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

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

***Abstract*—By building an optimization algorithm, one can understand the details of which factors contribute to creating a sufficient optimization algorithm. In this research, a new algorithm was crafted to experiment on how a combination of existing optimization techniques may improve the performance of the algorithm. The new algorithm, called the Directional Search Algorithm (DSA), utilizes a conglomeration of different techniques typically used in optimization algorithms. These techniques are carefully integrated into one algorithm that geared to finding the global minimum of a single-objective three-dimensional function. DSA is an algorithm derived from an idea that a single search particle extends out in different directions to "scope" out its surroundings. This particle scopes out in multiple directions and finds the solutions at those extended positions. With the incorporation of simulated annealing and a method called "Contraction-Retraction," the particle will be able to explore the search space expansively. ~~Also,~~ the algorithm also offers a unique technique that capitalizes a sufficient balance between exploration and exploitation. The first part of the research will explain and justify the utilization of specific techniques and parameters. The second part of the research will explore the effectiveness of the algorithm. The factors that determine the effectiveness of the algorithm are the following: convergence rate, precision, robustness, and performance. The research examined these four factors on DSA and analyzed the overall performance of the algorithm compared to that of a similar algorithm, the Particle Swarm Optimization (PSO).**

I. Introduction

Optimization problems require an immense amount of computational time as the problem scales increase. It is often unrealistic to compute the solution of an optimization problem using brute force because the problem has a nondeterministic polynomial computational time. Thus, metaheuristic algorithms are built to overcome this computational time problem. Although metaheuristic algorithms do not always output the global optimum, they provide efficiency in approximating optimal solutions. They make assumptions about the problem space to reduce search space and computation.

II. Background

It is often difficult to create an optimization algorithm because of its variety and complex relationships between parameters. Thus, in many cases, optimization algorithms are often inspired by phenomena in nature. The inspired algorithms are nature-proven methods of optimization, and they derive the properties from these natural behaviors that can assist in creating an optimization algorithm that works.

A. Simluated Annealing

One of the most prominent nature-derived methods is "Simulated Annealing." Its inspiration 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 its optimum cooling rate to create the strongest form of that material after cooling. Similarly, this method can be used to find a global minimum of a system or a problem. The mathematical function used in simulated annealing determines the probability in which the search will accept a worse solution. As the temperature within this function decreases, this probability also decreases. This means that the higher temperature allows for exploration in the initial phase of the search iterations. Once the temperature starts to cool down, the search begins its exploitation and hones down to one area in the search space. Simulated annealing is primarily a tool to balance the exploration and the exploitation within the algorithm. It also offers the probability characteristic that will allow for a more randomized search.

B. Beam Search

In contrast to the simulated annealing method, there is a straight forward concept to pick the solution each step in the optimization algorithm. As simulated annealing offers a probability whether to accept the solution or not, this classic method called "Beam search" is a greedy search algorithm that always accepts the best neighboring node. It explores all possible adjacent nodes and expands to the most promising node. Thus, the probability of accepting a worse solution is always zero. The benefit of the Beam search is that it will approach the solution much quicker than the methods that use probabilities. Also, another advantage is that it does not have any memory requirement. However, the disadvantage of Beam search is that it may converge quicker into a local optimum will not be able to get out. The Beam search does not explore enough of the search space, and its convergence rate is too high. Thus, once the search enters a local minimum, it has no chance of getting out.

C. Particle Swarm Optimization (PSO)

PSO is a metaheuristic that utilizes swarm intelligence to find the optimum solution of a system. Swarm intelligence capitalizes the collective knowledge of multiple particles. The

algorithm initiates with a certain number of particles, and it iteratively improves the positions of these particles by using formulas to determine the position and the velocity of each particle. The particles are guided towards the best solution. The benefit of PSO is that it is more thorough during the exploration process. The particles move across the search space, and they do not "teleport" across the search space. When particles teleport randomly throughout the search space, it loses its exploitative behavior will not be able to settle on a solution. PSO particles converge to the best solution gradually. However, this property can also be a disadvantage as the convergence rate might be too fast before conducting sufficient exploration. PSO does not have a feature that expands its search since it is designed only to converge. Thus, once enough particles swarm into a local minimum, they have no way of exiting that space.

