For the Activity Based Costing Analysis model, a “traditional” optimization was run with the following characteristics:

* Objective: Minimize total cost per product
* Parameters:
  + Resource A capacity: [1 .. 20 step 1]
  + Resource B capacity: [1 .. 20 step 1]
  + Mean process delay: [1 .. 12]
  + Conveyor speed: [0.1 .. 1.0]
* Model run time: 36500 days
* Requirements: The system was never overloaded during the entire simulation run (specifically that the `exceededCapacity` flag was never set to true)
  + This is tested after a run to determine whether the solution is considered feasible
* Replications: Between 40 and 100, with a confidence level of 99% and an error percent of 0.1
  + i.e., each configuration is run between 40 and 100 times
* Before each simulation run: Randomly vary the arrival rate between 0.5 and 2.0 (uniformly)

In short, the optimization experiment works as follows: a configuration of parameters is chosen by OptQuest engine. This configuration is then run between 40 and 100 times. The objective value (which can be thought of as the fitness score) is the mean of each run’s objective value (total cost per product). Based on this, the OptQuest engine chooses another configuration to try, which is based on its own internal logic, neural networks, etc.

Example of objective value calculation: if a configuration had 3 runs whose individual cost per product values were $80, $100, and $96, then the objective value used as a metric for that configuration would be 92 ().

Note 1: Because each run has a random arrival rate set, a large number of minimum replications were used. The chosen minimum (40) was found to have a high confidence (> 97%) that at least 5 runs would have an arrival rate in the lower 25% (less than 0.9) and at least 5 runs in the upper 25% (greater than 1.6). A relatively extreme upper bound was chosen to ensure that the confidence level would be reached.

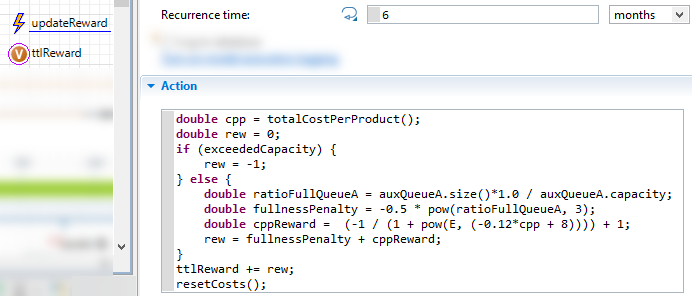
Note 2: For any given solution to be considered feasible, all 40+ replications have to pass the requirement check.+.-\*3

After running the optimization, the best configuration (i.e., the one that most minimized the cost per product) had a score of 48.427. In other words, the mean cost per product was $48.427. This was found to have a configuration consisting of each resource having 6 capacity, a mean delay time of 2.6 and a conveyor speed of 0.3.

With these values found, it’s desired to have a metric that enables comparison between the optimization results and the brain’s results. The best way to do this would be to fix the model’s random number generator to a known seed and then run the simulation twice – once with the optimized values and once with the brain being in control of the parameters. This would allow for an exact, side-by-side comparison between the two which compare a variety of different metrics with one another. A good example of this is the [traffic light example](https://www.youtube.com/watch?v=lgqwXM8CepQ&vl=en): the reward metric used in training was only based on the time in system of all cars; by directly comparing the optimization and trained AI in a simulation, it allowed us to see how the two differ in terms of speed distribution, mean time in system, and lengths of the traffic lights.

However, because predicting using an exported brain was not available at the time of this writing, there needed to be an alternative. The only other metric available from the brain is the latest mean reward shown in the Bonsai UI. This is not something that is currently in the model, but it could be easily added.

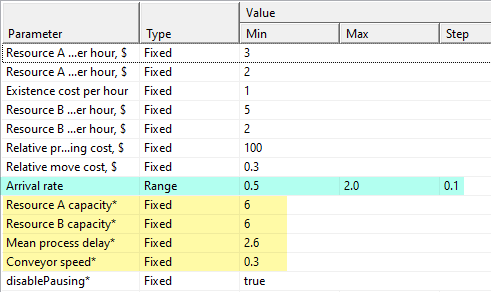
A new event was added in the model on the same recurrence as the brain observes, acts, and receives reward on (6 months). Its action field contains the same reward function used in the Inkling, and the resulting value for the current “iteration” is added to a double variable.



