Directional Search Algorithm and its Utilization of Simulated Annealing and Expand-Retract

Beom Jin An
United States Military Academy
Department of Electrical Engineering and Computer Science
danielan1996@gmail.com

Alexander S. Mentis
United States Military Academy
Department of Electrical Engineering and Computer Science
alexander.mentis@westpoint.edu

***Abstract*—We present the Directional Search Algorithm (DSA), a novel metaheuristic optimization algorithm combining a highly-simplified version of Particle Swarm Optimization (PSO) with ideas adapted from simulated annealing, and a dynamic "Expand-Retract" method, a novel technique to balance exploration and exploitation. We compare the speed and accuracy of DSA to PSO in finding the global minimum of some common artificial landscapes for single-objective optimization and find that DSA outperforms PSO for both criteria on the selected functions.**

***Index Terms*—Directional Search Algorithm, Particle Swarm Optimization, scope, global minimum, Expand-Retract, exploration, and exploitation**

I. Introduction

Optimization problems based on local search techniques suffer from the risk of getting stuck in local optima. It is often infeasible to overcome this problem using brute force search because of the size of the search space. Metaheuristic algorithms attempt to strike a balance between exploration of the search space and feasible computability. Although metaheuristic algorithms do not always find a global optimum, they have the potential to find solutions that are close to the optimal value by using techniques that allow the local search to consider exploration that is suboptimal in the near term, thus escaping local optima.[1] This research was motivated by a desire to learn about common building blocks of metaheuristic optimization algorithms such as stochasticity, fitness proportionate selection, decreasing exploration/favoring exploitation over time, and maintaining multiple candidate solutions in the local space. In the process, we created, and present here, a metaheuristic algorithm we call Directional Search Algorithm (DSA), which blends and modifies aspects of simulated annealing and particle swarm optimization (PSO) in a novel way. Initial tests and comparisons of DSA with PSO on some common artificial landscapes for single-objective optimization show that DSA finds better solutions than PSO in less time. In the remainder of this paper, we will describe the underlying metaheuristics upon which DSA is founded, explain the DSA algorithm, and present the results of our tests comparing DSA with PSO.

II. Background

Many metaheuristic optimization algorithms are inspired by phenomena observed in nature.[2] The DSA combines the idea of an interrelated collection of candidate solutions (swarm particles) from PSO and probabilistic acceptance of suboptimal solutions based on the technique used in simulated annealing. To these, we add the use of a damped sine wave to control the transition from exploration to exploitation over time.

A. Particle Swarm Optimization (PSO)

PSO is a metaheuristic that utilizes swarm intelligence to find a solution to a system. Swarm intelligence capitalizes on the collective knowledge of multiple particles. The algorithm randomly initializes a desired number of particles, and it iteratively adjusts the positions of these particles using formulas based on the position of each velocity relative to the others in the swarm and its velocity. This formula takes three parameters into account: social constant, cognitive constant, and weight. The social constant determines the importance of the collective's best solution. The cognitive constant determines the importance of the particle's personal best solution. The weight determines how hard or easy it is to change the particle's momentum.[3] With these parameters, the particles are guided toward better solutions. Depending upon the selection of hyperparameter values, particles may converge gradually or quickly. One must select a value that results in an appropriate balance between achieving fast computation time and allowing sufficient exploration.[4]

B. Simulated Annealing

The inspiration for simulated annealing comes from annealing metallurgy, a technique that involves cooling the heated material in a controlled manner to help the molecules form stable crystals. Each material has an optimal cooling rate to create the strongest form of that material after cooling. The simulated annealing algorithm uses a similar method in which a better solution is less likely to be abandoned for a worse solution as a simulated temperature becomes lower and as the magnitude of the difference in quality increases. This means that the higher temperature allows for exploration in the initial

phase of the search iterations, but as the temperature cools, the system starts stabilizing at the current best solution.[5]

III. Directional Search Algorithm (DSA)

The Directional Search Algorithm is an algorithm for optimizing continuous functions of any dimension. As a metaheuristic, it does not guarantee the discovery of the optimal solution. This section describes the algorithm and its parameters.

A. Initialization

DSA maintains multiple candidate solutions in the search space. One particle (the *agent*) is placed randomly within the given boundary constraints. The other particles (*scoping particles*) are initialized randomly along axes passing through the agent's coordinates. The number of scoping particles depends upon the dimension of the search space, and the total $PopSize$ of the particles is 3^{dim}, where dim (the "dimension of the scope") is one less than the dimension of the search space. Each particle has an associated *fitness*, and the particle with the highest fitness is maintained as the best value.