III. Direction Search Algorithm (DSA)

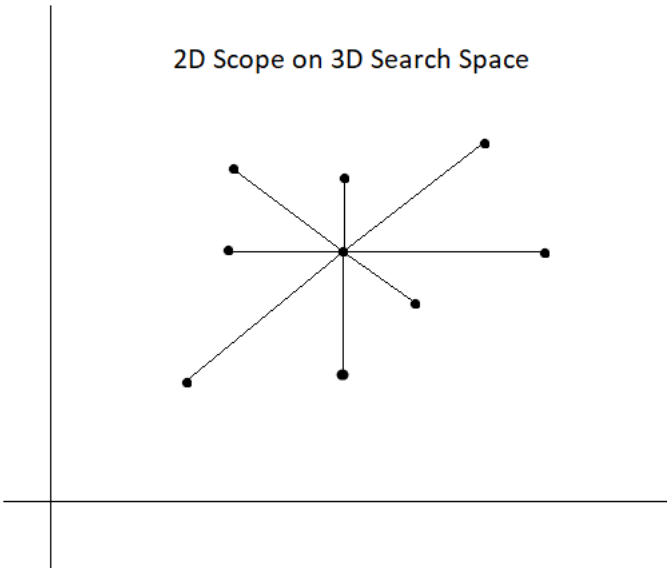
DSA is an optimization algorithm that takes a continuous function of any dimensions as input and outputs the global minimum of that function.

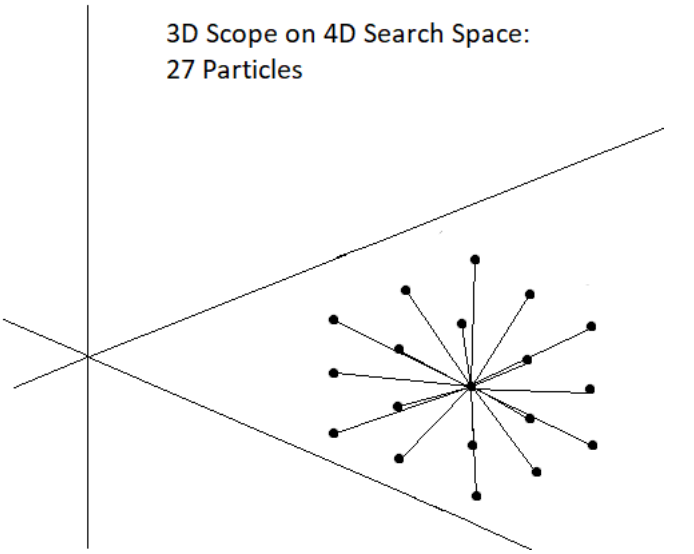
A. Initialization

DSA utilizes multiple agents or particles to share the search space. The algorithm initializes with one particle being placed randomly within the given boundary constraints. The other particles are determined by using the coordinate of the initial particle. DSA calls the initialized particle "the agent," and the other expanded particles "scoping particles." The number of the scoping particles varies depedning on the dimension of the search space. The total $PopSize$ of the particles will be: 3^{dim}. The dim is the "dimension of the scope" which is one less than the dimension of the search space which will be explained in the next subsection. The algorithm makes a matrix called $particles$ that contains the coordinates of the particles which is all initialized with zeroes. This matrix makes it easy for the algorithm to access and update the positions of the particles. Then, an array called fit with the size of the $PopSize$ is created to store the solutions or $fitness$ of the particles. One more additional array called $best$ is created to store the position of the best particle, in this case, the particle with the lowest calcuated value.

