Evolution of Fitness Functions to Improve Heuristic Performance*

Stephen Remde[1], Peter Cowling[1], Keshav Dahal[1] and Nic Colledge[1]

[1] MOSAIC Research Group, University of Bradford, Bradford, BD7 1DP, United Kingdom
{s.m.remde, p.i.cowling, k.p.dahal, n.j.colledge}@bradford.ac.uk
http://mosaic.ac/

Abstract. In this paper we introduce the variable fitness function which can be used to control the search direction of any search based optimisation heuristic where more than one objective can be defined, to improve heuristic performance. The method is applied to a multi-objective travelling salesman problem and the performance of heuristics enhanced with the variable fitness function is compared to the original heuristics, yielding significant improvements. The structure of the variable fitness functions is analysed and the search is visualised to better understand the process.

Keywords: Fitness Function, Evolution, Heuristic, Local Search, VFF.

1 Introduction

Optimisation heuristics are used when an optimal solution cannot be found in a reasonable amount of time. When the problem is just too complex to solve exactly, a heuristic method is used to find a sufficiently good or near optimal solution. One such type of heuristic is local search, which takes an initial solution and tries to improve it through a series of local perturbations. Local search has the problem of getting stuck in local optima. These are solutions from which a local change cannot improve the solution, however the solution is not globally optimal. Hence, a local move may not be the best globally, and the global fitness function may not be adequate for assessing local moves. Many ways to escape local optima have been introduced, including [1, 2, 3]. However, all these methods require modification of the local search. The method we introduce changes the search direction to avoid and escape local optima and can be applied to any search based optimisation heuristic without modification so long as two conditions are satisfied: (i) The objective function can be expressed in terms of two or more sub objectives (which is almost always the case in our experience of practical problems); and (ii) We have enough CPU time to run the heuristic many times. The search direction is determined by a Variable Fitness Function (VFF), which is evolved using a genetic algorithm. We apply this enhancement method to

* This work was funded by EPSRC and @Road Ltd, a Trimble Company under an EPSRC CASE studentship, which was made available through and facilitated by the Smith Institute for Industrial Mathematics and System Engineering

various heuristic methods for the TSP [4]. We show that given an optimisation heuristic *H*, we can create a better heuristic *VFF(H)* and that the extra time used by the VFF is far better than using random perturbations with the original heuristic.

In this paper we introduce the Variable Fitness Function and show how it can be used to enhance local searches for a multi objective problem, by varying the fitness function over the course of the search. Throughout this paper, the term *global fitness function* will be used where we mean the global fitness function that is used to assess the overall quality of solution so as to make a clear distinction between it and the *variable fitness function*.

The next section reviews literature related to the variable fitness function and the TSP test case. Section 3 describes how a variable fitness function is represented and evolved. Section 4 details the implementation of the variable fitness function and section 5 describes computational experiments and results. Section 6 draws final conclusions and analysis from the work and identifies future work.

2 Related Work

Local search (including constructive heuristics, which are a special case of local search) may fall well short of a global optimum when the search converges to a local optimum or a basin of attraction. A local optimum is a solution which has no better neighbouring solution, but such a solution may be far worse than the global optimum. Several methods have been described in literature. Here we describe three that are effective and have the practical advantage that they are easy to implement for complex, practical problems.

Simulated Annealing is a local search method that was inspired by the physical annealing process [3]. Simulated annealing works by changing the acceptance criteria of a local search operator. It will always accept moves which lead to a better solution, however it also has a chance to accept moves that make the solution worse. This probability of accepting a worse move is controlled by a cooling scheme and is inversely proportional to how bad the move is and how far into the search the process is. This helps diversity at the beginning and helps intensify the search toward the end. In the survey done by Kolisch and Hartmann [5] the heuristic was shown to be competitive and performed well ranking about midway of the tested heuristics for a the Resource Constrained Project Scheduling Problem (RCPSP).

Guided Local Search [2] attempts to modify the fitness function to change the direction the search heads when a local optimum has been found. Features of a solution are identified and penalties for solutions exhibiting these features are increased when the solution is stuck in a local optimum. It redefines the objective function thus:

$$f'(s) = f(s) + \lambda \sum_{i=1}^{F} p_i I_i(s)$$

where λ is the weighting for the guided local search, F is the number of features, p_i is the penalty value for the i-th feature and $I_i(s)=1$ when s exhibits feature i, 0

otherwise. When the search settles on a local optimum $s*$ the utility of penalising a feature is defined by:

$$util_i(s*) = I_i(s*) \times \frac{c_i}{1 + p_i}$$

The feature or features with the largest utility will be penalised by increasing their penalty values. This has the effect of changing the fitness function and forces the search to move in another direction.

