Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

STOCHASTIC SHIPYARD SIMULATION WITH SIMYARD

Oliver Dain, Matthew Ginsberg, Erin Keenan,
John Pyle, Tristan Smith, Andrew Stoneman

On Time Systems
1850 Millrace Drive, Suite 1
Eugene, OR 97403

Iain Pardoe

Charles H. Lundquist College of Business
1208 University of Oregon
Eugene, OR 97403*

Abstract

SimYard is a stochastic shipyard simulation tool designed to evaluate the labor costs of executing different schedules in a shipyard production environment. SimYard simulates common production problems such as task delays and labor shortages. A simulated floor manager reacts to problems as they arise. Repeatedly simulating multiple schedules allows the user to compare the schedules on many different metrics, such as expected labor costs and the probability of missing the deadline. A SimYard simulation is driven by many inputs that describe the shipyard being simulated. Determining the correct values for these inputs can be framed as a multivariate calibration problem, which can be solved using inverse regression methods. Predictive sampling from the resulting model provides an appropriate adjustment for statistical uncertainty.

1 INTRODUCTION

Billions of dollars are spent annually on construction, refit, and repair of ships and submarines. The problem of developing schedules that make efficient use of resources is critical, both to control costs and to meet shipyard needs without exceeding the available resources, such as time, dry-dock space, or increasingly scarce skilled manpower. Both construction and maintenance are extremely complex processes, involving thousands of activities with thousands of constraints among them. This complexity makes it virtually impossible for people to construct good schedules without the aid of software tools.

Software scheduling tools take a database of project information that includes a description of each task to be performed, and the constraints thereon. The task descriptions include information such as the expected task duration and the resources required to complete it. The constraints include both binary (or precedence) constraints and unary constraints. Binary constraints specify a relationship between two tasks. Thus a binary constraint between tasks A and B might specify that task B cannot start until two days have passed since task A was completed (to allow primer to dry before painting for example). Unary constraints specify constraints that affect only a single task. For example, we might specify that a task cannot start before a certain date (to account for the availability of materials).

A schedule is an assignment of start times to tasks. There are many ways to arrange the tasks in a project into a schedule that honors all the constraints and the project deadline. Some schedules will incur higher labor costs than others, and some schedules will be more robust than others. For example, one schedule might work two tasks that each require five welders during January. This schedule might then require no welders until March 1, at which point five welders will be required. In order to execute this schedule, it would be necessary to either lay off all ten welders on February 1 and then hire five welders on March 1, or lay off five of the welders on February 1, and pay the remaining five welders for the month of February during which they accomplish no useful work (this is referred to as an "undertime" cost). Both options incur costs. A different schedule might delay one of the tasks that was scheduled to work in January so that it starts in February. This schedule would require a constant level of five welders and avoid the hiring, firing, and undertime costs associated with the original schedule. However, delaying one of the tasks might make the new schedule more brittle. If something goes wrong with the delayed task, we might miss the deadline or incur higher overtime costs as workers struggle to correct the problem in time.

Given a set of candidate schedules it is difficult to know which is the best in a real production environment. Because the cost of building a ship is so high it is not possible to build the same ship several times with different schedules and compare the results. Even if it were possible, external factors such as absenteeism would obscure the comparison.

*The authors thank David Etherington, R. Dennis Cook, and Christopher Bingham for helpful discussions.

SimYard is a shipyard simulator that is designed to virtually "build" the same ship over and over, using the same or different schedules. SimYard models shipyards, including stochastic variability and the methods yards use to manage that variability. SimYard includes a virtual *Floor Manager* whose job is to make intelligent local repairs to the schedule in response to deviations that occur in practice.

Given a model of shipyard behavior, SimYard repeatedly executes the schedule under realistic conditions and estimates the total cost of each "build". Given enough such samples, we can derive estimates and confidence intervals for labor costs. In addition, the brittleness of schedules can be gauged in a variety of ways, such as the variance in the estimated cost over the set of simulations, the average amount by which the deadline is violated, and the average number of overtime hours required are good indicators of robustness. Since SimYard can repeat this process for different schedules, it is also possible to compare the likely cost of executing different schedules.