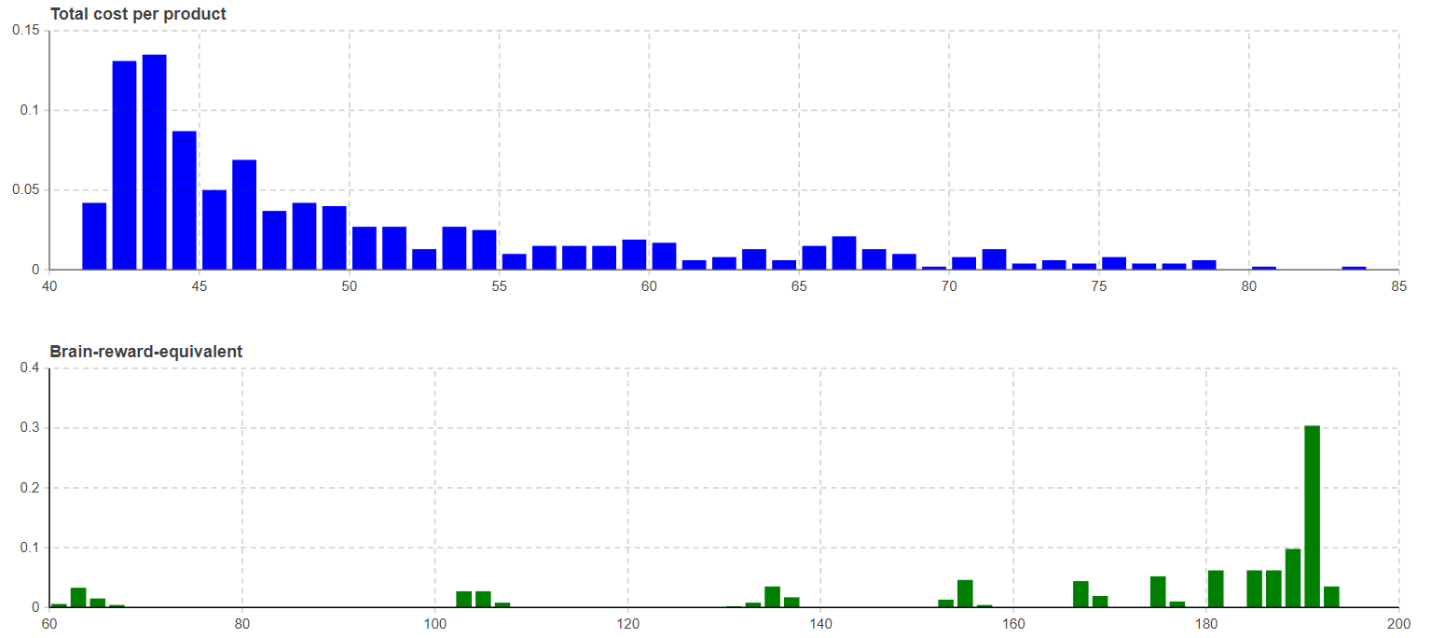
Now, every time the simulation is run, the value of the double variable will reflect what would have been the brain’s cumulative reward for the episode if it made no changes to its initial configuration.

Due to the inherit randomness of the model, even with a fixed arrival rate, it is not sufficient to run the model once and assume the output is the “true” value. By running the model multiple times and taking the mean of the outputted reward variable, it will start to converge on the “true” value. A common experiment used to run a model many times to see the distribution of values is a Monte Carlo experiment.

Here, we plug in the optimized values for the four parameters (yellow highlight in image below) and set the arrival rate to be varied between the allowed ranges (teal highlight in image below).



For each arrival rate in the specified range, there is a number of replications run. This varies with a minimum of 30 and a maximum of 100, stopping when a 99.9% confidence level, and an error percent of 0.01, of the reward output has been reached. The minimum, 30, was chosen based on personal discretion. One sample execution of this experiment resulted in 480 total simulation runs.



Two histograms (shown above) display the outputs. Each show the distributions and frequencies of a given value. The X-axis represents the value – cost per product for the blue histogram and the reward for the green histogram - and the Y-axis represents the frequency. The bar width, or bin size, determines the range of values encompassed by a given frequency, with the cost histogram having a bin size of 1 and the reward histogram having a bin size of 2.

Taking a look at the results, the tallest bin for the cost histogram is representative of simulations which returned a cost per product between 43 and 44, which were made up by approximately 13.5% of all the simulation runs. The tallest bin for the reward histogram signifies that approximately 30.4% of the simulation runs had a reward between 190 and 192.

Due to outliers, the mean values of each were offset from the most frequently occurring values. The mean cost was $51.57, and the mean reward was 163.51. It’s left up to the reader to decide which metric is more accurate of the “true” value. It’s for this reason that a direct comparison optimization vs policy is advised, as having the same RNG seed provides the most accurate way to compare.