Variable Neighbourhood Search (VNS) [1] is based on the idea of systematically changing the neighbourhood of a local search algorithm. Variable Neighbourhood Search enhances local search using a variety of neighbourhoods to "shake" the search into a new position after it reaches a local optimum. Several variants of VNS exist as extensions to the VNS framework [6] which have been shown to work well on various optimisation problems.

These local search metaheuristics all require modification of the local search. In the case of simulated annealing, only a minor change to the criteria of accepting a neighbouring solution is needed, however in guided local search and variable neighbourhood search, much larger changes are needed. The methods we introduce require no modification of the underlying local search and hence can easily be used to enhance any local search method. This becomes particularly important when trying to solve complex, real-world problems with a wide range of objectives and a detailed model, where the VFF approach provides a straightforward way to further enhance an existing approach.

The single objective travelling salesman problem [4] is a well known, well studied problem, often used to test single objective metaheuristics (for example [6] for VNS, [7] for GLS). [8] studies a multi-objective version of this which they call the J-objective TSP. Here, J different objectives are associated with travelling between each city. In practical applications these could represent factors such as distance, cost, travel time or some measure of traffic although uncorrelated objectives are used in [8]. Several greedy heuristics for initial tour construction exist for the TSP. Nearest neighbour and Multiple fragment are two we look at in this paper [9]. 2-opt is a local search heuristic often used to find good solutions quickly for the TSP. The basic move consists of choosing two edges and seeing if swapping them improves the tour. The result for Euclidean TSPs is that crossed edges get uncrossed and the tour is shortened. The TSP is a simple-to-express but difficult-to-solve problem which has been studied extensively and proven to be a good benchmark problem. It provides a very useful case study to investigate our VFF approach.

3 The Variable Fitness Function

The Variable Fitness Function describes how the weights of a weighted sum fitness function change over the iterations of a search process. Here we use a weighted sum linear objective, but the approach is immediately applicable to non-linear parameterised objectives. The variable fitness function is piecewise linear, describing the relative importance of objectives at each iteration. We consider two alternatives, the standard variable fitness function fixes the number of discontinuities and the

number of iterations between them. The adaptive variable fitness function allows the points of discontinuity to evolve along with the variable fitness function objective weights. These two methods will be described in detail in the next sections.

3.1 Standard Variable Fitness Function

We define a set of weights $\{W_{a,b}\}$ where a indexes the weight set ($a=0...A-1$) and b indexes the objective ($b=1...B$). We define I, the number of iterations between the weight sets. The variable fitness function is now defined as:

$$f(s,i) = \sum_{b=1}^{B} O_b(s) W_b(i)$$

where s is the solution to be evaluated and i is the iteration, $O_b(s)$ is the value of objective b for solution s, and

$$W_b(i) = W_{b,\lfloor i/I \rfloor} + \frac{i \bmod I}{I}(W_{b,\lfloor i/I \rfloor+1} - W_{b,\lfloor i/I \rfloor})$$

(i.e. the linear interpolation of the weight of objective b for iteration i)

Figure 1 shows how the weights of an example variable fitness function change over iterations. In this example, B=4, A=3 and I=100.

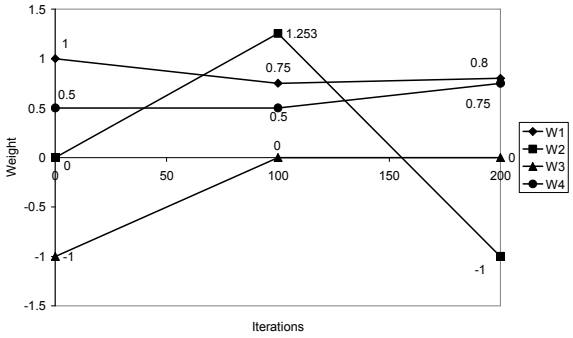

Figure 1. An example standard variable fitness function. The number of weight sets (3) and the number of iterations between them (100) are fixed.

3.2 Adaptive Variable Fitness Function

Initial experiments with the standard variable fitness function quickly showed its weakness. If the number of iterations was too small, the solution quality would suffer. If the number of iterations was to large, CPU time would be wasted. The adaptive variable fitness function does not require the optimal number of weight sets and iterations between them to be known. These are evolved along with the weight data to find appropriate values. This can lead to more complex variable fitness functions.