A single SimYard run is controlled by a number of input variables. Most of these control the probability distributions used to generate random numbers during the simulation. For example, there are two inputs to control the average and standard deviation of the factor by which task durations should be altered from their predicted values. These values are fixed for the duration of a single SimYard run. Given the mean and standard deviation, we have a probability distribution that can be used to generate random numbers. The random numbers are then used to perturb the task durations for each task in the project. The values of the input variables are specific to the shipyard being simulated; a yard that consistently underestimates how long it will take to complete a task would have a value greater than one for the mean task duration deviance factor, while a yard that consistently overestimates would have a value less than one.

Determining the correct values to use for the input variables requires performance data indicating how the yard actually executed a schedule. Given a set of performance data it is necessary to determine a range of inputs values that are likely to have generated the observed data. Discovering how input variables map to observable behavior (output variables) requires running SimYard many times with randomly chosen inputs picked from a reasonably large domain. SimYard uses this data and the performance data to estimate a range of inputs that are representative of the actual shipyard.

The SimYard simulator differs from other recent shipyard simulation projects (McLean and Shao, 2001; Williams et al., 2001; Asok and Aoyama, 2005; McDevitt et al., 2003), in three important respects. First, these other studies simulate at a more fine-grained level (including physical constraints such as crane alignment and shop-floor space), but at the cost of simulating entire shipyards for only very limited periods of time. Second, previous work has had only limited success in simulating behavior that matches observed real-world behavior; this is a major challenge that we address in this article. Finally, none of these other projects includes the ability to adapt to real-world changes, as we do with the virtual *Floor Manager* and *Personnel Manager*; we believe these components are crucial, given our observations of actual shipyards.

Section 2 gives an overview of a single simulation. Section 3 explains how the input variables are estimated. Results of running SimYard on real project data are presented in Section 4.

2 SHIPYARD SCHEDULING AND SIMULATION

The simulator in SimYard takes a shipyard schedule and a set of inputs. It then steps through a project while simulating changes that arise and shipyard responses to those changes. For example, on the day an activity is started, it might be discovered that more workers are needed than were anticipated; this might result in other activities being delayed. Each simulation produces corresponding outputs.

The inputs describe how the shipyard operates and consist of forty two variables. The outputs measure the outcome of an execution of a schedule and consist of twenty variables. Some, like the perceived cost of pausing a task that is in progress, are only inputs. These affect the simulation, but are not directly measurable. Others, like the fraction of tasks that were paused, are outputs only. These are directly measurable in performance data and the simulation results. Values can also be both inputs and outputs. For example, the measure of how task durations deviate from predictions are inputs which affect the simulation and are also directly measurable in performance data.

Most of the input variables describe a distribution. For example, there is a mean and standard deviation for task duration changes. During simulation a pseudo-random number is generated for each task. The random number is drawn from the distribution described by the task change input variables.

The input and output variables can be divided into five groups:

1. *Costs*: these are all outputs and include the total cost and components of the total cost such as overtime cost and workforce acquisition cost. The cost will be different for each run as variables such as task delays, worker absenteeism, and changes in manpower requirements will change.
2. *Shipyard Unpredictability*: These are all inputs, except for duration change values which are both inputs and outputs. This groups controls the distributions on the number of workers who will be sick in a day, how often tasks are delayed, and how much notice the shipyard will get that a task is going to be delayed.

3. *HR Issues*: These are primarily inputs that control the human resources department. These include the cost to hire and fire workers, the frequency with which staffing changes can be made, how much staffing should be above or below the projected labor levels, and the amount of overtime workers are allowed to work. This category also includes outputs that track the total amount of overtime worked on the project.
4. *Shipyard Timing*: These are primarily outputs that measure how often tasks were paused mid-execution, what percentage of tasks were paused, and the amount of free float that was present in the worked schedule (free float is the amount of time a task could be delayed without affecting any other tasks). The category also includes inputs describing the perceived cost to pause a task.
5. *Cheating*: While a project describes constraints on tasks, it is not uncommon for a yard to violate some of the constraints described in the project. Presumably there is a cost to violating a constraint. SimYard thus includes several inputs that describe the perceived cost of breaking binary, unary, release date, major milestones, and deadline constraints.

