SALO: COMBINING SIMULATED ANNEALING AND LOCAL OPTIMIZATION FOR EFFICIENT GLOBAL OPTIMIZATION

Rutvik Desai
Indiana University
rudesai@indiana.edu

Rajendra Patil
Los Alamos National Laboratory
rbp1@lanl.gov

Abstract

Simulated annealing is an established method for global optimization. Perhaps its most salient feature is the statistical promise to deliver a globally optimal solution. In this work, we propose a technique which attempts to combine the robustness of annealing in rugged terrain with the efficiency of local optimization methods in simple search spaces. On a variety of benchmark functions, the proposed method seems to clearly outperform a parallel genetic algorithm and adaptive simulated annealing, two popular and powerful optimization techniques.

1. INTRODUCTION

The goal of optimization is, given a system, to find the setting of its parameters so as to obtain the optimal performance. The performance of the system is given by an evaluation function. Optimization problems are commonly found in a wide range of fields, and it is also of central concern to many problems in artificial intelligence and machine learning. In situations where the space of parameters cannot be searched exhaustively and the evaluation function cannot be subjected to analytical methods, as is often the case in real world problems, heuristic methods have to be used. We sketch two such methods below.

1.1 Simulated Annealing

Simulated annealing (SA) [1] is an optimization technique inspired from Monte Carlo methods in statistical mechanics. It attempts to avoid local optima by probabilistically taking non-locally optimal steps in the search space. The probability of taking such steps decreases with the "temperature" of the system, which in turn decreases with time. SA is able deal with evaluation functions with quite arbitrary degrees of nonlinearities, discontinuities and stochasticity and can process quite arbitrary boundary conditions and constraints imposed on these evaluation functions [1,2]. It has been shown [3] that with a "large enough" initial temperature and a proper temperature schedule, SA guarantees a globally optimal solution.

We use Adaptive Simulated Annealing (ASA) [4], an implementation of a method known as Very Fast Simulated Re-annealing (VFSR) [5] which permits a very fast temperature annealing schedule, as our basic SA algorithm.

1.2 Local Optimization

A local optimizer or a hill climber is very efficient method for optimization in simple, unimodal spaces. But in most real-world problems, it easily gets trapped in non-optimal regions, mainly because of (1) local optima; (2) flat surfaces or (3) ridges [6]. The local optimization algorithm proposed in [7] attempts to tackle some of these problems while trying to maintain the efficiency by employing following ideas: (1) Adjust the size of the probing steps to suit the nature of the terrain, shrinking when probes do poorly and growing when probes do well. (2) Keep track of the directions of recent successes, so as to probe preferentially in the direction of most rapid descent. We choose this local optimizer for combining with ASA. The outline of this algorithm for finding a locally optimal point of function *f* starting from point $\vec{x}$ is given in Figure 1. $\vec{v}$ is the step vector which grows and shrinks and $\vec{u}$ is used to keep track of the direction of recent success. RANDOM-VECTOR($\vec{v}$) simply returns a random vector with the same size as $\vec{v}$. MAXITER is the number of iterations before shrinking of the vector is done, and THRESHOLD is the smallest step size allowed.

In *Proceedings of the 9th Florida AI Research Symposium (FLAIRS-'96),* Key West, FL, pp. 233-237, June 1996

```
LOCAL – OPTIMIZE(f, x) {
 INITIALIZE(v); u ← 0;
 while |v| ≥ THRESHOLD
   iter ← 0
   while f(x + v) > f(x) and iter < MAXITER
     v ← RANDOM – VECTOR(v)
     iter ← iter + 1
   if f(x + v) > f(x)
     v ← v / 2
   else if iter = 0
     x ← x + v; u ← u + v; v ← 2u;
   else if f(x + u + v) < f(x)
     x ← x + u + v;
     u ← u + v; v ← 2u;
   else
     x ← x + v; u ← v; v ← 2v;
 return x;
}
```