B. Directional Movement

A critical component of DSA is the directional search method. In a continuous search space, the location of each solution in the search space is specified by a tuple of coordinates. Every iteration, DSA calculates coordinates for the *scoping particles,* relative to the *agent* particle, by calculating every combination of 1) adding a scoping factor, 2) subtracting a scoping factor, or 3) making no change to each coordinate of the *agent*. For example, a two-dimensional problem would have the following particles, where (x, y) is the *agent*'s coordinates and each $+/-$ represents either adding or subtracting a scoping factor from the corresponding *agent* coordinate:

$$\begin{bmatrix} x & y \\ x & +y \\ x & -y \\ +x & y \\ +x & +y \\ +x & -y \\ -x & y \\ -x & +y \\ -x & -y \end{bmatrix}$$

C. Expand-Retract

The concept of Expand-Retract comes from the idea that the scope of the search will generally decrease over time but osciillates near the current area as it decreases. The Expand-Retract formula is a decreasing sinusoidal wave function, as illustrated in figure 1, and it is described by the following equations:

$$v = x_0^2 + 100x_0 \cdot \cos(0.05x_0\pi)$$

$$k(x_i) = \left| \left[x^2 + 100x \cdot \cos(0.05x\pi) \right] / v \right|$$

$$x_0 = countdown = iterations * PopSize * dimension$$

$x_i = current\ count$ (decremented every particle update)

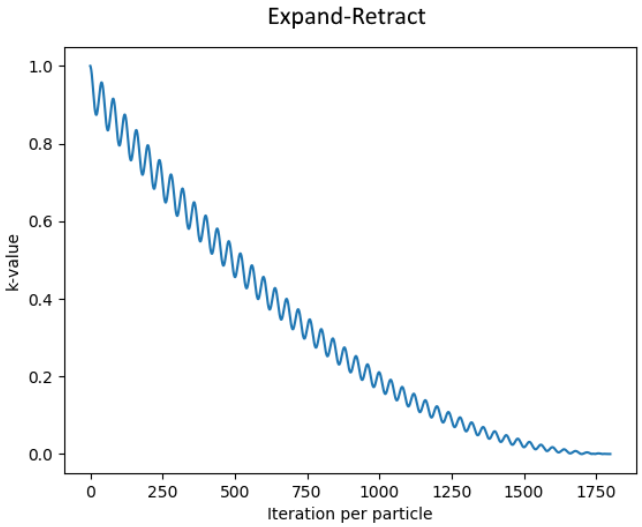


Fig. 1. Example of Expand-Retract behavior.

D. Scoping Factor

On each iteration, each particle receives its own, random scoping factor to encourage robust exploration. The scoping factor is calculated as follows:

1) Calculate v which is the Expand-Retract value (see section III-C)
2) Assign k equal to the absolute value of v after normalizing it to be within the search area boundaries
3) Let l be the current iteration
   a) $uppberbound$ and $lowerbound$ are hyperparameters that define the upper and lower limits of the search space
   b) $convergence$ is a hyperparameter that that defines the exponential rate by which the $ScopeFactor$ will converge
   c) Use the following formula to calculate $ScopeFactor$:

$$rand\_uniform(0, k) \cdot \left[\frac{(upperbound - lowerbound)}{(l+1)^{convergence}} \right]$$

Figures 2 and 3 illustrate the effect of the convergence rate.

E. Solution Acceptance

As the *agent* and *scoping particles* move through the search space in search of the optimal solution, DSA compares the *agent*'s solution with those found by the *scoping particles*. If the best solution found among the *scoping particles* is better than the *agent*'s current solution, the *agent* is replaced