Figure 2 shows an example adaptive variable fitness function. This describes how the weights change over the iterations, for example, that the weight of objective 1 (W1) starts off at 2 and then after iteration 200 its importance starts to decrease and objective 3 (that has weight W3) is to be minimized, and its importance is high at the start and end of the search process.

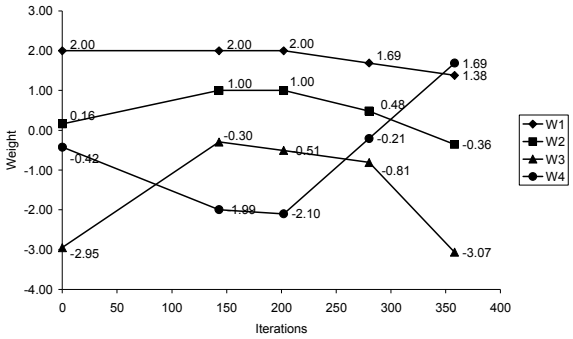

Figure 2. An example adaptive variable fitness function. In this example, the number of iterations between the weight sets and the number of weight sets may vary.

3.3 Evolution

Little work has been done in encoding piecewise linear functions such as these into chromosomes. [10] uses a complex encoding for polynomial expressions. The encoding is used to optimise a curve to fit a function described by a set of data points and is not an appropriate method in this case. The evolution here is similar to work done on tuning of parameters for another algorithm using genetic algorithms [11].

When optimizing the weights of the variable fitness function, each weight in the variable fitness function appears as a gene in a GA chromosome. When the adaptive variable fitness function is used, the iterations between the weight sets is also included. Figure 3 shows how the weight sets are mapped to the genes of a chromosome.

A modified version of 1 point crossover [12] will be used. It works the same way as normal 1-point crossover but the crossover point may only be on a weight set boundary. This method will keep mutually compatible weight sets together. The thick lines in Figure 3 show these crossover points. Each gene will have a chance to be mutated with a probability of p_{mut}, the mutation rate. Mutation will simply mutate the value of the gene by a random variable normally distributed around 0 and with the standard deviation defined for that weight. Hence $W_{a,b}$ is deviated by a value from the normal distribution $N(0, V_b)$ with probability p_{mut}. Where V_b is the standard deviation of mutation associated with objective b and p_{mut} is the probability of mutation for all alleles. This is similar to work done on mutation of artificial neural network weights

evolved using GAs [13] where the network weight is mutated by a random number selected from a normal distribution.

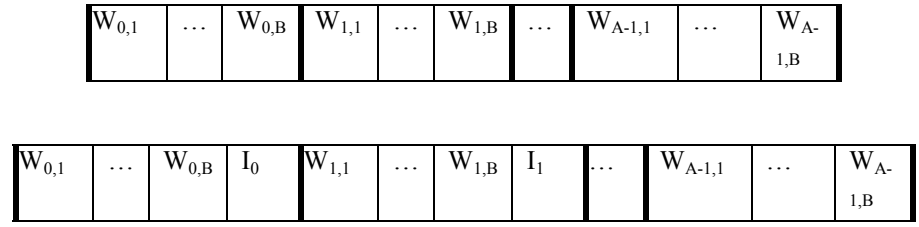

Figure 3. Mapping the weights to a chromosome for a standard variable fitness function (top) and an adaptive variable fitness function (bottom).

The initial population of variable fitness functions is generated at random. We may also seed the initial population with the global fitness function. These seeds are variable fitness functions where the weights are constant over all iterations and equivalent to the global fitness function. This may give the genetic algorithm a good individual to work from or provide good genetic material to create other individuals. For the random individuals $W_{a,b}$ is picked uniformly at random out of the interval $[L_b, U_b]$. Seeding the initial population with the global fitness function and using an elitist replacement scheme would ensure that in a worst case scenario, the best individual of the final population is a variable fitness function representing the global fitness function.

In the adaptive version there is also a p_{adapt} probability that the chromosome will change length. If a chromosome is to change length there is an equal probability it will either shrink or grow by one weight set. If it is to shrink, a random weight set is chosen and removed from the chromosome. If it is to grow, a new weight set is inserted between two randomly chosen adjacent weight sets. The inserted weight set does not change the shape of the variable fitness function as it is inserted exactly half way between the two adjacent weight sets and has weight values that are the mean of the bordering weight sets. The new weight set is then mutated. Lastly, the I_a genes also have a p_{mut} probability of being mutated. This gives the chromosomes a chance to get more and less complex and to also expand to more or less iterations.

