SIMULATION-BASED OPTIMISATION USING LOCAL SEARCH AND NEURAL NETWORK METAMODELS

Anna Persson
Henrik Grimm
Amos Ng
Centre for Intelligent Automation
Box 408
University of Skövde
Sweden
{anna.persson, henrik.grimm, amos.ng}@his.se

ABSTRACT

This paper presents a new algorithm for enhancing the efficiency of simulation-based optimisation using local search and neural network metamodels. The local search strategy is based on steepest ascent Hill Climbing. In contrast to many other approaches that use a metamodel for simulation optimisation, this algorithm alternates between the metamodel and its underlying simulation model, rather than using them sequentially. On-line learning of the metamodel is applied to improve its accuracy in the current region of the search space. The proposed algorithm is applied to a theoretical benchmark problem as well as a real-world manufacturing optimisation problem and initial results show good performance when compared to a standard Hill Climbing strategy.

KEY WORDS

Optimisation, Simulation, Local Search, Metamodel, Neural Network

1. Introduction

Real-world problems in optimisation often contain nonlinearities, combinatorial relationships and uncertainties that are too complex to be effectively modelled analytically [1]. For this reason, a simulation-based optimisation (SO) approach is often used to solve practical optimisation problems. The general problem in SO is to find a setting of decision variables that maximizes or minimizes a given objective function, assuming that the objective function cannot be computed analytically but must be estimated through simulation. In a general SO system (Figure 1), an optimisation procedure feeds input values into a simulation that estimates one or multiple performance measures. Based on the evaluation feedback, possibly in combination with previous evaluations, the optimisation procedure generates a new set of input values. The generation-evaluation process then iterates until a user-defined stopping criterion is satisfied.

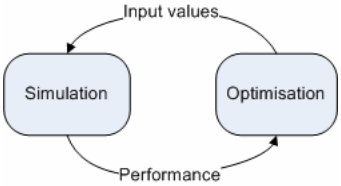

Figure 1: Simulation-Based Optimisation

Simulation runs are usually computationally expensive and it is not uncommon for large simulation models to run for hours. For practical applications of SO it is of critical importance that the optimisation process is constrained within reasonable time limits and the efficiency of the optimisation process is crucial [2][3][4][5]. One possible way to enhance the efficiency of SO and handle time-consuming simulation is to employ computationally cheap metamodels [6]. The purpose of metamodels, also known as surrogate or approximate models, is to approximate the relationship between the output variables and the decision variables by computationally efficient mathematical models. By adopting metamodels in simulation, the computational burden of the optimisation process can be greatly reduced since the computational cost associated with using metamodels is much lower than the standard approach of running all evaluations with the simulation models.

This paper presents a new optimisation algorithm that uses a metamodel for efficiency enhancement, called Metamodel Enhanced (MME) Local Search. Although the basic motivation for MME Local Search is to be used in SO, it can be applied to any general optimisation scenario in which a computationally expensive evaluation function can be approximated by some computationally cheap estimation function. The intended application of the MME Local Search Algorithm is, in first place, for problems suitable to solve with local search techniques. This may, for example, be single optimum problems or problems where a fairly good solution is known, for example, through a global search strategy, but needs to be improved.

The rest of the paper is organized as follows. The next section presents related work in the area of metamodelling in simulation. In Section 3, the MME Local Search algorithm is described, followed by a description of its application to a real-world optimisation problem in Section 4. Section 5 and 6 describe how the proposed algorithm is applied to solve two optimisation problems and an experimental comparison with a standard Hill Climbing algorithm is also presented. Conclusions of the paper and planned future work are presented in Section 6.

2. Related Work

The use of metamodels to reduce the limitations of time consuming simulations was first proposed by Blanning in 1975 [7]. Traditionally, regression and response surface methods have been two of the most common metamodelling approaches [8]. In recent years, artificial neural networks (ANN) have gained increased popularity, as this technique requires fewer assumptions and less precise information about the systems being modelled when compared with traditional techniques [9]. The first work providing the foundations for developing ANN metamodels for simulation was done by Pierreval [10] and Pierreval and Huntsinger [11]. Both of these studies yielded results that indicated the potential of ANNs as metamodels for discrete-event and continuous simulations, particularly when saving computational cost is important. Since then, many applications of ANN-based metamodels in simulation systems have been reported (see, for example, [12][13][14][15]). A majority of the proposed approaches for optimisation using ANN metamodels and simulation are based on some variants of global search optimisation. However, there are also some attempts in combining ANN metamodels and simulations with local search optimisation.