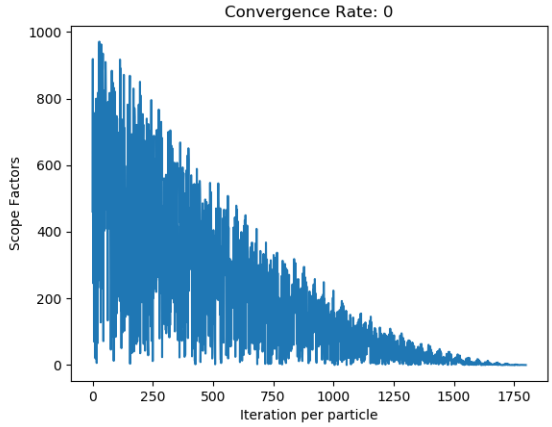


Fig. 2. Example of behavior with convergence rate = 0

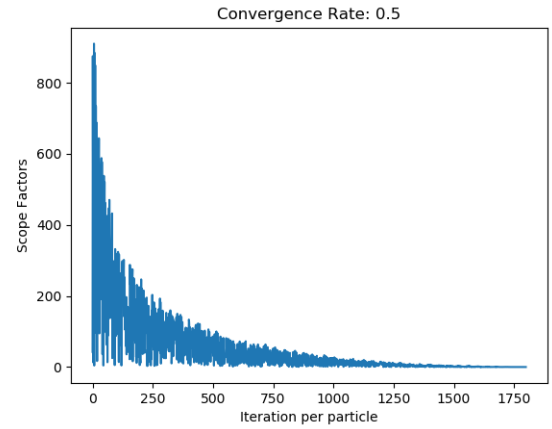


Fig. 3. Example of behavior with convergence rate = 0.5

by the best *scoping particle*. Otherwise, the best *scoping particle* replaces the *agent* based upon the same criteria used in simulated annealing: the worse solution replaces the *agent*'s with probability $p = \exp\left(\frac{-(spFit - agentFit)}{T}\right)$, where *spFit* is the *scoping particle*'s fitness and T is an output from a monotonically decreasing function representing the "temperature" of the system at the current iteration. The initial temperature and cooling rate are hyperparameters that will likely need to be set for each specific problem; however, for our experiments, we used $T_0 = 100$, and $T_{t+1} = 0.95 \cdot T_t$.

F. Pseudocode

Unifying the descriptions above, we present pseudocode for the complete algorithm here.

```
initialize particles matrix to zeroes
// size: PopSize x dimension
// first row will be agent's coords

randomly initialize agent particle's location
while iterations remain:
  for each dimension:
    for each particle (excluding agent):
      k = expand_contract_value
      scope_factor = random(0, k)
      // normalize scope_factor to be
      //  between search space
      //  lower and upper bounds
      // in current dimension:
      particle position =
        current position +/- scope_factor
  particle_fitnesses = all updated fitnesses
  if min(particle_fitnesses) < best_fitness:

    best_fitness = min(particle_fitness)
  else:
    use simulated annealing criteria
output best_fitness
```

IV. Experiments

Due to the similarities between DSA and PSO in maintaining populations of solutions that sample from the search space as a spatially-cohesive group, we conducted benchmarking experiments comparing the performance of DSA to PSO; however, metaheuristic optimization algorithms are characterized by varying numbers of hyperparameters controlling their behaviors, and there is not a one-to-one correspondence between the hyperparameter choices for DSA and PSO, making benchmarking under identical configurations impossible. Therefore, we instead tried a variety of hyperparameter configurations and tried to obtain the best performance from each of the algorithms for comparison. We implemented the DSA algorithm in Python 3.7, and used a Python implementation of PSO from [6]. All experiments were run on a MacBook Pro with a 2.9 GHz Intel Core i9 processor.

DSA has five independent variables (function to optimize, parameter bounds, iterations, trials, and convergence rate). PSO has eight independent variables (function to optimize, parameter bounds, iterations, trials, number of particles, momentum (w), cognitive constant (c1), and social constant (c2)). We ran DSA and PSO with a variety of combinations of hyperparameter configurations on the following four artificial landscape test functions:

1) Sphere function:

$$f(x) = \sum_{i=1}^{n} x_i^2$$

2) Beale's function:

$$f(x,y) = \left(1.5 - x + xy^2\right) + \left(2.25 - x + xy^2\right)^2 + \left(2.625 - x + xy^3\right)^2$$

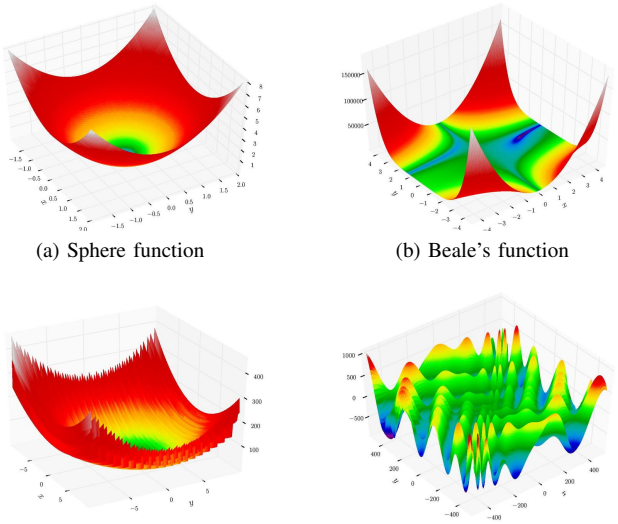
3) Lévi function no. 13:

$$f(x,y) = \sin^2 3\pi x + (x-1)^2\left(1 + \sin^2 3\pi y\right) + (y-1)^2\left(1 + \sin^2 2\pi y\right)$$

