3-25-1994

Rule-Based Run Control and Evaluation for Simulation

Robert F. Gordon Ph.D.
Molloy College, rfgordon@molloy.edu

Kow C. Chang

Edward A. MacNair

RC 19494 (84719) 3/25/94
Computer Science 7 pages

Research Report

Rule-Based Run Control and Evaluation for Simulation

Kow C. Chang, Robert F. Gordon and Edward A. MacNair

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

RULE-BASED RUN CONTROL AND EVALUATION FOR SIMULATION

Kow C. Chang
Robert F. Gordon
Edward A. MacNair

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598, U.S.A.

ABSTRACT

Modeling projects are often faced with a large parameter space that has to be explored in order to produce a set of performance measures representing the behavior of the systems under study. In this paper, we describe a software component that provides the analyst with the functionality to specify a design of experiments and execute a search algorithm over the resulting parameter space. The component invokes the associated simulation runs and compares the results to a goal to determine the solution. This component has been implemented as the run control mechanism in the RESearch Queueing Modeling Environment (RESQME).

We demonstrate the use of this experimental run control and evaluation component for simple enumeration of the parameter space, interactive evaluation, as well as generalized rule-based control.

1 INTRODUCTION

The developments in simulation software over the last decade have focused on providing graphical model specification and graphical output, including animation. The objectives of these graphical environments (Bell 1985, Bell 1991, Binnie and Martin 1988, Bishop and Balci 1990, Browne et al. 1985, Cox 1987, ElMaraghy 1982, Gordon et al. 1991, Hurrion 1986, Kurose et al. 1986, Sinclair, Doshi, and Madala 1985, White 1986) have been to make model building/debugging easier, faster, and more flexible and to improve the communication of the model and its results to other analysts and to management. The purpose of which is to arrive at better decisions by allowing the analyst to spend more time analyzing alternatives to make a recommendation and giving both the analyst and management more confidence in the decision. In this paper, we address the need to now provide tools to support the decision-making phase of the simulation process.

Because of the descriptive rather than normative aspect of simulation, it is necessary to design and run a series of simulation experiments, comparing the performance measures of each scenario, to arrive at a recommendation. For example, the analyst might vary a single parameter's value (say, number of servers) over a permissible range and examine the simulation output for each value to determine the performance improvement. He or she might weigh that improvement against the cost of the different parameter values to arrive at a decision. Realistically, the number of parameters and their possible value points may be quite large and the parameter interactions may be unclear a priori, resulting in a complex parameter space to examine.

The analyst needs a software tool to automate the specification of the parameter space and the specification of the algorithm to search that space and the goal to end the search. The software must then execute the algorithm to reach a decision, i.e., iteratively determine the next point in the parameter space, produce the associated simulation run, display and/or examine the performance results comparing them to the goal, and then either interactively or rule-based determine the next iteration or end the search. This experimental run control and evaluation tool therefore needs to provide functionality to:

1. Specify the experiment (parameter space and algorithm),
2. Execute its design, and
3. Compare and/or display the results and decide on the next action.

We describe in section 2 the experimental run control and evaluation component in the RESearch Queueing Modeling Environment, RESQME. Section 3 provides an example of its use. Section 4 summarizes its benefits to the analyst.

2 EXPERIMENTAL RUN CONTROL AND EVALUATION COMPONENT

The experimental run control and evaluation component in RESQME can be thought of as a control shell on top of the graphical simulator. This component supports simple enumeration of the parameter space, interactive evaluation, as well as generalized rule-based control.

In RESQME, we view the modeling process as consisting of three phases: create/edit the model, evaluate the model, analyze the results of the simulation. The analyst can move back and forth through these phases until a decision is made. We incorporated the experimental run control and evaluation component in the evaluate phase. When the analyst selects this phase from the command menu, a pop-up window appears. In that window, the analyst can specify the parameter space and the algorithm to execute the experimental design. The first prompt in the pop-up window allows the analyst to specify an upper limit to the number of simulation runs to produce:

```
MAX NO RUNS/EVALUATION: _
```

The analyst then uses a C-like syntax to specify the experiment, execute its design, compare the resulting performance measures, and determine the next action.