We have introduced a lot of parameters in this section but our experiments using the TSP and other problems show that the performance of VFF is not sensitive to these parameters. L_b and U_b can be set to -1 and 1 respectively without losing any information as any weighted sum objective function can be normalised and the individual weights will lie within this range. We have found that V_b set to 5% of the range ($V_b = 0.05(U_b - L_b)$) also works well. All the experiments here have used these default values. In the experiments carried out in this paper, $p_{mut} = p_{adapt} = 0.05$.

4 Application to the Traveling Salesman Problem

The Travelling Salesman Problem (TSP) is a well studied optimisation problem which is often solved using a basic local search operator 2-opt [14]. We study a multi-objective variant of the TSP and use the variable fitness function to guide a 2-opt local search to find better solutions than 2-opt alone.

The TSP consists of a set of n cities, and a cost matrix c_{ij} such that $1 \leq i,j \leq n$ that defines the cost of travelling from city i to city j. The aim of the TSP is to determine a tour of minimum length visiting each city only once and returning to the starting city. The Symmetric TSP (STSP) add a further constraint that $c_{ij} = c_{ji}$ for all i,j. We study a variant of this such that each journey has B uncorrelated objectives associated with it. The global objective is a weighted sum of the B objectives. It should be noted that these problems can easily be converted into an STSP (and in fact we do to solve them exactly). This way of creating multi objective TSP problems is related to the work of Jaszkiewicz [8] who use uncorrelated objectives to make the symmetric TSP into a multi-objective problem.

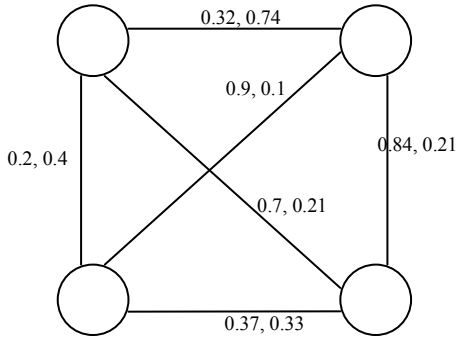

Figure 4. Example MO-TSP with 4 cities and 2 objectives

Figure 4 shows an example problem. The edge costs are uncorrelated. In the problems we look at, all costs are generated uniformly at random between 0 and 1. "Splitting" each edge in this way provides a way to convert a single-objective TSP to a multiple objective problem, which is readily applicable to other problems. In most of the real-world problems which we study, there are already many (often too many) objectives.

4.1 Variable Fitness Function Usage

We will use 3 different initial solution generation methods and improve them using 2-opt. We will then use the variable fitness function to enhance these methods in two ways. Firstly, just enhancing the 2-opt part and secondly enhancing the initial solution generation method as well. The methods are shown in Table 1. All of these heuristics for the TSP are well known – see [4] for details.

The three initial solution generation methods we will use are an arbitrary solution (*AR*), nearest neighbour (*NN*) and multiple fragment (*MF*). The arbitrary solution is simply the tour where all the nodes are visited in numeric order. Nearest neighbor starts the tour at a given point and repeatedly adds the nearest unvisited city to the city at a fixed end of the current partial tour until a complete tour is found. Multiple fragment is similar to nearest neighbor as at each iteration it adds an edge between the two closest unconnected cities whose connection does not form a cycle (unless it is the last edge to be added).

Table 1. Hueristics and variable fitness function enhanced versions used in our experiments

Heuristic	Description
AR	Solution is an arbitrary solution
NN	Solutions are generated using a stochastic nearest neighbor algorithm
MF	Solutions are generated using a stochastic multiple fragment algorithm
AR + 2opt	Solutions are generated using AR and improved using a stochastic 2-opt
NN + 2opt	Solutions are generated using NN and improved using a stochastic 2-opt
MF + 2opt	Solutions are generated using MF and improved using a stochastic 2-opt
AR + VFF(2opt)	Solutions are generated using AR then improved using 2-opt where the fitness function for 2-opt is evolved
NN + VFF(2opt)	Solutions are generated using NN then improved using 2-opt where the fitness function for 2-opt is evolved
MF + VFF(2opt)	Solutions are generated using MF then improved using 2-opt where the fitness function for 2-opt is evolved
VFF(NN + 2opt)	Solutions are generated using NN and improved using 2-opt where the fitness function for both the NN and the 2-opt algorithm is evolved
VFF(MF + 2opt)	Solutions are generated using MF and improved using 2-opt where the fitness function for both the MF and the 2-opt algorithm is evolved

