

# Heuristic Optimization as a V&V Tool for Software Process Simulation Models



Wayne Wakeland\*<sup>+</sup> Stephen Shervais and David Raffo  
 Portland State University, P.O. Box 751, Portland, OR 97207, USA  
 Eastern Washington University, 316 Kingston, Cheney, WA 99004 USA  
 Portland State University, P.O. Box 751, Portland, OR 97207 USA

## Research Section

This work illustrates the use of heuristic algorithms to improve the verification and validation of software process simulation models. To use this approach, an optimization problem is formulated to guide a heuristic search algorithm that will attempt to locate particular combinations of parameter values that yield surprising results. These surprising results often help the modeler to identify flaws in the model logic that would otherwise remain undetected. The general concepts are discussed and a simple example is provided. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: software process simulation; verification; validation; heuristic algorithm

## 1. INTRODUCTION

Verification and validation (V&V) are ongoing issues in the development of effective software and simulation models. It is simply not possible to test the full parameter space of large, complex software process simulation models (SPSMs), therefore V&V approaches emphasize a variety of methods, including bounds-checking, sensitivity analysis, and scenario analysis. This article discusses an idea we refer to as heuristic V&V (HVV), where the SPSM is exercised over a broad range of its parameter space using a heuristic search algorithm. This idea is not new, of course, but we were reminded of its value once again during research that employed genetic algorithms to optimize discrete event simulation models. In this case, the use of a genetic algorithm

unexpectedly led to the discovery of a subtle programming error in the simulation model. We then began to look for other examples where heuristic search algorithms can help with V&V. This article focuses on using HVV to improve the verification and validation of SPSMs.

Heuristic search algorithms have two characteristics useful for verification and validation of simulation models. First, they search a broad range of values in the model's parameter space. This allows testing of unusual combinations of parameter values that might not be found by *ad hoc* experimentation, bounds-checking, sensitivity analysis, and the like. Second, they attempt to 'exploit' any and all combinations of parameter values *whether or not such combinations actually make sense, and whether or not such combinations were intended by the designer.*

The concept is illustrated in this article using an SPSM that was developed using a commercial simulation language with linkages to a commercial optimizer that employs a scatter/tabu search algorithm. The HVV approach was able to detect an unintentional design error that was missed by *ad hoc* V&V methods. In HVV, the objective is *not* system

\* Correspondence to: Wayne Wakeland, Portland State University, P.O. Box 751, Portland, OR 97207, USA

<sup>+</sup>E-mail: wakeland@pdx.edu

Contract/grant sponsor: This work was supported in part by NASA Grant; contract/grant number: NAG5-12736



1 optimization per se, although the optimized results  
2 may be useful. Rather, the heuristic algorithm is  
3 used primarily as a means for thoroughly exploring  
4 the parameter space of the model.

5  
6

## 7 2. VERIFICATION AND VALIDATION

8

9 According to the DMSO glossary (DMSO 2005),  
10 verification is 'the process of determining that  
11 a model or simulation implementation accurately  
12 represents the developer's conceptual description  
13 and specification,' whereas validation is 'the process  
14 of determining the degree to which a model or  
15 simulation is an accurate representation of the  
16 real world from the perspective of the intended  
17 uses of the model or simulation'. The objective  
18 of verification is to determine that the intent of  
19 the modeler has been captured by the simulation,  
20 whereas the goal of validation is to determine  
21 that the model on which the simulation is based  
22 is an acceptably accurate representation of reality  
23 (Giannanasi *et al.* 2001) (Sadoun 2000).

24 It is quite difficult to verify computer simulation  
25 models. When attempting to do so, one is faced  
26 with the same problems that one must confront  
27 when verifying any complex computer program:

- 28 1. Large programs (or systems) have a number  
29 of residual bugs that will *never* be completely  
30 eliminated (Yourdon 1979, pg. 26).
- 31 2. Owing to the complexity of the simulation soft-  
32 ware, complete removal of bugs is infeasible.  
33 It has long been established in the practice  
34 of software engineering that complete removal  
35 of bugs by testing, even for relatively simple  
36 codes, is impossible from a practical standpoint  
37 (Froehlich *et al.* 2000, pg. 421).
- 38 3. It is almost impossible to verify totally a model  
39 for a complex system (Kelton *et al.* 2001, pg.  
40 513).

