Adaptive Stochastic Search Methods for Parameter Adaptation of Simulation Models

George D. Magoulas, Tillal Eldabi, and Ray J. Paul

***Abstract*—Adaptive stochastic search methods are expected to lead to "optimal" or "near-optimal" configurations of a simulation model as they manage to escape from sub-optimal (local) solutions. In that sense, they provide an automated "optimization" approach that adapts the parameters of a model in order to handle uncertainty that arises from stochastic elements in either the environment (process noise/concept drift) or the objective function evaluation process (observation noise) and improves the performance of the model. The paper reviews the fundamentals of adaptive stochastic search methods and explores their behavior for the adaptation of the parameters of a steelworks model. Experimental results illustrate the effectiveness of the methods, and particularly of swarm intelligence in this task.**

***Index Terms*—Adaptive stochastic search methods, Optimization, Parameter adaptation, Uncertainty, Simulation modeling.**

I. Introduction

IT has long been established that simulation is a powerful tool for aiding decision-making. This is due to a number of factors, such as the inherent ability to evaluate complex systems with large numbers of variables and interactions for which there is no analytic solution.

Simulation can model the dynamic and stochastic aspects of systems, generating more precise results when compared with static and deterministic spreadsheet calculations. It is also considered as a tool to answer "What if" questions. Simulation itself is a solution evaluator technique, not a solution generator technique [1]. This scenario could be changed with the aid of optimization procedures. In this case, simulation could answer not only "What if" questions but also answers "How to" questions [2], providing with the best set of input variables that maximize or minimize some performance measure(s) (based on the modeling objectives).

According to [3], simulationists can be classified into 3 categories with regards to the optimization of quantitative variables: the first category tends to use the "trial and error" method, varying the input variables in order to find which set gives the best performance. The second category tends to systematically vary the input variables, to see their effects on the output variables. The third category will apply an automated simulation optimization approach.

In the paper we are concentrating on the last category. That is, an automated "optimization" approach based on adapting the parameters of the model in order to handle uncertainty that arises from stochastic elements in either the environment (process noise/variables drift) or the objective function evaluation process (observation noise), and we particularly investigate the use of adaptive stochastic search methods in this context.

Adaptive search strategies include methods like simulated annealing [4]-[5], genetic and evolutionary algorithms [6], and swarm intelligence [7]-[8]. The paper is organized as follows. In the next section, three popular adaptive search strategies are reviewed. We then proceed by describing a typical simulation model that we use as a benchmark in our study. Finally, results are presented and discussed, and the paper ends with concluding remarks.

II. Adaptive Stochastic Search Strategies

A. Simulated Annealing

Simulated Annealing (SA) is a method based on Monte Carlo simulation, which solves difficult combinatorial optimization problems. The name comes from the analogy to the behavior of physical systems by melting a substance and lowering its temperature slowly until it reaches freezing point (physical annealing).

Simulated annealing was first used for optimization by Kirkpatrick et al. [5]. In this context, SA is a procedure that has the capability to move out of regions near local minima [4]. SA is based on random evaluations of the objective function, in such a way that transitions out of a local minimum are possible. Thus, candidate solutions are updated following the relation

$$x_{k+1} = x_k + \Delta x, \tag{1}$$

where Δx is random noise from a uniform distribution. The method applies the *Metropolis* criterion, i.e. it either accepts or rejects a candidate solution depending on the probability

$$P(\mathbf{x}_k \to \mathbf{x}_{k+1}) = \begin{cases} 1 & \text{if } \Delta f = f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k) < 0 \\ e^{-\frac{\Delta f}{T}} & \text{otherwise} \end{cases}. \tag{2}$$

This, if the sign of $\Delta f = f(\mathbf{x}_{k+1}) - f(\mathbf{x}_k)$ is *negative*, then the new point can be accepted with probability 1; otherwise, it depends on the probability value and the threshold value θ, i.e. $P(\mathbf{x}_k \to \mathbf{x}_{k+1}) = \exp(-\Delta f / T) > \theta, \quad \theta \in (0,1)$.

