An Optimisation Methodology for Multi Parameter Heuristics

Susan E. Bedingfield
School of Business Systems
Monash University
Victoria 3150
Australia
sue.bedingfield@infotech.monash.edu.au

Kate A. Smith
School of Business Systems
Monash University
Victoria 3150
Australia
Kate.smith@infotech.monash.edu.au

Abstract – **This paper proposes a methodology for overcoming the 'curse of dimensionality problem' associated with optimising a multi parameter heuristic over its entire parameter space. The solution space associated with a given heuristic and associated problem is modeled with a well behaved function which can then be used to estimate parameter inputs to the heuristic which provide local optima for the problem. A case study demonstrating the methodology, using an evolutionary algorithm as the heuristic, is presented. Extensions of the proposed methodology to global search algorithms are discussed.**

I. INTRODUCTION

There are many heuristics available for solving multicriteria optimization problems, including evolutionary algorithms, simulated annealing, neural networks, etc.. Due to the size of the parameter space, a major problem encountered when using these techniques is finding the optimal combination of parameters, since a global search of the parameter space is frequently infeasible due to time or computing resource constraints. Many researchers have proposed techniques for finding local and global optima for a given heuristic and problem specification, either by deriving analytical solutions to the optimal parameter combination problem, and by an experimental trial-and-error approach to exploring the parameter space. For example, Kamgar-Parsi [1] presents a technique for optimizing a Hopfield network solution of the TSP. Gates and Merkle [2] et. al. report on their empirical studies on run-time parameters for genetic algorithms using a real-world application. Hoffmann and Salamon [3] present both a numeric and an analytic solution to the problem of finding an optimal cooling schedule for simulated annealing. These techniques however, are either problem specific or heuristic specific and not readily generalisable.

This paper presents a methodology, based on the work of Smith and Gupta [4,5], for finding locally optimal parameter combinations of a given heuristic on a given problem without the necessity of performing an evaluation of all points in the parameter space. The methodology generalizes, regardless of the heuristic or problem domain. A neural network is used to learn a smooth approximation to the heuristic parameter search space, based on limited sample data (pairs of parameter combinations and heuristic evaluation points). Locally optimal points of the resulting smooth approximation surface are then obtained using a gradient descent method. This approach is used to determine the optimal parameter combinations of an evolutionary algorithm performing a classification task on a credit scoring problem.

In section 2 of the paper we describe the methodology in general terms, for any heuristic method and any problem domain. Section 3 presents the details of the the case study used to illustrate the approach: the evolutionary algorithm heuristic is described, and the credit scoring data is discussed. The experimental procedure is described in Section 4, and results presented in Section 5. Finally, conclusions are drawn in Section 6.

2. METHODOLOGY

Suppose we are attempting to solve an optimization problem using a heuristic H on a problem domain D which involves N parameters $P_1, \ldots, P_N$ (where $N \geq 1$) and that each of these parameters may assume M_{P_i} possible values, where each M_{P_i} may be integer valued or real valued. Note that the parameters may come from both the heuristic and the problem domain.

We denote an N-tuple of these parameters by g. I.e. $g = (g_1, g_2, \ldots, g_N)$ where $P_1 = g_1$, $P_2 = g_2, \ldots\ldots$, $P_N = g_N$. We denote the problem solution using heuristic H and parameter values g by $s(g)$.

Step 1

We first take a sample set of Q input vectors $\{g^i\}_{i=1}^{Q}$ to the heuristic H, then evaluate them using the heuristic. Thus we arrive at a grid $G = \{g^i, s(g^i)\}_{i=1}^{Q}$ of Q combinations of parameters with corresponding solution provided by the heuristic, where Q is an integer to be determined by experimentation.