B. Directional Movement

As the name describes it, a critical component of DSA is the directional search method. In most cases, the search space is a continuous function. This means that the movement through the search space on a Cartesian coordinate system requires the algorithm to specify the grid coordinates. In contrast, a graph with nodes has a set of adjacent nodes that restricts the movement through the graph. Thus, even within a continuous function, a way of establishing a set of adjacent nodes is required to move through the search space. DSA uses a simple way of choosing these adjacent nodes or in this case, adjacent coordinates. From the starting point of the search space, which is initialized randomly, the adjacent coordinates are determined by adding or subtracting certain "values" from the initial coordinate. (These "values" will be discussed in the next subsection)





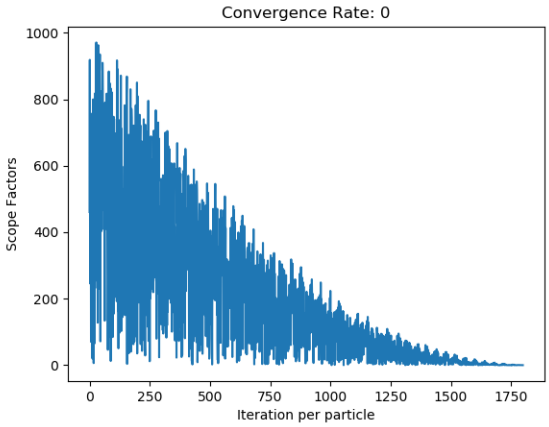
These adjacent coordinates are called "scoping particles" and the initial or the center coordinate is called the "agent". The scoping particles are on a dimension that is one less than the search space. For example, when searching through a three-dimensional function, you will need two-dimensional scoping particles. One way to look at this is to observe the input variables and the output variable. The particles in the search space are calculating solutions on their respective coordinates. On a three dimensional function, there are two input variables and one output variable. There will always be just one output variable. The two input variables are the dimensions you need to output a solution. Therefore, the particles will have a dimension equal to the number of input variables.

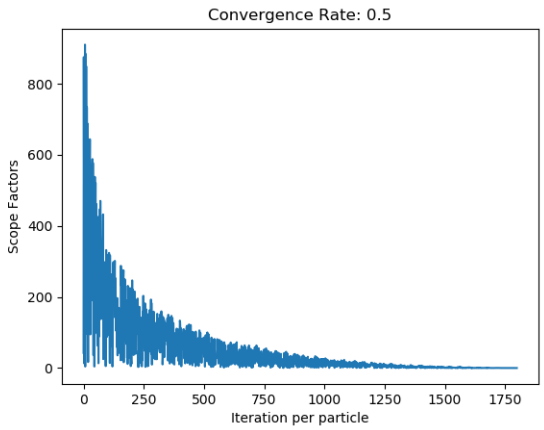
C. Scoping Factor

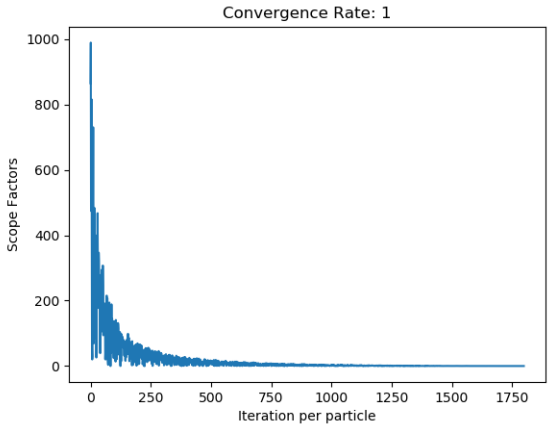
As mentioned above, certain values are added or subtracted from the coordinate values of the agent which is the center coordinate. This allows the scoping particles to expand out and explore the search space. The algorithm randomizes these values to ensure a robust explorative feature. The randomization of the scoping factor is as follows:

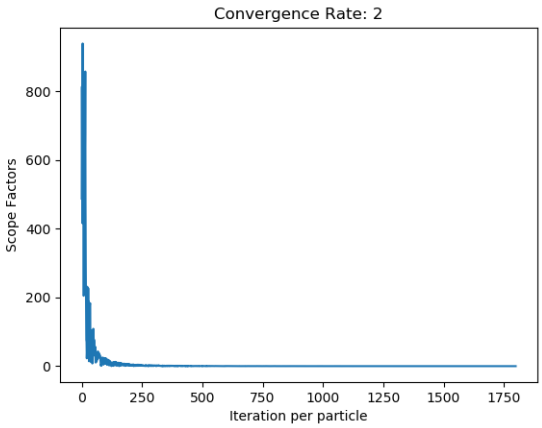
1) Calculate v which is the Contract-Retract value (covered in 3.4)
2) Normalize v and find the absolute value of v which becomes k
3) l is the current iterations
4) $uppberbound$ and $lowerbound$ are inputted values that define the upper and lower limits of the search
5) $convergence$ is an inputted value that that defines the exponential rate in which the $ScopeFactor$ will converge.
6) Use the following formula to calculate $ScopeFactor$: $random.uniform(0, k) * [(upperbound - lowerbound)/(l + 1)^{convergence}]$

