Function Optimization using Guided Local Search

Chris Voudouris and Edward Tsang
{voudcx,edward}@essex.ac.uk

Technical Report CSM-249
September 1995

Department of Computer Science
University of Essex
Colchester, CO4 3SQ, UK

Abstract

In this report, we examine the potential use of Guided Local Search (GLS) for function optimization. In order to apply GLS, the function to be minimized is augmented with a set of penalty terms that enable local search to escape from local minima. The function F6 is used to demonstrate the proposed technique.

1. Introduction

In this report, we present preliminary findings on the potential use of Guided Local Search (GLS) for function optimization. GLS is a meta-heuristic for guiding local search [3] to escape local minima and visit promising solutions. GLS has been used to tackle difficult combinatorial optimization problems [7,5] and derives itself from the GENET network for constraint satisfaction problems [6]. Function optimization can be seen as a combinatorial problem by encoding real variables as binary strings [2]. In the simple case of binary encoding, binary string values are converted to integers which then are scaled by the appropriate coefficient to give real values in the desired range (i.e. domain of variable) [2]. A typical move used is to flip the value of a bit in the binary string representing the solution (comprises the binary strings of the function's variables) [1,8]. In the x-y plane, bit flips translate to "jumps" in either x or y direction. The more significant the bit changed, the larger the step of the "jump" performed. Local search starting from a random binary string examines all possible bit changes performing that which results to the maximum reduction in cost (minimization case). The new better solution replaces the old solution and the procedure continues until a solution is reached for which no further improvement is possible. This solution is called a local minimum. GLS iteratively uses such a local search procedure modifying the cost function each time local search reaches a local minimum.

2. F6 Function

GLS may be used as a last resort method for optimizing functions with numerous local minima. Example of such function, mentioned many times in the literature, is F6 [1,2,8]:

$$F6(x,y) = 0.5 + \frac{\sin^2\sqrt{x^2+y^2} - 0.5}{\left[1.0 + 0.001\cdot\left(x^2+y^2\right)\right]^2} \quad (1)$$

minimized in the domain [-100,100]×[-100,100]. F6 has been suggested as a benchmark for Genetic Algorithms [4]. A cross section of the function is shown in figure 1. The global minimum of F6 is located at (0,0) where the function takes the value 0. The basin of the global minimum is very narrow and therefore difficult to reach unless a lucky start is made from within the domain of attraction of the global minimum. The many local minima of the function are arranged in co-centric cycles around the global minimum and they can easily trap a hill-climber. In F6, local gradients provide limited if any information on the location of the global minimum. GLS may be utilized for such problems to help local search to escape from local minima.