Figure 1. An outline of the local optimization algorithm

```
SALO(f) {
 INITIALIZE(x, t);
 x_bsf ← x;
 do
   do
     y ← PERTURB(x);
     z ← LOCAL - OPTIMIZE(f, y);
     Δf_xy ← f(z) - f(x);
     if (Δf_xy ≤ 0) or
        (RANDOM() < EXP(-Δf_xy / T(t))))
        x ← y;
        if ( f(x) < f(x_bsf)) x_bsf ← x;
   while(equilibrium has not been reached);
   INCREMENT(t);
 while(stop criterion has not been met);
 return x_bsf;
}
```

Figure 2. An outline of the SALO algorithm

2. COMBINING SIMULATED ANNEALING AND LOCAL OPTIMIZATION

The strengths and weaknesses of the two methods outlined above are complementary: SA avoids local optima by jumping away from them, but it sacrifices efficiency by doing so; hill climbers employ a "greedy" strategy and quickly reach to the nearest local optimum, but have no way of getting out of a local optimum if it is not indeed the global optimum. We attempt to combine the ability of SA to get out of local optima and the efficiency of the local optimizer in relatively "simple" regions of the space. SA helps in locating good regions of the search space, while the local optimizer is used to rapidly hit the optimum. We call this hybrid method SALO (Simulated Annealing with Local Optimization).

From every point in the space generated by the SA process, we start the local optimizer and allow it to converge to the nearest local optimum. The value of the function at the local optimum is used as the evaluation value for the original point, and an acceptance or rejection decision is made according to the metropolis criterion. The basic high-level outline of SALO for minimization of a function $f(\vec{x})$ is given in Figure 2. $\vec{x}_{bsf}$ is the best $\vec{x}$ encountered so far, PERTURB($\vec{x}$) returns a point in the neighborhood of $\vec{x}$, and LOCAL-OPTIMIZE(f, $\vec{x}$) returns a locally optimal point in the neighborhood of $\vec{x}$ for function f.

Statistical guarantee and SALO

The statistical promise of finding the globally optimal solution is considered an important feature of SA. This ensures a uniform sampling of the search space, which is reassuring when little is known about the nature of the space. Attempts to speed up SA, such as simulated quenching (SQ), usually trade this promise off with the gain in efficiency [2]. Below we argue that SALO maintains the statistical promise of SA.

Employing SALO for finding the optimal point of a function f can be viewed as using ordinary SA on a transformed function f'. The exact nature of f' depends on f and the characteristics of the local optimizer used. A transformation of a function f for a typical local optimizer is shown in Figure 3. Typical local optimizers have a "flattening" effect on the function being optimized. However, regardless of the behavior of the local optimizer (excluding some trivial cases, *e.g.*, a "local optimizer" which always returns a fixed point), f' will contain all the local optima of f, including the global optimum. In

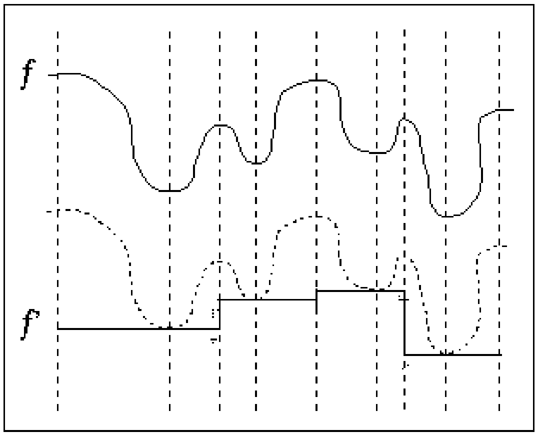

Figure 3. A sample function *f* and its transformation *f'* after applying a local optimizer

other words, using SALO on a function *f* is the same — including the cost and parameter cooling schedules and space sampling characteristics — as using SA on the transformed function *f'*. Thus SALO(*f*) ≡ SA(*f'*) and if *optimum(g)* is a function returning the globally optimal point of the function *g*, then *optimum(f) = optimum(f')*. Since SA(*f'*) is statistically guaranteed to find the global optimum of *f'*, which is the same as that of *f*, we are statistically guaranteed of finding the global optimum of *f*. Note that SALO does not perform "true" annealing of *f*, but it effectively performs true annealing of *f'*.

