A Heuristic Algorithm for Function Optimization

Janez Brest, Sašo Greiner, Borko Bošković and Viljem Žumer
University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova 17, 2000 Maribor, Slovenia
Phone: ++386-2-220-7452 Fax: ++386-2-211-178 E-mail: janez.brest@uni-mb.si

Abstract

In this paper we present a heuristic algorithm for optimization of mathematical functions. Our algorithm has been tested on several function optimization problems. The performance of the heuristic algorithm is compared with the performance of genetic algorithms.

Experimental results using benchmark optimization problems confirm that our algorithm is comparable with algorithms from literature considering the quality of solutions found.

I. INTRODUCTION

From the beginning of Genetic Algorithms (GAs) research it has been known that choosing the best parameters and operators has significant impact on the optimization process [8]. However, it was demonstrated that every particular problem needs to have its parameters adjusted to its own requirements.

Finding robust variation operators or control parameter settings is not a trivial task since their interaction with the performance of an evolutionary algorithm is complex and the optimal choices are problem dependent [2].

There are many papers that proposed techniques for adapting control parameter setting associated with genetic operators [6, 2]. The goal is to obtain fuzzy rule bases that produce suitable control parameter values for allowing the genetic operator to show an adequate performance on the particular problem to be solved. The empirical study of an instance of the technique has shown that it adapts the parameter settings according to the particularities of the search space allowing significant performance to be achieved for problems with different difficulties [2].

Many researchers [5] have shown that genetic algorithms (GAs) perform well for global searching, but they usually take a relatively long time to converge to the optimum. On the other hand, local search methods can quickly find the local optimum of a small region of the search space, but are typically poor global searchers. Therefore, local search methods have been incorporated into GAs in order to improve their performance through what could be termed as learning. Such hybrid GAs often known as memetic algorithms have been used successfully to solve a wide variety of realistic problems [9].

Davis [1] argues that genetic algorithms when hybridized with the most successful local search methods or memes for a particular problem give the best of both worlds. If implemented correctly, these algorithms should do better than the traditional genetic algorithms or local search alone. Nevertheless, this also means that unless one correctly chooses the right meme, a memetic algorithm may not perform at its optimum, or it may even be worse than using the genetic algorithm or the local improvement procedure itself [9].

Winter et al. [8] introduced a new and more flexible internal structure design, where the parameters, the operators and the structure can adapt themselves at each optimization step. This new structure is named a Flexible Evolution Agent (FEA).

The performance results for different Evolution Strategies (ESs) and optimization with flock (OF) are compared by Črepinšek et all. in paper [4].

In this paper we propose a heuristic algorithm based on local search optimization. Results obtained by our algorithm are comparable with results from literature.

In [8] authors present some sampling methods that are used in a process to get new solution x_{new}: only over any of the variables or over the all variables of x_{old}. We introduce a method to calculate x_{new} which is based on random over all variables x_{old}.

The rest of paper is organized as follows. In the next section the widely used local search heuristic is presented. In Section III our heuristic algorithm is described. Numerical benchmark functions that we used for evaluating our algorithm are analyzed in Section IV. In Section V results for all algorithms on benchmark problems are presented. Finally, Section VI concludes the paper.

II. LOCAL SEARCH

Local optimization is a well known and widely used general purpose heuristic. A generic procedure for local search optimization is shown in Figure 1 [7, 3]. Step 1 is the *initialization step* which produces initial solution S, step 2 is the *optimization step* which attempts to improve the existing solution through the local search.

```
1      find initial solution S
2      while not done do
2.1      transform S into S'
2.2      if S' is better then S then S = S'
3      output S
```

Figure 1: Generic local search procedure

Finding initial solution may be random or can be obtained by some algorithm. The result in step 2 is usually something like "no improvement for some (long) time". Transformation in 2.1 should be cheap in comparison to finding initial solution in step 1.

III. HEURISTIC ALGORITHM

When our algorithm was constructed, we were inspired by local search procedure, which is described in previous section.

Initial solution in our algorithm is obtained randomly using `rand()` function in C/C++ programing language.