41  
42 As with verification, validation is also not a goal  
43 that can actually be achieved, but is rather an ideal  
44 for which to strive. Some authors insist that, since  
45 there is no such thing as absolute validity, the word  
46 validity should not even be used; instead, modelers  
47 should simply discuss how their model has been  
48 tested (Forrester and Senge 1980, pg. 123) (Law and  
49 Kelton 2000, pg. 279) (Serman 2000, page 846). See  
50 also (Balci 1998) for a thorough overview of model  
51 verification, validation, and testing.

David Kelton and his colleagues (Kelton *et al.* 52  
2001) propose one reason for the lack of trust in 53  
V&V – that interactions among the many different 54  
simultaneous activities occurring in a model can 55  
cause unintended consequences that can be very 56  
difficult to detect. This means the modeler must 57  
'develop tests that will allow you to ferret out the 58  
offending interactions or just the plain and simple 59  
modeling mistakes' (page 512). They recommend 60  
creating various scenarios and replicating these 61  
scenarios in the model to see that the model still per- 62  
forms adequately, and point out the importance of 63  
running the simulation for extended periods of time. 64

John Serman (Serman 2000, page 845) argues 65  
that the emphasis should be on model testing – the 66  
process to 'build confidence that a model is 67  
appropriate for the purpose' – not on V&V. His 68  
45-page discussion on model testing provides much 69  
detail on the model testing process, including the 70  
importance of determining whether or not the 71  
model is robust in the face of extreme variations in 72  
input conditions or policies. He suggests a number 73  
of 'assessment' tests that serve to thoroughly 74  
exercise a model over a wide range of parameter 75  
values. 76

Law and Kelton's often-cited discrete system sim- 77  
ulation textbook (Law and Kelton 2000) includes 25 78  
pages on the subject of building valid and credible 79  
simulation models. The authors recommend that 80  
the modeler 'run the simulation under a variety of 81  
settings of the input parameters and check to see 82  
that the output is reasonable' (page 270). Of interest 83  
to us is their insistence that 'The measures of perfor- 84  
mance used to validate the model should include 85  
those that the decision maker will actually use for 86  
evaluating system designs' (pg. 265), suggesting to 87  
us the possibility of using optimization methods 88  
based on plausible objective functions. 89

Regardless of whether one calls it V&V or simply 90  
testing, everyone agrees that it is important, and is 91  
often not done as thoroughly as it should be done. 92  
One way to increase the thoroughness of the testing 93  
process, without resorting to exhaustive parameter 94  
search, is to use a heuristic search algorithm, as we 95  
describe here in the context of SPSM. 96

97  
98

## 99 3. HEURISTIC SEARCH ALGORITHMS

100  
As mentioned in the introduction, heuristic search 101  
algorithms are interesting with regard to V&V 102



1 because they conduct a broad search of parameter  
2 space, and they seek to achieve a specific goal  
3 without regard for the intentions of the person who  
4 designed the simulation model. Let us look at each  
5 of these aspects in turn.

6 Often, when testing simulation models, a limited  
7 number of operating conditions are evaluated,  
8 focused perhaps in the neighborhood of current  
9 and/or highly plausible future operating parameters.  
10 These neighborhoods might not include  
11 regions of parameter space where possible flaws  
12 in model logic would become obvious.

13 The heuristic algorithm, however, concentrates  
14 its search through parameter space in those regions  
15 that provide high benefit in terms of the objective  
16 function. The algorithm has one goal, to optimize  
17 the fitness function, subject to whatever constraints  
18 and business rules the designer built into that  
19 function. Since the algorithm is only minimally  
20 constrained, it will 'exploit' whatever loopholes  
21 and errors it discovers in the course of its search.  
22 Errors might include poorly written branching logic  
23 or incorrectly defined process steps. The hope is  
24 that the heuristic search algorithm will identify  
25 combinations of parameter values that make the  
26 results of that error more obvious and easier to  
27 detect.

28 Scatter/tabu search is a heuristic search algorithm  
29 that incorporates decision rules and constraints as  
30 inputs (Glover 1998) (Glover *et al.* 2000). The process  
31 operates as follows:

- 32 1. A set of candidate solutions are generated using  
33 a problem-specific heuristic, and a subset of the  
34 best are designated as *reference solutions*.
- 35 2. Linear combinations of the elements of the  
36 reference solutions produce new solutions.
- 37 3. A new set of reference solutions is selected from  
38 the population, and the solution generation  
39 heuristic (1) is applied again.

40 Scatter search algorithms are readily available  
41 in commercial packages such as OptQuest from  
42 OptTek Systems, Incorporated. Such tools have also  
43 been integrated with commercial simulation tools,  
44 making it possible for HVV to be employed easily  
45 and quickly. For our illustrative application, we  
46 use the academic version of the Arena simulation  
47 software from Rockwell Software, which features  
48 an integrated version of OptQuest.