The language is comprised of the standard C operators, operands, statements, and blocks. The C built-in functions for *fabs* (absolute), *sqrt* (square root), *exp* (exponent), *ln* (natural log), *ceil, floor. max, min, pow* and *fmod* are supported.

The language supports as operators, the model performance measures. These are the keywords defined in the RESQ modeling language that store the results, such as QT for queueing time, QL for queue length, UT for utilization. They take as an argument the names the analyst gives to nodes and queues in the model. The

language operands include any symbolic names used in the model to refer to constants, variables, parameters, and network elements.

The C programming language constructs for while loops and if-else statements are supported. The C standards for expressions are followed. An expression becomes a *statement* when it is followed by a semicolon. Braces { and } are used to group statements into a compound statement or block, so that they are syntactically equivalent to a single statement.

Two keyword commands are added:

1. *Evaluate* initiates a simulation run based on the current values of the parameters.
2. *Quit* ends the execution of the rules.

As an example, we use this syntax to execute a simple enumeration of three parameter values. The analyst enters this code fragment in the above-mentioned pop-up window.

```
param1 = 0;
while (param1 < 0.5){     /* loop 1 */
  param1 += 0.1;
  param2 = 0;
  while (param2 < 0.4){   /* loop 2 */
    param2 += 0.1;
    param3 = 0;
    while(param3 < 0.3){  /* loop 3 */
      param3 += 0.1;
      evaluate;
    }
  }
}
```

Alternatively, the component allows the analyst to enter a matrix of parameter values to specify the enumeration. In this view, each parameter appears as the label of a row. The initial values of the parameters are entered by the analyst to form the first column. The analyst can supply the parameter values for each run desired. Column i contains the values for run i of the experiment. To reduce the input required, the last value given for a parameter is taken as fixed for any further runs of the experiment.

More generally, we provide an example of using the component's syntax for rule-based control of the simulation experiment. Here the control is used to execute a number of experiments, evaluating a performance measure, ql, (mean queue length) and adjusting a parameter, svrate1 (mean service rate for the server) until a stopping condition is reached (number of runs equals 10 or service rate reaches 0). The name given to the queue is "que1".

```
MAX NO RUNS/EVALUATION: 10
    svrate1 = 0.3;
    evaluate;                        /* first run */
    while (1) {
       if (ql(que1) <= 5){
          svrate1 -= 0.1;            /* decrease service rate */
          if (svrate1 <= 0)
             quit;
       }
       else
          svrate1 += 0.1;            /* increase service rate */
       evaluate;
    }
```

3 EXAMPLE

To illustrate using the experimental run control feature, we examine alternatives for a given model by varying two of its parameters. Each parameter can be varied over a range of nine values. The objective is to find the combination of parameter values that produces the worst case in terms of the mean queue length for a particular service center in the model.

The structure of the model is not of interest for this discussion; the model was constructed so that we could produce a convex response surface within the range of the parameter values, resulting in a single "optimum" point.

3.1 Simple Enumeration

We name the parameters *iat1* and *st1*. The *iat1* parameter will be varied over the nine values 1.8, 1.6, ..., 0.2. The *st1* parameter will be varied over the nine values 0.1, 0.3, ..., 1.7. We evaluated the model over all combinations of the two parameter values.

The following is the code fragment used to evaluate the model over the entire range of both parameter values.

```
iat1 = 1.8;
while (iat1>=0.2){
   st1 = 0.1;
   while (st1<=1.7){
      evaluate;
      st1 += 0.2;
   }
   iat1 -= 0.2;
}
```

This resulted in the eighty-one runs whose mean queue length values are shown in the following table.

P		Parameter 2								
a		0.1	0.3	0.5	0.7	0.9	1.1	1.3	1.5	1.7
r	1.8	0.0502	0.1679	0.3108	0.4919	0.8186	0.5377	0.3306	0.1732	0.0523
a	1.6	0.0665	0.2306	0.4573	0.7894	1.3567	0.8076	0.4582	0.2259	0.0645
m	1.4	0.0792	0.2813	0.5674	1.0316	2.0654	1.2058	0.6936	0.3246	0.0832
e	1.2	0.0929	0.3560	0.7775	1.5537	3.8715	1.6495	0.8050	0.3527	0.0935
t	1.0	0.1054	0.3973	0.8821	2.0201	7.6098	2.3581	0.9953	0.4210	0.1080
e	0.8	0.0900	0.3274	0.6914	1.3878	3.5007	1.5807	0.7895	0.3574	0.0917
r	0.6	0.0758	0.2607	0.5050	0.8668	1.8800	1.0967	0.6042	0.2863	0.0754
	0.4	0.0633	0.2181	0.4218	0.7042	1.1412	0.7242	0.4321	0.2159	0.0617
1	0.2	0.0518	0.1727	0.3168	0.5049	1.0972	0.6732	0.3796	0.1911	0.0562