2-opt is a simple local search heuristic which improves a TSP solution by finding edges (i, j) and (k, l) in the current tour such that $c_{ij} + c_{kl} > c_{ik} + c_{kl}$ and replacing edges (i, j) and (k, l) with (i, k) and (j, l). For each of the NN, MF and 2-opt we consider stochastic versions where instead of greedily choosing the best at each iteration, we choose peckishly [15], with equal probability, one of the best two possibilities at each iteration, so allowing us to use extended CPU time sensibly.

Table 2 shows the parameters used in the evolution. Picking the weights between -1 and 1 gives us the possibility to start the search in every direction, including those negatively correlated with the global fitness function.

Table 2. Parameters used to evolve the variable fitness functions for the MOTSP problems

Objective $b_.$	Initial Value $L_b...U_b$	Standard Deviation V_b
all	-1…1	0.1

5 Computational Experiments

Each of our ten methods was run 5 times for each of 5 problem instances. They were given the same CPU time (15 minutes) in which to find a solution and were restarted if they completed before the allotted time. The optimal solution of each of the 5 instances was found using CONCORDE [16]. The five instances have 100 cities and 2 objectives, equally weighted in the global fitness function. The quality of a method will be assessed by the average deviation from the optimal tour, measured by this global objective, over the 25 runs.

To tune the genetic algorithm parameters for the variable fitness function evolution, the genetic algorithm was run for 1000 fitness evaluations with different population sizes of 10, 20 and 40 in an attempt to find the best parameters. The population was seeded with *0*, *1* and *All* global fitness functions to see the difference. The parameter tuning experiments show that the genetic algorithm was not very sensitive to the parameters. A Population size of 20, and seeding with no global fitness functions were among the best set of parameters and were used for the rest of the experiments.

The comparative results are shown in Figure 5, showing each method's average deviation from the global optimum over 5 runs of 5 problems instances together with 90% confidence intervals. We can see that NN provides the weakest result. In fact, the stochastic version of *NN* is worse on average than normal *NN* (not shown here). 2-opt can be seen to improve the NN and MF heuristics considerably as expected. When we enhance the 2-opt with the variable fitness function, we can see significant further improvements. When also enhancing the NN or MF as well as the 2-opt we see different results. In the case of *VFF(NN + 2opt)* the solution quality improves again, however, for *VFF(MF + 2opt)* the solution gets worse when compared to *MF + VFF(2opt)*. This decrease in solution is, however, statistically insignificant at the 90% confidence level and could be because *MF + VFF(2opt)* is already producing very good solutions. Overall the results demonstrate that the *MF + VFF(2opt)* and *VFF(MF + 2opt)* perform the best. For every heuristic H in our experiments, VFF(H) performs significantly better than H (at the 90% confidence level). These results provide good evidence that the variable fitness function can be used to enhance a simple local search without needing knowledge of the problem or modification of the search technique.

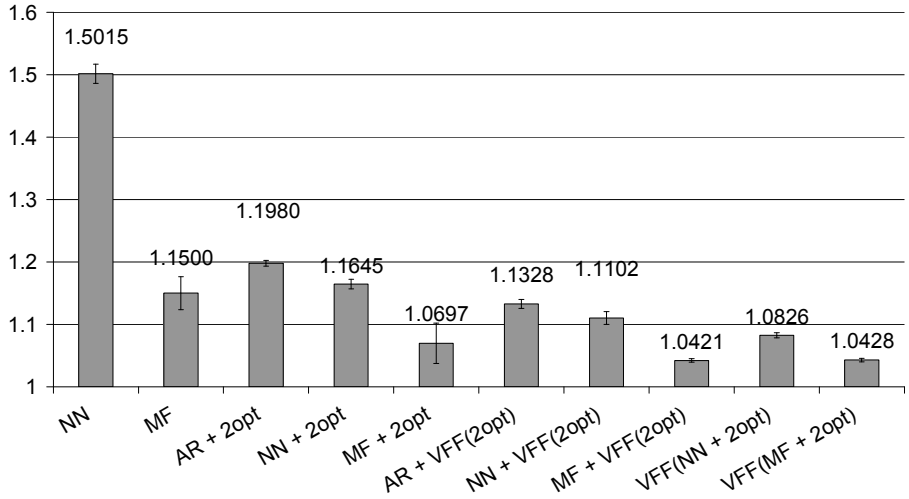

Figure 5. Average Deviation of the six tested methods from the optimal solution. Error bars show 90% confidence intervals.