#### 4. APPLICATION OF HVV TO SOFTWARE PROCESS SIMULATION

52 One use, among many, for SPSMs is to find  
53 the 'appropriate' staffing level for a project,  
54 where appropriate means that a balance has been  
55 struck between low utilization (indicating improper  
56 resource balance) and lengthy project duration due  
57 to insufficient resources. Typically, performance  
58 improves dramatically when resources are added  
59 to a system that has insufficient resources. How-  
60 ever, owing to other constraints, there are typi-  
61 cally diminishing returns from adding additional  
62 resources beyond some point. If the logic of the  
63 system has not been properly modeled, then the  
64 performance of the system might appear to improve  
65 beyond what is actually possible.

66 The modeler may, based on prior experience,  
67 believe that the proper resource balance should  
68 occur when the ratio of coders to testers is, for  
69 example, approximately 2:1, and therefore might  
70 never try running the model with ratios far different  
71 than this. But, HVV would do exactly that, and,  
72 when it results in superior, but impossible results,  
73 logical error(s) in the model are likely to be revealed.

74 Consider the situation, common in SPSMs, where  
75 multiple resources are needed to complete a partic-  
76 ular task, such as a code review. The modeler  
77 might inadvertently fail to seize all of the necessary  
78 resources. In some cases, the syntax checker would  
79 catch this error. In other cases, watching an anima-  
80 tion and/or carefully studying the outputs might  
81 reveal the flaw to the modeler. But, in other cases,  
82 particularly with more complex models, such an  
83 error might remain in the model undetected.

84 By employing HVV and fully exploring different  
85 possible combinations of resources, the modeler is  
86 more likely to discover such a flaw. For example,  
87 HVV may report that it is possible to complete  
88 the software project more quickly by adding more  
89 testers and reducing the number of coders. This is  
90 likely to look suspicious to the modeler, and lead to  
91 discovery of the flawed logic.

##### 4.1. The Model

92 The model is simple, but not completely trivial. We  
93 wanted the model to be as simple as possible, but  
94 we learned as we developed the illustration that  
95 Arena, to its credit, does a good job of revealing  
96 obvious errors to the modeler, especially when the



1 logic is simple. In order to make a compelling case  
 2 for HVV, we felt that the flaw had to be subtle  
 3 enough that an experienced modeler could easily  
 4 miss it. This necessitated building a slightly more  
 5 complex test case than we had originally thought  
 6 would be necessary, as shown in Figure 1.

7 The model shown in Figure 1 represents a scoped  
 8 portion of a software development process. Units  
 9 of code to be developed are created at the start  
 10 of the simulation and assigned a unique id tag. A  
 11 copy of each code unit is routed along a second  
 12 path to initiate the creation of a test package for  
 13 each code unit. Code units are programmed and  
 14

15 then inspected. Inspection requires a walkthrough  
 16 with the coder and a tester. If the inspection reveals  
 17 errors, the coder does the rework and the code unit  
 18 is re-inspected. Once the code passes inspection,  
 19 the coder is released and the code unit is routed to a  
 20 special queue to await unit test.

21 Test packages are created and routed to the same  
 22 special queue module, where code units and test  
 23 packages are matched and grouped together on the  
 24 basis of their unique id. Unit test is then performed.  
 25 If the code unit passes, both the code unit and test  
 26 package are counted, and exit the model. In a more  
 27 complete model, the code units would be checked in  
 28