Section 3 explains how values are determined for the inputs. For the moment we assume we are given values and focus on how a single simulation proceeds. The simulator consists of three main components:

1. *Reality* models how things change under real-world, yard-dependent conditions. This component makes changes to the project based on the input variables.
2. The virtual *Floor Manager* adapts the schedule in response to *Reality*, deciding which tasks to work on and which to put off. This component is intended to model the actual floor manager of a real shipyard. Therefore, the decisions made are based on the knowledge that a real manager would be expected to have (based on conversations with employees in various shipyards). For example, if work levels exceed available manpower the *Floor Manager* decides if tasks should be delayed or overtime hours assigned.
3. The virtual *Personnel Manager* reacts to the overall changes from *Reality* and from the *Floor Manager*'s responses to those changes by revising staffing decisions (work-force acquisition and reduction).

Each simulation begins with the *Reality* component randomly picking tasks that will be delayed. For each such task the length of the delay, and the amount of notice the floor manager will have of the delay are randomly determined. For example, it might be determined that task will be delayed by five days and the floor manager will not know that the task will be delayed until one day before it was scheduled to start. Simulation then proceeds one day at a time. At the start of each day the *Reality* component adjusts the state of the yard as follows:

- Tasks that are scheduled to start on the day in question have their duration and manpower needs randomly perturbed.
- The number of workers who called in sick is randomly generated.
- Delayed tasks will be revealed if the current simulation day falls within the task-delay-notice window.

The *Floor Manager* then considers the available work force and the tasks that are could be started. For each such task, the *Floor Manager* compares the cost of working the task today with the cost of delaying it until tomorrow. The cost calculation includes the manpower costs (including overtime and undertime) as well as the perceived cost of putting off the work. For example, putting off a task that has already started involves pausing the task, and it might also involve violating one or more constraints. Each task has had a pause cost and constraint violation costs randomly assigned according to the input distributions. If the cost of putting off the work appears less expensive than the cost of doing the work today, the task will be put off. The fourth time a task is put off it will be moved into the future far enough for the *Personnel Manager* to react to it and hire workers. The *Personnel Manager* reviews staffing levels periodically. The frequency with which staffing levels are reviewed is determined by an input variable. If the *Personnel Manager* decides to increase or decrease the staffing level it cannot be done immediately; there is a lag between the decision to change staffing levels and the levels actually changing. This lag is determined by another input variable. When hiring workers the lag models the delay between posting a job and the availability of trained workers. Similarly, it is generally not possible to fire or reassign workers immediately.

The simulation continues until all tasks in the project have been completed. At this point it is possible to calculate all the output values, including all the labor costs. Running the simulation many times makes it possible to estimate the probabilities of the output variables. If multiple schedules are simulated many times, it is possible to compare the schedules for cost and robustness.

3 Determining Input Values

One of the challenges in developing SimYard was determining a procedure for "tuning" it by selecting appropriate sets of inputs so that its behavior matches that of an actual production shipyard. The problem is that many inputs cannot be obtained from existing data. For example, if a task

requires more work hours than expected, the *Floor Manager* could assign overtime, rearrange other tasks, or postpone the task in question. The likelihood of each decision depends in turn on other factors; postponing the task, for example, may depend on how willing the *Floor Manager* is to break deadlines or constraints with other tasks. Quantifying, *a priori*, the input values that determine this behavior (for example, the perceived cost of missed deadlines) is almost impossible.