A new $ScopeFactor$ will be calcuated for every particle on every iterations. This ensures the randomness of the scope and allows the particles to explore the search more robustly.





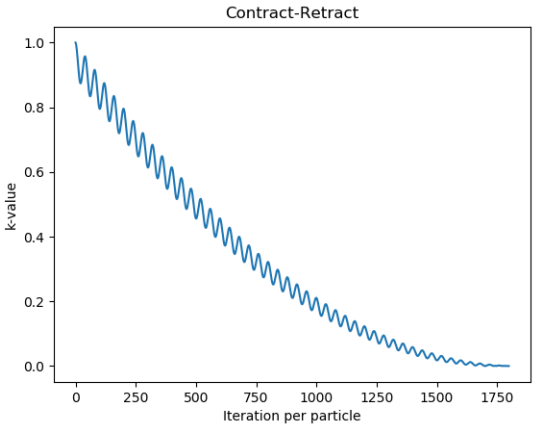




D. Contract-Retract

The concept of Contract-Retract comes from the idea that the scope of the search will generally decrease over time but can also increase for a short period of time to explore outside the scope. The Contract-Retract formula is a decreasing sinusoidal wave function. The equation is:

$v = x_0^2 + 100x_0 * cos(0.05x_0\pi)$
$k(x_i) = |[x^2 + 100x * cos(0.05x\pi)]/v|$
$x_0 = countdown = iterations * PopSize * dimension$
$x_i = current\ count$ (decremented every particle update)



E. Particle Update

The particles in the matrix $PopSize$ are update every iteration individually. The update is simple as add or subtract the $ScopeFactor$ from the agent particle. It will update the matrix by columns.

Two-dimensional scope will have the following PopSize matrix:

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

Three-dimensional scope will have the following PopSize matrix:

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

The values x, y, z are the coordinates of the agent particle which is always at the first row of the matrix. The "+" adds the ScopeFactor, and the "-" subtracts the ScopeFactor from the agent particle's coordinates. There is a pattern that associates with how the ScopeFactor is added or subtracted. Using the patterns, the matrix is efficiently updated depending on the dimensions of the scope. The counts of repeated values have a pattern. The pattern is: $count = \frac{3^{dimension-1}}{3^{column}}$. Count is the number that the values will be repeated before switching to the next value. For example, the first column of the three-dimensional matrix will be $count = \frac{3^2}{3^0} = 9$. The first column will have each of the three possible values, x, +x, -x, repeated 9 times, thus 27 total rows. The second column will have 3 repeated values.

F. The Best Particle

The scope will move through the search space to find the optimum solution. Unlike the Beam Search, DSA is not a greedy search. DSA utilizes simulated annealing to accept worse solutions. The selection process for the best particle is as follows:

Initialization: $T_0 = 100$, $cooldown = 0.95$

1) Update the particles
2) Find the best solution among the scope particles (particles below the first row of the matrix)
3) Compare the first row particle (agent particle) with the best scope particle
4) If the best scope particle is better than the agent particle, replace that scope particle as the new agent particle. Update the first row to reflect the new coordinates. Move to step 6.
5) If the best scope particle is not better than the agent particle, only accept with a probability $p = e^{\frac{-(bestFit - oldbestFit)}{T}}$. If accepted, replace the current agent particle with the new particle and update.
6) Once the agent particle is updated, repeat the process until the end of iterations. The temperature T will cool down with a rate of 0.95.