Authors are with the "Centre for Applied Simulation Modelling" (CASM), Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex UB8 3PH, U.K

The effectiveness of the method depends the parameter T that is called temperature; it controls the noise reduction rate:

$$T(k) = \frac{T_0}{1 + \ln k} \quad (3)$$

When the temperature takes high values SA behaves like a random search, whilst at low values it works like a hill climbing procedure. Usually one starts with a high temperature value and gradually reduces: it is important to start at a temperature where most changes are accepted, i.e. probability is 1 in (2), then we can reduce the temperature in steps of a few percent while maintaining each temperature just long enough to reach an approximate solution.

SA does not guarantee, of course, to find the global minimum, but if the function has many good near-optimal solutions, it should find one. In particular, SA is able to discriminate between "gross behavior" of the function and finer "wrinkles". First, it reaches an area in the function domain space where a global minimizer should be present, following the gross behavior irrespectively of small local minima found on the way. It then develops finer details, finding a good, near-optimal local minimizer, if not the global minimum itself.

B. Genetic Algorithms

Genetic Algorithms (GA) are simple and robust search algorithms based on the mechanics of natural selection and natural genetics. The mathematical framework of GAs was developed in the 1960s and is presented in Holland's pioneering book [9]. GAs have been used primarily in optimization and machine learning problems.

The fundamental principle of GAs is that at each generation of a GA, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than their progenitors, just as in natural adaptation. A high level description of the simple GA is shown in the code below.

```
STANDARD GENETIC ALGORITHM MODEL
{
//initialize the time counter
t := 0;
//initialize the population of individuals
InitPopulation(P(t));
//evaluate fitness of all individuals
Evaluate(P(t));
//test for termination criterion (time,
//fitness, etc.)
while not done do
t := t + 1;
//select a sub-population for offspring
//production
Q(t) := SelectParents(P(t));
//recombine the "genes" of selected
//parents
Recombine(Q(t));
//perturb the mated population
//stochastically
Mutate(Q(t));
//evaluate the new fitness
Evaluate(Q(t));
//select the survivors for the next
//generation
P(t + 1) := Survive(P(t), Q(t));
end
}
```

More specifically, a simple GA processes a finite population of fixed length binary strings called genes. GAs have two basic operators, namely: crossover of genes and mutation for random change of genes. The crossover operator explores different structures by exchanging genes between two strings at a crossover position and the mutation operator is primarily used to escape the local minima in the weight space by altering a bit position of the selected string; thus introducing diversity in the population. The combined action of crossover and mutation is responsible for much of the effectiveness of GA search. Another operator associated with each of these operators is the selection operator, which produces survival of the fittest in the GA. The parallel noise-tolerant nature of GA and their hill-climbing capability make GA eminently suitable for simulation optimization, as they seem to search the parameter space efficiently.

C. The Particle Swarm Optimization Method

In Particle Swarm Optimization (PSO) algorithm the population dynamics simulates a "bird flock's" behavior where social sharing of information takes place and individuals can profit from the discoveries and previous experience of all other companions during the search for food. Thus, each companion, called particle, in the population, which is now called swarm, is assumed to "fly" over the search space in order to find promising regions of the landscape. For example, in the minimization case, such regions possess lower functional values than other regions visited previously. In this context, each particle is treated as a point in an n-dimensional space which adjusts its own "flying" according to its flying experience as well as the flying experience of other particles (companions).

There are many variants of the PSO proposed so far, after Eberhart and Kennedy introduced this technique [7]-[8]. In our experiments we have used a version of this algorithm, which is derived by adding an *inertia weight* to the original PSO dynamics [10]. This version is described in the following paragraphs.

First let us define the notation used in this PSO: the i-th particle of the swarm is represented by the n-dimensional vector $x_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ and the best particle in the swarm, i.e. the particle with the smallest function value, is denoted by the index g. The best previous position (the position giving the best function value) of the i-th particle is recorded and represented as $p_i = (p_{i1}, p_{i2}, \ldots, p_{in})$, and the position change (velocity) of the i-th particle is $V_i = (v_{i1}, v_{i2}, \ldots, v_{in})$.

