11-21-2013

Starfish Search

Ethan Criss

Starfish Search

Ethan Criss

November 21, 2013

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Science

Supervised by

Dr. Roger Gaborski
Department of Computer Science
B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, New York
November 2013

Approved By:

Dr. Roger S. Gaborski
Professor, Department of Computer Science
Chair

Dr. Peter Anderson
Professor, Department of Computer Science
Reader

Dr. Joe Geigel
Associate Professor, Department of Computer Science
Observer

Abstract

Starfish Search is a swarm optimization algorithm that operates in the same vein as Particle Swarm Optimization and the Firefly Algorithm. This search algorithm attempts to find global optimal solutions to optimization problems by dispersing agents into the search space. Each agent consists of many nodes that represent candidate solutions to the problem being solved. Agent's nodes are formatted in a parent-child hierarchy, similar to tree structures, which facilitates information passing to a root node. With this structure, it becomes possible to determine the likely direction in which an optimal lies. By using a form of linear regression, the fitness values and positions of each node in an agent are used to evaluate a vector, known as the Local Gradient. This vector points along the slope of the search space, and its magnitude represents the steepness of this slope. In this way, an agent has an understanding of the local area and can make intelligent decisions about which direction to search for additional candidate solutions. With this additional information, agents also have the ability to execute behaviors based on the type of topology encountered. These behaviors can be specifically tailored to individual problems and situations to help agents correctly solve the problem.

Starfish Search has been applied to problems such as, search space optimization, k nearest neighbors classification, and k means clustering. By tailoring fitness functions and behavior execution, evidence has been gathered to support the algorithms use over traditional techniques. This paper dives into the details of the algorithm's implementation, calculations, and behaviors as well as explain the tests and evidence gathered to support the use of Starfish Search.

Contents

1 Introduction

The quest for the best solution to a problem has been a long sought after achievement. The process for finding better and better answers to age old and brand new problems is constantly evolving and changing. Fields such as mathematics, engineering, business and computer science are interested in completing tasks in the most optimal way possible. This saves time, resources, upkeep and cost. New technologies, approaches and ideas are being implemented to discover the most optimal solution to these problems. Algorithm development for use on computers has been a driving force behind these solution discovery methods. With a computer's ability to process vast amount of information in varying problem types and fields, they have become the ideal solution finding mechanism. Many algorithms have been designed for computer use in order to reliably and accurately find optimal solutions in a wide range of problem spaces.

Problems faced today by modern engineers and business men are complicated problems that depend on a multitude of variables. This creates nonlinear problem spaces that ungulate and twist to form almost surreal shapes and forms. Attempting to solve for optimality in these conditions by hand become a near impossible task, filled with frustration and time consuming calculations. Computers offer a significant advantage in these cases by being able to quickly check through vast amounts of information and data, while also being able to efficiently solve complicated equations. In cases where it would take a long amount of time or many complicated calculations to solve the problem, computers are able to quickly and accurately find solutions. To find optimal solutions to such nonlinear problems requires efficient optimization algorithms.[4]

2 Background

Computer algorithms for solution discovery can be classified into two categories, deterministic and stochastic. Deterministic algorithms, like hill climbing or beam search, produce the same answer each time the algorithm is run if the starting positions are the same. This makes them efficient at consistently finding the exact same solution, but lack the ability to overcome local optima. Deterministic algorithms can offer find a solution that is not the global optima. The algorithm then becomes stuck on this solution with no way to discover a better solution. [4] Some processes, such as random restart have attempted to overcome these limitations running the program multiple times with different starting positions. [16] The other type of algorithms used are known as stochastic. At their simplest these algorithms build on the principles set down by deterministic solutions. To improve on the discovered results though a random component is added into the evaluation. This random component is meant to allow the agents in the algorithm to avoid being trapped by local optima. The random components range from randomly changing the position or movement vectors of an agent during each iteration.[4] More complicated methods such as simulated annealing allow for selection of less optimal solutions over more optimal ones. This probability of this preferred selection slowly lowers over time until only optimal solutions are selected.[15] Some examples of stochastic algorithms include genetic algorithms, Particle Swarm Optimization and Firefly Algorithm.

To illustrate the effectiveness of using computer algorithms to find optimal solutions to problems their application in an engineering example will be explored. In this example pressure vessels are being manufactured with certain size and strength requirements. These requirements can be described in a series of equations and variables. The variables represent things like size, wall thickness and required pressure. The equations set the bounds for these variables and determine if the specifications such as volume and strength are met. Ultimately the manufacturer wants to meet these needs while minimizing the cost of production, another equation in the setup. To effectively find an optimal, lowest cost, solution to this problem the PSO and FA were used. Each was able to search through the equations and boundary conditions to discover solutions that fit the manufacturers needs while minimizing costs. In fact it was shown that in this specific case that FA outperformed PSO by three percent.[4]

2.1 Search Space Optimization

2.1.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) finds its origins located in a very different field from optimization algorithms. The original idea for PSO comes from computer simulations of birds in flight and their flocking behavior. Computer scientists were interested in modeling the way in which birds grouped themselves, simultaneously and the aesthetics of those interactions.[1] Two models of these interaction, one by Reynolds and the other by Heppner and Grenander were at the forefront of this endeavor.[1] Both attempted to create flocks of creatures that maintained optimal distances between each member of the flock and its neighbors.[1] These simulations were able to show groups of organisms traversing through a space in unison and with coordination. To add to the simulation, Heppner added what was called a roost within the simulation space. This roost was simply a location within the environment that would attract the members of a flock to it.[1] This idea quickly developed into simulations of flocks finding food, which are similar to roosting points. This was the foundation of PSO and its use in optimization algorithms.