G. Usage

The DSA is a function that takes the following inputs: $DSA(lowerbound, upperbound, ScopeDimension, iterations, functio$

The lowerbound and upperbound create a square boundary on a two-dimensional search space and a cube boundary on a three-dimensional search space. The ScopeDimension is always one less than the function dimensions (search space).

IV. Results

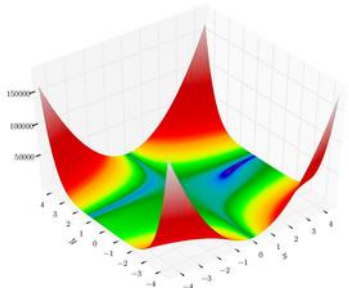
A. Test Functions

In order to test the accuracy and precision of DSA. These four functions are used to test the algorithm.

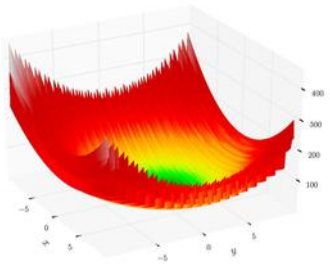
Function 1:

$y = x^2 + z^2$

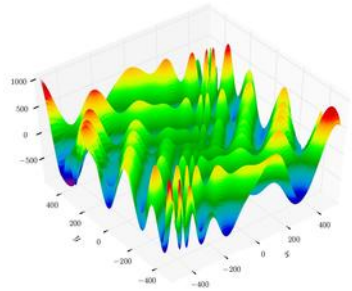
Function 2, Beale Function:



Function 3, Levi Function:



Function 4, Eggholder Function:



B. Experiment

In order to figure out the effectiveness of the new algorithm, an experiment must be done. The DSA takes in multiple inputs that can affect the accuracy of the results. The test functions have their own unique features that make difficult for any static algorithms. Thus, the parameters on DSA may need to be adjusted to fit the test functions. At the same time, numerous trials with man y different functions can determine the optimal parameter settings for DSA. For the purpose of this experiment, four different test functions were used. The four functions are on 4.1.

C. Method

The DSA is written in Python 3.7. The computational time for DSA is $O(dim * iterations * PopSize^2)$. The python file DSA.py was experiment on a MacBook Pro 2.9 GHz Intel Core i9. The method for this experiment is to test all four functions and figure out which parameters change the accuracy of that function. There are five independent variables: function, bounds, iterations, trials, convergence rate. There are two dependent variables: accuracy and calculation. The function variable is slecting the one out of the four functions. The bound parameter sets the area of search within the search space. The iteration parameter determines how many times the scope particles will expand and move per DSA function call. The trial parameter determines how many times the experiment will call the DSA function. The convergence rate determines how fast the search scope will converge to an area (refer to 3.3). For the results of the experiment, the accuracy and the time are the outputs. The accuracy is determined by how many time the algorithm converged to the global optimum out of the number of trials. The calculation time is determined by calculating the time between the start of the first trial and the end of the last trial.

DSA Experiement Results

Function	bounds	iterations	trials	c-rate	accuracy	time
1	-10,10	500	100	0	100%	3.18s
1	-10,10	500	100	1	100%	3.43s
2	-10,10	500	100	0	92%	3.51s
2	-10,10	500	100	1	67%	3.62s
2	-10,10	500	100	0.5	74%	3.59s
2	-10,10	500	100	0.1	83%	3.52s
2	-5,5	500	100	0	92%	3.45s
2	-10,10	2000	100	0	96%	13.95s
3	-10,10	500	100	0	100%	3.66s
3	-10,10	500	100	1	100%	3.84s
4	-512,512	500	100	0	87%	5.28s
4	-512,512	500	100	1	6%	5.12s
4	-512,512	1000	100	0	92%	10.17s
4	-512,512	2000	100	0	94%	19.89s
4	-512,512	4000	100	0	96%	40.80s
4	-512,512	10000	100	0	93%	102.66s

TABLE I
RUNNING DSA ON THE TEST FUNCTIONS WITH DIFFERENT PARAMETERS

D. Comparison with PSO

Particle Swarm Optimization uses particles to explore the search space. The information of the particles are centralized, and the particles move in accordance with the centralized information. These two concepts are analogous to Directional Search Algorithm. Thus, a comparison can be make between these two algorithms to determine the benefits and disadvantages of the both. The experiment for the PSO is the same as the experiment for the DSA. The parameters will be tweaked in order to figure out how the parameters affect the accuracy.