Step 2

The grid G is then fed through a multi layer feed forward neural network (MFNN) in an attempt to model the surface determined by G. The MFNN has N input nodes (where N is the number of parameters required by the heuristic), and one output. The number of hidden nodes however cannot be specified in general and will depend on the complexity of the heuristic and the problem domain. Likewise the most appropriate activation functions at the output and hidden neurons (tanh, linear, logistic etc.) may differ depending on the problem context. Note that we need make no assumptions about the differentiability of the surface we are attempting to model. Conversely, the model surface is continuously differentiable and thus much easier to work with in terms of finding local optima. Once we

have trained the MFNN we consider the resulting value of R^2 on both the training data and a reserved validation data set to see whether the model is a satisfactory fit for the data grid G. An acceptable value of R^2 is to be determined be the user. If value of R^2 is not sufficiently high, then step 1 may be repeated with a more refined grid.

Step 3

The MFNN has now produced a continuously differentiable function which models the output surface of the heuristic H of the form

$$\hat{h}(x) = f_O\left(\sum_{j=1}^{k} V_j^* f_h\left(\sum_{i=1}^{l} W_{ij}^* g_i\right)\right) \quad (1)$$

where l is the size of the input layer, k the size of the hidden layer and g an input vector.

$f_0()$ and $f_h()$ are the activation functions at the input and hidden layers respectively. W^* and V^* are the hidden layer and output weights respectively generated by the MFNN .

We use the surface defined by $\hat{h}$ to search for local optima using the gradient descent technique (although any search technique could be used to locate local or global optima of the surface). For each input g we have:

$$g_i(t+1) = g_i(t) - \alpha \frac{\partial h}{\partial g_i}$$

where $\frac{\partial h}{\partial g_i} = \sum_{j=1}^{m} V_j^* W_{ij}^* f_o'(net_o) f_h'(net_j)$

m is the number of hidden nodes and α is the step size of the gradient descent.

The local optima of the approximated surface $\hat{h}$ should be close to a local optima of the original heuristic parameter surface, provided that the surface has been well approximated by the MFNN.

3. CASE STUDY

The heuristic used to trial this methodology is an evolutionary algorithm (EA) which generates rule systems for classification from databases containing any number of condition attributes and one consequence attribute. The EA has been fully described and reported upon in [2]. However for the purposes of this paper we provide a brief description as follows. At each generation of the EA, a rule system is generated of the form ....

A rule system comprises a finite set of rules {rule1, rule2,...,rulek} where 1≤k≤K, K being a parameter defining the number of allowable rules in a rule set. Each rule takes the form

$(\alpha_1 \propto_1 y_1, \alpha_2 \propto_2 y_2, ..., \alpha_{m-1} \propto_{m-1} y_{m-1}, \alpha_m = y_m)$

where $\alpha_1, \alpha_2, ..., \alpha_{m-1}$ are non-consequence database attributes, and α_m is the consequence attribute.

$(\propto_1, \propto_2, ... \propto_{m-1})$ are comparison operators ($\leq$, $\geq$ or $=$). The value of m is variable, but less than some fixed upper bound. An example of a rule is (age>50, sex = 0, income > 20,000, age <55, married = 1, income < 35,000, credit class = 1).

The German credit database has a binary valued decision attribute (equivalent to 'good' or 'bad' risk). This leads to two row accuracy measures C_1 and C_2, and two column accuracy measures C_3 and C_4. C_1 and C_2 effectively measure the coverage by the rule system of the 'good' and 'bad' objects respectively. Whereas C_3 and C_4 measure the accuracy of the 'good' and 'bad' predictions. These four expressions should optimally be equal to 1. Analogous to the accuracy measures C_1 and C_2 are two measures of inaccuracy C_5 and C_6 which should optimally be equal to 0. The fitness function used for the work presented in this paper has the following form:

$$F = \lambda F^1 + (1-\lambda) F^2 \text{ where } 0 < \lambda < 1 \text{ and}$$

$$F^1 = \left(\frac{\sum_{i=1}^{4} \omega_i C_i}{1 + \sum_{i=5}^{6} \omega_i C_i}\right), \quad F^2 = \left(\sum_{i=1}^{4} \omega_i C_i + \sum_{i=5}^{6} \frac{1}{1 + \omega_i C_i}\right) \quad (2)$$

where $\omega_i \geq 0$, i = 1,...6

This is a weighted function of the six components C_1-C_6 which attempts to generate rules over time with maximal coverage, maximal accuracy and minimal contradictions.

