Proceedings of the 2005 International Conference on Simulation and Modeling
V. Kachitvichyanukul, U. Purintrapiban, P. Utayopas, eds.

A GENETIC ALGORITHM FOR THE MULTICRITERIA OPTIMISATION OF COMPUTER SIMULATION MODELS

F. Van Utterbeeck, R. Van Loock and H. Pastijn

Royal Military Academy
Avenue de la Renaissance, 30
B-1000 Bruxelles, BELGIUM

ABSTRACT

We try to find the best configuration of a stochastic discrete-event simulation model when multiple criteria are simultaneously involved in the decision making process. We develop and investigate the performance of a genetic algorithm that makes use of an interval-based variant of the Prométhée MCDM-method to calculate the fitness of the chromosomes. We illustrate the performance of this algorithm by applying it for the optimization of a simulation model for incident management in a call centre.

1 INTRODUCTION

While the optimization of simulation models is extensively studied in the literature, it is almost exclusively done from a single-response point of view. In reality however, one often encounters problems where the assessment of the behaviour of a system depends on multiple performance measures. In these cases, the solution that is perceived as the "best" solution will often turn out to be a compromise solution, which may differ significantly from the optimal solutions that would be found when following a single-response approach. Different system configurations will typically improve some performance measures while deteriorating others. The selection of the best candidate (system configuration) among a finite set of alternatives assessed for a finite set of criteria (performance measures) is a typical multicriteria decision making problem.

The algorithm that we propose in this paper combines a classical MCDM-method with a genetic algorithm, which enables us to use the algorithm for the selection of the best system configuration in a combinatorial optimization problem, where the set of candidates, although finite, is prohibitively large.

This paper is organized as follows. In section 2, we start with a general description of the studied optimization problem. In section 3, we give a brief overview of the Prométhée MCDM-method and its interval-based variant Prométhée-i. In section 4, we propose a genetic algorithm that makes use of the Prométhée-i method to calculate the fitness of the chromosomes. In section 5, we give a brief description of a simplified call centre model. In section 6, we present the results of a series of experiments that made use of our genetic algorithm to optimize the configuration of the simulation model proposed in section 5. Finally, we present our conclusions.

2 DESCRIPTION OF THE OPTIMISATION PROBLEM

Our aim is to find the best feasible configuration (according to the Prométhée MCDM-method, which we describe in the following section) for a stochastic discrete-event simulator when the number of possible configurations is prohibitively large (i.e. evaluating all the candidates is impossible within the allocated time-budget).

Every configuration of the simulator can be defined by a vector $\boldsymbol{\alpha} = (x_1, x_2, \ldots, x_m)$, where x_i represents the setting for the i^{th} input parameter of the simulator. The n corresponding performance measures are defined by the vector $\boldsymbol{\beta} = (y_1, y_2, \ldots, y_n)$ where $\boldsymbol{\beta} = f(\boldsymbol{\alpha})$. We consider the form of the function f to be unknown, the values of the various performance measures are estimated using simulation. A configuration is considered feasible if a given set of constraints of the form $g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \geq c$ are simultaneously satisfied. Note that the feasibility of a configuration can in general only be verified after the execution of the simulation, as the feasibility constraints are functions of the performance measures.

We define the best configuration $\boldsymbol{\alpha}_{opt}$ as the feasible configuration that would be selected by the Prométhée MCDM-method if we applied this method on the set of all possible feasible configurations.

3 THE PROMETHEE AND PROMETHEE-i MCDM-METHODS

For a complete overview of the original Prométhée method we refer to Brans et al. (1986). This MCDM-method is based on the assessment of a finite number of *n* candidates (configurations) on *k* criteria (performance measures).

For each criterion a pairwise comparison (difference of assessment) of candidates *a* and *b* is translated on the interval [0,1] into a preference indicator $P_j(a,b)$. One of six different types of preference indicator functions can be selected. These $P_j(a,b)$ are aggregated over the set of all criteria by :

$$\pi(a,b)= \Sigma_j \omega_j \, P_j(a,b),$$

with ω_j in [0,1] being the normalized weight of criterion *j*. Then we calculate for each candidate *a* the strength $\phi^+(a)$ and the weakness $\phi^-(a)$:

$$\phi^+(a)=(1/(n-1)).\Sigma_x \pi(a,x),$$
$$\phi^-(a) =(1/(n-1)).\Sigma_x \pi(x,a).$$

Finally we calculate the net dominance $\phi(a)$:

$$\phi(a) = \phi^+(a) - \phi^-(a).$$

The best alternative is the one with the highest net dominance.