Therefore, given performance data for a real project that has been completed, the goal is to create sets of inputs for which SimYard's behavior on the project would be similar to that observed in the performance data. One way to approach this is to try to model the relationship between inputs and outputs in SimYard. However, estimating such a model and then finding appropriate inputs is not straightforward for several reasons:

- Many inputs are not outputs. Therefore, observed performance data will not indicate what settings should be used.
- Other variables are outputs only. There will be observed values for these and an accurate SimYard should result in similar output values. Achieving this restricts how the inputs can be used.
- Many variables are interdependent.
- Because the simulation is stochastic, it is impossible to calculate exact relationships between the inputs and outputs from a randomly generated sample of SimYard runs.
- As there are more inputs than outputs, there is a variety of input values that should yield identical measured outputs. It will be necessary to characterize the portion of the input space that yields the desired outputs so that SimYard can be run on many points in this space to produce accurate cost estimates.

These restrictions suggest a statistical modeling approach. Given a project and a schedule one can generate data suitable for building a statistical model by running SimYard thousands of times on randomly selected input values. Each run will yield a pair of vectors, (X_i, Y_i), relating the inputs, X_i, used in the run to the outputs, Y_i, produced by the run. Given this data the challenge is to build a suitable statistical model that relates inputs and outputs, so that an appropriate set of inputs can be generated (sampled) for any given set of outputs (i.e., real shipyard data). The model needs to be stochastic rather than deterministic to account for modeling uncertainty (the model can only hope to approximate SimYard relationships, not replicate them exactly).

A SimYard evaluation thus consists of five steps:

1. Run SimYard many times with different, randomly generated, inputs on a schedule that was actually used to construct a ship.
2. Build a statistical model relating inputs to outputs.
3. Given performance data from the actual construction that followed the schedule used in step (1), use the model developed in step (2) to generate a probability distribution over the input variables. This distribution indicates the joint probability that any set of input variables generated the observed performance data.
4. Randomly generate input values according to the probability distribution generated in (3).
5. Use the input values generated in (4) to run SimYard on one or more schedules.

If performance data is not available it is still possible to perform the first two steps. The statistical model developed in step (2) is valuable even if the other steps are not performed, as it indicates how certain input values affect observable shipbuilding metrics. For example, the model might indicate that incorrect estimates of task duration are an important driver of total cost and missed deadlines. This type of knowledge may help a yard control its business practices. A detailed discussion of the statistical modeling used in step (2) appears in the next section.

3.1 Calibration and Inverse Regression

Given a set of (X_i, Y_i) pairs, we must generate a statistical model. A multivariate regression model with outputs as the dependent variables and inputs as the independent variables might seem a natural way to approach the problem, but runs into difficulties because there are many more inputs than outputs. For example, solving twenty linear equations (one for each output) for forty two unknowns (the inputs) leads to a high-dimensional space of possible solutions. Sampling correctly from this solution space is not straightforward. Further, based on initial work using this approach, the solution space often leads to implausible ranges for the input variables. An alternative approach seems prudent.

If, in tuning SimYard, both inputs and outputs can be considered stochastic (a reasonable assumption here), then inference about the joint probability distribution of inputs and outputs can also be based on multivariate "inverse regressions" of inputs (dependent variables) on outputs (independent variables). This approach is particularly advantageous in this context, because it is straightforward to calculate expected input values from observed output values using the estimated (inverse) regression results. Uncertainty is accounted for using standard formulas for prediction (confidence) regions. Also, having the number of inputs exceed the number of outputs does not lead to problems with this

approach since the prediction region formulas automatically handle this.

Thus, we used an inverse regression approach to tune SimYard. In particular, we first sampled a large number (two thousand) of sets of inputs from appropriate distributions. To allay concerns over biasing the simulations, these were selected to be uniform over reasonable, meaningful ranges for each input variable (some inputs were also transformed if, for example, they were constrained in reality to be positive). (We also experimented with Latin hypercube sampling (McKay et al., 1979) to sample the inputs, but this made little difference to our results or computing times.) SimYard then produced a set of outputs for each set of inputs (this is step (1) as described in Section 3). The entire collection of simulated inputs and outputs was then modeled using inverse regressions of the input values on the output values.