Step 2.1 from figure 1 is calculation of a new solution vector from temporary best solution vector. Next lines of original C/C++ code present the calculation of new `X` vector:

```
// random real number [0, 1)
double mRandom() {
  return rand()/(RAND_MAX + 1.0);
}

void Border(double X[MAX_N], int n, double border)
{
  for (int i=0; i<n; i++) {
    if(X[i] < -border) X[i] = -border;
    if(X[i] > border)  X[i] = border;
  }
}

// calculate new vector X from vector Xbest
void newX(int n, double border, double X[MAX_N],
          double Xbest[MAX_N], int k)
{
  if (0 == k) {                 // Step 1
    for (int i=0; i < n; i++) {
      X[i] = -border + (2*border)*mRandom();
    }
  }
  else {                        // Step 2.1
    for (int i=0; i < n; i++) {
      const double mulConst = 100.0;
      double p = pow(10, mulConst*mRandom());
      X[i] = Xbest[i] +
           (-border + (2*border)*mRandom()) / p;
    }
    Border(X, n, border);  // border check
  }
}
```

Function `newX()` includes both initialization and optimization steps. When new `X` vector is calculated, all its components are changed. Function `Border()` checks interval bounds of the vector's definition space. Note, that definition interval is symmetric. If any of the vector's components is out of bounds, we assign a new value to that component where the value is equal to the bound value. For example, by f_{Gri} function the definition space is $-600.0 \leq x_i \leq 600.0$ and we use value -600.0 if the value of x_i component is less than -600.0 and we use value 600.0 if the value of x_i component is greater than 600.0.

Step 2.2 from figure 1 is also straightforward, new evaluated function value is used in comparison with the temporary best function value. If new value is better than the temporary best function value, the new one is stored and the new `X` vector is also stored.

The next code fragment illustrates an example of function calculation for f_{sch} function:

```
// Spherical
void F1(int nEvaluation, int n, double border) {
  double X[MAX_N], Xbest[MAX_N], f, fbest=DBL_MAX;

  for (int k=0; k < nEvaluation; k++) {
    newX(n, border, X, Xbest, k);
    f = 0.0;

    for (int i=0; i < n; i++) {    // function
      f += X[i]*X[i];
    }
    better(f, fbest, X, Xbest, n); // Step 2.2
  }
  printResult(Xbest, fbest, n);
}
```

`F1` function represents f_{sch} function. Function `F1` is called with values 600 000, 25, 5.12 for number of evaluations, size problem (dimensionality), interval definition, respectively.

Our algorithm stores only one vector as the temporary best solution, while GA needs more vectors to store a whole population.

At a higher level of abstraction, maybe one can look at our heuristic algorithm as GA with the population size of 1.

IV. NUMERICAL BENCHMARKS FUNCTIONS

For the experiment, we have considered six frequently used test functions [2]:

1. $f_{sph}(\overrightarrow{x}) = \sum_{i=1}^{n} x_i^2$
   $-5.12 \leq x_i \leq 5.12$
   $f_{sph}(x^*) = 0$

2. $f_{Ros}(\overrightarrow{x}) = \sum_{i=1}^{n-1}(100 \cdot (x_i^2 - x_{i+1})^2 + (x_i - 1)^2)$
   $-5.12 \leq x_i \leq 5.12$
   $f_{Ros}(x^*) = 0$ (see Figure 2)

3. $f_{Sch}(\overrightarrow{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$
   $-65.536 \le x_i \le 65.536$
   $f_{Sch}(x^*) = 0$

4. $f_{Ras}(\overrightarrow{x}) = a \cdot n + \sum_{i=1}^{n} (x_i^2 - a \cdot \cos(\omega \cdot x_i))$
   $a = 10, \omega = 2\pi$
   $-5.12 \le x_i \le 5.12$
   $f_{Ras}(x^*) = 0$ (see Figure 3)

5. $f_{Gri}(\overrightarrow{x}) = \frac{1}{d} \sum_{i=1}^{n} (x_i^2) - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$
   $d = 4000$
   $-600.0 \le x_i \le 600.0$
   $f_{Gri}(x^*) = 0$

6. $ef_{10}(\overrightarrow{x}) = f_{10}(x_1, x_2) + ... + f_{10}(x_{n-1}, x_n) +$
   $+ f_{10}(x_n, x_1)$
   $f_{10}(x, y) = (x^2 + y^2)^{0.25} \cdot [\sin^2(50 \cdot (x^2 + y^2)^{0.1}) + 1]$
   $x, y \in (-100, 100]$
   $ef_{10}(x^*) = 0$