The Prométhée-i method is an interval-based variant of this method that can be used when the assessments are not crisp data, but are defined by intervals. Pastijn et al. (2003) showed how this method can be used for the selection of an optimal configuration amongst a limited number of candidates for the configuration of a stochastic discrete-event simulator. The crisp assessments of Prométhée are then replaced by either the interquartile interval or a confidence interval of the mean. They compared the application of the original Prométhée method using the mean values across replications with the Prométhée-i method and found that the use of Prométhée-i with interquartile intervals generally gives the best results (requiring the smallest number of replications for a correct ranking). The following paragraph summarizes the method:

When we have executed *m* replications of a stochastic discrete-event simulation, then we obtain for each alternative configuration *m* assessments for each criterion (performance measure). These *m* assessments can be represented for alternative *a* by an interval $[a^l,a^u]$, which we can take either as the interquartile interval, or as a confidence interval on the mean. All the arithmetic of Prométhée is now extended, keeping intervals all along the calculations, by means of the following definitions:

$$[a^l,a^u] + [b^l,b^u] = [a^l+ b^l \, , \, a^u+ b^u],$$
$$[a^l,a^u] - [b^l,b^u] = [a^l- b^u \, , \, a^u- b^l].$$

We obtain consecutively:

$$P_j(a,b) = [P_j^l(a,b) \, , \, P_j^u(a,b) \,],$$
$$\pi(a,b) = [\pi^l(a,b) \, , \, \pi^u(a,b) \,],$$

where

$$\pi^l(a,b) = \Sigma_j \omega_j \, P_j^l(a,b) \text{ and}$$
$$\pi^u(a,b) = \Sigma_j \omega_j \, P_j^u(a,b).$$

Then we calculate

$$\phi^+(a) = [\phi^{+l}(a) \, , \, \phi^{+u}(a)],$$
$$\phi^-(a) = [\phi^{-l}(a) \, , \, \phi^{-u}(a)],$$

where

$$\phi^{+l}(a) =(1/(n-1)).\Sigma_x \pi^l(a,x),$$
$$\phi^{+u}(a) =(1/(n-1)).\Sigma_x \pi^u(a,x) \, ,$$
$$\phi^{-l}(a) =(1/(n-1)).\Sigma_x \pi^l(x,a),$$
$$\phi^{-u}(a) =(1/(n-1)).\Sigma_x \pi^u(x,a).$$

And finally

$$\phi(a) = \phi^+(a) - \phi^-(a) = [\phi^l(a) \, , \, \phi^u(a) \,].$$

In addition the original Prométhée method is applied by taking into account all the worst bounds of the assessment intervals $[a^l,a^u]$ and another time by taking all the best bounds of these assessment intervals $[a^l,a^u]$ for all candidates on all criteria. This yields for each candidate another interval $[\phi^{l'}(a) \, , \, \phi^{u'}(a) \,]$.

Finally this Prométhée-i procedure returns a trapezoidal fuzzy number $[\phi^l(a) \, , \, \phi^{l'}(a) \, , \, \phi^{u'}(a) \, , \, \phi^u(a) \,]$ for each candidate *a*. On these fuzzy numbers we apply the Yager operator Ψ (Yager, 1981), (Detyniecki et al., 2001), and the best candidate corresponds to the highest value for this Yager operator Ψ.

4 THE GENETIC ALGORITHM

For an in-depth discussion of genetic algorithms, we refer the reader to Goldberg (1989), Davis (1991), Chambers (1995) and Reeves (1997). The pseudo-code of our genetic algorithm is represented below:

```
Program Begin
Generate Random First Generation of Chromo-
somes;
While Stopping Criterion not reached
   Begin
      Current chromosome =
      First Chromosome of Current Generation;
      Repeat
         Initialize Simulation Model with
         Configuration Represented by
         Current Chromosome;
         Run m Replications of the Simulation;
         Save Performance Measures;
         Current Chromosome =
         Next Chromosome of Current
         Generation;
      Until End Of Generation Reached
      Rank Configurations of Current Genera-
      tion with Prométhée-i;
      Modify Ranking to Promote Feasible
      Configurations;
      Attribute Fitness to Chromosomes;
      Generate New Current Generation;
    End
Program End
```

The following paragraphs briefly describe the different features and options that have been implemented in the different parts of the code.

4.1 Chromosome representation

The chromosomes in our algorithm are strings of bits, whose problem-dependent size varies depending on the number and value range of the input parameters. Every possible configuration $\boldsymbol{\alpha} \in A$ has to be mapped into exactly one chromosome. The first generation is generated randomly.

4.2 Stopping Criterion

The stopping criterion can be defined either as a maximum number of generations or as a maximum number of non-improving iterations.