The goal of identifying which inputs are most likely to have produced some actual observed outputs, bears some similarities to multivariate calibration problems. There, the "inputs" might be "gold standard" (hence expensive) measurements of some kind and the "outputs" are inexpensive alternative measurements. The goal is then to build inverse regression models relating the gold standard measurements to the inexpensive alternative measurements, so that future alternative measurements can be accurately recalibrated. Brown (1982) reviews the inverse regression procedure employed by SimYard in this context.

To understand this approach, consider a hypothetical example where there are six inputs, $(X_1, \ldots, X_6)$, and three outputs, (Y_1, Y_2, Y_3). Furthermore suppose the true relationship between the inputs and outputs are given by the following:

$$\begin{aligned} Y_1 &= X_1 + X_2 + e \\ Y_2 &= X_3 + e \\ Y_3 &= X_4 + e \end{aligned}$$

Here, e represents 1% random error. Note that two inputs, X_5 and X_6, do not affect any outputs.

It is straightforward to generate a large number of observations from this setup, with the inputs ranging uniformly over [0,1]. The resulting input and output values can then be used to estimate inverse regression models of X on Y. Finally, given a particular set of observed output values, say $Y_1 = Y_2 = Y_3 = 0.5$, the estimated inverse regression models can be used to generate a large number of appropriate "likely" input values (drawn from the predictive distribution, see Section 3.2).

We ran this experiment using the R statistical software (R Development Core Team, 2005) to produce the scatterplots in Figure 1.

The generated values of X_1 and X_2 in the left-hand plot show that they satisfy the first linear equation, $X_1 + X_2 = 0.5$ (with a small amount of variation due to error). The values of X_3 and X_4 in the center plot show that they correctly cluster tightly around 0.5 (which is required to satisfy the second and third linear equations). Finally, since X_5 and X_6 do not affect the outputs, the right-hand plot correctly shows sampling approximately uniformly over the range [0,1].

3.2 Predictive Sampling

Step (4) of a SimYard evaluation, as explained in Section 3 requires us to generate random input values according to the joint probability distribution generated by the model described in the previous section. While it is straightforward to calculate expected input values from observed output values using the estimated (inverse) regression results, it is not immediately obvious how to generate a set of possible input values from an appropriate prediction region. The multivariate probability distribution required for drawing these predictive samples, a matrix-t distribution (Raiffa and Schlaifer, 2000; Keyes and Levey, 1996, pp. 256), is non-standard and reasonably difficult and computationally intensive to sample from. Due to the relatively large size of the simulations, we therefore used a multivariate normal distribution (which is easy to sample from) as an approximation here. Since the number of input/output observations used to tune SimYard and estimate the inverse regression models is two thousand, this should be a reasonable approximation.

3.3 Further Validation of the Regression Model

In order to confirm the correctness of the inverse regression models, we randomly selected SimYard output values and checked if SimYard simulations based on input values generated from the corresponding inverse regression models actually produced output values similar to those initially selected. In particular:

1. We ran one thousand SimYard simulations using the default uniform input ranges to generate corresponding outputs, and then estimated the inverse regression models (based on the one thousand sets of inputs/outputs).
2. We selected one of the thousand runs at random and used the output values produced by that run as assumed "real" performance data.
3. We used the predictive sampling (of Section 3.2) to produce one hundred sets of "likely" input values that could have produced the selected set of "real" output values.