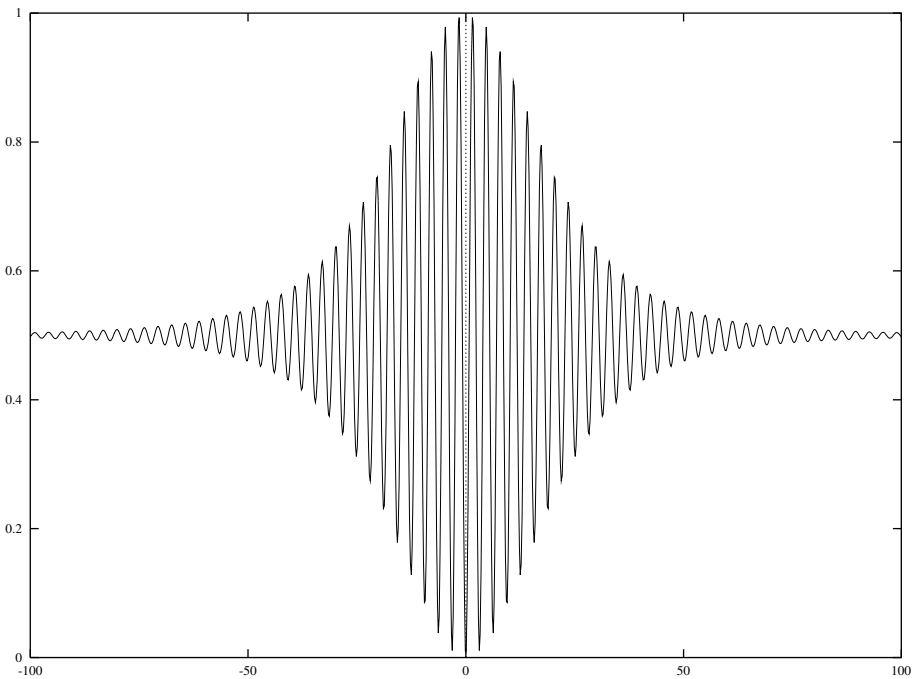


Figure 1. Cross section of F6.

3. Guided Local Search

GLS introduces constraints that modify the landscape and guide local search out of local minima. Constraints are based of information gathered during the search process. For example, if local search reaches a local minimum then a guess can be made that the global minimum is unlikely to reside in the surrounding area. Constraints could be then introduced that exclude this area from being searched in future iterations. These constraints are essentially soft because we can not be sure that local search thoroughly searches the space around the local minimum solution.

In GLS, a set of features is defined that allow us to constraint solutions. A feature can be any solution property represented by an indicator function (see [7]). A simple setting for function optimization is to divide the domains of variables in a number of non-overlapping and equally-sized intervals. Let us consider the variable $x \in (a,b]$, a set of features f_i, i=1, ...,n, can be defined by the intervals *(a0=a,a1], (a1, a2], ..., (an-1, an=b]* as follows:

$$I_i(x) = \begin{cases} 1, & x \in (a_{i-1}, a_i] \\ 0, & otherwise \end{cases} \tag{2}$$

Each feature f_i is attached a penalty parameter p_i which allows us to penalize solutions that are characterized by the feature such that they are being avoided. The cost function is augmented with penalty terms to form the augmented cost function. This function takes the place of the original function and it is minimized instead. The augmented version of F6 is defined as follows:

$$H(x,y) = F6(x,y) + \lambda \cdot \left(\sum_{i=1}^{n} I_{xi}(x) \cdot p_{xi} + \sum_{j=1}^{m} I_{yj}(y) \cdot p_{yj} \right) \quad (3)$$

where n the number of features defined over the domain of x, m the number of features defined over the domain of y, and λ the parameter that controls the relative importance of constraints with respect to the primary cost term (i.e. function to be minimized). Initially, all penalty parameters of features are set to 0 ($p_{xi} = 0$, $p_{yj} = 0$, $i = 1, ..., n$, $j = 1, ..., m$). Each time local search settles in a local minimum, we simply increment by one the penalty parameters of the features exhibited by the local minimum (only two at a time). This increases by $2*\lambda$ the cost of all solutions that lie in the intersection of the zones corresponding to the penalized features and by λ the cost of all solutions that lie in either one of these zones (see figure 2). As a result, local search will mainly avoid the rectangular area with centre the local minimum and also in a lesser degree the two zones that run parallel to the coordinate axes as shown in figure 2. This simple technique can be used to minimize arbitrary functions. In fact, there is nothing that binds the method to F6 and may be used for other functions too. In the continue, we examine the results obtained for F6.