The particles are manipulated according to the relations

$$v_{in} \leftarrow w \cdot v_{in} + c_1 \cdot r_1 (p_{in} - x_{in}) + c_2 \cdot r_2 (p_{gn} - x_{in}) \quad (4)$$

and

$$x_{in} \leftarrow x_{in} + v_{in}, \quad (5)$$

where n=1,2,…,N; i=1,2,…,NP and NP is the size of population; w is the inertia weight; c_1 and c_2 are two positive constants; r_1 and r_2 are two random values in the range [0,1].

The first relation is used to calculate i-th particle's new velocity by taking into consideration three terms: the particle's previous velocity, the distance between the particle's best previous and current position, and, lastly, the distance between swarm's best experience (the position of the best particle in the swarm) and i-th particle's current position. Then, following the second equation, the i-th particle flies toward a new position. In general, the performance of each particle is measured according to a predefined fitness function, which is problem-dependent.

The role of the inertia weight, w, is considered very important in PSO convergence behavior. The inertia weight is employed to control the impact of the previous history of velocities on the current velocity. In this way, the parameter w regulates the trade-off between the global (wide-ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine-tuning the current search area. A suitable value for the inertia weight w usually provides balance between global and local exploration abilities and consequently a reduction on the number of iterations required to locate the optimum solution.

A general rule of thumb suggests that it is better to initially set the inertia to a large value, in order to make better global exploration of the search space, and gradually decrease it to get more refined solutions, thus a time decreasing inertia weight value is used.

From the above discussion it is obvious that PSO, to some extent, resembles GAs. However, in PSO, instead of using genetic operators, each individual (particle) updates its own position based on its own search experience and other individuals (companions) experience and discoveries. Adding the velocity term to the current position, in order to generate the next position, resembles the mutation operation in evolutionary algorithms. Note that in PSO, however, the "mutation" operator is guided by particle's own "flying" experience and benefits by the swarm's "flying" experience. In another words, PSO is considered as performing mutation with a "conscience", as pointed out by Eberhart and Shi [10].

III. The Steelworks Model

In this section, we use a typical simulation model as a benchmark for comparing the various methods. We will first explain the system to be modeled and we will proceed by specifying the parameters that are considered important to adaptation.

The steelworks simulation model is an example of a manufacturing simulation model fully described in [11]. A brief description is provided here.

There are two blast furnaces, which melt iron at certain daily volumes, in a plant that blows and fills as many torpedoes as available; these are used to transport molten iron. If no torpedo is available, the molten iron is dropped on the floor and waste is produced. Each torpedo can hold a fixed quantity of molten iron. All torpedoes with molten iron travel to a pit, where crane(s)-carrying ladles are filled from torpedoes, one at a time. The ladle holds 100 tons of molten iron, which is exactly the volume of a steel furnace that is fed from the crane. There are five steel furnaces, which produce the final product of the steelworks. Figure 1 shows the schematic layout (not to scale) for the steelworks plant.

When a blast furnace has emptied its blast into the minimum number of torpedoes required (if available) all torpedoes with molten iron in go to the pit (by railway-the torpedoes run on a railway track). This includes partially full torpedoes.

The crane travels along an overhead gantry and carries a ladle (large spoon shaped vessel). The crane is filled at the pit from one torpedo at a time. The pit is two-sided to avoid any delay if the crane requires more than one torpedo to load it (yet one at a time). In the event that two torpedoes are insufficient to load the crane, the time it takes to discharge the second torpedo can be considered sufficient for a third torpedo (if there is one) to take the place of the first, ready for unloading. Note that the pit's function is to catch any spillage of molten iron. It does not hold molten iron.