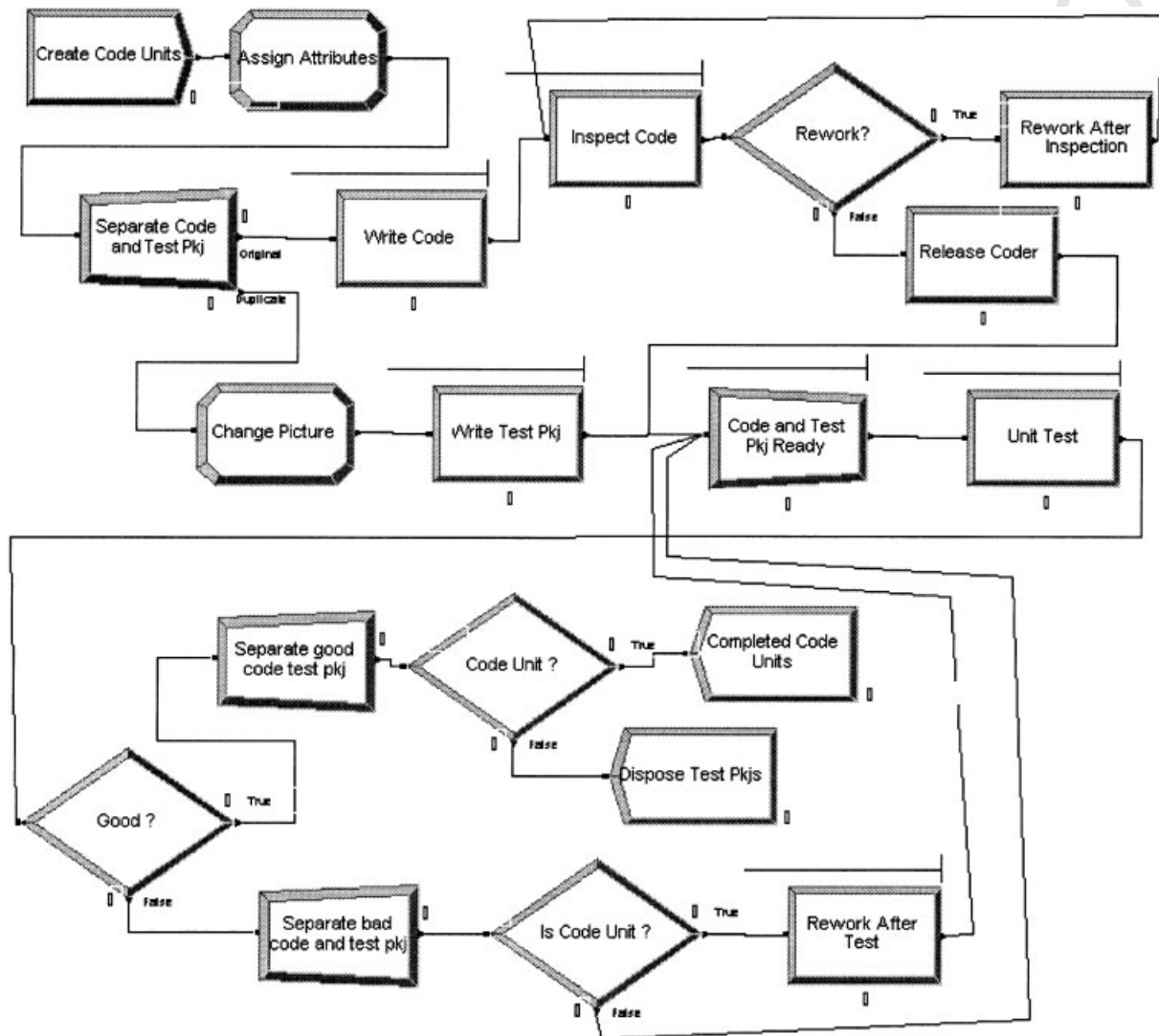


Figure 1. Arena software process simulation model for illustrating heuristic verification and validation



1 to await integration and system test, etc. Code units  
 2 that fail unit test are reworked, and routed back to  
 3 be re-matched with its test package and re-tested.  
 4 The associated test package is routed back as well,  
 5 since it will be needed again.

6 Task times are modeled using triangular proba-  
 7 bility density functions. The specific data utilized  
 8 is illustrative only. The 'Write Code' step requires  
 9  $T(1,2,4)$  days, where  $T(x, y, z)$  represents the tri-  
 10 angular probability distribution with a minimum,  
 11 mode, and maximum of  $x, y,$  and  $z$  respectively.  
 12 Other steps and their required times are as follows:  
 13 Inspect Code,  $T(1,3,8)$  hours; Rework after Inspect,  
 14  $T(1,4,8)$  hours; Write Test Package,  $T(.5,1,2)$  days;  
 15 Unit test,  $T(2,5,8)$  hours; and Rework after test,  
 16  $T(5,10,20)$  hours.

17 Initially, our plan was create a deliberate flaw in  
 18 the model that HVV would subsequently reveal.  
 19 This flaw was to have been to inadvertently seize  
 20 the tester to perform the rework after test, when it  
 21 should have been the coder. In fact, however, when  
 22 we applied HVV, we found that the model contains  
 23 a subtle flaw in its logic that we had not introduced  
 24 deliberately! This flaw is that the coder waits until  
 25 a tester is available to do the code review, which  
 26 is not realistic. This error was not easily caught  
 27 because the utilization figures for the resources  
 28 appeared to be plausible. Another reason the flaw  
 29 was subtle was the fact that, in some cases, or for  
 30 certain organizational processes, such logic might  
 31 *not* be flawed – people might actually wait until the  
 32 next work package arrives. We considered it a flaw

in our case because that was not our intention. The 33  
 HVV method served to reinforce one of the main 34  
 benefits of building process simulation models – to 35  
 help surface hidden assumptions. 36