Chen and Yang [16] use an ANN metamodel together with a stochastic local search approach based on simulated annealing (SA) for optimisation of manufacturing systems design. The ANN is trained using back-propagation based on data obtained from a simulation program. The proposed method is applied to a design problem of an example manufacturing system and shows to be efficient in the problem solving process. The SA-based optimisation algorithm is constructed to exploit the search space extensively in order to escape local optima while the number of time-consuming simulation runs is reduced. Instead of triggering the detailed simulation program, the optimization procedure evaluates solutions using the neural network metamodel.

Altiparmak et al. [17] use an ANN metamodel together with simulated annealing (SA) to obtain optimal buffer sizes in a theoretical asynchronous assembly system. The ANN metamodel is trained to predict the production rate of a system given a set of buffer sizes, jammed stations, number of pallets, and jam clear time. A three layers feed-forward network is constructed with thirty input neurons and one output neuron. The data which is used to train the ANN is generated by simulating random samples from the set of possible configurations using a discrete-event simulation. The trained ANN is used during the search for optimal buffer sizes to estimate production rate based on different configuration settings. The ANN metamodel together with SA is used to find the optimal buffer size configuration for an assembly system with 15 stations.

Ong et al. [18] propose a framework for efficient aerodynamic shape design that combines evolutionary optimisation, gradient-based local search optimisation, and metamodels such as ANNs. The developed metamodel approximates a computationally expensive design model and is used for fast evaluations of solutions during the search for an optimal design. The authors apply their proposed approach to a 2D shape design problem and the results indicate that global convergence can be reached within a limited time budget.

The literature reports a handful of approaches that make use of local search optimisation, simulations and ANN-based metamodels. In these studies, however, either the metamodel is used exclusively in the SO process or the procedure of switching between the metamodel and the simulation is not fully addressed. In contrast to previous studies, this paper presents an approach for using a simulation model and an ANN metamodel in conjunction in the SO process and the procedure of switching between the two models are explicitly described.

3. The MME Local Search Algorithm

The MME Local Search algorithm is based on the concept of steepest ascent Hill Climbing. The algorithm guides a series of local evaluations with the goal of finding better solutions, according to the flowchart presented in Figure 2. A move in the landscape is based on the evaluation results from a simulation, and a metamodel is used to determine what solution in the neighbourhood of the current state to simulate. It should be noted that the metamodel has no need to be fully trained and validated prior to the searching starts and its computational time is assumed to be negligible when compared to that of the simulation. As the metamodel evaluates solutions very fast, a large neighbourhood search area is possible, preventing the MME Local Search to get stuck in a local optimum.

After each simulation evaluation, the metamodel is trained on-line using the simulation input-output sample and in this way the precision of the metamodel is constantly improved. By training the metamodel with samples from the current region, the local accuracy of the metamodel is increased. In local search, a locally correct metamodel is of more practical importance than a globally correct one.