The EA population consists of P rule systems each containing up to k individual rules, P and k are also parameters associated with the EA. At each generational change the population increases via crossover and various mutations, and is then reduced to the fixed population size P via an elitist selection process. The EA is run on a training subset of the dataset. Once the EA has converged, the value of the loss incurred due to misclassifications from the resultant rule system is calculated on a test set (i.e. the remaining values in the dataset). This is done using ten training and test set pairs, then the loss values are averaged and this final value is effectively the solution value. The input parameters together with the loss evaluated by the rule system as described above define the surface which we are attempting to model. The parameters involved in this EA are listed in Table 1.

3.1 Data

The data used in the experiment is the well known German credit classification data available from the UCI machine learning repository. The data contains 1000 examples of credit applicants, 700 of which are known to be good credit risks, and 300 of which are known to be bad credit risks. This data has been used for previous experimentation with the EA described in this paper [6][7].

Table 1

Parameter name	Possible values	Explanation
PopSize	Integer valued	The number of rule sets in each population
MaxAtoms	Integer valued	Maximum initial number of atoms in each rule
RulesPerSet	Integer valued	The initial number of rules in each rule set
ChgComponent	[0,1]	Probabilty of changing a component of an atom
SwapAtom	[0,1]	Probabilty of swapping atoms between rules
DelRule	[0,1]	Probabilty of deleting rule from a rule set
AddRule	[0,1]	The probability of adding a new rule to a rule.
AddAtom	[0,1]	Probability of adding an atom to a rule
CrossOver	[0,1]	Probabilty of crossing two rule sets
$w_1, w_2, \ldots, w_6$	>=0	Weightings for the fitness function components
λ	[0,1]	Fitness function parameter

Table 2. Parameter search space

Parameter name	Possible values
x_1: CrossOver	[0,1]
x_2: ω_6	[0,3]
x_3: λ	[0,1]

Table 3. Fixed parameter values

Parameter name	Fixed values
PopSize	10
MaxAtoms	6
RulesPerSet	10
ChgComponent	0.5
SwapAtom	0.2
DelRule	0.2
AddRule	0.2
AddAtom	0.2
$\omega_1, \ldots, \omega_5$	1

4. EXPERIMENTAL PROCEDURE

Step 1

To test the viability of the proposed methodology, a subset of the parameter space was selected as a sample space on which to optimize the average loss derived from running the evolutionary algorithm. In this case the search space was limited by varying only three parameters: lambda, ω_6 and the crossover value as outlined in Table 2. All other parameters were fixed as in Table 3. We will subsequently refer to those three parameters as x_1, x_2, x_3. The grid comprised a collection of 60 sample points distributed evenly within the search space defined by table 2.

Step 2

The points in the grid were first linearly scaled so that all values lay in the range [-1,1]. The resulting grid was feed through a MFNN with an input layer of three neurons and a hidden layer of nine neurons. A linear activation function was used for the input layer and the hyperbolic tangent was used as the activation function for the hidden layer. An R^2 value of .9658 was obtained on the training set and .9593 on the test set. This indicates that the MFNN has managed to approximate the solution surface generated by the EA with a high degree of accuracy.

Step3

Using the weights W* and V^* generated by the MFNN (eq.(1)), the gradient descent heuristic was run ten times from randomly generated start values for x_1, x_2 and x_3. The final values for each of the ten runs were then used as parameter values and run through the EA. Each run incorporated a ten fold cross validation. The loss incurred using the first rule system from each final population was then calculated.