Next, we describe the *ad hoc* process of experi- 37  
 menting with the model in order to determine the 38  
 'best' mix of resources. During this process, the 39  
 modeler specifies the number of resources (coders 40  
 and testers) that are available, and then runs the 41  
 model. At the end of each experiment, the modeler 42  
 reviews the output report to check the project dura- 43  
 tion and the utilization for each of the resources. 44  
 Each experiment consists of 20 replications in order 45  
 to assure sufficient sample size for estimating the 46  
 utilizations. The experimental objective is to find 47  
 the 'best' mix of resources that can be relied upon 48  
 to complete the project quickly. 49

After a number of iterations, the modeler has 50  
 determined that 6 coders and 3 testers is the 'best' 51  
 mix of resources. This resource level and mix is 52  
 also consistent with *a priori* expectations. Table 1 53  
 shows the output from Arena, indicating resource 54  
 utilizations, including confidence interval data for 55  
 these estimates. 56

Notice in Table 1 that the utilization of the 57  
 resources (coders and testers) is approximately 93%, 58  
 suggesting that a proper ratio had been established 59  
 and the total number was neither too high nor too 60  
 low. The project duration (not shown) ranged from 61  
 27.8 to 32.7 days, with a mean value of 30.7 days. 62  
 The duration is based on working 24 hours per day, 63  
 7 days per week. This figure must be multiplied by 64

Table 1. Output report from flawed arena model

ARENA simulation results					
Wayne Wakeland - License: Student					
Output summary for 20 replications					
Project: EVV for SPSM					Run execution date : 4/10/2004
Analyst: Wakeland					Model revision date: 4/10/2004
OUTPUTS					
Identifier	Average	Half-width	Minimum	Maximum	# Replications
CodeUnits.NumberIn	161.25	2.0701	154.00	170.00	20
CodeUnits.NumberOut	161.25	2.0701	154.00	170.00	20
Coder.NumberSeized	63.250	2.0701	56.000	72.000	20
Coder.ScheduledUtilization	.93668	.00954	.88281	.96197	20
Tester.NumberSeized	209.35	4.0836	189.00	224.00	20
Tester.ScheduledUtilization	.93907	.01730	.85985	.99661	20
System.NumberOut	49.000	.00000	49.000	49.000	20
Simulation run time: 0.05 minutes.					
Simulation run complete.					



1 about 4 to obtain actual project duration in calendar  
 2 days based on working 5 days per week, 8 hours  
 3 per day. This could have been calculated within  
 4 the model by introducing resource schedules, but  
 5 adding this complexity was not felt to be necessary  
 6 for this illustrative example. Thus, the 30.7 days  
 7 corresponds to the project duration was about  
 8 123 days or 6 months. The modeler was ready to  
 9 recommend this mix of resources for the project.

10  
 11 *4.1.1. The Experiment*

12 What might the modeler learn by utilizing HVV? To  
 13 apply HVV, one must first be more explicit about  
 14 what a 'good' result is. In this case, we want to  
 15 minimize cost, subject to the constraint that the  
 16 schedule is completed in each of the 20 replications  
 17 for a specific configuration of resources. In Arena,  
 18 costs may be easily incorporated by specifying  
 19 how much each resource costs per hour. In this  
 20 illustration, coders cost \$100 per hour whether they  
 21 are busy or idle, and testers cost \$50 per hour  
 22 whether they are busy or idle.

23 The HVV strategy then uses the OptQuest opti-  
 24 mization algorithm that is integrated with Arena  
 25 to explore different combinations of resources. If  
 26 there are flaws in the model, the [relatively] uncon-  
 27 strained optimization algorithm may find that it  
 28 can exploit those flaws and produce a lower cost  
 29 configuration – that may or may not actually make  
 30 sense. Even if the model is not flawed, OptQuest  
 31 may well find attractive solutions that the modeler  
 32 did not think of trying. To use OptQuest, we sim-  
 33 ply indicate the following: (a) the objective function  
 34 (minimize total cost), (b) which control variables  
 35 to experiment with (the number of each resource),  
 36 (c) the range to try for each control variable (very