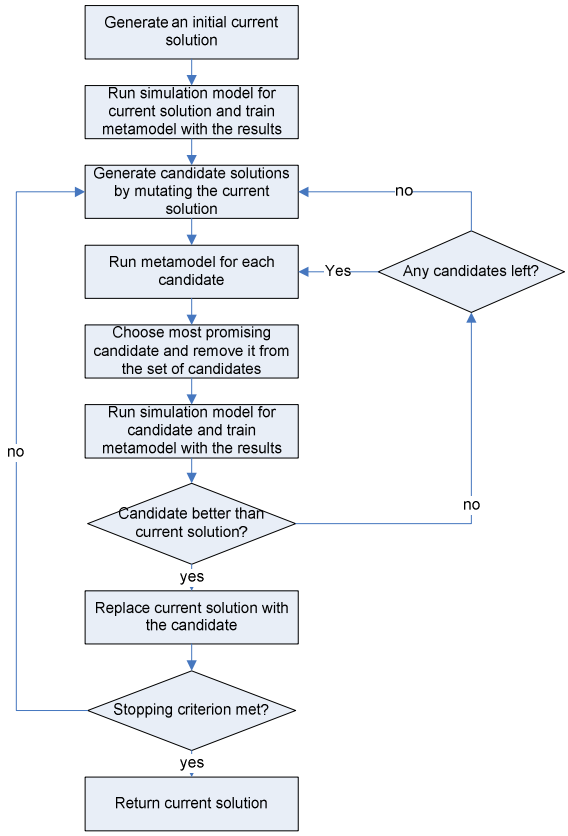

Figure 2: Flowchart of the MME Local Search Algorithm

The core of the algorithm is illustrated with pseudo-code in Section 3.1. The algorithm calls a number of problem-specific functions, which are described in Section 3.2.

3.1 Algorithm Core

In the algorithm, a solution is defined by a triple $(input, mm\_output, sim\_output)$ where $input$ is an input sample, mm_output is the output produced by the metamodel, and sim_output is the output produced by the simulation. Any of these attributes can be unassigned. For example, if sim_output is unassigned, this means that the solution has not yet been simulated. In order to refer to the attributes of a solution, a subscript notation is used, e.g., i_{input} is the *input* attribute of solution i.

The main function MME_Local_Search (Figure 3) is passed with an initial solution, which is iteratively improved and later returned when a user-specified stopping criterion is met. How the initial solution is selected is unspecified. One way to do this is to randomly generate a set of points in the search space and pick the best one. Alternatively, a more structured approach, such as Design of Experiments (DOE), can be used. In some cases, a fairly good solution may already exist, e.g., from earlier optimisations or experiments.

```
function MME_Local_Search(current)
Input: Initial solution.
Returns: Best solution found.
  current_sim_output ← Run_Simulation(current_input)
  Train_Metamodel(current)
  while (not Stop_Optimization()) do
    candidates ← Generate_Candidates(current)
    current_updated ← false
    while (not current_updated and candidates ≠ ∅ and not End_Evalution()) do
      foreach i in candidates do
        i_mm_output ← Run_Metamodel(i_input)
      end
      j ← Choose_Solution(candidates)
      candidates ← candidates − {j}
      j_sim_output ← Run_Simulation(j_input)
      Train_Metamodel(j)
      if (Compare_Solutions(j, current)) then
        current ← j
        current_updated ← true
      end
    end
  end
  return current
```

Figure 3: Algorithm Core

3.2 Problem-Specific Functions

This section describes the problem-specific functions called by the algorithm.

function Run_Simulation(*input*)
Input: An input sample.
Returns: Output response from simulation.

This function runs the accurate, but time-consuming, simulation.

function Run_Metamodel(*input*)
Input: An input sample.
Returns: Output response from metamodel.

This function is assumed to be many orders of magnitude faster than running the simulation. Note that the structure of the output returned from this function may not be the same as returned from the simulation. The metamodel may, for example, return an objective function value as output instead of the performance measure data returned by the simulation.

function Train_Metamodel(*solution*)
Input: A simulated solution.

This function trains the metamodel with the given solution.

function Generate_Candidates(*solution*)
Input: The parent solution.
Returns: A set of mutated solutions.

Employing the common diversification method in evolutionary optimisation strategies, this function

generates a set of mutated candidates based on the parent solution while considering possible constraints. Generation of candidates could, for example, be done by applying random perturbations to the parent.

function Choose_Solution(*solutions*)
Input: A set of solutions.
Returns: One of the solutions.

In the simplest case, just returns the solution with the best objective function value. Another approach is to choose the solution probabilistically based on their objective function values.

function Compare_Solutions(*solution1*, *solution2*)
Input: Two solutions.
Returns: True if solution1 is better than solution2, false otherwise.

Usually, this function applies an objective function to both solutions. For a maximisation problem it then returns true if *solution1* has a higher objective function value than *solution2*. For a minimization problem, it instead returns true if *solution1* has a lower objective function value than *solution2*.

function Stop_Optimization()
Returns: True to stop the optimisation.

This can, for example, be based on the quality of the current solution, on time elapsed, or on the number of iterations since the current solution last was updated.

function End_Evaluation()
Returns: True to end the evaluation of the candidates.