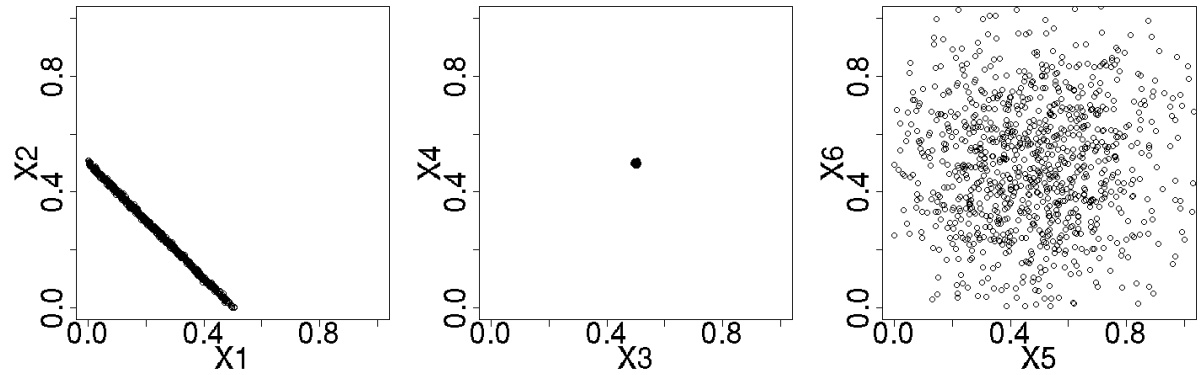

Figure 1: Scatterplots for the hypothetical six input/three output experiment showing X_1 vs. X_2, X_3 vs. X_4, and X_5 vs. X_6, respectively.

4. We ran a SimYard simulation for each of these one hundred sets of input values to produce one hundred corresponding sets of output values.
5. Finally, we looked at these one hundred sets of output values and compared them to the original randomly selected set of "real" output values. In particular, for each output, we calculated the percentile of "real" output value in the SimYard simulation values (e.g. if the "real" value was the highest value produced its percentile is one hundred. If the "real" value is the median of the values produced its percentile would be fifty).
6. We repeated steps 2-5 one hundred times to obtain an estimate of the distribution of percentiles (from step 5) for each output.

Since SimYard is stochastic, any particular output value will differ from its predicted value based on the inverse regression models. If it is close to its predicted value, then the percentile in step 5 should be close to 50%. If it is far from its predicted value (i.e., it is somewhat of an outlier), then the percentile might be closer to 0% or 100%. However, on average, repeating this experiment should, if SimYard is not producing biased output values, result in percentiles centered around 50% (although, as just noted, individual values might be as extreme as zero or one hundred).

Since there are twenty outputs, there is a good chance that with any randomly chosen set of output values, a few could be relative outliers. We observed this in the above experiment; as expected the output variables with extreme percentiles varied from experiment to experiment, and some experiments produced no outliers. Overall however, no individual outputs exhibited a tendency to consistently produce extreme percentiles. For example, Figure 2 shows a histogram of the percentiles produced for one particular output.

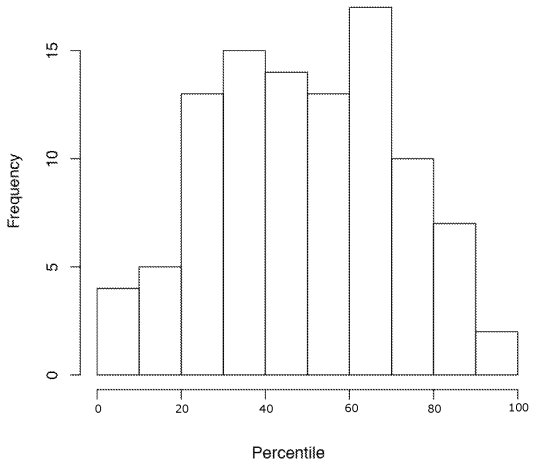

Figure 2: Histogram of percentiles for a randomly selected output value with respect to SimYard simulations of that particular output.

This exhibits the desired behavior, being centered at 50% with no particular tendency to always produce extreme percentiles close to 0% or 100% (such a tendency would correspond to a bias on the part of SimYard for the corresponding output). Other outputs produced broadly similar results, with none showing signs of severe bias.