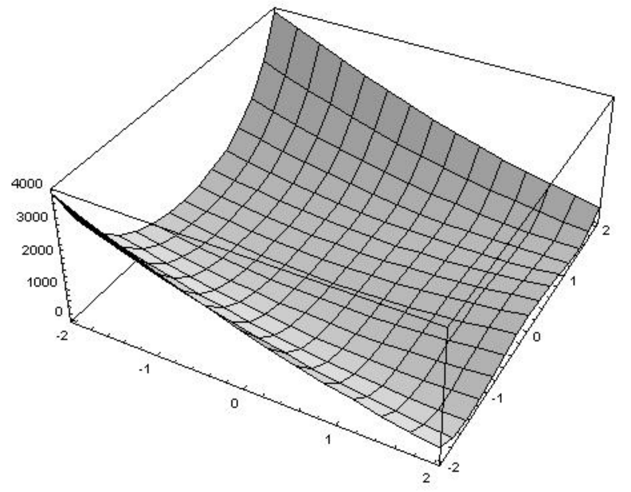

Figure 2: 2D graph of $f_{Ros}(x)$ function

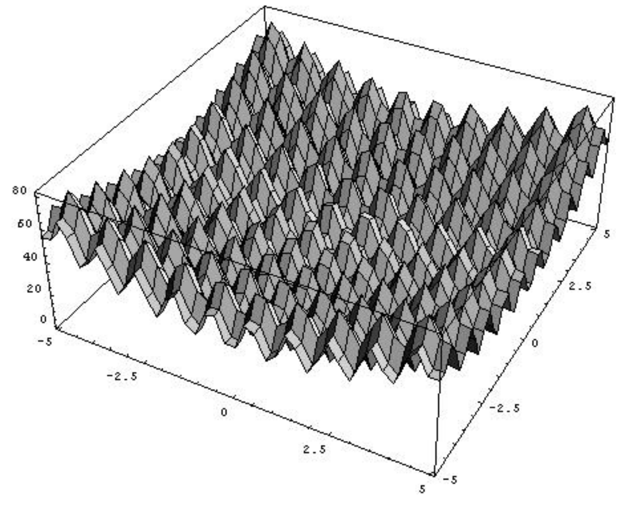

Figure 3: 2D graph of $f_{Ras}(x)$ function

f_{sph} is a continuous, strictly convex, and unimodal function.

f_{Ros} is a continuous and unimodal function, with the optimum located in a steep parabolic valley with a flat bottom. This feature will probably cause slow progress in many algorithms since they must continually change their search direction to reach the optimum. This function has been considered by some authors to be a real challenge for any continuous function optimization program. A great part of its difficulty lies in the fact that there are nonlinear interactions between the variables, i.e., it is nonseparable.

f_{Sch} is a continuous and unimodal function. Its difficulty concerns the fact that searching along the coordinate axes only gives a poor rate of convergence, since the gradient of f_{Sch} is not oriented along the axes. It presents similar difficulties to f_{Ros}, but its valley is much narrower.

f_{Ras} is a scalable, continuous, separable, and multimodal function, which is produced from f_{sph} by modulating it with $\alpha \cdot \cos(\omega \cdot x_i)$.

f_{Gri} is a continuous and multimodal function. This function is difficult to optimize because it is nonseparable and the search algorithm has to climb a hill to reach the next valley.

f_{10} is a function that has nonlinear interactions between two variables. Its expanded version ef_{10} is built in such a way that it induces nonlinear interaction across multiple variables. It is nonseparable as well.

V. RESULTS

Algorithm T-FBs is the best of six algorithms by Herrera et al. [2]. The T-FBs algorithm was executed 30 times with 10000 generations, and the population size was 60 chromosomes. Therefore, we can calculate the number of function evaluations: 600000 $(10000 \cdot 60)$.

We want to make a performance comparison of two different algorithms and we choose the equal number of evaluations. Our algorithm was executed 30 times, and the stop condition was 600000 evaluations for each function. The dimension of the search space is 25.

In our experiment the same six mathematical function were used as in [2]. The obtained results are presented in Table 1. For each function the best value (Min.) and the average value (Avg.) are presented for the T-FBs algorithm and our algorithm, respectively.