By capitalizing on the ideas of using flocking behaviors to find food resources scattered in an environment, Kennedy and Eberhart were able to develop an efficient means to search this environment for these resources. Each agent in the flock represents a potential solution to a given problem.[2] In the case of food finding, each agent would represent how much food was at a particular location in the environment. Each particle adjusts its speed dynamically according to the comprehensive analysis individual and population flying experience, and fly to the best position that it experienced and other particles have.[2] Here agents have been replaced with the term particle and instead of being in a flock they now belong to a swarm. This was done by Kennedy and Eberhart for a specific reason. In their research the term swarm was used in much of the supporting literature. It was determined that their flock acted more as a swarm for these reasons.[1]

- Proximity principle: the population should be able to carry out simple space and time computation
- Quality principle: the population should be able to respond to quality factors in the environment

- Principle of diverse response: the population should not commit its activities along excessively narrow channels
- Principle of stability: the population should not change its mode of behavior every time the environment changes
- Principle of adaptability: the population must be able to change behavior mode when its worth the computational price

The term particle was selected as a compromise. While it could be argued that the population members are mass-less and volume-less, and thus could be called points, it is felt that velocities and accelerations are more appropriately applied to particles, even if each is defined to have arbitrarily small mass and volume. [1] Thus was born the Particle Swarm Optimization Algorithm.

PSO works by distributing a number of particles randomly through the search space of a problem. Each particle knows its position in the search space, its velocity through that space, and the best position it has visited.[7] The best position represents the particles position at a certain iteration where it achieved the highest fitness rating for the given problem. In early application and models of PSO, particles followed their velocity vectors to update their positions. These velocities were changed by a certain increment to attract them to their best position.[1] Therefore each particle acted independently of one another and was only attracted to and tried to improve its best results. In later developments of the algorithm the best position of all agents was taken into account. A global best position found by any one of the agents was determined and all agents were then attracted to that location as well as their own best location.[2] Assuming the function has some form or continuity by attracting agents to the location of good solutions there is a chance that they will discover better solutions. By approaching these areas from different directions there is a high probability that the best solution in the area will eventually be found.[7]

PSO has been used in a wide variety of real life applications in both the industrial industries and laboratory research. One of the major branches of its use has been in training neural networks. A hybrid method of it has been shown to have a distinct improvement in classification problems as well as in function approximation.[7] Another application involved finding the optimal mixing solution to facilitate the growth microorganisms and showed a dramatic improvement over traditional methods. [7] As stated above there

are also application of this algorithm in manufacturing processes to maximize profit.

2.1.2 Firefly Algorithm

Firefly Algorithm (FA) evolves the method of attraction for the particles in PSO. In nature fireflies flicker and flash for mating, hunting and communication.[5] Xin-She Yang used this phenomenon in conjuncture with ideas from PSO in order to find optimal solutions to similar types as PSO. Similar to PSO, FA attracts agents in the population swarm to more desirable locations in the search space. It does this by using the brightness of a firefly to determine how attractive these agents are.[4] The algorithm can be summed up by these rules.[4]

- All fireflies are unisex so that one firefly is attracted to other fireflies regardless of their sex.
- Attractiveness is proportional to their brightness, thus for any two flashing fireflies, the less bright one will move towards the brighter one. The attractiveness is proportional to the brightness and they both decrease as their distance increases. If no one is brighter than a particular firefly, it moves randomly.
- The brightness or light intensity of a firefly is affected or determined by the landscape of the objective function to be optimized.

By using these rules a dramatically different behavior pattern for the system emerges. Instead of each agent, no matter their location, being attracted to one other agent, fireflies are attracted to the best in their local area. This attraction falls off over greater and greater distances. Due to this fall off, fireflies behave very differently than particles in PSO.

The attractiveness of a firefly agent is proportional to the fitness of it's location's solution. This means that more fit fireflies are brighter than unfit ones. Also a firefly of fitness X would appear brighter to a closer agent than to one that is further away.[5] The algorithm used to determine this attractiveness is the light intensity equation, a law of nature.

Here Io represents initial brightness, or in our case fitness. Y is the absorption rate. r is the distance between fireflies.[4]

This equation is used to attract agents in the population with the hopes of finding a global optima. The execution of the search routine can be described in the following manner.[4]

```
Objective function f(x), x = (x1, ..., xd)T
Initialize a population of fireflies xi (i = 1, 2, ..., n)
Define light absorption coefficient

while (t <MaxGeneration)
        for i = 1 : n all n fireflies
                for j = 1 : i all n fireflies
                        Light intensity Ii at xi is determined by f(xi)

                        if (Ij > Ii)
                                Move firefly i towards j in all d dimensions
                        end if

                        Attractiveness varies with distance r via exp[r]
                        Evaluate new solutions and update light intensity

                end for j
        end for I

        Rank the fireflies and find the current best
end while
```

The applications of this algorithm follow in the same footsteps as PSO. Because each algorithm hails from very similar backgrounds, they can be applied in similar fields. Training of neural networks, manufacturing optimization and use in research are all possible application for FA.

2.1.3 Comparison of PSO and Firefly

Due to their similar applications, comparison of PSO and FA can occur on an even playing field. By allowing each algorithm to have the same population size and number of execution epochs, an accurate comparison of their performance can be achieved.[5] In [5] just such a test was conducted on a list of continuous optimization problems. The tests showed when comparing the

most advanced PSO algorithms to FA, PSO found more optimal solutions in eleven of the fourteen test cases. This number drops to eight when firefly is tuned for the specific search.[5] These are not shabby comparisons. PSO is more than a decade older and has been able to develop itself much more precisely than the simple approach of the FA algorithm. These results were the initial stepping for the emergence of FA.

2.2 Clustering Algorithms

In computer science clustering data refers to partitioning entries into different groups based on their similarity. [17] The top-down view regards clustering as the segmentation of a heterogeneous population into a number of more homogeneous subgroups. A bottom-up view defines clustering as finding groups in a data set 'by some natural criterion of similarity'[12][17] Clusters are groups of data points that are more similar to each other in some manner.[17] The objects are clustered or grouped based on the principle of maximizing the inter-class similarity and minimizing the intra-class similarity[30, Page 25][17] An example of this could be a group of fiends talking at a mall. They are huddled together and form a tight boundary which defines the size and shape of their cluster of friends. Similarly data groups together in a search space to create define borders. In certain fields of computer science it is useful to be able to discover the boundaries of these clusters, or to classify data as belonging to one cluster or another.

