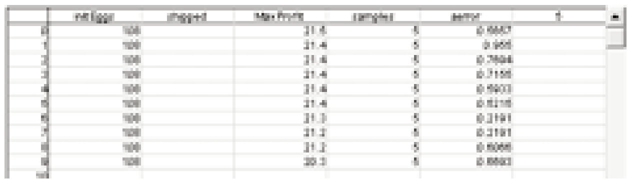
A Global Constraint can force the rejection of an entire case depending on case values. The equation needed on the Constraints tab of the Optimizer to force the rejection of even numbered dozens of raw eggs is:

```
// if even dozen, reject this case        (4)
if (initEggs/24.0 == int(initEggs/24.0))
  Reject = TRUE;
```

THE EXTENDSIM OPTIMIZER

Final results

Notice how you need to buy 108 eggs or 9 dozen to make the best profit. This makes sense, because it is more probable that you would make a higher profit buying 9 dozen eggs and throwing some out, than buying 7 dozen and throwing none out.



	intEggs	dragged	Max Profit	samples	error	f
0	108		21.5	5	0.5667	
1	108		21.4	5	0.955	
2	108		21.4	5	0.7694	
3	108		21.4	5	0.7155	
4	108		21.4	5	0.5903	
5	108		21.4	5	0.6216	
6	108		21.3	5	0.2191	
7	108		21.2	5	0.2191	
8	108		21.2	5	0.6066	
9	108		20.3	5	0.6693	

FIGURE 9. ODD DOZEN GLOBAL CONSTRAINT ADDED

Designing the ExtendSim Optimizer

When adding features to any application, there are always design tradeoffs. We wanted an optimizer for the ExtendSim simulation applications, but we wanted it to match ExtendSim's interface design so it would feel very familiar to a user. How should we proceed?

How to add features to ExtendSim

There are three routes that we can take to adding a feature to ExtendSim:

1 Add the code and user interface elements to ExtendSim's C++ code base, or
2 Implement the feature as a new block in a library that ships with ExtendSim, or
3 Interface to another application that implements the feature.

If we hard coded an Optimizer into the application code, the source code would be invisible to the user and not available for clarification or possible modification. The user would not be able to further customize the Optimizer for a specialized application.

This would violate ExtendSim's design culture where it is as open source and extensible to the user as possible.

We could save much design time by using an existing optimization application, but at the expense of ease-of-use. We could save many user executed steps needed by external applications by integrating our optimization process into ExtendSim's simulation engine, and we would be able to keep control over the user interface, a most critical part of our design.

If we could create an Optimizer library block with ExtendSim's built-in block development environment, the Optimizer could be open source and available to the user for modification, if needed, for a specialized use. Additionally, any added user interface elements would be available for use in other types of blocks. We chose this method to add optimization capabilities to ExtendSim.

Creating the ExtendSim Optimizer block

We discuss the internals of the Optimizer here. It is beyond the scope of this paper to show the procedures that we used to come to this design. Please note that there are no precise analytical methods available to justify any particular design of an optimizer.

Human Interface Considerations: Drag and Drop

In order to run cases using parameter sets, the Optimizer needs to gather information about the locations of variables contained in the parameter set.

The easiest metaphor to use for the connection between the Optimizer block and model data is the "drag and drop" design. This will allow a user to drag a desired variable to the Optimizer, and the Optimizer will then extract the data location. To facilitate this, we added a message to ExtendSim which will be sent to a block that has a variable dragged onto it.

The procedure to use the drag and drop interface is:

1 Open the dialog containing the desired variable.
2 Using the clone tool, drag the data to the optimizer block on the model worksheet.
3 When the user releases the mouse, ExtendSim sends the Optimizer a 'DragCloneToBlock' message.

The Optimizer block will then execute its 'DragCloneToBlock' message handler, interrogating the remote (dragged from) block, extracting the data's pointer information and saving it in its dialog Variables table.

ExtendSim's optimization algorithm

ExtendSim's optimization algorithm is essentially a survival of the fittest solution within a steady state population of solutions. Good fitness is defined as a maximum resulting profit or minimum resulting cost of the objective function used.

The algorithm developed is based on genetic algorithm concepts, but with enhancements to more closely emulate a DNA based evolutionary process with its larger effective number of 'bits' (floating point is used) and correspondingly finer precision of mutation and mating attributes. Algorithms similar to the one we are using have historically been called evolutionary algorithms to differentiate themselves from classical bitwise genetic algorithms as developed by John Holland in the late 1960s.

Calculating fitness of stochastic processes

Because of noise in a stochastic process, calculating the fitness of a solution becomes complex. One method relies on making *N* runs of a model and taking the mean of the objective function results. Alternately, the median can be substituted for the mean in some very noisy environments.

Evolutionary Steady State algorithm summary

Here is a brief pseudo code representation of the optimization algorithm developed for ExtendSim:

```
Generate double size population;
Throw away worst half;
While (population not converged)
    {           // Main optimization loop
    Tournament(2) select parent 1;
    Tournament(1) select parent 2;

    If (mating condition satisfied)
            Mate parents to create offspring;
    Else
            Use parent 1;

    If (mutation condition satisfied)
            Mutate offspring;
    Else
            Don't mutate offspring;

    Replace worst member with offspring;

    Calculate fitness of offspring N samples;

    Sort population for fitness;

    If (all population has used N samples)
            Increment N samples;
    }
Post final parameters to model;
Run model once to correlate results;
```

FIGURE 10. PSEUDO CODE OF OPTIMIZER BLOCK

Steady state population of solutions

Each member of the double size initial population is generated by randomizing the parameter set values using a uniform distribution, obeying limits specified by the user, and meeting all constraint equation requirements.