This function can be used to avoid having to simulate all the candidates. For example, the candidates can be discarded when the solutions are below a certain quality threshold, followed by the generation of new candidates in a larger region of the search space.

4. Benchmark Optimisation Problem

To demonstrate the MME Local Search algorithm, it is applied to the 2-D Rosenbrock function (Equation 1 below). The fitness landscape of the Rosenbrock function (plotted in Figure 4) has a global minimum of 0 at the point (1, 1).

$$f(x,y) = 100\left(y - x^2\right)^2 + (1-x)^2 \qquad (1)$$

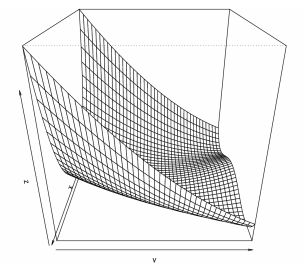

Figure 4: The 2-D Rosenbrock Function

A four-layer feed-forward ANN with two input nodes, 20 nodes in each hidden layer and one output node is developed to approximate the fitness landscape of the Rosenbrock function. The ANN is trained off-line with 1000 training samples randomly generated. During the search, the ANN is further trained continuously with the data from the last simulation using back-propagation for 5 epochs with a learning rate of 0.7. In each iteration of the MME Local Search algorithm, ten candidate solutions are generated and evaluated using the ANN.

In order to draw a comparison, a standard Hill Climbing algorithm is implemented for the same optimisation problem and evaluation function. This algorithm uses the same representation, initial solution, objective function, and mutation operator as in the MME Local Search implementation.

In Figure 5, average results from 1000 replications of the experiments are shown. The chart shows the best objective function value found against number of function evaluations.

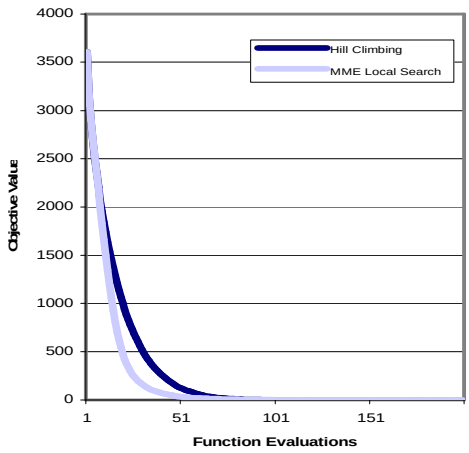

Figure 5: Results from Experiments

As shown in Figure 5, the MME Local Search Algorithm converges faster than the Hill Climbing algorithm. After about 75 iterations, the algorithms reach the same objective function value.

5. MME Local Search Applied to a Complex Real-World Problem

To further evaluate the MME Local Search algorithm, it is also applied to a complex real-world multi-objective optimisation problem at the aircraft engine manufacturer Volvo Aero, Sweden. The objective of the optimisation is to minimise the total lead time and simultaneously minimise the total tardiness of different products in a multi-process machining line.

A discrete-event simulation model of the production system is developed using the SIMUL8 software package (www.simul8.com). The simulation takes planned lead times of different product types as inputs and the total tardiness of different product types as outputs. For the scenario considered, 11 product types are included and a

duration corresponding to one year of production is simulated. An ANN is developed as a metamodel of the simulation model. The ANN is trained to estimate tardiness of products as a function of planned lead times. A four layers feed-forward network is constructed with eleven neurons in each layer (Figure 6). The data set which is used to train the ANN consists of 1000 input-output pairs generated from the simulation of random points in the search space.

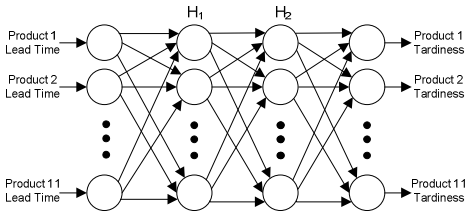

Figure 6: Conceptual Illustration of the ANN

The next section describes how the MME Local Search algorithm uses the simulation model and the metamodel for the optimisation of minimal lead time values in combination with minimal tardiness. Section 4.2 then presents the results of the optimisation.