2.2.1 K Nearest Neighbors

To solve the second type of problem, a simple and commonly used classification algorithm.[8] The k nearest neighbor algorithm looks at the k closest data points to a given unknown data point. The classes of these neighbors are polled, and a majority vote is taken in deciding which class to use for the unknown point. This means that if an unknown data point X is surrounded by three Y points and one Z point (k = 4), X will be classified as belonging to the Y cluster. This method for classifying unknown data is simple yet efficient classification algorithm widely in use.[8]

The k nearest neighbor (KNN) algorithm can come in one of two flavors, weighted and unweighted. Both versions of the algorithm use some sort of distance calculation, Euclidean etc. to discover the k nearest neighbors and how far they are from the query point.[9] In the unweighted version of the

algorithm the output, or class, of each neighbor is summed then divided by the value of k. This gives a simple popular vote for the output of the query point.[9] In the weighted version the distance neighbors are from the query point changes its contribution to the average calculation. Neighbors that are further away have less of a vote in the output of the query point than neighbors that are much closer.[9]

Problems do arise with this algorithm once the dimensionality of the search space begins to grow. One of the major slow down factors of the algorithm is the required distance calculations. As dimensionality and k grow, distance calculations begin to use more and more time to finish calculating.[8] Furthermore it is difficult to consistently reduce the number of distance calculations needed when finding the k nearest neighbors. In most cases the distance between the query point and all of the known points needs to be calculated to determine the closest neighbors.

Furthermore it has been shown in [10] that as dimensionality grows to the double digits and beyond, the nearest neighbor becomes more and more ambiguous. This is due to two different kinds of problems. The first of these is that as the number of dimensions grow, the distance between the closest and furthest neighbor shrinks.[10] Imagine standing in the center of a circle of stones and asking to choose which was closest. It is a fruitless exercise because while there may be a stone that is slightly closer than any other, it is not really much closer than any other stone. This makes the nearest neighbor rather ambiguous. Secondly when data is represented in some number of dimensions, computer algorithms can cause a heuristic error. [10] This sort of error occurs when a neighbor's distance calculation is thrown off by the way it is being represented.[10] Back to the rock example, if these rocks were floating at varying heights above the ground, an observer in the center of the circle on the ground would calculate different distances. Conversely an observer from a plane would only see a person in the center of a circle of rocks. These two perspectives differ in their view point and can therefore differ in distance calculations, similar to heuristic errors in computing.

2.2.2 K Means Clustering

K means classification differs from KNN in both the computations that are performed as well as the results. Instead of working on a set of known classified data, k means is given unclassified data points and asked to find relatively good clusters for them.[12] The algorithm arbitrarily creates clus-

ters within the data set by placing k centers in the search space. Each center represents a cluster and points belong to the closest center point. The algorithm looks at the variance within those clusters to determine the center of the cluster based on the squared distance of each point in the cluster. The center is then moved to this location and the process repeats.[11] The algorithm converges when there is no further change in assignment of members to clusters[12] In pseudo code [13]

```
Place k centers
While not converged
        Classify data into clusters
        Calculate mean position of cluster members
        Move k to mean of members
Loop
```

Applications for the K means algorithm include similarity grouping, relevant classification, approximating distributions, testing for relationships between variables in a sample and creating partition trees based on distance.[11] Concrete examples of these application include pulling k colors out of an image, known as color quantization.[14] Other real world applications include attribute prediction based on limited data, applications in computer vision for feature learning and use in primary component analysis.[11]

There are however drawbacks to the algorithm. Firstly this is what would be called a deterministic algorithm. There is no random component and will therefore reproduce the same results given the same starting position. Furthermore due to the way in which the centers are calculated and moved, there is the possibility that the centers will oscillate and thereby never converge on a single solution. This makes it difficult to determine when the algorithm should be halted.[11] Finally the algorithm is poorly suited to solve clustering problems that rely on an understanding of the density of the cluster. In certain applications this means that the algorithm finds very poor solutions to problems that have clearly defined clusters obvious to the human eye.[13]

3 Starfish Introduction

The Starfish Algorithm is like the above algorithms in many ways. The Starfish Algorithm uses the same principals of PSO and FA, creating and distributing a population, gathering information about candidate solutions, moving population members to new locations based on findings in an attempt to find an optimal solution to a problem. The algorithm creates a population of starfish, and then scatters them through the search space. Each starfish then gathers information from its surrounding area by looking at the fitness of candidate solution. The algorithm then uses information gathered by each starfish about the search space to move them to new locations in an attempt to find an optimal solution.

The algorithm implements all of these same ideas and implements them in different ways. Creation of the population is much the same in each algorithm. When the information gathering stage implements its routines the Starfish Algorithm looks at several candidate solutions in a given area around each population member. This information allows each member to gain an insight about its local surroundings and make more informed decisions about where to move next. Additionally each member is equipped with a set of behaviors designed to assist candidate evaluation and movement in special situations. These two differences, local area evaluation and executable behaviors, are the key differences between the Starfish Algorithm and other algorithms.

The Starfish Algorithm completes its task by executing a number of steps. These steps include accepting a list of initialization parameters, creating a Starfish population, performing calculations, executing behaviors and updating positions of population members.

3.1 Parameters

A parameter list is fed into the algorithm at the beginning of execution. This parameter list tells the algorithm how to execute as well as including necessary information about the search space. Parameters in this list include

- Bounds Represents the limits on the size of the search space. Possible solutions outside of these bounds will not be considered.
- Population Size The number of Starfish created for the population.

- Epochs The number of execution loops for the search.
- Attribute Array Values that determine when certain behaviors are executed and how those behaviors are executed.

3.2 Population

In order to discover optimal solutions in the search space a population of Starfish must be created. This population is tasked with exploring the search space and reporting the best solutions it finds.

- A population consists of a number of Starfish agents.
- Each Starfish agent consists of a number of nodes.
- Each node represents a potential candidate solution in the search space. Each node in an agent is placed in a specific pattern determined by the user. This pattern can vary in shape, size, number of nodes and the position of those nodes. In this way the user has complete control over the physical structure of a Starfish agent.

Each node in an agent is placed in a specific pattern determined by the user. This pattern can vary in shape, size, number of nodes and the position of those nodes. In this way the user has complete control over the physical structure of a Starfish agent.

3.3 Attributes

During the execution of the Starfish Algorithm, information is gathered by each member of the population. This information is then translated through mathematical calculations to allow the members to decide what to do next. These calculations include

- Fitness How well a potential solution meets the criteria of the problem being solved.
- Local Gradient (LG) The direction and estimated distance towards better solutions.
- Best Node The most fit solution found by an agent.