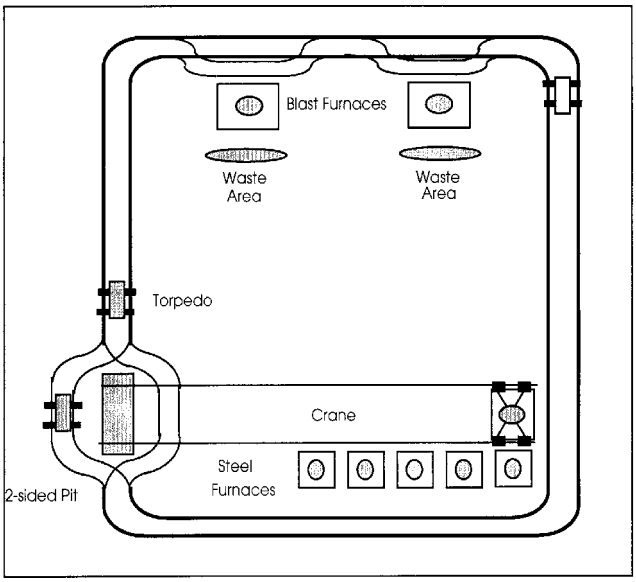

Fig. 1. Layout for the Steelworks Plant

The simulation model was built using Simul8 package running on Windows NT (see Figure 2). Simul8 is a simulator package, which in its simplest form allows the building of models by using graphics (e.g. dragging and dropping icons), by selecting items from menus and filling in dialog boxes. Due to its two-way interface facilities with Microsoft Excel and Visual Basic for Applications (VBA) it

helps the user to build extra routines for the mode. This adds to the software's flexibility and power.

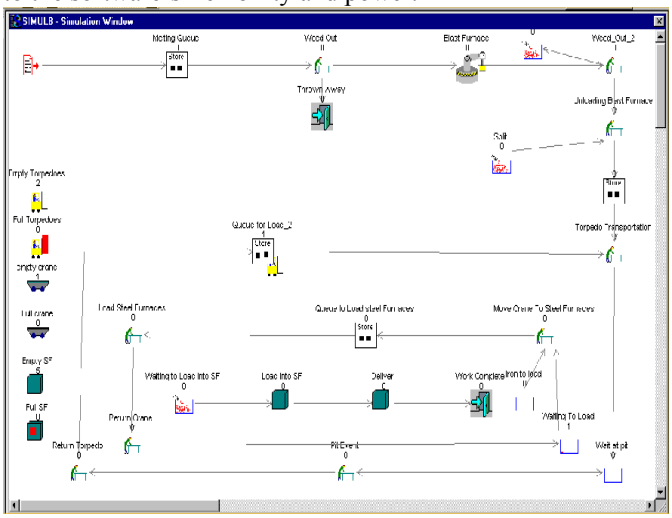

Fig. 2. Simulation of the steelworks model

IV. Parameter Adaptation

This section shows the use of SA and PSO in adapting the parameters of the steelworks model with the aim to maximize profit and at the same time minimize waste. Of course the fitness of a solution changes, gradually or abruptly, as conditions change within the environment. This is simulated in our model through several stochastic variables; for example, the model incorporates randomness in the way the weight of the molten iron is distributed and in the number of minutes that it takes to get from the blast furnace to the pit.

Figure 3 shows the interaction between Simul8 and Excel. The model has been built using Simul8 and the optimization algorithm has been implemented in Visual Basic. Excel acts as the "middle-man" sending information back and forth between the two applications. However, at this point we must note that the end user will only be using Excel as his/her point of interaction. The other applications will run in the background.

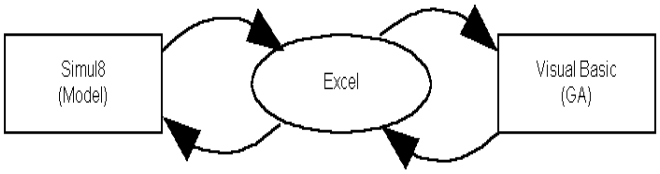

Fig. 3. Interactions between the software components

In Simul8 we used Visual Logic (a logic based component within Simul8) to pass arguments to Excel and Excel's macro language (VBA) was used to implement the optimization algorithms (see in Figure 4 the optimization worksheet of the PSO algorithm). The data exchange between Simul8 and the optimization algorithm was performed via an Excel spreadsheet using DDE facilities.

Simul8 provides an Excel/VB Library (a list of all the functions within SIMUL8 which Excel/VB logic can reference) and Excel/VB Signals (a list of the special signals which SIMUL8 sends to Excel/VB logic while the model runs), which will be employed.