4 RESULTS

SimYard was run on a project obtained from a commercial shipyard for a single large hull consisting of over seven

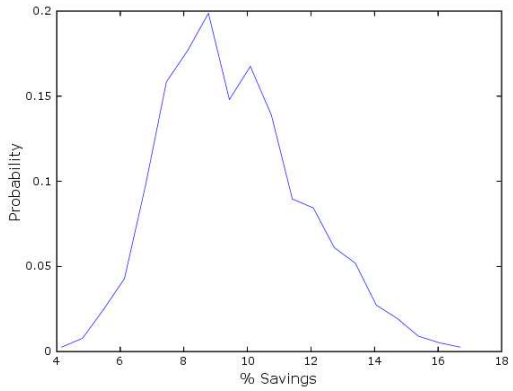

Figure 3: Percentage cost savings of the ARGOS schedule versus the standard schedule. The curve represents a probability density function estimate based on one thousand observations.

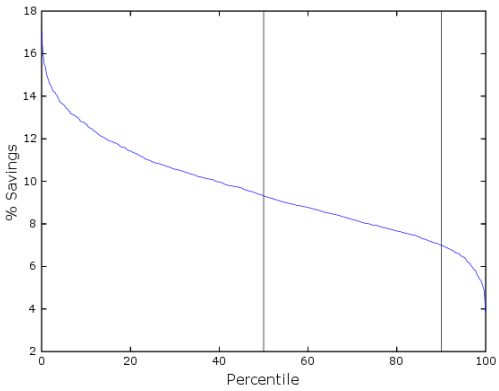

Figure 4: Cost savings confidence of the ARGOS schedule versus the standard schedule, showing expected percentage savings as a function of the confidence percentile. The two vertical lines correspond to the fiftieth and ninetieth percentiles.

thousand activities spanning seven years. The schedule used by the shipyard, referred to as the "standard schedule", was compared to the schedule produced by On Time Systems' ARGOS scheduling tool (Dain et al., 2005), referred to as the "ARGOS schedule".

For each schedule, we generated one thousand sets of outputs and associated costs, as detailed in Section 3. Figure 3 shows the cost savings of the ARGOS schedule relative to the standard schedule.

Over all the simulations done, the ARGOS schedule saves between 4% and 17% of the labor costs, with most runs saving between 7% and 13%. Figure 4 shows a savings confidence plot for the ARGOS schedule relative to the standard schedule.

The ARGOS schedule saves more than 10% half the time and saves more than 7% 90% of the time.

In addition to comparing the savings of one schedule to another, the simulations described in this article enable consideration of how those savings vary as input values change. For example, Figure 5 shows the savings sensitivity of the ARGOS schedule relative to the standard schedule based on the input that measures the average change in manpower requirements from unexpected shipyard events.

As tasks tend to require more work than expected, the savings of the ARGOS schedule go down, presumably because the resulting uncertainty is forcing SimYard to depend more on the floor manager and less on the details of the original schedule. However, overall, the savings achieved by the ARGOS schedule appear to be relatively robust to changes in input values. For example, Figure 6 shows that ARGOS savings depend little on the fraction of tasks that get unexpectedly delayed.

5 DISCUSSION

SimYard was designed to simulate shipyard operations under a variety of input conditions. It can be used to compare the labor costs incurred by various schedules and to evaluate the robustness of these schedules. Generating input values that are likely to correspond to observed outputs from real shipyard projects is accomplished via multivariate inverse regressions to infer the joint probability distribution of inputs and outputs.

SimYard has the potential to provide a number of benefits to shipyards. In particular, SimYard could be used to:

- synthesize shipyard performance data and answer questions such as "how often were tasks paused?"
- analyze input/output dependencies to answer questions such as "which factors influence the amount by which deadlines are broken?"
- investigate hypotheses concerning shipyard behavior (for example, "does advanced warning of task delays help the yard meet the deadline?")
- compare different shipyards based on performance data
- help understand the impact of additional work
- select the best schedule for a project.