broadly in the case of HVV – in this case, from 1 37  
 to 10 coders and testers), (d) constraints amongst 38  
 the values of the control variables (none in this 39  
 case), and (e) any 'requirements' (that all of the 40  
 code units must be completed for each replication, 41  
 and that the utilization for each resource must be 42  
 less than 0.95). Table 2 shows the results. In this 43  
 case, HVV indicates that the project will cost less if 44  
 we increase the number of resources by over 50%. 45  
 This does not make sense. This result was quite 46  
 robust regardless of the task times, and even robust 47  
 with respect to whether or not we introduce or 48  
 remove the flaw associated with whether or not the 49  
 correct resource is utilized to perform the Rework 50  
 after Test. We began to realize that HVV was indi- 51  
 cating a truly unintentional flaw in the model logic. 52  
 A closer inspection of the statistical reports showed 53  
 that while the coder is utilized at 93%, much of that 54  
 time is wait time rather than productive time. 55

56 This led to the realization that having coders  
 57 simply wait in the model until a tester became  
 58 available to do the code inspection is not correct.  
 59 Instead, the coder would start to work on another  
 60 code unit until a tester is available to do the code  
 61 review on the completed code unit. The model  
 62 logic was corrected to reflect this discovery. Table 3  
 63 shows the results of rerunning the optimization  
 64 after correcting the error.

65 Here, we see that in fact only 3 coders and  
 66 2 testers are able to complete the project, with  
 67 the same 93% utilization reported earlier. This is  
 68 very interesting! The project duration (not shown)  
 69 varied from 42 to 52 days, with a mean value of  
 70 47 days, which, as mentioned earlier, would be  
 71 188 days working 40 hours per week, or about  
 72 9 calendar months. Thus, if project cost is the

Table 2. HVV results on flawed model

Simulation	Minimize System. TotalCost	Optimizing...			Coder	Tester
		Requirement System. NumberOut 49 <= Value	Requirement Coder. Utilization Value <= .95	Requirement Tester. Utilization Value <= .95		
1	545 579	49.0000	0.938730	0.952586-Infesible	6	3
2	589 382	49.0000	0.924375	0.535514	6	6
7	570 967	49.0000	0.878959	0.923354	10	5
11	555 085	49.0000	0.909490	0.945184	8	4
12	552 112	49.0000	0.883171	0.821833	10	6
Best: 13	539 917	49.0000	0.907194	0.877228	9	5
Current: 78	614 724	49.0000	0.891522	0.515660	8	8



Table 3. HVV results with corrected model

Simulation	Optimizing...					Coder	Tester
	Minimize System. TotalCost	Requirement System. NumberOut 49 <= Value	Requirement Coder. Utilization Value <= .95	Requirement Tester. Utilization Value <= .95			
11	496 191	49.0000	0.895980	0.695603		8	7
13	492 347	49.0000	0.904216	0.715038		7	6
22	490 868	49.0000	0.895242	0.783879		8	6
23	480 494	49.0000	0.915391	0.748194		6	5
37	473 447	49.0000	0.903576	0.859275		7	5
Best: 38	443 420	49.0000	0.935244	0.927659		3	2
Current 77	537 170	49.0000	0.930894	0.561403		8	9

1 primary driver, the resource mix of 3 and 2 would  
 2 be preferred. One can see from Table 3, however,  
 3 that doubling the resources does not double the  
 4 cost. By adding resources, one can significantly  
 5 shorten the project duration. Thus, if time to market  
 6 is the driver, one might utilize more resources  
 7 even though it is somewhat more costly and  
 8 less efficient. The modeler can proceed to report  
 9 model results such as these to the client with  
 10 higher confidence than would have been warranted  
 11 without conducting HVV.

12  
 13 5. GENETIC VERSUS HEURISTIC  
 14 ALGORITHMS  
 15

16 We used the OptQuest optimizer because it was  
 17 readily available inside the Arena simulation tool,  
 18 but presumably *any* optimization technique is  
 19 capable of discovering regions of the solution space  
 20 that appear to be optimal, but do not actually  
 21 make sense in the real world. Some possible  
 22 search techniques depend on a cellular automata  
 23 metaphor, as in Stochastic Learning Automata  
 24 (Mars *et al.* 1996), a thermodynamic metaphor, as  
 25 in Simulated Annealing (Davis 1987), parameter  
 26 adaptation to local conditions – a mostly European  
 27 development, which includes Evolution Strategies  
 28 and Evolutionary Programming (Bäck 1996) – or  
 29 on a genetic metaphor via Genetic Algorithms  
 30 (Holland 1975). We chose to test the use of a Genetic  
 31 Algorithm (GA) as an example of an optimization  
 32 technique that is distinctly different from OptQuest.