For sphere model function (f_{sph}) the T-FBs algorithm gets better values, but it can be noticed that our algorithm gets good solution, too.

By Generalized Rosenbrock's function (f_{Ros}) our algorithm outperforms T-FBs algorithm.

By Expansion of f_{10} function (ef_{10}) T-FBs algorithm outperforms our algorithm. Neverless, our algorithm gives good results.

Algorithms perform the same in two cases, Schwefel's Problem 1.2 (f_{Sch}) and Griewangk's function (f_{Gri}).

Table 1: Results of experiments and their comparison with results from literature.

Algorithm	f_{sph} Min.	f_{sph} Avg.	f_{Ros} Min.	f_{Ros} Avg.	f_{Sch} Min.	f_{Sch} Avg.
T-FBs	2.76e-201	2.69e-200	8.31e-2	1.02e+1	5.88e-11	9.25e-9
Our	1.96e-37	8.71e-36	5.57e-26	1.65e-24	1.82e-10	1.26e-8
Algorithm	f_{Ras} Min.	f_{Ras} Avg.	f_{Gri} Min.	f_{Gri} Avg.	ef_{10} Min.	ef_{10} Avg.
T-FBs	0	3.32e-2	0	0.00e0	1.58e-48	3.55e-25
Our	0	1.11e-17	0	5.33e-2	5.81e-9	9.85.e-9

T-FBs algorithm gets better average solution value by f_{Sch} function.

By Generalized Rastrigin's function (f_{Ras}) our algorithm gets better average solution value.

The advantages of our algorithm are:

- it requires a small amount of memory during execution. For example, GAs need much more main memory to store population.
- has simple implementation.
- no need for parameters and/or operators adjustments, that can be found in GAs, etc.,
- and finally, but not less important, it is fast.

A feature of this algorithm is a natural mapping onto coarsely grained parallel architectures. We plan to utilize networked workstations and PCs to solve function optimization in parallel on more difficult problem classes. Future work includes research and study of cases, when search space becomes more complex.

VI. CONCLUSION

In the paper a heuristic algorithm for numerical function optimization is presented. The algorithm is based on a generic local search optimization procedure.

Performance of our algorithm is compared to the algorithm taken from the literature and the obtained results shows that our heuristic algorithm is comparable with other well known algorithms.

References

[1] Lawrence Davis, editor. *Handbook of Genetic Algorithms*, New York, 1991. Van Nostrand Reinhold.

[2] F. Herrera and M. Lozano. Adaptive Genetic Operators Based on Coevolution with Fuzzy Behaviors. *IEEE Transaction on Evolutionary Computation*, 5(No. 2):149–165, April, 2001.

[3] Janez Brest and Janez Žerovnik. An approximation algorithm for the asymmetric traveling salesman problem. *Ric. oper.*, 28:59–67, 1999.

[4] Matej Črepinšek, Marjan Mernik, Viljem Žumer. Using flocks for solving numerical optimization problem. In Diana Šimić Vlado Glavinić, Vesna Hljuz Dobrić, editor, *Proceedings of the 24th International Conference on Information Technology Interfaces*, pages 395–400, Cavtat, Croatia, 2002.

[5] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, 1996. Contains introductory chapter on LCS.

[6] Mitchell A. Potter and Kenneth De Jong. A cooperative coevolutionary approach to function optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature – PPSN III*, pages 249–257, Berlin, 1994. Springer. Lecture Notes in Computer Science 866.

[7] R. Sosič and G. D. Wilby. Using the Quality-Time Tradeoff in Local Optimization. In *Proceedings of IEEE Second ANZIIS Conference, Brisbane*, pages 253–257, Dec, 1994.

[8] Gabriel Winter, Blas Galvn, Silvia Alonso, and Begoña González. Evolving from genetic algorithms to flexible evolution agents. In Erick Cantú-Paz, editor, *Late Breaking Papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 466–473, New York, NY, July 2002. AAAI.

[9] K. W. Wong Z. Ning, Y. S. Ong and M. H. Lim. Choice of memes in memetic algorithm. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2003), Special Session on Optimization using Genetic, Evolutionary, Social and Behavioral Algorithms*, Singapore, 2003. http://ntu-cg.ntu.edu.sg/ysong/conference/ysongCIRAS03.pdf.