This potential rests heavily on whether output variable estimates from SimYard can be considered accurate and reliable. Initial experiments indicate that this is the case. The costs estimates generated by SimYard are similar to the cost estimates generated by McDevitt et al. (2003) on similar data

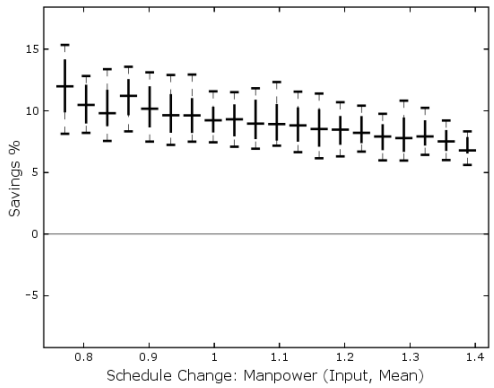

Figure 5: Savings sensitivity to the input measuring the average change in manpower requirements from unexpected shipyard events. The box plots represent medians, upper and lower quartiles, and tenth ninetieth percentiles for the generated savings estimates.

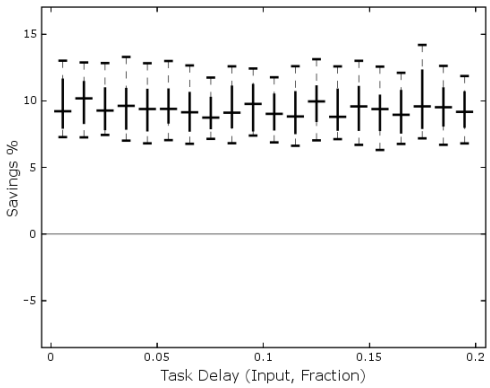

Figure 6: Savings sensitivity to the the input measuring the fraction of tasks that get unexpectedly delayed.

(this is particularly interesting as the simulation approaches are significantly different). Conversations with shipyard personnel about the dependencies discovered by SimYard indicate that the results are accurate. Furthermore, running SimYard on several different projects from the same shipyard and discovering input values that could have produced the observed outputs on those projects results in very similar inputs. The input values discovered seemed reasonable to personnel from the shipyard in question. This indicates that the simulation and inversion techniques employed by SimYard are capable of discovering the true operating parameters of the simulated yard.

To further validate SimYard it would be beneficial to get more real data and to discuss SimYard results with shipyard personnel to make sure the results reflect reality. We are currently working with the United States Navy to begin such experiments. Additionally, we intend to adapt SimYard for use in a repair environment. This will involve the ability to handle variablity specific to repair environments such as "open and inspect" tasks that reveal new work.

References

Asok, K. A., and K. Aoyama. 2005. Risk management in modular ship hull construction considering indefinite nature of tasks. In *Proceedings of the 12th International Conference on Computer Applications in Shipbuilding*. Busan, Korea.

Brown, P. J. 1982. Multivariate calibration (with discussion). *Journal of the Royal Statistical Society, Series B (Methodological)* 44:287–321.

Dain, O. M., D. W. Etherington, M. L. Ginsberg, E. O. Keenan, and T. B. Smith. 2005. Automated scheduling to minimize shipbuilding cost. In *Proceedings of the 12th International Conference on Computer Applications in Shipbuilding*, 41–54. Busan, Korea.

Keyes, T. K., and M. S. Levey. 1996. Goodness of prediction fit for multivariate linear models. *Journal of the American Statistical Association* 91:191–197.

McDevitt, M. E., M. W. Zabarouskas, and J. C. Crook. 2003. Ship repair workflow cost model. *MOR Journal* 10 (3): 25–44.

McKay, M. D., W. J. Conover, and R. J. Beckman. 1979. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* 21:239–245.

McLean, C., and G. Shao. 2001. Simulation of shipbuilding operations. In *Proceedings of 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 870–876. Arlington: The Society for Computer Simulation International.

R Development Core Team 2005. *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing.

Raiffa, H., and R. Schlaifer. 2000. *Applied statistical decision theory*. New York: Wiley.

Williams, D. L., D. A. Finke, D. J. Medeiros, and M. T. Traband. 2001. Discrete simulation development for a proposed shipyard steel processing facility. In *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 882–887. Arlington: The Society for Computer Simulation International.