4.1 Implementation

An input sample consists of a vector of 11 real values, corresponding to lead time for each product type. The values are bounded by a lower and upper bound, specific for each type. An output sample consists of a vector of 11 real values, corresponding to the total tardiness of each product type. To compare solutions the input and output values are converted to an objective function value, described by $\sum_{i \in D} \left(i_{planned\_lead\_time} + 10\, i_{tardiness} \right)$ where D is the set of all products. The goal of the optimisation is to minimise this value. Note that both the input and output values are used to calculate the objective function value.
The optimisation starts from the best input sample used to train the ANN and runs for 2000 simulations. After each simulation, the metamodel is trained using back-propagation for 5 epoch with a learning rate of 0.7. Only data from the last simulation is used to train the network, as using the complete training set would be too time-consuming. In each iteration of the algorithm, 100 candidate solutions are generated and evaluated using the ANN. During mutation, the lead time for each product type is modified by a random number taken from a Gaussian distribution with a deviation of 2. If any of the values are outside of the allowed bounds, the value is discarded and a new one is generated.

4.2 Results

This section presents the results of the MME Local Search implementation described in the previous section. To investigate if the optimisation benefits from using the metamodel, experiments without using the metamodel (it always returns zero for all output values) but with an identical experimental setup in all other aspects have also been carried out.

As with the theoretical problem described in the previous section, a standard Hill Climbing algorithm is also implemented for the same optimisation problem and simulation model. This algorithm uses the same representation, initial solution, objective function, and mutation operator as the MME Local Search implementation.

In Figure 7, average results from eight replications of the three experiments are shown. The chart shows the best objective function value found against number of simulations.

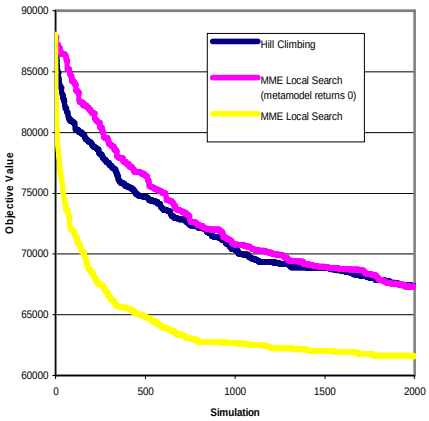

Figure 7: Comparison of Experiments

As the chart shows the MME Local Search algorithm has considerably faster convergence than the standard Hill Climbing algorithm.

The MME Local Search algorithm is also considerably better than the MME Local Search algorithm without the metamodel. When comparing the average objective function value using the metamodel and not using it, the benefit of using the metamodel is evident.

Using on-line training, the accuracy of the metamodel is continuously improved, as shown in Figure 8. This figure presents the Mean Square Error (MSE) of the metamodel based on the input-output from the last simulation. From an average of 10 replications, the MSE presented is a rolling average of 100 estimations.

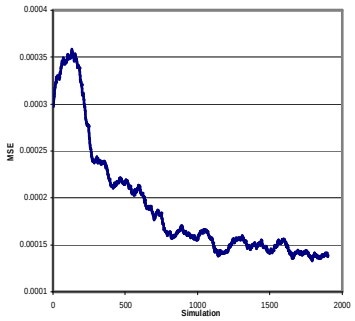

Figure 8: Estimated Metamodel MSE

6. Conclusions and Future Work

This paper presents a new algorithm for optimisation using a simulation model in conjunction with a simulation metamodel. The proposed algorithm is based on a heuristic local search procedure and the concept of steepest ascent Hill Climbing. In contrast to other approaches that use a metamodel for simulation optimisation, this algorithm alternates between the metamodel and its underlying simulation model, rather than using them sequentially. When using only the metamodel for evaluations, as in many other approaches, there is a risk that the search is guided in a more or less wrong direction, as the metamodel provides only an approximation function for the underlying simulation.

The proposed algorithm applies on-line training of the ANN-based metamodel after each simulation evaluation and in this way the precision of the metamodel is constantly improved. Furthermore since the metamodel is trained with samples from the local search area, the accuracy is improved in the part of the search space that the algorithm is currently exploring.

The proposed algorithm is applied to two optimisation problems, one theoretical and one real-world, and shows good performance compared to a standard Hill Climbing algorithm. In these specific implementations an ANN was used as the metamodel, although the algorithm allows for any kind of metamodel to be used. In the same way, the simulation can be any time consuming procedure.