4) Eggholder function:

$$f(x,y) = -(y+47)\ \sin\sqrt{\left|\frac{x}{2}+(y+47)\right|} - x\ \sin\sqrt{|x-(y+47)|}$$

These functions are illustrated in figure 4. For each configuration, we recorded the accuracy and runtime of the algorithms. The accuracy is the number of times the algorithm converged to the global optimum divided by the number of trials. The calculation time is the time between the start of the first trial and the end of the last trial as measured by the benchmark program using the system clock.



(a) Sphere function (b) Beale's function

(c) Lévi function no. 13 (d) Eggholder function

Fig. 4. Artificial landscape test functions[7]

V. Results

Tables I and II and figures 6and 5 show the results of our experiments comparing DSA and PSO. Both algorithms yielded high accuracy rates on functions 1 and 3; however, DSA was faster than PSO on both and slightly outperformed PSO on function 3, reaching 100% accuracy on both trials. Functions 2 and 4 were more challenging for these algorithms, but with appropriate hyperparameter selection, DSA was able to achieve 96% accuracy on both, whereas the best accuracies we were able to obtain with PSO were 54% and 68%, respectively. The trends show that it is fairly easy to find hyperparameter configurations that work well for DSA than with PSO and that DSA consistently runs faster than PSO, even when many iterations are required in order to improve accuracy.

VI. Analysis

It is not surprising to find that DSA is faster than PSO. The runtime complexity for DSA is $O(dim \times iterations \times PopSize^2)$, so the runtime only grows linearly with increased iterations. While the runtime grows quadratically with the size of the population, the population is determined by the number of dimensions in the problem and is not a value that would be adjusted to alter performance. Contrast this with PSO, where the number of particles is a hyperparameter and each particle must calculate updates based on the particles defined as its neighbors. Design choices include not only the number of particles to use but also the topography of their neighborhoods. As the number of particles and inter-particle connections grow, the computation time required grows quickly. For these tests, the resulting increased computational requirement did not yield a significant improvement on accuracy.

DSA reduces the particle population and the computations required, but it retains some of the behavior arising from the social constant and momentum of PSO with the Expand-Retract functionality. Combined with simulated annealing's probabilistic acceptance of suboptimal solutions, DSA is able to avoid getting trapped in local minima.

The simulated annealing cooling rate and the damped Expand-Contract behavior, controlled by the convergence rate, also helps avoid converging too quickly to local minima. We can see the effect of converging too quickly in the algorithms' performances on the Eggholder function. If the scope converges too quickly, the accuracy drops from 87% to 6%.

TABLE I
DSA Experiment Results

Cfg	Fn	Bnds	Iter	Trls	C-Rate	Acc	Time
1	1	±10	500	100	0	100%	3.18s
2	1	±10	500	100	1	100%	3.43s
3	2	±10	500	100	0	92%	3.51s
4	2	±10	500	100	1	67%	3.62s
5	2	±10	500	100	0.1	83%	3.52s
6	2	±5	500	100	0	92%	3.45s
7	2	±10	2000	100	0	96%	13.95s
8	3	±10	500	100	0	100%	3.66s
9	3	±10	500	100	1	100%	3.84s
10	4	±512	500	100	0	87%	5.28s
11	4	±512	500	100	1	6%	5.12s
12	4	±512	1000	100	0	92%	10.17s
13	4	±512	2000	100	0	94%	19.89s
14	4	±512	4000	100	0	96%	40.80s

TABLE II
PSO Experiment Results

Cfg	Fn	Bnds	Iter	Trls	Part	w, c1, c2	Acc	Time
1	1	±10	500	10	50	0.5, 1, 2	100%	8.61s
2	1	±10	500	10	100	0.5, 1, 2	100%	17.65s
3	2	±10	500	10	50	0.5, 1, 2	18%	8.79s
4	2	±10	500	10	100	0.5, 1, 2	21%	17.29s
5	2	±10	500	10	200	0.5, 1, 2	29%	35.04s
6	2	±5	500	10	100	0.1, 1, 2	48%	17.18s
7	2	±10	2000	10	100	0.1, 2, 2	54%	17.03s
8	3	±10	500	10	50	0.1, 2, 2	96%	9.56s
9	3	±10	500	10	100	0.1, 2, 2	100%	18.91s
10	4	±512	500	10	50	0.1, 2, 2	16%	17.45s
11	4	±512	500	10	100	0.1, 2, 2	35%	35.02s
12	4	±512	500	10	200	0.1, 2, 2	59%	69.85s
13	4	±512	500	10	400	0.1, 2, 2	68%	140.89s
14	4	±512	500	10	200	0.5, 2, 2	55%	68.87s

Beale's function is difficult because a large part of the function is flat and local search is unable to differentiate between neighboring values. Here, exploration is the key. A higher iteration number combined with a lower conversion rate improves DSA's accuracy on this function.