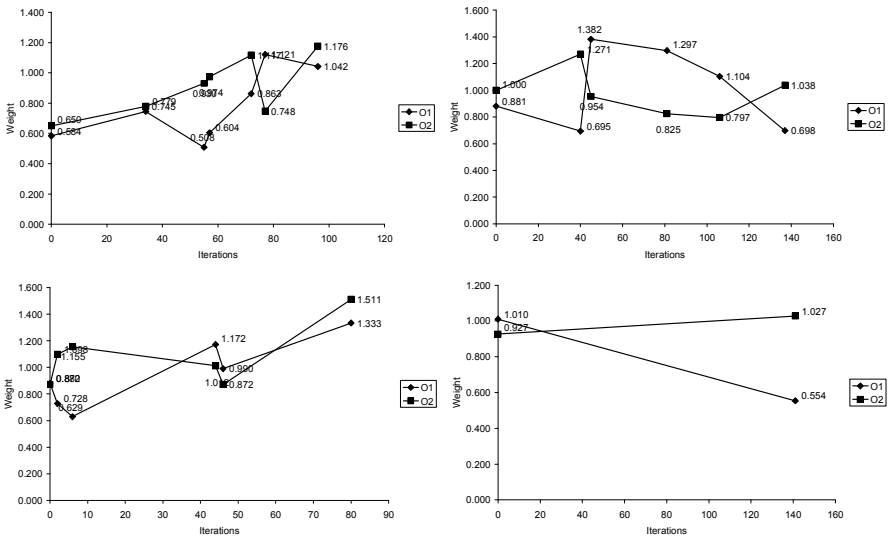

Figure 6. Sample of the best variable fitness functions evolved for the *VFF(NN + 2Opt)* heuristic for different MO-TSP problems.

Figure 6 show a sample of the best variable fitness functions evolved for different MO-TSP problem instances. They are quite different and have very little in common, ranging from really simple to quite complex. This implies, not surprisingly, that there is not a single good variable fitness function for this set of uncorrelated MO-TSP problems instances. This can be seen where the weights of variable fitness function change priority (for example at approximately iteration 75 of the top left graph). This

could be because the search has reached a local optimum and changing the direction toward the other objective avoids or escapes it. We will investigate this in more detail below. Each one is quite different because the problem instances have nothing in common. This is as expected because the objectives were generated randomly and uncorrelated.

To show that the evolved variable fitness functions exploit individual problem instances characteristics rather than the characteristics of the TSP itself, we used each of the evolved variable fitness functions for *VFF(NN + 2opt)* on the other problem instances for which it was not evolved. Figure 7 shows this comparison and indicates that using an incorrect variable fitness function is worse than using the global fitness function (comparing *Mismatch VFF(NN + 2 opt)* to *NN + 2 opt*).

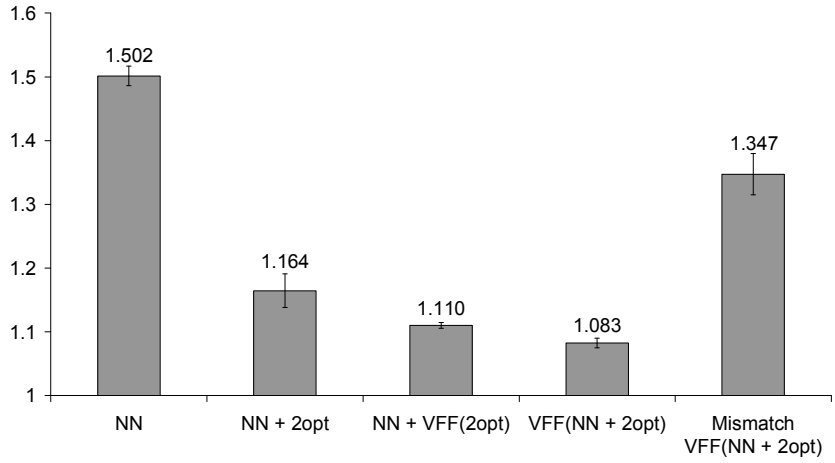

Figure 8. *Mismatch VFF(NN + 2 opt)* represents the average deviation from the optimal solution when using variable fitness function evolved for other problem instance.