PSO Experiement Results

func	bounds	iter	trials	#particles	(weight,c1,c2)	accuracy	time
1	-10,10	500	100	50	(0.5,1,2)	100%	8.61s
1	-10,10	500	100	100	(0.5,1,2)	100%	17.65s
1	-10,10	1000	100	50	(0.5,1,2)	100%	17.61s
2	-10,10	500	100	50	(0.5,1,2)	18%	8.79s
2	-10,10	500	100	100	(0.5,1,2)	21%	17.29s
2	-10,10	500	100	200	(0.5,1,2)	29%	35.04s
2	-5,5	500	100	100	(0.1,1,2)	48%	17.18s
2	-10,10	2000	100	100	(0.1,2,2)	54%	17.03s
3	-10,10	500	100	50	(0.1,2,2)	96%	9.56s
3	-10,10	500	100	100	(0.1,2,2)	100%	18.91s
4	-512,512	500	100	50	(0.1,2,2)	16%	17.45s
4	-512,512	500	100	100	(0.1,2,2)	35%	35.02s
4	-512,512	500	100	200	(0.1,2,2)	59%	69.85s
4	-512,512	500	100	400	(0.1,2,2)	68%	140.89s
4	-512,512	1000	100	100	(0.1,2,2)	30%	69.09s
4	-512,512	500	100	200	(0.5,2,2)	55%	68.87s

TABLE II
RUNNING PSO ON THE TEST FUNCTIONS WITH DIFFERENT PARAMETERS

E. Analysis

1) DSA Analysis: DSA managed to output a high accuracy for the difficult functions. The most difficult function was the Eggholder Function, which had a given boundary (-512,512). It has many local minima that are not easy to distinguish. The simulated annealing and the Contract-Retract were able

to mitigate the search from converging too fast on to a local minimum. They also help the scope to escape local minima by reaching outside the local area. If the convergence rate is increased on the Eggholder function, the scope converges too quickly and the accuracy drops from 87% to 6%. Also, the bounds are also important. If the search area is too large, it may take more iterations to find the global optimum. Another difficult function was the Beale Function. It is difficult because a large part of the function is flat. Thus, if it converges too slowly when the bounds are too large, it might take a long time to reach the bottom of the function, and the search scope will linger around a local optimum once it reaches the bottom. A higer iteration number improves the accuracy on function simple because it has more opportunities to find the global optimum. However, increasing the iteration also increases the calculation time. The iteration linearly increases the computational time. The trial was set to 100 to easily calculate the percentage of accuracy. Overall, DSA proved to be accurate most of the time (greater than 50% accuracy). It is also time efficient because of its utilization of limited number of particles and the use of matrices which allows updating of particles at constant time.

2) PSO Analysis: PSO is an algorithm that relies on the particles to find their way to the global minimum. Thus, more particles present in the search space, more accurate the solution will be. Through the experiment, PSO proved to be sufficient at finding the global minimum of a bowl-shaped function (100% accuracy in function 1 and 3). However, PSO had a difficult time dealing with functions with irregular placement of local minima. On the functions that were not bowl-shaped, PSO had a low accuracy on finding the global minimum. Increasing the number of particles slightly increased the accuracy, but the calculation time was also increased proportionally. The PSO could not surpass DSA on calculation time. The biggest factor in the time discrepancy is the use of particles. PSO relies heavily on the characteristics of the particle: velocity and position. It constantly updates the velocity and the position of numerous particles each iteration. On the other hand, DSA only updates the position of the particle for limited amount of particles. In addition, the PSO code used in this experiment utilizes a Particle class that had attributes and functions. Creating these Particle objects may take much longer time than simply referencing and updating arrays. Therefore, DSA proves to be the structurally faster than PSO.

V. Conclusions

Directional Search Algorithm is an algorithm that combined the basic ideas to form an efficient system that can calculate the global minimum of a function.

VI. Appendix

A. Code