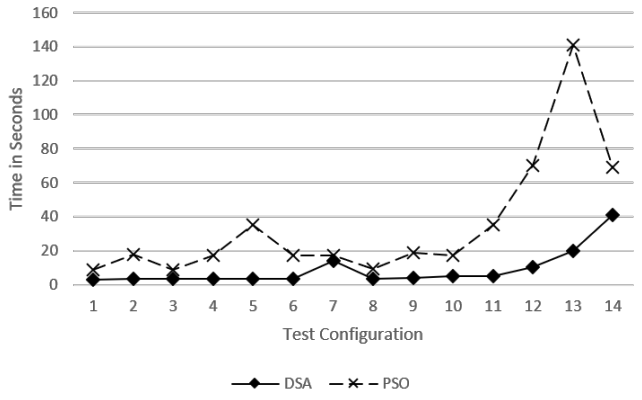


Fig. 5. Comparison of DSA and PSO runtime for different test functions and configurations

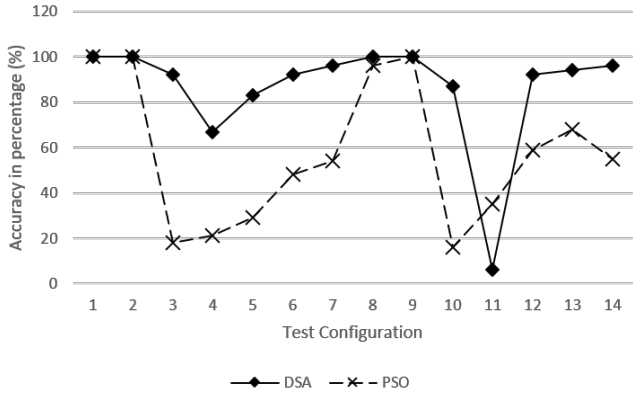


Fig. 6. Comparison of DSA and PSO accuracy for different test functions and configurations

VII. Future Work

As noted previously, it is difficult to identify equivalent configurations between DSA and PSO for benchmarking. The results here have shown enough promise to continue testing with more deliberate experiment design. Taking care to connect particles in PSO with a neighborhood similar to that used by the *agent*-centric scoping particles would help isolate the difference in the quality of the social/cognitive trade-off of PSO and the Expand-Contract functionality of DSA. It could also be more useful to compare the accuracy of PSO versus DSA for fixed runtimes instead of varying the iterations for the two algorithms. Also, while some of DSA's functionality was inspired by PSO, DSA also shares many attributes with simulated annealing. It would be illuminating to benchmark the performance of DSA against simulated annealing as well. Finally, the algorithm must be benchmarked against more test and real-world functions.

VIII. Conclusion

These experiments have been a successful proof of concept for the idea of a small-particle-population using simulated annealing's probabilistic acceptance of suboptimal solutions during searching and facilitating exploration with an Expand-Contract function. Directional Search Algorithm is an algorithm that combined these basic ideas to form an efficient system that can find the global minimum of a function. Inspired by Particle Swarm Optimization and simulated annealing. DSA also incorporates a novel Expand-Contract method that allows for a good balance between exploitation and exploration.

References

[1] J. Dreo, P. Siarry, A. Petrowski, and E. Taillard, *Metaheuristics for hard optimization : methods and case studies*. Springer-Verlag, 2006.

[2] X.-S. Yang, *Nature-Inspired Metaheuristic Algorithms*. 07 2010.

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," 1995.

[4] M. Juneja and S. K. Nagar, "Particle swarm optimization algorithm and its parameters: A review," in *2016 International Conference on Control, Computing, Communication and Materials (ICCCCM)*, pp. 1–5, Oct 2016.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.

[6] N. Rooy, "Simple particle swarm optimization with python." https://nathanrooy.github.io/posts/2016-08-17/simple-particle-swarm-optimization-with-python/, August 2016. Accessed July 13, 2019.

[7] Wikipedia contributors, "Test functions for optimization — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Test_functions_for_optimization&oldid=896257708, 2019. [Online; accessed 13-July-2019].