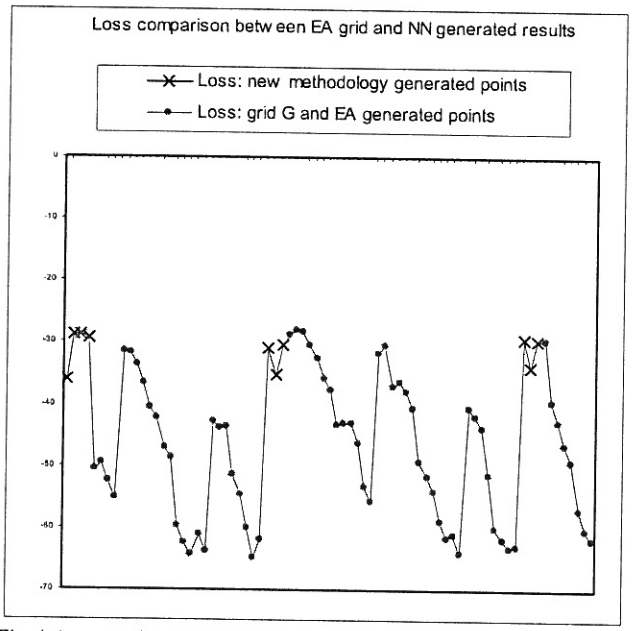

Fig. 1 A comparison of the values derived from the grid G and the points generated by the NN descent technique. The horizontal axis represents all combinations of λ, ω_6 and the crossover value.

5. RESULTS

The results are illustrated in Figure 1. the x-axis indicated the combined values of the three input parameters (λ, ω_6 and the crossover value). From the chart it appears that the points generated by the gradient descent algorithm lie within a neighbourhood of a local optima. Using the surface generated by the MFNN the gradient descent algorithm is able to interpolate between the points in the sample grid. This is particularly encouraging due to the complexity of the parameter space defined by the heuristic and problem domain.

6. CONCLUSIONS

The results of this initial experiment are promising. As far as the problem domain described in this paper is concerned, the next step is to create a grid using all parameters and thus attempt to locate a global optima. The methodology can then be tested on other multi parameter heuristics and other problem domains. In general however, there are several areas of possible experimentation:

- The number of points required in the input grid *G*: further investigation regarding the dependency between the number of heuristic parameters and the problem domain and the number of grid points necessary to generate a surface which accurately models the solution surface is required.
- The gradient descent technique used in this paper could be substituted with a global search strategy less prone to entrapment in a local optima such as tabu search or a genetic algorithm.
- The architecture of the MFNN used including the number of neurons in the hidden layer and the activation functions used. At this stage it is unclear what the optimal number of hidden neurons for a given heuristic and problem domain is, or how the optimal number varies with the number of parameters. Also further work is required to establish whether the choice is activation function is significant and if so which combinations of activation functions work best for a given heuristic and problem domain.

7. REFERENCES

[1] B. Kamgar-Parsi and B. Kamgar-Parsi, "Dynamical stability and parameter selection in neural optimization", *Proceedings of the International Joint Conference on Neural Networks*, vol. 4, pp. 566-571, 1992.

[2] Gates G.H. Jr. Merkle L.D. Lamont G.B. Pachter R., "Simple genetic algorithm parameter selection for protein structure prediction" 1995 IEEE International Conference on Evolutionary Computation vol.2. New York, NY, USA.

[3] Hoffmann KH. Salamon P., "The optimal simulated annealing schedule for a simple model" , *Journal of Physics A-Mathematical & General*, vol.23, no.15, 1990,UK.

[4] Smith, K. A. and Gupta, J. N. D., "Continuous function optimisation via gradient descent on a neural network approximation function", *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, Lecture Notes in Computer Science, vol. 2084, Springer-Verlag, Berlin, pp. 741-748, 2001.

[5] Smith, K. and Gupta, J., "Integrating Feedforward and Feedback Neural Networks for Optimisation", in C. Dagli et al. (Eds.), *Intelligent Engineering Systems Through Artificial Neural Networks: Neural Networks, Fuzzy Logic, Evolutionary Programming, Data Mining, and Rough Sets*, ASME Press, vol. 8, pp. 69-74, 1998.

[6] Bedingfield, S. and Smith, K. A., "A Comparison of Fitness Functions for Evolutionary Rule Generation", in M. Mohammadian (ed.), *Advances in Intelligent Systems: Theory and Applications*, IOS Press, 2000.

[7] Bedingfield, S. and Smith, K. A., "Evolutionary rule generation and its application to credit scoring", L.Reznik and V.Kreinovich (eds.), *Soft Computing in Measurement and Information Acquisition*, Physica-Verlag, Heidelberg, 2003.