3.4 Behaviors

In order to increase the chances of finding optimal solutions, each Starfish can perform a number of actions or behaviors when the appropriate situation arises. Under these situations, the Starfish Algorithm can execute behaviors that some other algorithms are incapable of executing. This allows the algorithm to perform specialized tasks to help solve problems at hand. Some of these behaviors are

- Splitting The Starfish agent breaks in two to facilitate searching is more than one direction.
- Competition After a split the two halves compete against one another. The half that encounters more fit solutions regenerates the missing half, while the other decays and is then dropped from the population.
- Growth/Shrink The agent increases or decreases in size without changing shape. This is used for facilitating accurate solution discovery.
- Expansion Nodes in an agent move and distort the shape of the agent. This is used to find cluster bounds.
- Attraction and Repulsion agents are pushed away or pulled towards each other. This is a mimic of similar behaviors found in other algorithms.

3.5 Movement

The final portion of the algorithm is actually updating the position of each member of the population. Like in other algorithms the population members move through the search space in search of optimal solutions. Starfish perform the same action but use different criteria to determine how the position is updated.

- LG Following the LG from the calculation phase to update position.
- Best Node Moving towards the best found node in an agent.

4 Starfish Algorithm

The Starfish search algorithm is an attempt to improve upon other search algorithms by utilizing information, gathered during the search routine, to improve the process of candidate selection in order to increase the accuracy of results. Search and clustering algorithms, such as PSO, KNN and Kmeans, gather significant amounts of information during their search routines. However, this information is rarely shared, between the actors in the search algorithm, to make decisions about where to search for better candidate solutions. Thus decisions are made more or less independently based on information that is local to each actor, rather than utilizing all available information from all actors to make informed decisions about most efficiently expending resources where better solutions are more likely to be found.

There is a need to employ a more refined, controllable candidate selection process in search algorithms. The Starfish Algorithm aims to improve the process of candidate selection in order to increase the accuracy of results and the efficiency of resource utilization. It does this by evaluating how well a particular solution meets the requirements of a problem. Then, by sharing this information with other actors in the evaluation, an understanding of where additional, more desirable candidates are likely to be found is developed. Using this understanding, the algorithm can then make more informed decisions about where subsequent evaluations should take place, as well as what actions and behaviors to perform in a given situation. Being able to accurately pick better candidates for solution generation could mean a reduction in resources consumed during the evaluation. Additionally being able to gather more information about a problem without the need for additional querying allows algorithms to gain an insight into the inner workings of the problems and how factors affect the solutions.

Two attributes of candidate selection are integral to the Starfish search algorithm, Fitness how well a candidate solution solves the problem, and Local Gradient (LG) the direction in which candidates with better fitness may be found.

4.1 Fitness

Within the search space of a problem, there are nearly countless candidates that offer potential solutions. The fitness of a candidate is used to describe just how well a candidate solves a given problem. Solutions with

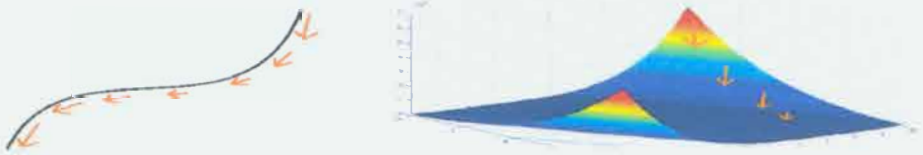
better fitness values are considered to solve the problem better than ones with worse fitness values. The Starfish Algorithm tries to find the candidate in the search space with the best fitness value.

As an example, consider hiking to the top of a mountain, where the problem would be defined as finding the highest point (altitude) on the mountain. Each different latitude and longitude combination on the mountain is a potential solution to this problem. Some latitude longitude parings end up putting a person very high up on the mountain, while others put them much closer to the base. Finding the latitude and longitude combinations that are higher on the mountain are better, or more fit, than the ones lower down, thus better candidates for solving the problem of getting to the top of the mountain.

4.2 Local Gradient

This attribute represents a direction in which more fit candidate solution may be found. LG is calculated by looking at the difference in fitness values of candidate solutions at given intervals over a certain area. The LG gives a basic understanding of which direction candidates with better fitness values may be found. It is assumed that by traveling in the direction of the LG it is more likely that more fit candidates will be encountered.

Drawing parallels to the hiking example, say there are several teams trying to reach the summit. Each team knows their position and how high up they are. If it is noted that the teams that are further and further west are getting higher and higher, more than likely traveling in that direction will take them closer to the summit. This basic example is the essence behind the LG calculations and decision making in the Starfish Algorithm.



The Starfish search algorithm uses information about candidate fitness and the LG to make informed decisions about where to look for candidate solutions that are most likely to solve the problem. By looking at the differences in fitness values and the direction in which more fit candidates lie, the algorithm is directed to search only in the areas that are likely to contain the

best candidate solutions.

4.3 Search Parameters

Starfish search has been designed to accept a number of different parameters in order to allow it to work with a large variety of problem sets. These parameters define how the algorithm functions and behaves while it runs. The parameters are upper and lower bounds, a number of allowed turns called epochs, the size of the starfish population, and an attribute array used for defining constants.

Upper and lower bounds of each dimension must be given to algorithm. These bounds tell the algorithm how far it can search in any given direction. This gives users the ability to limit the range of searching in a particular dimension if certain factors or requirements for solutions are already known.

The epoch variable is used to determine how many iterations the algorithm has in order to discover a solution. Population size is used to set how many members are generated for the population array. The population array contains all candidate solutions for the problem space.

The attribute array can hold any values the designer chooses it to hold for a specific application. It's main purpose is to allow the user to set certain properties of the algorithm. This gives great flexibility and robustness with the search algorithm as well as testing procedures. It allows users to tweak different variables for different applications of the algorithm. It also allows for other techniques to discover what variable combinations are most potent for solution finding in a problem space.

5 Structure

The Starfish Algorithm consists of three main classes. These classes are the search, starfish and node classes. The purpose of these classes are to organize information used in the searching process. The algorithm consists of three main parts, the Search class, the Starfish class, and the Node class.