4.3 The Simulation

The replication length and the number of replications used are problem dependent and require some experience with the underlying simulation model. Our simulation models were implemented in ARENA, and we make use of common random numbers to reduce the variability of the difference between the performance measures for two configurations.

4.4 Prométhée-i ranking

All configurations in the current generation are ranked using the Prométhée-i method based on the interquartile intervals for the elements of $\boldsymbol{\beta}$, the vector of performance measures.

4.5 Promotion of feasible configurations

This procedure verifies whether the performance measures of the configurations fulfill the set of i constraints $g_i(\boldsymbol{\alpha}, \boldsymbol{\beta}) \geq c_i$. Every constraint has an associated weight. We define the feasibility score of a configuration as the sum of the weights of all the satisfied constraints.

The final ranking of the configurations is now calculated as follows: all configurations are ranked by decreasing feasibility score. All ties are broken in favour of the configuration with the highest Prométhée-i ranking. If no constraints were imposed, then the Prométhée-i ranking remains unmodified.

4.6 Attribute Fitness

The chromosomes are assigned a fitness value that is a (scalable) linear function of their final ranking.

4.7 Creation of the next generation

The algorithm makes use of elitism, crossover and mutation.

The elitism ensures that a specified number of the best chromosomes are copied into the next generation without modification. Moreover, the first-ranked chromosome is marked as immune to mutation. Not only does this guarantee the survival of the fittest genes, it also removes the need to save the best configuration.

One of three different crossover operators can be selected: the classic one-point and two-point crossover, and uniform crossover. The uniform crossover copies a bit-value from parents to offspring if both parents' chromosomes agree on the value of the specified bit position. The bitvalue is randomly assigned if the parents disagree.

One of two different parent-selection methods can be selected: the classic roulette-wheel selection and the less well-known stochastic universal selection (Baker 1987).

Mutation takes place using the classical mutation operator.

Crossover and mutation rate can be fixed constants, or can be selected to increment/decrement linearly between an initial value and a final value.

Whenever a new child chromosome has been generated, we verify whether it is unique within the current generation. If an identical chromosome already exists, we generate another child and replace the duplicate chromosome.

5 THE SIMULATION MODEL AND THE PERFORMANCE MEASURES

The stochastic discrete-event simulator used is a model of an incident management process of a call centre. A complete description of this model can be found in Van Loock et al. (2003). We briefly summarize the key elements of this simulation below. Figure 1 gives a schematic overview of the process-flow of the model.

The *incidents* are initiated by the customers of the call centre. These incidents are represented by the calls that arrive at the centre. These *incoming calls* follow a stochastic arrival pattern. The calls are subdivided into categories and subcategories, depending on the area of expertise required by the customer. Each category has a specified probability of occurrence, while the subcategories within a certain category are assumed to be equiprobable.

The *resources* in our model are the dispatcher(s) and the system engineers. Each resource has its own weekly working schedule, an hourly cost (based on the number of skills known) and a FIFO queue associated with it. Incoming calls will wait in the FIFO queue if the resource is busy. A call will be *rejected* (and leaves the system immediately) if the time spent waiting in a FIFO queue exceeds a certain fixed threshold.

Every system engineer has his own areas of expertise, which are specified in the *skills matrix*. Every line in the matrix represents a subcategory, while every column represents a system engineer.

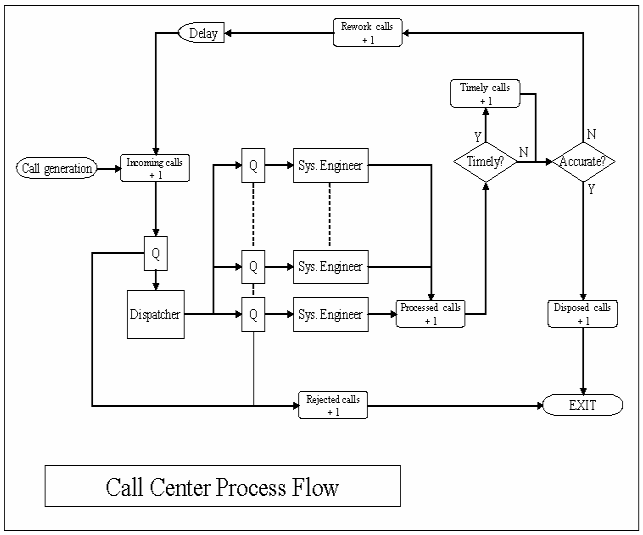

Figure 1: Call Centre Process Flow

The *process flow* used in our model can be summarized as follows. Every incoming call must pass through a dispatcher. The dispatcher will route the call to a system engineer whose area of expertise covers the category and subcategory of the call. If multiple system engineers are eligible, the dispatcher will route the call to the resource with the shortest queue. Ties are broken in favour of the resource located the most to the left in the skills matrix. The *dispatching time* (the time needed by the dispatcher to decide on the routing of the call) follows a stochastic distribution.