33 In practice, a GA is often seeking to optimize  
 34 a fitness function. The way in which the function  
 35 itself evaluated is of limited theoretical, but  
 36 much practical, importance. It is most often some

37 computable value – given inputs X and Y, produce  
 38 output Z – but need not be. At the extreme, the  
 39 fitness of a proposed solution could be tested in  
 40 the physical world. In our case, we used the Arena  
 41 SPSM model as the fitness function.

42 A typical GA program begins by creating a popu-  
 43 lation of some specified number of (usually) binary  
 44 strings, each of which encodes a specific set of  
 45 characteristics – parameter values for the SPSM in  
 46 our case. The program then evaluates the *relative*  
 47 fitness of each member of the population, and  
 48 rank-orders them in terms of their fitness score.  
 49 The program then selects two of the top solutions  
 50 and creates a new individual through the *crossover*  
 51 operator. This operator splits the chromosomes of  
 52 the two parents and uses one piece from each to  
 53 build the child. It also does a small amount of gene  
 54 flipping – random changes of values – to simulate  
 55 *mutation*. The subroutine continues creating indi-  
 56 viduals until it has created a new population, often  
 57 with some portion of the highest scoring individuals  
 58 retained unchanged from the previous generation.  
 59 The genetic subroutine then passes this new popu-  
 60 lation back to the fitness function, and the process  
 61 repeats until stopping criteria are met.

62 We took an existing GA program, written in  
 63 Microsoft Visual Basic (VB) 6.0, and modified it  
 64 to call Arena, passing the set of parameters to  
 65 be run. The Arena model ran 20 replications for  
 66 the specified parameter values, and then used the  
 67 internal VBA functions of Arena to write summary  
 68 data to a file. In the GA, the number of Coders  
 69 and the number of Testers was represented by a bit  
 70 string corresponding to integer values between 1  
 71 and 10. The population of the GA was set at 30, and  
 72 the system was allowed to run for 10 generations.



1 Normally, the number of individuals and the  
2 number of generations would have been larger,  
3 but when we used larger values, the run times were  
4 unacceptably long, due, in part, to bookkeeping  
5 overhead necessitated by the prototype nature  
6 of our implementation. Even with the smaller  
7 population and smaller number of generations, a  
8 typical run took 45 minutes.

9 When applied to the flawed version of the Arena  
10 SPSM, the GA found a local optimum of 9 Coders  
11 and 4 Testers, which is almost identical to the  
12 OptQuest solution of 9 Coders and 5 Testers (note  
13 that because of the limited scale of the run, the GA  
14 did not actually try this particular combination).  
15 As was the case with HVV, this result would  
16 have been sufficiently surprising to motivate the  
17 modeler to carefully scrutinize the model logic and,  
18 presumably, find the flaw. However, since nothing  
19 new was discovered using the GA, and its run  
20 times were longer (without finding the OptQuest  
21 optimum), it would appear that the OptQuest  
22 heuristic approach is superior in this case. We do not  
23 know if there is a particular optimizer that works  
24 best for HVV in general.

## 25 26 27 6. CONCLUSIONS AND FUTURE WORK

28 We have illustrated how the use of heuristic  
29 algorithms may potentially enhance the verification  
30 and validation of software process simulation  
31 models. In this case, it helped to reveal a subtle  
32 model design flaw that went undetected during *ad*  
33 *hoc* testing and model experimentation.

34 However, there is no guarantee that by using  
35 HVV all important model errors will be detected. It  
36 might be the case, for example, that the flawed logic  
37 is rarely executed, and is therefore not triggered  
38 in the cases generated by the heuristic search  
39 algorithm. Further, it is well known that there is no  
40 guarantee that a heuristic algorithm will locate all of  
41 the pertinent locally optimal configurations. And,  
42 once located, it may not be possible to determine  
43 whether the configuration is truly a local optima or  
44 the result of subtle flaws in the logic of the model.

45 What HVV does do well is to exercise the model  
46 over a broad region of the input parameter space,  
47 and this significantly increases the likelihood that  
48 the modeler will uncover hidden flaws in the model  
49 logic that may otherwise go undetected. The fact  
50 that heuristic optimization algorithms are now  
51

being bundled with popular simulation software 52  
packages makes it surprisingly easy to perform 53  
HVV analysis. Given this ease of use, there is no 54  
reason not to include HVV as a standard component 55  
of the SPSM model testing process. 56