The Search class contains the logic for how a search is executed. It decides which actions to perform when in order to give the best possible chance to find desirable solutions. The class is passed a list of parameters that tell it how to behave and setup its variables. This class also contains a list of objects that actually perform the searching operations. This list is called the population. These objects are allowed to independently decide what to do and where to look in the search space. They are known as Starfish agents.

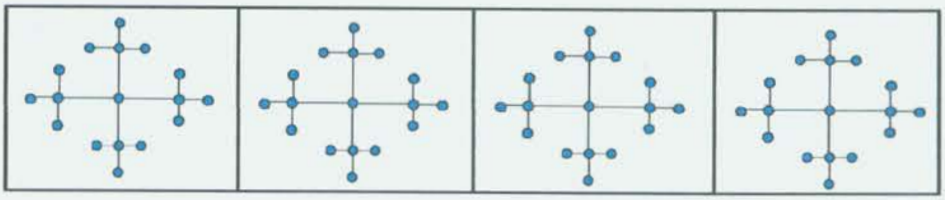
Starfish class objects are the Starfish agents that perform the actual searching in the algorithm. Each object in the population pick locations in the search space to check for candidates and then decide where to move and what actions to perform. Their decision process is as follows, calculate LG vector, decide what behaviors to execute, execute behaviors, and determine next location to look at. Each Starfish object contains a list of nodes. These nodes represent candidate solutions in the search space. The Starfish keeps track of these nodes' locations and fitness values in order to calculate its LG and make decisions.

Nodes represent candidate solutions in the search space. Each node belongs to a single Starfish agent. Nodes understand their own location as well as how fit of a solution they represent.

5.1 Search

The Search class is the driving force behind solution finding in the algorithm. This class contains the high level structure of the code that determines the behavior of a single starfish. It also contains functionality for creating and maintaining a population, looping for a set number of epochs, and recording statistics about performance.

The population of the Starfish Algorithm is a list of candidate solutions known as Starfish agents. Each member of the population is updated during each epoch.



5.2 Starfish Agent

A starfish agent is the most basic searching structure in the algorithm. Each starfish is tasked with trying to locate the best solution that it can find. Each agent knows certain attributes about itself and can perform a number of behaviors. Agents contain a number of what are called nodes. Nodes are simply a candidate within the search space and will be discussed later. The most important node to the agent is called the root node. This root node is what the Starfish Algorithm is attempting to optimize or find the best solution for. All other nodes belonging to the agent are children of this root node in some way. These children are used to determine the gradient of the search space at the location of the root node.

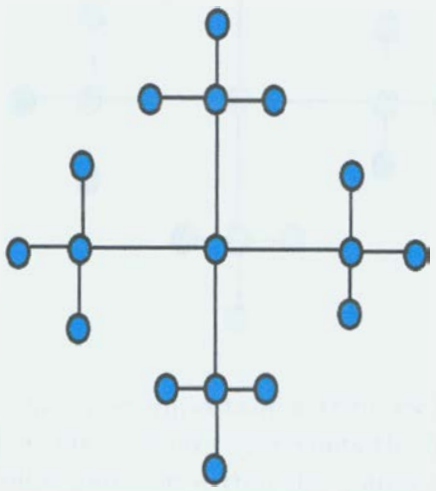
The gradient of the agent is known as the Local Gradient, or LG. This vector represents the direction and steepness of the. This LG property is the driving force behind traversing each agent through the search space in the hopes of finding more desirable solutions. A complimentary property is the Normative LG of an agent. This is simply the LG vector after it has been normalized by dividing the LG by its magnitude.

Since the LG moves an agent from one position to another, the agent must also keep track of its position. This position vector represents the variables that it is trying to optimize. Going back to the example of a hiker, the latitude and longitude of said hiker analogs the position vector of an agent. Each is trying to discover combinations of values that derive the best answer, or altitude in the hikers case.

Other parameters of the agent class include a State property. This property is used to create a finite state machine for the agent. When the agent is called upon by the Starfish class, its state determines what actions and behaviors will be used during a certain epoch. There also exists a Radius value for the agent. This value is used to give a basic understanding of how much the agent has grown or shrunk during execution. Furthermore it is used to update the position of the children of the root in the event that the

agent tries to grow or shrink.

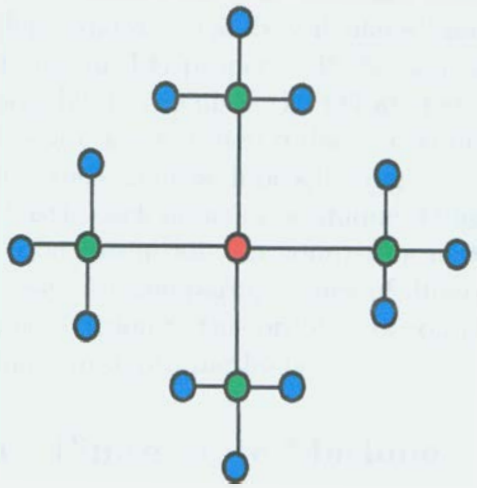
A final parameter of the agent class is known and the Competitors List. This property will be discussed in detail later. The Comp List will simply be described now as referencing the other half of an agent after a split occurs during the searching procedures.



5.3 Node

The node class is the most basic unit in the Starfish Algorithm. It represents a single candidate solution within the problem space. Nodes are hooked and grouped together in such a way that an accurate determination of the LG at a given spot can be determined. All nodes must belong to an agent in the population. Nodes can be classified as root, middle or leaf. Root nodes are the point that an agent is trying to optimize. All other nodes in the agent are children, children of children, etc, of the root node. Like in a tree structure, only this root node needs to be referenced because all other members are referenced by this node. This parental hierarchy allows for information passing between different nodes. Leaf nodes are nodes that have no children and a parent. They are the end of the line for calculations and recursive methods. Middle nodes are as they sound, between root and leaf nodes. Each middle node contains a list of children and a parent. This hierarchical structure is very similar to those seen in tree traversal algorithms

where a node has a parent higher in the tree and children lower in the tree.