Future work will focus on many different aspects of this ongoing research. These include verification of the proposed algorithm by applying the approach to different optimisation problems with various properties, and further benchmarks against some other state-of-the-art algorithms. A number of different variants of the basic MME Local Search algorithm can also be tested. For example, instead of evaluating one solution at a time, a set of solutions can be chosen and simulated, possibly in parallel. Ideas and concepts from other local search strategies, such as Simulated Annealing and Tabu Search, can also be incorporated into the algorithm.

Future work also includes extending a global search procedure with a metamodel for efficiency enhancement. Combining global and local search procedures with a metamodel may also be a viable path.

References

[1] J. April, M. Better, F. Glover and J. Kelly, New advances for marrying simulation and optimization, *Proceedings of the 2004 Winter Simulation Conference*, Washington, DC, 80-86.

[2] I. Mezgti, C.S Egresits and L. Monostori, Design and real-time reconfiguration of robust manufacturing systems by using design of experiments and artificial neural networks, *Computers in Industry 33*, 1997, 61-70.

[3] F. Azadivar, Simulation optimization methodologies, *Proceedings of the 1999 Winter Simulation Conference*, Squaw Peak, AZ, 1999, 93-100.

[4] J. Boesel, R.O. Bowden, F. Glover, J.P. Kelly and E. Westwig. Future of simulation optimization, *Proceedings of the 2001 Winter Simulation Conference*, Arlington, VA, 1466-1469.

[5] J.S. Carson, Introduction to modeling and simulation, *Proceedings of the 2005 Winter Simulation Conference*, Orlando, FL, 16-23.

[6] F. Alam, K.R. McNaught, and T.J. Ringrose, A comparison of experimental designs in the development of a neural network simulation metamodel, *Simulation Modelling Practice and Theory 12*(7-8), 2004, 559–578.

[7] R.W. Blanning, The construction and implementation of metamodels. *Simulation*, 1975, 177–184.

[8] P.A. Fishwick, Neural network models in simulation: a comparison with traditional modeling approaches, *Proceedings of the 1989 Winter Simulation Conference*, Washington, DC, 702–710.

[9] M. Padgett and T.A. Roppel, Neural networks and simulation: modeling for applications, *Simulation 58*, 1992, 295–305.

[10] H. Pierreval, Training a neural network by simulation for dispatching problems", Proceedings of the Third Rensselaer International Conference on Computer Integrated Engineering, New York, 1992, 332–336.

[11] H. Pierreval and R.C. Huntsinger, An investigation on neural network capabilities as simulation metamodels. *Proceedings of the 1992 Summer Computer Simulation Conference*, San Diego, CA, 1992, 413–417.

[12] F. Alam, K.R. McNaught; and T.J. Ringrose, A comparison of experimental designs in the development of a neural network simulation metamodel, *Simulation Modelling Practice and Theory 12*(7-8), 2004, 559–578.

[13] D. J. Fonseca, D.O. Navaresse and G. P. Moynihan, Simulation metamodeling through artificial neural networks, *Engineering Applications of Artificial Intelligence 16*(3), 2003, 177-183.

[14] L. Monostori and Z.J. Viharos, Hybrid, AI- and simulation-supported optimisation of process chain and production plants, *Annals of the CIRP 50*(1), 2001, 353-356.

[15] I. Sabuncuoğlu and S. Touhami, Simulation metamodeling with neural networks: an experimental investigation, *International Journal of Production Research 40*(11), 2002, 2483-2505.

[16] M.C. Chen and T. Yang, Design of manufacturing systems by a hybrid approach with neural network metamodelling and stochastic local search, *International Journal of Production Research 40*(1), 2002, 71-92.

[17] F. Altiparmak, B. Dengiz, and A.A. Bulgak. Optimization of Buffer Size in Assembly Systems using Intelligent Techniques, *Proceedings of the 2002 Winter Simulation Conference*, San Diego, CA, 1157-1162.

[18] Y.S. Ong, K.Y. Lum, P. B. Nair, D.M. Shi, and Z.K. Zhang, Global Convergence of Unconstrained And Bound Constrained Surrogate-Assisted Evolutionary Search In Aerodynamic Shape Design, *Proceedings of 2003 Congress on Evolutionary Computation*, Canberra, Australia, 1856-1863.