For problems where there is significant correlation between the objectives in different instances, a VFF generated using historical problem instances is likely to show good performance on future problem instances. Our initial work on real-world scheduling problems (forthcoming) show that this is indeed the case.

Figure 9 shows a visualisation of the search process of a variable fitness function for a *AR + VFF(2opt)* search. The top plot shows the evolved variable fitness function and the plot below it attempts to visualise the search. *Current Solution Fitness* shows the fitness of the solution at each iteration. *Moves to Local Optima* and *Local Optima Fitness* show the number of 2-opt moves the current solution is to the 2-opt local optimum that would be found if the weights were fixed (at the VFF values) and the fitness of that local optimum. This shows us many things. When the *Moves to Local Optima* reaches zero the search is at a local optimum. When it increases after being zero it has changed search direction and escaped a local optimum. When *Local Optima Fitness* stays constant, the search is heading for the same local optima, and when it changes it has changed direction. When the *Local Optima Fitness* is worse than the *Current Solution Fitness* the search is heading in a

non intuitive way (in terms of the global fitness function), away from a globally unpromising region.

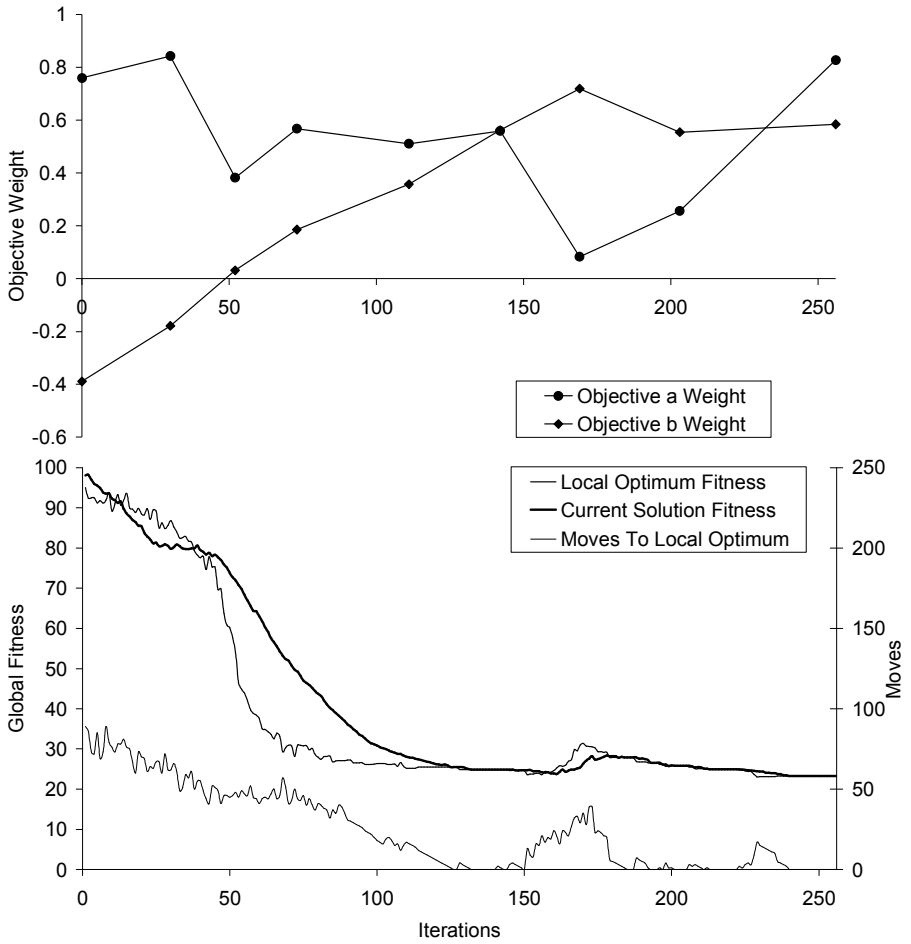

Figure 8. Visualising the search. Fitness use the right axis and are measured in term of the global fitness function.