The *processing time* (the time the system engineer needs to handle a call) follows a stochastic distribution, regardless of the subcategory. For every *processed call*, there is a fixed probability that the customer is not completely satisfied with the assistance provided. These customers will call back after a stochastic delay. These subsequent *rework calls* will result in a decrease in the performance of the call centre. If the customer is satisfied with the assistance provided, the call is *disposed* and leaves the system.

We use four performance measures: waiting times in queues, resource utilization or productivity, service level and system cost. Waiting times in queues and resource utilization are average values obtained from standard ARENA statistics. Service level is expressed as the percentage of arriving calls which are finally disposed after a successful handling by the available resources (and as a consequence were not ejected from the system). The cost of a system engineer depends on his degree of polyvalence (number of skills). The overall system cost is a stochastic entity due to the fact that the resources continue to work at the end of their daily schedule until all calls waiting in their queue at the end of the working day have been processed.

6 EXPERIMENTAL RESULTS

6.1 Problem description

We want to determine the best combination of input parameters (the number of dispatchers, the number of system engineers and the elements of skills matrix) for our call centre simulation. For this experiment, we define the range for the of number of dispatchers as 1 to 4, the range for the number of system engineers as 1 to 10 and the size of the skills matrix as 6 rows (corresponding to the number of skills in our model) multiplied by the number of columns (corresponding to the number of system engineers).

We will compare the results of 5 scenarios. The first scenario, labeled "MCDM" uses 4 criteria. The best configuration will be determined using the Prométhée parameters summarized in table 1 below. We refer to Brans et al. (1986) for a detailed description of the 6 types of preference function and the associated parameters p and q.

Table 1: Prométhée Parameters

Performance Measure	Weight	Min/max	Type	Q	P
Productivity	6	Max	5	0.1	10
Service Level	8	Max.	3	0	5
Queue W. Time	3	Min.	6	0.5	0
Cost	9	Min.	3	0	500

We impose 4 equally important feasibility constraints, one for each performance measure, as summarized in table 2 below.

Table 2: Feasibility Constraints

Performance Measure	Weight	Constraint
Productivity	1	≥ 50%
Service Level	1	≥ 90%
Queue W. Time	1	≤ 2 minutes
Cost	1	≤ 5000 Euro

The other 4 scenarios will be used as a benchmark. The setup of these scenarios is completely identical to the first scenario, except for the fact that we only try to optimize one of the four performance measures (using the corresponding parameters from table 1), while imposing the same 4 constraints. These scenarios are labeled "Max Productivity", "Max Service Level", "Min Queue Waiting Time" and "Min Cost".

6.2 Chromosome Representation

Our chromosome representation requires 2 bits to represent the number of dispatchers. The skills matrix has 6 rows (constant, equal to the number of skills in the model) and 1 to 10 columns (variable, equal to the number of system en-

gineers) and consists of elements that can be either true (if the engineer possesses the skill) or false. This requires 10 groups of 6 bits. These 60 bits implicitly define the number of system engineers, as every group containing at least one bit set to true corresponds with a system engineer. This gives us a fixed chromosome length of 62 bits per configuration. To obtain a unique one-to-one relationship between configurations and chromosomes, we need to apply a transformation to the chromosomes: the 10 bit groups representing the engineers are ranked from left to right in decreasing order of the integer value corresponding with their 6-bit code. This ensures that "empty" bit groups (which are present in configurations where the number of system engineers is inferior to 10) are shifted to the right-hand side of the chromosome.

6.3 Initialization

The initialization phase is an important step in the algorithm, as we have to ensure that the selected initial chromosomes cover the entire search-space as uniformly as possible. Failure to do this might cause our algorithm to overlook potentially interesting portions of the search-space. To ensure this property, every initial chromosome is generated in two steps: first we randomly determine the number of dispatchers and system engineers, and then we randomly generate as many 6-bit sequences as we have system engineers. At least one bit in these sequences has to be set to true. The chromosome is then right-filled with zeros.

6.4 Results

Table 3 below contains the performance measures corresponding to the optimal configurations found by our 5 scenarios.