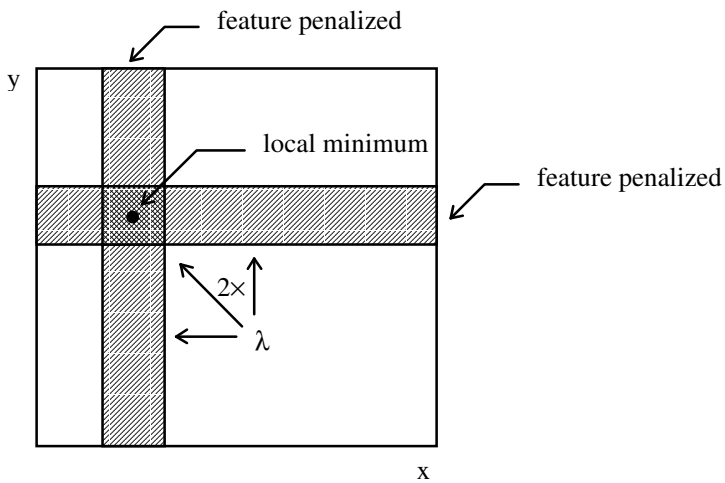


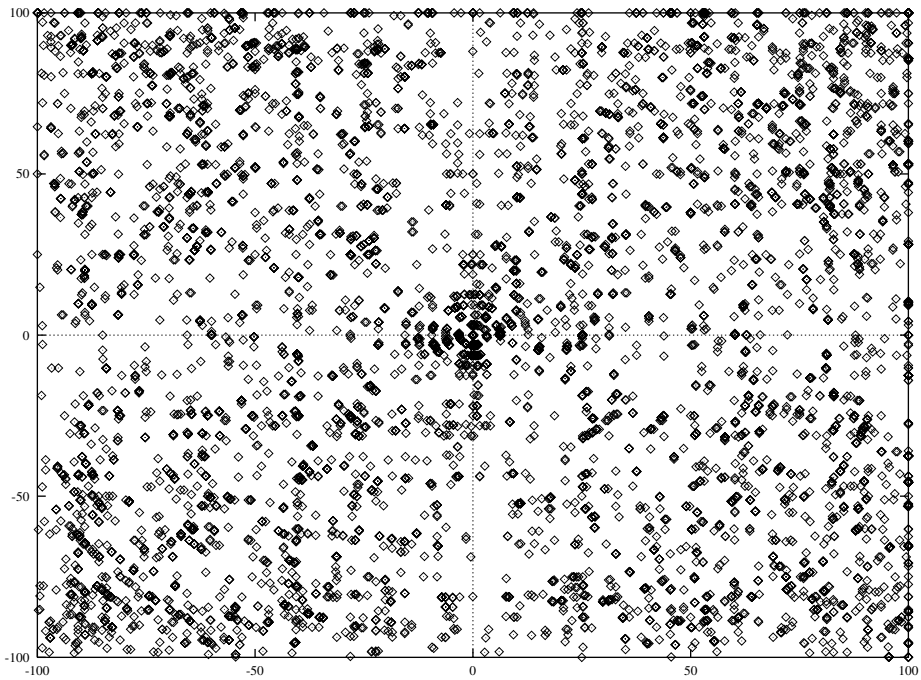
Figure 2. Changes in cost due to penalizing the features exhibited by a local minimum.

4. Results

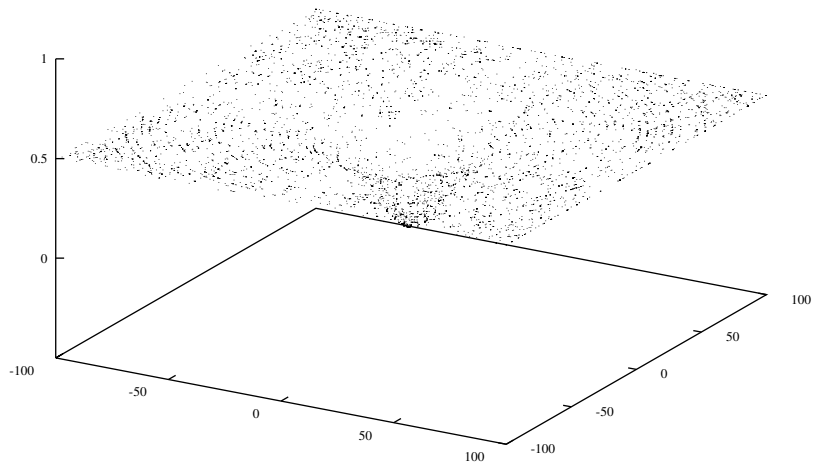
We used 22-bits for representing each variable. An equal number of features was used to cover the domain of each variable (n=m). The algorithm was relatively insensitive to parameter λ and performed well for values of λ greater than 0.2. The value 0.25 for λ was used in the tests. Experiments performed for varying n (i.e. number of features per variable) to determine how this parameter affects GLS. The values tried for n were 5,10,15,20,50,100. Fifty runs from random solutions (random binary strings) performed for each value of n considered with the iteration limit set to 10K local search cycles. Table 1 illustrates the results obtained. The best setting proved to be n=m=5. Under this setting, the algorithm succeeded in finding the exact optimal solution (0,0) in 100% of the runs. Under all settings, the algorithms found the exact optimum many times.