If the cost of performing the transformation *f*→*f'* turns out to be less than the efficiency gained by annealing on a (hopefully) simpler surface (*f'*), SALO will be more efficient than SA. Otherwise, using SA will be more beneficial than using SALO. In the following section, we try to empirically determine which one of the above two cases is likely to be encountered more often in real-world situations by comparing SALO with SA and a parallel genetic algorithm, on several "interesting" functions.

3. EXPERIMENTAL RESULTS

For comparing the performance of SALO with other methods, we use several well-known benchmark problems, listed below. These problems represent various characteristic terrain found in real-world problems, *e.g.*, unimodal/multi-modal, with/without plateaus and ridges, high/low dimensional.

f1: Sphere model [8]

$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2$$
$$-5.12 \le x_i \le 5.12; \min(f) = f(0,\ldots,0) = 0$$

This is a smooth, unimodal, symmetric function and does not have any problems of ridges, plateaus or foothills. The performance on this function is a measure of the general efficiency of the optimization algorithm.

f2: Rosenbrock's function [8]

$$f(\vec{x}) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)$$
$$-5.12 \le x_i \le 5.12; \min(f) = f(1,\ldots,1) = 0$$

This is a unimodal and bi-quadratic function with a very narrow ridge which runs around a parabola and has a very sharp tip. The progress of many algorithms is very slow because they are unable to discover a good search direction.

f3: Step function [8]

$$f(\vec{x}) = 6n + \sum_{i=1}^{n} \lfloor x_i \rfloor$$
$$-5.12 \le x_i \le 5.12;$$
$$\min(f) = f([-5.12,-5),\ldots,[-5.12,-5)) = 0$$

This function contains flat regions joined by steep slopes. It is a representative of plateau problems with linear and discontinuous properties. Flat regions do not give any information as to which direction is favorable.

f4: Plateau function [9]

$$f(\vec{x}) = \sum_{j=1}^{4} 2500 \cdot \max_i \lfloor 1000 \cdot |x_i| \rfloor$$
$$(j-1)h < i \le jh; h = n/4;$$
$$\min(f) = f(\max_{1 \le i \le n} |x_i| < 10^{-3}) = 0$$

This function has a large number of flat regions whose value gradually decreases towards the global minimum near the origin.

f5: Sines function

$$f(\vec{x}) = 1 + \sin^2(x_1) + \sin^2(x_2) + -0.1\exp(-x_1^2 - x_2^2)$$
$$-10 \le x_1, x_2 \le 10; \min(f) = f(0,0) = 0.9$$

This is a multi-modal function with a large number of local minima which have very similar values.

f6: Goldstein and Price

$$
\begin{aligned}
f(\vec{x}) = {} & [1+(x_1+x_2+1)^2(19-14x_1+3x_1^2 \\
& -14x_2+6x_1x_2+3x_2^2)]\cdot[30+(2x_1-3x_2)^2 \\
& (18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)] \\
& -2 \le x_1, x_2 \le 2; \min(f) = f(0,-1) = 3.0
\end{aligned}
$$

This is a multi-modal and nonlinear function. The problem optimization algorithms face with this function is the peak response of about five orders of magnitude greater than the minimum in the neighborhood of the minimum.

f7: Rastrigin's function

$$
f(\vec{x}) = 10n + \sum_{i=1}^{n} x_i^2 - A\cos(2\pi x_i)
$$
$$
-5.12 \le x_i \le 5.12; \min(f) = f(0,\ldots,0) = 0
$$

This is a scaleable, multi-modal function made from the *sphere model* by modulating it with $A\cos(2\pi x_i)$ [10]. Far away from the origin this functions looks like the sphere model, but with smaller x_i the effect of the modulation grows and dominates the shape. The multi-modality here presents substantial difficulty to many optimization algorithms.

Table I. The number of function evaluations required by various methods for finding the global optimum with the accuracy of 10^{-5}. n is the number of dimensions.