All numerical results were obtained using a stopping criterion of 200 generations or 15 non-improving generations, a generation size of 40, elitism for the top 4 chromosomes, a crossover-rate decreasing from 60% to 20% by 1% decrements, a mutation rate increasing from 0.1% to 0.5% by increments of 0.01% and a linear fitness decreasing from 100 to 2.5 by 2.5 unit decrements.

A comparison of the MCDM-scenario with the other four scenarios clearly shows that the configuration selected by this scenario represents a middle ground between the other four. If we focus e.g. on the productivity performance measure, we note that the MCDM-scenario returns the second-best result. Max Productivity obviously returns the highest productivity, while in 2 out of the other 3 scenarios the productivity drops down to just above the imposed lower constraint of 50%.

Our experiments indicate furthermore that our algorithm is quite robust. Consecutive runs of the algorithm give very comparable results, as can be seen in table 4, which shows the mean, standard deviation, minimum and maximum values of the performance measures for the optimal configuration obtained during 15 consecutive runs of the MCDM-scenario. The variability in the observed performance measures is of the same order of magnitude as the variability that is observed when executing 15 replications of the same configuration.

Table 3: Experimental Results

	Productivity	Service Level	Queue W. Time	Cost
MCDM	56,86	90,46	0,62	1449
Max Productivity	**64,6**	90,09	0,51	2402
Max Service Level	50,9	**97,11**	0,18	4080
Min Queue Waiting Time	50,7	96,73	**0,18**	3756
Min Cost	54,71	90,12	0,52	**1236**

Table 4: MCDM-scenario Results

	Productivity	Service Level	Queue W. Time	Cost
mean	56,86	90,46	0,62	1448,80
std dev	1,37	0,35	0,06	54,59
min	54,87	90,05	0,49	1356,00
max	58,95	91,16	0,68	1557,00

7 CONCLUSIONS

We have described a genetic algorithm that allows us to find the optimal configuration for a stochastic discrete-event simulator when multiple performance measures have to be considered simultaneously. This type of algorithm may prove particularly interesting when the decision making authority is shared by multiple decision makers with conflicting priorities. The optimal solutions found with this algorithm typically represent a middle ground solution that may be acceptable to all the involved parties.

The multi-criteria approach relies on an interval-based variant of the Prométhée method, which is combined with a feasibility score to obtain the ranking of the chromosomes within a certain generation of the genetic algorithm.

REFERENCES

Baker J.E. 1987. *Reducing Bias and Inefficiency in the Selection Algorithm*, in Proceedings of the 2nd International Conference on Genetic Algorithms, J.J. Grefenstette (ed.), Lawrence Erlbaum Associates, Hillsdale, NJ: 14-21

Brans J.P., P. Vincke and B. Mareschal. 1986. *How to Select and How to Rank Projects: The Prométhée Method*, European Journal of Operational Research, 1986(24): 228-238

Chambers L.(ed.). 1995. *Practical Handbook of Genetic Algorithms: Applications, Volume I*, CRC Press, Boca Raton, Fl.

Chambers L.(ed.). 1995. *Practical Handbook of Genetic Algorithms: New Frontiers, Volume II*, CRC Press, Boca Raton, Fl.

Davis L.(ed.). 1991. *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

Detyniecki M. and R.R. Yager. 2001. *Ranking Fuzzy Numbers Using α-weighted Valuations*, International Journal of Uncertainty, Fuzziness and Knowledge-based Systems, Vol.8(5), 2001: 573-592

Goldberg D.E. 1989. *Genetic Algorithms in Search, Optimisation and Machine Learning*, Addison-Wesley, Reading, MA.

Le Téno J.F. and B. Mareschal. 1992. *An Interval version of Prométhée for the Comparison of Building Products' Desig with Ill-defined Data on Environmental Quality*, unpublished working paper, Université Libre de Bruxelles

Pastijn H., F. Van Utterbeeck and R. Van Loock. 2003. *Promethee-i selecting the best simulation model configuration based on multiple performance measures*, ESS2003, Delft, The Netherlands, Oct 26-29 2003

Reeves C.R. 1997. *Genetic Algorithms for the Operations Researcher*, INFORMS Journal on Computing, Vol.9, No.3: 231-250

Van Loock R. and H. Pastijn. 2003. *Performance Measures in Simulating Incident Management Processes of a Call Centre*, Orbel 17, Brussels, Belgium, Jan 23-24 2003

Yager R.R. 1981. *A Procedure For Ordering Fuzzy Subsets of the Unit Interval*, Information Science, 1981(24): 143-161

AUTHOR BIBIOGRAPHIES

F. VAN UTTERBEECK Email: filip.van.utterbeeck@rma.ac.be

R. VAN LOOCK

H. PASTIJN