From Figure 8 we see that the search is both escaping local optima and changing direction to avoid local optima throughout the search process. Until a good local optimum is reached the *Moves to Local Optimum* stays high, keeping the search away from local optima. After a good local optima is reached, the *Moves to Local Optimum* is increased by more radical changes in the objective weights. During the beginning of the search the *Current Solution Fitness* does not improve much and the *Local Optimum Fitness* varies a lot. This could indicate the search is moving to a different area of the search space. We then see a period of intensification where there is a large improvement in the solution quality and the *Local Optimum Fitness* varies less and the *Moves to Local Optimum* steadily decreases indicating it is heading toward the

same local optima. At around iteration 140, the search has reached a local optima after which, we see a change in the priority of weights in the variable fitness function which leads the search to another local optima at around iteration 200. During this period of "diversification" we can see that the *Local Optimum Fitness* is worse than the *Current Solution Fitness.* This is because the *Objective a* W*eight* has become very small and the search is probably pushing toward the other objective at the expense of this one.

6 Conclusions

The variable fitness function has demonstrated its ability to enhance local search methods to provide better solutions. We have shown that given a optimization heuristic *H* and a sufficiently large quantity of CPU time we can produce a better heuristic *VFF(H)*. We have also sown evidence that the variable fitness functions are learning to move the search to different parts of the search space when local optimum are encountered

The generation of a solution using VFF takes longer due to the evolutionary process. However, if the time is available, this method appears to be better than random perturbations to a local search and requires no modification of the local search unlike most common meta-heuristics. Although we have not empirically tested this method against and in conjunction with other common meta-heuristics, we feel that from limited testing, the variable fitness function would result in improvements and we will try this in future work.

Preliminary experiments with more complex, real world problems indicate that when using variable fitness functions on problems with correlated objectives and rather unusual, non-linear, real-world objectives, variable fitness functions can be evolved to work on a range of problem instances and hence the VFF can be evolved offline and then used online with no increase in CPU time.

References

1. Mladenovic, N. and Hansen, P.: Variable neighborhood search. Computers & Operational Research 24 (11): 1097-1100 Nov (1997)
2. Tsang, E. and Voudouris, C.: Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. Operations Research Letters 20 (3): 119-127 (1997)
3. Aarts, E.H.L. and Korst, J.H.M.: Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Computing, Whiley Chichester, 1989
4. Lawler, E.L. , Lenstra, J.K., Rinnooy Kan, A.H.G. and Shmoys, D.B. (Eds.): The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, Wiley, Chichester, 1985.
5. Kolisch, R. and Hartmann, S.: Experimental Investigations of Heuristics for RCPSP: An Update. European Journal of Oper. Res. 174 (1): 23-37 (2006)
6. Hansen, P. and Mladenovic, N.: Variable neighborhood search: Principles and applications. European Journal of Oper. Res. 130 (3): 449-467 May 1 (2001)

7. Burke, E.K, Cowling, P.I and Keuthen, R.: Effective Local And Guided Variable Neighbourhood Search Methods for the Asymetric Travelling Salesman Problem. Applications Of Evolutionary Computing, Proceedings Lecture Notes In Computer Science 2037: 203-212 2001
8. Jaszkiewicz, A.: Genetic Local Search for Multi-Objective Combinatorial Optimisation. European Journal of Operational Research 137 (1): 50-71 Feb 2002.
9. Johnston, D.S. amd McGeoch, L.A.: The Travelling Salesman Problem: a case study in local optimisation. In E.J.L Aarts J.k. Lenstra (Eds.), Local Search in Combinatorial Optimisation, Wiley, London (1997)
10. Potgieter, G. and Engelbrecht, A. P.: Genetic Algorithms for the Structure Optimisation of learned Polynomial Expressions. Applied Mathematics and Computation 186 (2): 1441-1466 Mar 2007
11. Shimojika, K., Fukuda, T., Hasehawa Y.: Self-Tuning Fuzzy Modeling with adaptive membership function, rules, and hierarchical structure-based on Genetic Algorithm. Fuzzy Sets And Systems 71 (3): 295-309 (1995)
12. Reeves, C.R.: Genetic Algorithms and Combinatorial Optimization. In V.J. Rayward-Smith (Ed.), Applications of Modern Heuristic Methods, Alfred Waller, Henley-on-Thames, 1995, pp. 111–125.
13. Yao, X.: Evolving artificial neural networks.: Proceedings Of The IEEE 87 (9): 1423-1447 SEP 1999
14. Croes, G. A.: A Method for Solving Traveling Salesman Problems. Operations Res. 6 (1958), 791-812.
15. Corne, D. and Ross, P.: Peckish Initialisation Strategies for Evolutionary Timetabling. Proceedings of PATAT. LNCS 1153, pp 227-240, 1995
16. Applegate, D., Bixby, R., Chvatal, V., Cook, W.: Concorde TSP Solver. Website http://www.tsp.gatech.edu/concorde.html