The most important attributes in the node class is its position vector. This position vector represents the node's position. Root nodes are given an absolute position within the search space. All other nodes do not know their absolute position in the search space, rather only how far they are away from the root node. By updating only the root node's position the entire starfish can be moved through the search space without needing to calculate each node's exact position. This was done to allow for fewer updates to the node as the search progressed. Allowing agents to grow, shrink and split was cause for serious concern about calculations of all the nodes' positions. To alleviate this problem relative positions were used instead.

As discussed above, each node also has a single parent and a list of children. Root nodes have no parents, only children. Middle nodes have both children and a single parent. A middle node's parent can be either another middle node or the root node. Likewise its children can be other middle nodes or leaf nodes. Leaf nodes have no children and a single parent that is either the root node or a middle node. By setting up this structure used in tree hierarchies, information can be passed from parent to child or vice versa.

Each node also has what is known as a fitness value. This fitness value represents how well the candidate solution represented by the node solves the problem. This value is used to determine the best solution found by an agent

which has repercussions in movement decisions as well as other behaviors.

Since each node is a candidate solution within the search space, the LG of that candidate can be calculated much like the root node. Therefore each node has an LG property. If the node in question has no children, then it is impossible to calculate the LG at that position, and it is set to 0. These LG values are used to determine if certain behaviors can be performed on the agent, such as growth or splitting.

Lastly each node has a unique string name within an agent. These names are used specifically for comparing nodes within an agent. These names are not used for comparing nodes of different agents. It allows for a quick and simple solution to this problem of comparing nodes without needing to resort to more in depth methods.

5.4 Finite State Machine

In order to control the execution of behaviors by each agent of the population, a finite state machine(FSM) is used. This FSM gives agents the ability to choose what behaviors to execute in a given situation. Specifically it is useful for chaining behaviors together one after the other. For example, a starfish could execute a certain behavior until some criteria was met. The state of the FSM could then be changed to allow for different actions to be executed once this criteria was met. The FSM allows the agent to execute the first behavior completely, and then move to the second one using convenient easy to understand code. Furthermore this allows an agent to set its state and change what behaviors it will perform while executing a behavior without the need for a return parameter. This led to cleaner, more readable code while using recursive method calls.

5.5 Population Creation

To create the population methods to create both agents and nodes are evoked. The agent creation method is passed information about the absolute location of the root node and starting state for the FSM. All other attributes, such as LG, are set to initial default values. At this point the agent itself begins creating nodes.

To create nodes, a relative position to the root, a name, and the number of children must be provided. The position defines where the node is in relation to the root of the agent. This vector must contain the same number

of dimensions as the problem itself. Along with the name, this vector is used to set these two parameter of the node.

Creating children is much more intensive. A list, containing the number of children a node has is passed to the node creation method. The first value of this list is popped off and used to iterate a loop. The position and name lists are structured and used in the same manner, popping off the first value and using it as a parameter for the new node. This loop recursively calls the node creation method, passing the name, position, and number of children list minus the first values. In this manner new nodes are created and returned once the method finishes executing. The returned node is then added to the calling nodes children list. This process repeats until the lists are exhausted.

```
Names = [1,2,3,4,5]
Positions = [0,0; 0,1; 0,-1; 1,0; -1,0]
NumberChildren = [4, 0, 0, 0, 0]

function [E] = Makeagent(Names, Positions, NumberChildren, StartPosition)

        E = agent()
        E.Position = StartPosition
        E.LG = [0,0,0]
        E.NormLG = [0,0,0]
        E.radius = 1;
        E.State = 1;
        E.Comps = []

        E.root = MakeNode([], Names, Positions, NumberChildren)

end

function [N] = MakeNode(Parent, Names, Positions, NumberChildren)

        N = node()

        N.parent = Parent
        N.position = Positions.pop()
        N.name = Names.pop()

        for i = 1:NumberChildren.pop()
```

```
            C = Makeagent(N, Names, Positions, NumberChildren)

            N.children = [N.children; C]

        end

end
```

6 Basic Operations

The Starfish Algorithm consists of a set of basic operations, calculations and decisions that are performed during each epoch and collectively known as behaviors. These behaviors define the functionality of the algorithm as well as the solutions it generates.

The first set of operations that each agent must perform during an epoch, is to update its attributes. These attributes need to be recalculated during each epoch in order to accurately determine what the agent should do in order to discover better solutions. The calculations that need to be performed are as follows.

This information is then used to decide how to move and update the position of the root node within the search space as well as determine what behaviors need to be executed during the current epoch.

There a wide range of behaviors that can be executed by an agent during an epoch. This paper discusses several that were designed and used for specific purposes in solving problems in the applications of this algorithm. There is however no reason to believe that additional behaviors can be designed and used. Most of the time the behavior executions occur before movement does but is not a requirement. This is because once the attributes of the agent are calculated, they describe what actions the agent should take while at that position, not the one it is about to move to.

Each behavior first needs to determine if it has met the criteria necessary for execution. This prescreening happens before the behavior method is executed. The agent uses the values of the attributes to determine if a certain behavior is likely to produce better results in the given situation. If this test is passed the then behavior is executed.

Behaviors are designed to increase the likelihood that an agent discovers better candidate solutions than the ones it is currently exploring. These behaviors will be discussed in detail later but include splitting, attraction, repulsion, growth and shrink, expansion, competition, regrowth and decay. Behaviors can kick off an opportunity for more behaviors to execute as well as change the state of the agent based on the results of its execution.

After the attributes of an agent have been updated and behaviors have been executed, the agent needs to decide where it needs to next check in hopes of finding more desirable candidate solutions. A movement method is called and passed information relating to the movement speed, LG, and most fit node of the agent. This method then decides whether it wants to

move along the LG vector, or towards the most fit node in the agent. The position is then updated and behavioral execution takes over.

6.1 Fitness Calculation

To do so a function must be provided for each problem that takes a candidate solution, and returns the fitness value of that solution. This is known as the Fitness Function. The Fitness Function can either be developed by the user to grade candidates in a custom methodology, or be determined by some known mathematical function. In search space optimization the latter is used. Functions that create 3D surfaces, such as the Michalewicz function, both define the search space as well as the fitness of the candidate solution. By passing in an X Y location pairing to the Michalewicz function, a value for the height at that point can be generated and used as a fitness value.