3.2 Interactive

To illustrate the interactive approach to finding the worst case for the mean queue length, we select a number of points at random in the parameter space and evaluate the model at these points. From these runs, we pick the one with the largest queue length. At the largest one, we evaluate all eight surrounding points. If the selected point is the maximum, we stop. If it is not the maximum, we move to the maximum and evaluate any unknown surrounding points until we have a maximum.

We arbitrarily pick the four points with values for (iat1,st1) at (1.6,0.3), (0.8,0.5), (1.4,1.5), and (0.4,1.3). These points have mean queue lengths of 0.23056, 0.69141, 0.32464, and 0.43214, respectively. The code fragment in the run control component to perform these evaluations are:

```
iat1 = 1.6;
 st1 = 0.3;
evaluate;
iat1 = 0.8;
 st1 = 0.5;
evaluate;
iat1 = 1.4;
 st1 = 1.5;
evaluate;
iat1 = 0.4;
 st1 = 1.3;
evaluate;
```

The second one, (0.8,0.5), gives the maximum. We select this point and evaluate all unknown surrounding points using the following code:

```
iat1 = 1.0;
 st1 = 0.3;
while (iat1>.5){
   while (st1<0.7){
      evaluate;
      st1 += 0.2;
   }
   iat1 -= 0.2;
   st1 = 0.3;
}
```

This will reevaluate the known middle point, and we could avoid this reevaluation if we wanted to. Of these eight new evaluation points, (1.0,0.7) is the maximum at a mean queue length of 2.02010. We move to this point and evaluate five new points.

```
iat1 = 1.2;
 st1 = 0.5;
while (st1 < 0.9) {
   evaluate;
   st1 += 0.2;
}
iat1 = 1.0;
while (iat1 > 0.8) {
   evaluate;
   iat1 -= 0.2;
}
```

Of these five new evaluation points, (1.0,0.9) is the maximum at a mean queue length of 7.60982. We move to this point and evaluate three new points.

```
iat1 = 1.2;
 st1 = 1.1
while (iat1 > 0.8) {
   evaluate;
   iat1 -= 0.2;
}
```

This gives us the local maximum at (1.0,0.9).

3.3 Rule-Based

In practice, the nature of the response surface would not be known, and so we would not know exactly how to go about finding the maximum mean queue length. For illustrative purposes, we will assume that we know that there is one maximum and that the results are well-behaved as illustrated by the values presented in Section 3.1.

Given the nature of the results, a simple algorithm could be to start at a corner of the region, say the lowest value for both parameters (0.2,0.1) and run the simulation to get the mean queue length for that point. Then, increase iat1 to (0.4,0.1) and see if there is an improvement and compare it to increasing st1 to (0.2,0.3). Whichever is higher, move there and again increase separately iat1 and then st1. Compare and move to the point with higher mean queue length. When neither parameter change increases the mean queue length, stop. It would take seventeen evaluations to reach the optimal value and two more to confirm it with this algorithm. This can be accomplished with the following code where ql is the keyword for the mean queue length and node1 is the name of the node that we are interested in:

```
iat1 = 0.2;
 st1 = 0.1;
evaluate;
maximum = ql(node1);
while (1) {
   iat1 += 0.2;
   evaluate;
   next1 = ql(node1);
   iat1 -= 0.2;
    st1 += 0.2;
   evaluate;
   next2 = ql(node1);
   if (next1 > next2 && next1 > maximum) {
      iat1 += 0.2;
       st1 -= 0.2;
      maximum = next1;
   }
   if (next2 > next1 && next2 > maximum)
      maximum = next2;
   if (maximum > next1 && maximum > next2)
      quit;
}
```

4 SUMMARY

We described the experimental run control and evaluation component of RESQME and gave an example of its use in searching the parameter space. In that example, we considered search techniques from simple enumeration to interactive evaluation to generalized rule-based control.