$f(\vec{x})$	n	PGA	SA	SALO
f1	2	2212	292	81
	15	17525	5925	575
f2	2	2550	3073	343
	4	197574	229390	35172
f3	5	2312	2968	2413
f4	2	?	488	142
	4	7935	1384	245
	8	17240	10114	2829
f5	2	1987	603	477
f6	2	4355	387	103
f7	2	3811	474	95
	4	?	1597	229
	8	?	7718	5199
f8	2	4762	2184	297
	10	?	28656	480

f8: Griewank 1

Table I shows the results in terms of the number of function evaluations required to find the global minimum with the precision of 10^{-5} for a parallel genetic algorithm (PGA), ASA and SALO. The number of function evaluations reported are the averages of 10 independent runs

$$
f(\vec{x}) = \frac{1}{d}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1
$$
$$
d = 2; -100 \le x_i < 100; \min(f) = f(0,\ldots,0) = 0
$$

for each method. Lack of consistent convergence is indicated by a "?". n indicates the number of dimensions of the function.

The multi-population genetic algorithm used was PGA v2.7 [11]. The following parameters were used in PGA: number of populations 5, number of individuals in each population 20, number of bit per variable 16, selection type rank selection, mutation rate 0.005, crossover rate 1.0, crossover type two-point crossover, and migration interval 10. The same set of parameters were used in optimizing all test functions.

For ASA and SALO, a couple of parameters, which control the cost and parameter temperature annealing schedules, need to be adjusted for some functions to achieve consistent convergence. These are determined by a few trials, without fine-tuning them or exploiting the knowledge of the function terrain.

It should be noted that these results are obtained for particular implementations of the algorithms. The performance of the PGA can probably be improved marginally by experimenting with various parameters such as population size, crossover type, crossover and mutation rates, etc.

4. CONCLUSIONS AND FUTURE DIRECTIONS

Results on a variety of functions with different characteristics suggest that SALO provides a powerful optimization method. While experimentation on many real-world problems is desirable, this method clearly compares favorably to ASA and PGA, two popular optimization methods on most test functions. Once in a region near the global optimum, SALO is able to reach it rapidly due to the presence of a good local optimizer which has a variable step size and can handle ridges effectively. Annealing enables it to get out of local optima and allows it to sample different regions of the search space. From another viewpoint, the cost of "flattening" the function terrain is outweighed by the efficiency gained by doing annealing on a simpler terrain. This is achieved by what is essentially a simple addition of a function call to the local optimizer in an existing SA algorithm.

As a potential improvement to SALO, the local optimizer could be run with a probability inversely proportional to the current temperature. This could save some effort spent in initial random regions of the space. The THRESHOLD parameter in the local optimizer can also be made adaptive, keeping it high initially so as to stop the local optimizer from spending effort in trying to fine-tune its solution, and lowering it gradually so that it can reach the global optimum with the required precision, when good regions of the space are found.

References

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, 220, pp. 671-680, 1983.

[2] L. Ingber, "Simulated Annealing: Practice versus Theory," *J. Mathl. Comput. Modelling,* 8, pp. 29-57, 1993.

[3] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distribution an the Bayesian Restoration in Images," *IEEE Trans. Patt. Anal. Mac. Int*., 6, pp. 721-741, 1984.

[4] L. Ingber, *Adaptive Simulated Annealing (ASA)* [ftp.caltech.edu:/pub/ingber/asa.Z], Software package documentation, 1995.

[5] L. Ingber, "Very Fast Simulated Re-Annealing," *J. Mathl. Comput. Modelling*, 12, pp. 967-973, 1989.

[6] P. Winston, *Artificial Intelligence*, Addison-Wesley Publishing Company, Reading, MA, third edition, 1992.

[7] D. Yuret, "From Genetic Algorithms to Efficient Optimization," *MS Thesis, Dept. of Electrical Engineering and Computer Science, MIT*, 1994.

[8] K. De Jong, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems," *PhD Thesis, University of Michigan, Diss. Abstr. Int. 36(10), 5140B,* University Microfilms No 76-9381, 1975.

[9] D. J. Ackley, "An Empirical Study of Bit Vector Function Optimization," in *Genetic Algorithms and Simulated Annealing*, edited by L. Davis, London, Pitman, pp. 194-200, 1987.

[10] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, "Applications of Evolutionary Algorithms," *Report of the Systems Analysis Research Group SYS-2/92, University of Dortmund, Department of Computer Science*, 1992.

[11] P. Ross, "About PGA v2.7," *Dept. of AI, University of Edinburgh*, 1994.