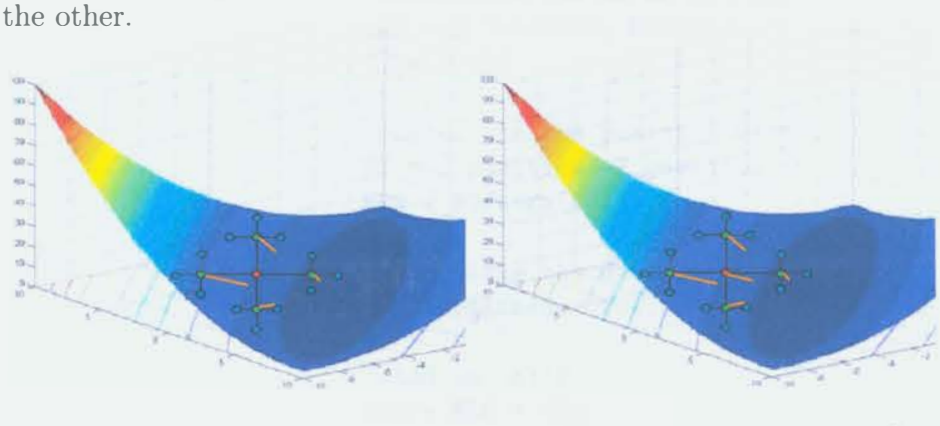
User defined function offer the ability for the users to determine what makes one solution better than another. For example, pretend a user would like to find the most densely populated portion of a cluster of points. They can define a function that returns better fitness values for areas that have more points in a given radius of a candidate, than a candidate with fewer points in that radius. In this way a user can define the criteria for how a problem is solved. It offers flexibility and modularity that can be applied to many varying types of problems.

6.2 Local Gradient Calculation

The LG of an agent is calculated by looking at the direction a child node is from its parent, as well as the difference in fitness values between the two nodes. By looking at this difference in fitness values, the algorithm determines whether a child node pushes or pulls a parent node. Pushing a parent node means that a force vector is applied to the node in such a way that it moves away from the child's position. Pulling a parent node means that a force vector is applied to the node in such a way that it moves towards the child's position. The difference in fitness values are then multiplied by the directional vector between parent and child to generate a force vector. By summing these force vectors up along with the LG of each child, the local gradient for a node can be calculated.

By performing the calculations in this way, a very accurate vector of the LG is created. This vector points along the LG or slope of the search space with a high degree of accuracy and its magnitude represents the steepness. For example, if this calculation were done on the side of a steep hill, the vector could be calculated to point directly up the hill and the magnitude would be very large. This would cause an agent to rapidly traverse up the hill. If the same calculation were done near the top of the hill, where it had leveled off more and become less steep, a very different LG would be calculated. This LG would still point along the slope of the hill, but its magnitude would be much less because the hill has leveled off. This calculation does not prevent overshooting of the hilltop. PSO and Firefly have similar problems with overshooting desirable solutions which can be negotiated in a number of ways. Starfish search addresses this problem by changing the speed based on the gradient. Even if overshooting occurs, as the agent approaches the hilltop the overshoot becomes less and less.

With this simple calculation the algorithm gains insight into how the search space is structured. Using that information each agent can make intelligent decision about where to move in order to find more fit candidates. Furthermore, by performing this calculation on each node in an agent, the algorithm understands what the topology of the search space. It can then use that information in deciding which behaviors are most appropriate to execute when a specific topology is encountered. An example of this occurs with the splitting behavior. If the algorithm determines that a child is being pulled in a different direction than the parent, then it can kick off a split behavior and explore in both directions instead of needing to choose one or the other.



`PLG = Parent's LG = n dimensional array of 0s`

```
For each child of parent

        NP =  normalized position relative to parent
        DF = difference in fitness values
        CLG = child's LG

        PLG += CLG
        PLG += DF * NP

Next child


function [PLG] = CalcLG(P, Bounds)

        N = # dimensions in search space
        PLG = Parent's LG = zeros ( N,1)  //N by 1 array of 0s

        //For every child of P
        For k = 1:NumberChildren

                C = P.children(k)

                //If the child is in the search space
                if  InBounds(C.position, Bounds)

                        //Calculate the LG of the child
                        CLG = C.CalcLG(C, Bounds)
                        C.LG = CLG

                        //Calculate the position vector
                        PS = C.position  P.position
                        NPS = PS/norm(PS)

                        //Calculate the difference in fitness
                        DF = C.fitness  P.fitness

                        //Add in child's LG
                        PLG = PLG + CLG
```

```
                    //Add in child's push or pull
                    PLG = PLG + DF * NPS
                end
            end
            P.LG = PLG
        end
```

6.3 Movement

Moving agents through the search space is the primary way in which more fit candidate solutions are discovered and evaluated. Each agent starts at a randomly selected starting position. This position is then modified during each epoch unless this action is specifically prohibited. The agent can decide to move in one of two ways, each of which has its advantages. Either the agent can follow the LG vector, or decide to move towards its most fit node. Allowing these two separate movement types overcomes difficulties faced by only allowing movement of one type. The DecideToMove method was designed to take four different parameters. These parameters are the best node in the agent, a movement speed factor, a minimum value for the LG magnitude, and the search space bounds.

Movement by following the LG is normally the most common way in which an agent chooses to change its position. This type of movement is ideal in steep or undulating topology. Under these circumstances the LG has a large enough magnitude to allow for speedy progress towards more desirable candidates. Using the calculated LG, the agent updates its root position. The root position is calculated to be

agent.root.position += agent.LG * MovementSpeed

In this way the user and the algorithm has the ability to decide by what percentage of the LG the agent will move. This allows for dynamic changes to the speed of the agent in given situations. Due to the way the agent's nodes are structured, with relative positions to the root node, no more action needs to be taken after the root position is updated.