The simulation model has many parameters that could be used for adaptation. We have chosen to keep the number of blast furnaces constant (i.e. two) throughout the experimentation and adapt the number of torpedoes, number of steel furnaces, volume that the torpedoes can hold and the number of cranes. This would keep in line with other publications of the same model [12]-[13].

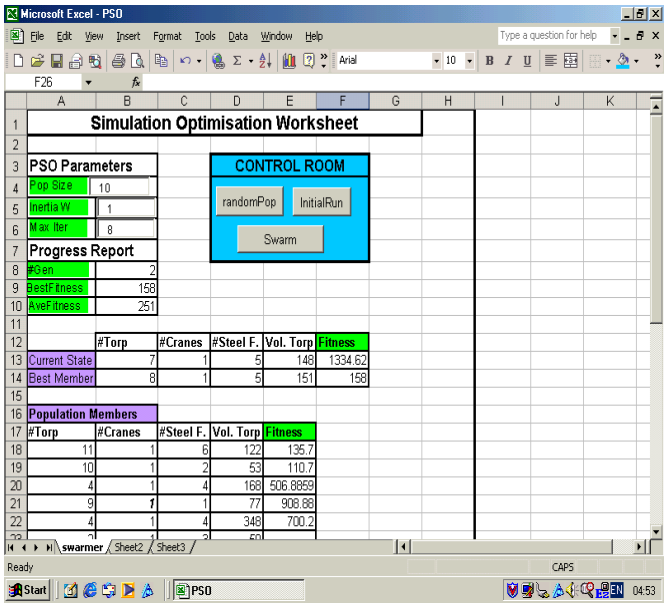

Fig. 4. PSO simulation optimization worksheet in Excel

Figure 5 presents the behavior of the best particle in the swarm for various sizes (the results are from 20 independent runs). A summary of the best results is shown in Table I. Note that no special fine-tuning of PSO heuristics was done; default values suggested in the literature were used to initialize the parameters which were adapted in run time (see [14]-[15] for PSO details and parameter control techniques).

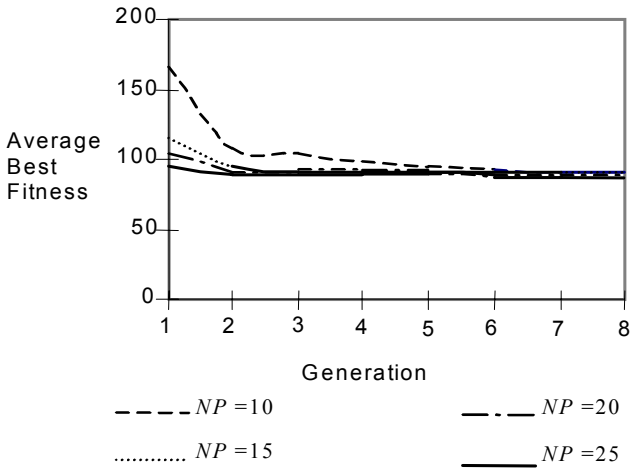

Fig.5. Average best fitness for various swarm sizes

We conducted additional experiments to compare PSO

with SA. PSO is a population-based method while SA works with a single point. That results of course in totally different converges behavior. However, we decided to run several experiments with the SA and record the initial point that leaded to the best available solution found. We then initialized the swarm and the GA in the neighborhood of the same point. According to our experiments PSO offers superiority over SA. The criteria used to arrive at this conclusion are two fold.

TABLE I
SUMMARY OF THE BEST PSO RESULTS FOR EACH SWARM SIZE

Population (*NP*)	10	15	20	25
#Torpedo	4	5	5	4
#Cranes	2	2	1	1
#Furnaces	4	4	4	4
Torpedo Volume	350	350	240	236
Objective function (£K)	94	92	86	84

Firstly the number of iterations required to reach a "global" optimum is less, and secondly the value of the fitness value reached is lower than that achieved by SA. This may be a fair comparison because the swarm was deliberately initialized in a region analogous to that of the SA test. The SA found a feasible solution after 130 function evaluations that correspond to searching the 0.3% of the search space. The GA needed 460 function evaluations using 20 individuals. In the PSO test, the swarm consisted of 10 particles and a typical result is shown in Table II. An average of 6 iterations was needed in these tests that corresponds to searching the 0.14% of the search space. The best result obtained with PSO had a value of 79; a value that is significantly better than other results obtained using SA and GAs.