This work represents our initial efforts at explor- 57  
ing how optimization-based HVV can be used as 58  
an adjunct to other V&V tools. There are additional 59  
questions that can be asked about the topic. For 60  
example, we do not know if there is a preferred 61  
optimizer for this kind of a task. Much of the cur- 62  
rent literature does not apply, since the role of the 63  
optimizer is not actually to optimize the system, 64  
but instead to search the solution space for unreal- 65  
istic solutions. Similarly, question of scaling arises; 66  
many simulations are much larger than the model 67  
we used here, will HVV work on those systems? 68  
It would appear, then, that the optimizer selection 69  
criteria should be based on coverage and efficiency, 70  
rather than on an ability to find global optima. These 71  
topics will be the subject of future work. 72

## 73 74 75 REFERENCES

- 76  
77 Bäck T. 1996. *Evolutionary Algorithms in Theory and*  
78 *Practice*. Oxford University Press: New York, USA.  
79  
80 Balci O. 1998. Verification, validation, and testing. *The*  
81 *Handbook of Simulation*, Banks J (ed.). John Wiley and  
82 Sons: New York, USA, Chapter 10.  
83  
84 Davis L (ed.). 1987. *Genetic Algorithms and Simulated*  
85 *Annealing*. Pitman Publishing: London, UK.  
86  
87 DMSO. 2005. Defense Modeling and Simulation Office.  
88 <https://www.dmsomil/public/resources/glossary/>  
89 (accessed 1/15/2005).  
90  
91 Forrester JW, Senge PM. 1980. Tests for building  
92 confidence in system dynamics models. *TIMS Studies*  
93 *in the Management Sciences* **14**: 209–228.  
94  
95 Froehlich G, Ogden BA, Bylec K. 2000. Software quality  
96 assurance in the 1996 performance assessment for the  
97 waste isolation pilot plant. *Reliability Engineering and*  
98 *System Safety* **69**: 1–3.  
99  
100 Giannanasi F, Lovett P, Godwin AN. 2001. Enhancing  
101 confidence in discrete event simulations. *Computers in*  
102 *Industry* **44**: 141–157.  
103  
104 Glover F. 1998. A template for scatter search and path  
105 relinking. In *Lecture Notes in Computer Science, 1363*, Hao J-  
106 K, Lutton E, Ronald E, Schoenauer M, Snyers D (eds).  
107 Springer-Verlag: Heidelberg, Germany, 13–54.



1	Glover F, Laguna M, Martí R. 2000. Fundamentals of	Mars P, Chen J, Nambiar R. 1996. <i>Learning Algorithms: Theory and Applications in Signal Processing, Control, and Communications</i> . CRC Press: Baton Rouge, LA.	13
2	scatter search and path relinking. <i>Control and Cybernetics</i>		14
3	<b>39</b> (3): 653–684.		15
4	Holland J. 1975. <i>Adaptation in Natural and Artificial</i>	Sadoun B. 2000. Applied system simulation: a review	16
5	<i>Systems</i> . The University of Michigan Press: Ann Arbor,	study. <i>Information Sciences</i> <b>124</b> : 173–192.	17
6	MI.		18
7		•, Stermann JD. 2000. <i>Business Dynamics: Systems Thinking</i>	19
8	Kelton WD, Sadowski RW, Sadowski DA. 2001. <i>Simula-</i>	<i>and Modeling for a Complex World</i> . Irwin/McGraw-Hill.	20
9	<i>tion with Arena</i> , 2nd edn. McGraw-Hill: London, UK.		21
10		Yourdon E. 1979. <i>Managing the Structured Techniques</i> .	22
11	Law AM, Kelton WD. 2000. <i>Simulation Modeling and</i>	Yourdon Press: Raleigh, NC.	23
12	<i>Analysis</i> , 3rd edn. McGraw-Hill: New York, USA.		24

AQ1

UNCORRECTED PROOFS

1		52
2	<b>QUERIES TO BE ANSWERED BY AUTHOR</b>	53
3		54
4	<b>IMPORTANT NOTE: Please mark your corrections and answers to these queries directly onto the proof</b>	55
5	<b>at the relevant place. Do NOT mark your corrections on this query sheet.</b>	56
6		57
7	<b>Queries from the Copyeditor:</b>	58
8	AQ1 Please provide the place of publication for this reference.	59
9		60
10		61
11		62
12		63
13		64
14		65
15		66
16		67
17		68
18		69
19		70
20		71
21		72
22		73
23		74
24		75
25		76
26		77
27		78
28		79
29		80
30		81
31		82
32		83
33		84
34		85
35		86
36		87
37		88
38		89
39		90
40		91
41		92
42		93
43		94
44		95
45		96
46		97
47		98
48		99
49		100
50		101
51		102

UNCORRECTED PROOFS