The population is then sorted according to fitness and the worst half is thrown out by reducing the population to the specified size.

In the main optimization loop, the algorithm then generates an offspring and replaces only the worst member of the population with that offspring. Some algorithms replace the entire population of parents with offspring, but the method used here keeps the population size constant and gives the offspring a chance to compete with its parents for fitness; the parents could be better than the off-spring.

The main loop continues until the convergence condition is satisfied.

Tournament selection

To select a good set of parents out of the population, Tournament Selection is used with a parameter S, Selective Pressure, that specifies how many population members compete for selection. If the parameter S is 1, one member is randomly selected from the population. If S is 2, then two members are selected and the best one of the two is used as a parent. If S is too high, only the best members of the population are mated, causing premature convergence to a false local minima or maxima.

Our algorithm specifies that one parent randomly selected from the population is mated with the best one of two other competing candidates.

Mating

Mating occurs at a specified rate. Considering that each member solution is composed of genes that represent an independent parameter of the model in question, mating represents a combination of the genes of two members, using a weighting factor, to create an offspring. The weighting factor for each separate gene is calculated as a uniformly distributed real value between 0 and 1 that specifies what proportions of member 1 gene and member 2 gene are linearly combined:

$$offspringG = (member1G * w) + (member2G * (1-w)) \quad (5)$$

Mutation

Mutation occurs at a specified rate for each gene, and this rate depends on how many genes (parameters) are contained in a population member.

If the mutation rate condition is satisfied for a gene, the change to the gene value is calculated using a Normal distribution with a specified standard deviation. This weights smaller value mutations higher than larger value mutations, but still allows radical changes that could find a more distant minima or maxima.

Sample reduction in stochastic models

Previously, we talked about finding the fitness of a solution in a noisy environment. However, in order to find the best solution in the least time, we must try to use as low of a sample rate as possible.

Our strategy is to use an incremental sample rate, starting at 1 sample. Incremental sampling tends to find stronger (more sensitive) parameters first, and then as the sample rate increases, finds weaker parameters more easily after the stronger parameters are found. This fundamental principle of signal processing uses the analogue of enhancing signal-to-noise ratio to extract weaker data.

Confidence intervals tend to overestimate the needed number of samples because all parameters are evaluated in the confidence of a fitness calculation, even the hard to find weak ones which tend to bump up the number of samples needed for the entire run.

Convergence

Our convergence metric is defined as the following, with *best* meaning the fitness value of the best member of the population and *worst* meaning the fitness value of the worst member of the population. It is the relative distance from best to worst member:

$$\text{Convergence} = 1 - (2 * (\text{best} - \text{worst}) / (\text{best} + \text{worst})) \qquad (6)$$

When convergence is above a threshold (e.g. 0.95 or 95%), the optimization run stops and posts its best results to the model. It then runs the model once more to correlate results data with that parameter set.

CONCLUSION

We have shown that setting up an optimization run in ExtendSim can be a relatively quick process, reducing the normal roadblocks and making optimization more accessible to the modeler. We have also covered the basic operation of the Optimizer and invite examination of the Evolutionary Optimizer block's open source code.

ACKNOWLEDGEMENTS

When you're designing an Optimizer, it sure helps to have a wonderful daughter who just happens to be a geneticist working in transgenics. Thanks Jenn for those conversations and explanations about the mysteries of genes, chromosomes, and DNA.

The Optimizer would never have happened without my muse, my wife Pat who constantly inspires me and who put up with noisy computers running simulations at all hours of the day and night.

And thanks to the writers that wow me constantly with all their ideas.

REFERENCES

Aarts, E., and Lenstra, J. K. et al. 1997. Local Search in Combinatorial Optimization. West Sussex, England: John Wiley & Sons Ltd.

Glover, Fred, and Laguna, Manuel. 1997. Tabu Search. Norwell, MA: Kluwer Academic Publishers.

Law, A. M., and W. D. Kelton. 2000. Simulation modeling and analysis, 3rd ed. New York: McGraw-Hill.

Michalewicz, Zbigniew. 1999. Genetic Algorithms + Data Structures = Evolution Programs, 3rd ed. New York: Springer-Verlag.

Rustagi, J. S. 1994. Optimization Techniques in Statistics. San Diego, CA: Academic Press, Inc.

AUTHOR BIOGRAPHY

BOB DIAMOND is the creator of the ExtendSim simulation application (and before ExtendSim, the DesignScope simulation application, used in signal processing) and the founder of Imagine That, Inc. While in school, Bob was recruited by NASA Apollo to create the Rocket-Drop simulation of combustion within the Saturn booster engine. As Principal Scientist at Commodore Intl., he used simulation extensively to validate new technology during the birth of the personal computer. At Consolidated Video Systems, he designed broadcast digital video signal processing hardware using simulation and was the recipient of a Rockefeller/Ford Foundation grant for his work as technologist/artist-in-residence at WNET/NY. His multimedia collaborations with Fluxus artist Bob Watts and composer David Behrman have been shown in the Whitney in NYC, the Smithsonian Institution and in other venues worldwide. His email address is <bobd@ExtendSim.com>