TABLE II
RESULTS FOR SIMULATED ANNEALING AND PARTICLE SWARM

Method	SA	PSO	GA
#Torpedo	6	4	4
#Cranes	2	2	2
#Furnaces	5	4	5
Torpedo Volume	260	250	235
Objective function (£K)	112.70	92	108.5

V. CONCLUDING REMARKS

Adapting the parameters of a simulation model is a powerful tool to improve model's performance and answer "What if" questions. Adaptive stochastic search methods are eminently suitable for this task. These methods do not require derivative-related information, and provide global exploration of the search space. As the environment changes, information contained within the population of solutions allows efficient discovery of better-adapted solutions.

REFERENCES

[1] C. Harrel and K. Tumay, *Simulation Made Easy: A Manager's Guide*, Norcross, Georgia IE: Management Press, 1994.

[2] F. Azadivar, "Tutorial on simulation optimization," in *Proceedings of the 1992 Winter Simulation Conference, Arlington*, J. Swain, D. Glodsman, R. C. Crain, and J. R. Wilson, Eds. New York : Institute of Electrical and Electronics Engineers, 1992, pp. 198–204.

[3] B. Stuckman, G. Evans, G., and Mollaghasemi, M. Comparison of global search methods for design optimization using simulation, in *Proceedings of the 1991 Winter Simulation Conference*, Phoenix, B. L. Nelson, W. D. Kelton, and G. M. Clark, Eds. , New York: Institute of Electrical and Electronics Engineers, 1991, pp. 937–943.

[4] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the simulated annealing algorithm," *ACM Transactions on Mathematical Software,* vol. 13, no. 3, pp. 262–280, 1987.

[5] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. "Optimization by simulated annealing", *Science*, vol. 220, pp. 671–680, 1983.

[6] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*, New York: Springer, 1996.

[7] R. C. Eberhart, P. K. Simpson, and R. W. Dobbins, *Computational Intelligence PC Tools*. Boston, MA: Academic Press Professional, 1996.

[8] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings IEEE International Conference on Neural Networks*, Piscataway, New Jersey: IEEE Press, 1995, pp. 1942–1948. Available online <http://dsp.jpl.nasa.gov/members/payman/swarm/> [accessed February 16, 2001]

[9] J. H. Holland, *Adaptation in Neural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press, 1975.

[10] R. C. Eberhart and Y. H. Shi, "Evolving artificial neural networks", in *Proceedings International Conference on Neural Networks and Brain*, PL5–PL13. Beijing, P.R. China: Publishing House of Electronics Industry, 1998.

[11] R. J. Paul and D. W. Balmer, *Simulation modelling*. Lund: Chartwell-Bratt, 1993.

[12] R. J. Paul and T. S. Chanev. "Optimizing a complex discrete event simulation model using a genetic algorithm," *Neural Computing and Applications*, vol. 6, pp. 229–237, 1997.

[13] R. J. Paul and T. S. Chanev, Simulation optimization using a genetic algorithm, *Simulation Practice and Theory*, vol. 6, no. 6, pp. 601–611, 1998.

[14] K. Parsopoulos, V. P. Plagianakos, G. D. Magoulas, and M. N. Vrahatis, "Improving the particle swarm optimizer by function stretching", in *Advances in Convex Analysis and Global Optimization: Noncovex Optimization and its Applications*, vol. 54, N. Hadjisavvas and P. Pardalos Eds. Dordrecht: Kluwer Academic Publishers, 2001, pp.445–457.

[15] K. Parsopoulos, V. P. Plagianakos, G. D. Magoulas, and M. N. Vrahatis, "Stretching technique for obtaining global minimizers through particle swarm optimization," in *Proceedings of Particle Swarm Optimization Workshop (PSOW)*, Y. Shi, Ed. Indianapolis, Indiana: Purdue School of Engineering and Technology, IUPUI Press, 2001, pp. 22–29.