No. of features	n=m=5	n=m=10	n=m=15	n=m=20	n=m=50	n=m=100
Mean Cost	0.00E+00	4.55E-11	3.19E-10	2.73E-10	1.97E-04	3.21E-04
Best Solution	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Worse Solution	0.00E+00	2.28E-09	2.28E-09	2.28E-09	9.72E-03	9.72E-03
Mean Iterations	2287.32	2566.22	2954.08	3526.9	4132.66	3738.48
Mean Time	2.823333	3.150668	3.634334	4.382333	5.188333	4.654
Mean Funct. Eval.	104958.6	117778.8	135588	161878.4	189675.6	171578.5
Optimal Runs	50	49	43	44	31	22
Total runs	50	50	50	50	50	50

Table 1. GLS performance on F6 (Time in CPU seconds on a DEC Alpha 3000/600).



(a)



(b)

Figure 3. (a) Points visited during the first 10K iterations of local search. (b) 3-D View of the points.

This performance further improves if more time is given to the algorithm. For example, in the case (n=m=*100*) where most failures recorded (28 out of 50 runs), we performed the same experiment but this time allowing the algorithm to complete 100K local search iterations. The performance of GLS significantly improved and the algorithm found the exact optimum in 50 out of 50 runs (no failures).

The main observation made was that GLS performance degraded as the number of features used increased. More features meant more effort to leave a particular area but also more careful exploration. For this particular function, diversification of search to cover the whole search space proved important to find quickly the global minimum. The distribution of points visited for n=m=10 during 10K iterations of local search is shown in figure 3a. During the particular run that generated figure 3a, the optimal solution was found early and after 1965 iterations. Despite that, the algorithm left to continue until 10K iterations were completed. As one can see in figure 3a, the algorithm distributed its efforts over the whole of the search space but visiting mainly local minima. That is why points in figure 3a are arranged in co-centric cycles around the point (0,0). This is more clearly demonstrated in figure 3b where a 3-D view of the visited points is shown. The shape formed in figure 3b is exactly the bottom part of F6 which suggests that the points are actually local minima in the great majority.

5. Conclusions

GLS for combinatorial optimization problems may be used as a last resort method to optimize functions with numerous local minima considered difficult for hill-climbers. The application of GLS to optimize the F6 function, a benchmark for Genetic Algorithms [4], was examined. GLS repeatedly located the exact optimum of the function and that despite the quantization errors due to the binary string representation used.

References

[1] R. Battiti and G. Tecchioli, "The Reactive Tabu Search", *ORSA Journal on Computing*, Vol. 6, 126-140 (1994).

[2] L. Davis (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[3] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall,1982.

[4] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization", In *Proc. 3rd Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, 51-60 (1989).

[5] E. Tsang and C. Voudouris, "Fast Local Search and Guided Local Search and Their Application to British Telecom's Workforce Scheduling Problem", Technical Report CSM-246, Department of Computer Science, University of Essex (1995).

[6] E. Tsang, *Foundations of Constraint Satisfaction*, Academic Press, 1993.

[7] C. Voudouris and E. Tsang, "Guided Local Search", Technical Report CSM-247, Department of Computer Science, University of Essex (1995).

[8] D. L. Woodruff and E. Zemel, "Hashing vectors for tabu search", *Annals of Operations Research* 41, 123-137 (1993).