Several recent developments have made it appropriate to incorporate such a component in simulation packages. First, current graphical, PC/Workstation-based, packages have made model building easier and faster, freeing the analyst from having to develop and debug textual programs to represent the real-world system. The result is that the analyst has the time and the inclination (not having invested so much in building one model) to examine different scenarios in making a decision. Second, the graphical depiction of the model and the results, combined with animation, enables the fast communication of the model, along with a range of alternatives, to the decision-makers. Third, the low-price processing power of the PC/Workstation platform supports executing search algorithms that could require a large number of simulation runs to examine the parameter space, especially considering that each point examined might require several replications.

The component we described gives the analyst direct access to the performance measures of each simulation run to compare results interactively as well as automatically. It further provides, through its C-like language, the ability to implement any user-defined search algorithm.

ACKNOWLEDGMENTS

We are grateful to the many colleagues and RESQME users who have helped improve RESQME over the years.

REFERENCES

Bell, P. C. 1985. Visual Interactive Modelling in Operational Research: Successes and Opportunities. *Journal of the Operational Research Society* 36:975-982.

Bell, P. C. 1991. Visual Interactive Modelling: The Past, the Present and the Prospects. *European Journal of Operational Research* 54:274-286.

Binnie, J. M., D. L. Martin. 1988. The Role of Animation in Decision-Making. In *Proceedings of the 1988 Winter Simulation Conference*, ed. M. A. Abrams, P. L. Haigh and J. C. Comfort, 272-276. Institute of Electrical and Electronics Engineers, San Diego, California.

Bishop, J. L., and O. Balci. 1990. General Purpose Visual Simulation System: A Functional Description. In *Proceedings of the 1990 Winter Simulation Conference* ed. O. Balci, R. P. Sadowski and R. E. Nance, 504-512. Institute of Electrical and Electronics Engineers, New Orleans, Louisiana.

Browne, J. C., D. Neuse, J. Dutton, and K.-C. Yu. 1985. Graphical Programming for Simulation of Computer Systems. In *Proceedings of the 18th Annual Simulation Symposium* ed. A. Miller, 109-126. IEEE Computer Society, Los Alamitos, California.

Cox, S. W. 1987. The Interactive Graphics and Animation of GPSS/PC. In *Proceedings of the 1987 Winter Simulation Conference* ed. A. Thesen, H. Grant and W. D. Kelton, 276-285. Institute of Electrical and Electronics Engineers, Atlanta, Georgia.

ElMaraghy, H. A. 1982. Simulation and Graphical Animation of Advanced Manufacturing Systems, *Journal of Manufacturing Systems* 1:53-64.

Gordon, K. J., J. F. Kurose, R. F. Gordon and E. A. MacNair. 1991. An extensible visual environment for construction and analysis of hierarchically-structured models of resource contention systems. *Management Science* 37:714-732.

Hurrion, R. D. 1986. Visual Interactive Modelling. *European Journal of Operational Research* 23:281-287.

Kurose, J. F., K. J. Gordon, R. F. Gordon, E. A. MacNair, and P. D. Welch. 1986. A Graphics-Oriented Modeler's Workstation Environment for the RESearch Queueing Package (RESQ). In *1986 Proceedings ACM/IEEE Fall Joint Computer Conference*, ed. H. S. Stone, 719-729. Institute of Electrical and Electronics Engineers.

Sinclair, J. B., K. A. Doshi, and S. Madala. 1985. Computer Performance Evaluation with GIST: A Tool for Specifying Extended Queueing Network Models. In *Proceedings of the 1985 Winter Simulation Conference* ed. D.T. Gantz, G.C. Blais and S.L. Solomon, 290-300. Institute of Electrical and Electronics Engineers, San Francisco, California.

White, D. A. 1986. PCModel and PCModel/GAF --- Screen Oriented Modeling. In *Proceedings of the 1986 Winter Simulation Conference*, ed. J.R. Wilson, J.O. Henriksen and S.D. Roberts, 164-167. Washington, District of Columbia.

Copies may be requested from:

IBM Thomas J. Watson Research Center
Distribution Services F-11 Stormytown
Post Office Box 218
Yorktown Heights, New York 10598