The second way an agent moves is by moving to its most fit node. The most fit node in an agent is discovered by simply querying all nodes and finding the one with the best fitness. This type of movement is ideal when the topology of the local area is much more level. Under these circumstances there is not a lot of difference between the fitness of nodes in the agent. This

means that the magnitude of the LG is very low, which in turn produces very slow movement. For a time this was an active problem with the algorithm. Agents would hit flat areas in the search space and get stuck moving very slowly. Allowing agents to ignore the LG and move directly towards their best node alleviated this problem. This process is done by moving the best node's parent to the best nodes position. The rest of the agent is then drug along with the parent node. In this manner the best node's position is passed up the tree and eventually to the root which is the goal of the algorithm. Once again due to the structure of the agent and its nodes, after the root position is updated, no further action needs to be performed.

```
//Passed the best node and the agent it belongs to
MoveToBestNode(N,E)

        DPos = difference in position between child and parent
        Dpos = N.position  N.parent.position

        E.root.position += Dpos
end
```

7 Behaviors

7.1 Attraction

Attraction is a behavior in the Starfish Algorithm that pulls agents in the population closer towards each other. Each agent in the population is attracted towards the agent with the most fit root node. The force of attraction can be varied by the user using the Starfish's parameter list or can be coded to dynamically alter under different circumstances. This behavior was included in order to mimic the behavior of attraction seen in PSO and Firefly search algorithms. These two highly popular algorithms use an attraction behavior to move members of their populations in order to find new candidate solutions for evaluation. Since this behavior worked so well for these two search algorithms, it was included for use in the Starfish Algorithm as well.

7.2 Repulsion

Repulsion is the exact opposite behavior of attraction. While in certain circumstances and problems attraction results in better solution finding, it others it reduces the quality of the solutions. This can be clearly seen in clustering algorithms. In clustering, the user hopes to find unique sets of data that do not overlap with a clear boundary. If an attraction force were applied to the agents in the population, it would break this principal and cause poor clusters to be formed. Instead we would like the agents in the population to be forced away from each other in order to create these divergent sets.

In the Starfish Algorithm this repulsive force is calculated in a nearly identical fashion as the attractive force. Each agent finds the closest other agent to it. It then applies a force to this agent in the opposite direction to push it away. The force of this push can be altered by the parameter list and by the distance between the two agents. Adding this repulsive force helps to prevent overlapping of agents in the search space and has led to better results in clustering tests than without it.

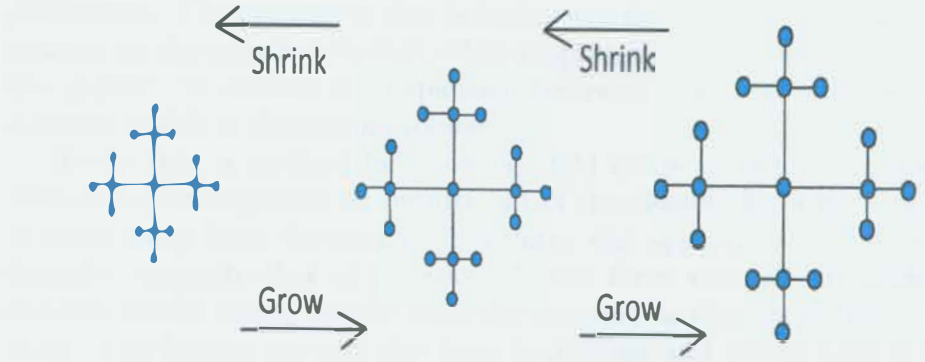
7.3 Growth

agents in the population define the position of their middles and leaf nodes relative to its root node. This means that no matter what the position

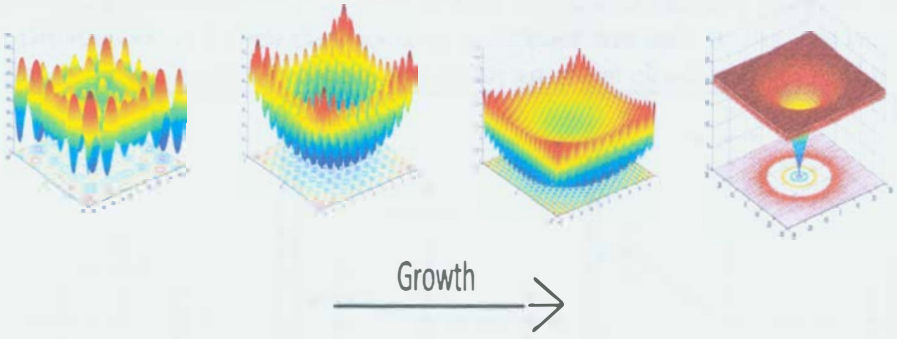
of the root is, every other node will be the same distance away from it. These distances are stored in the node's position property. The growth behavior represents the ability to change these relative position values by a certain percentage. Values greater than one cause the agent to uniformly grow in size. Values less than one cause the agent to uniformly shrink in size. This behavior does not change the overall shape and composition of the agent. Allowing the agent to change its size by either growing or shrinking helps it to find more precise candidate solutions. This is important for finding the optimal solution in a certain area, as well as ensuring that solution is not beaten by another some distance away.

Shrinking of the agent was designed to center the root node on the optimal solution in the local space. Once the root node of the agent is determined to be its best node, the LG of all children will point towards the root. In this instance there is no way to determine what direction to move the agent to find the optimal solution. Decreasing the size of the agent brings all of the root's children closer and closer to it. This gives the children a chance to encounter a slightly more optimal solution that the one located at the root. If a more optimal solution is found then the root is moved towards this position. By repeating this process and continuing to check for the best node and LG, the agent centers itself on the best candidate. This is an incredibly beneficial side effect of this simple behavior. It allows for more accurate final solutions when exploring potential optima.

Growing the agent is a response to two difference events. This can occur after shrinking the agent to a certain size, or by recognizing that the local topology ungulates too much. The first instance allows the agent to ensure that the optimal solution it has found is indeed the optimum. Growing the agent extends the children away from the root. Doing this allows each child to possibly encounter a more optimal solution. More importantly though it allows the children to calculate the LG further away from the root. If the LG begins to pull a child away from the root in this instance, then a split can occur. This behavior then allows the split child to continue to explore along that gradient. If it so happens that the gradient produces a better solution then the searching continues. This growth factor allows fro the agent to continue exploration of the local area without needing to lose the answer it already found.



The other opportunity to use the growth behavior is if the topology is very hilly. Under these circumstances the LG has little meaning and generally points towards local optimum, or bad directions of travel. By recognizing that this has occurred, the size of the agent can be changed to better fit the topology. Growing the agent in these circumstances given the agent a better understanding of the overall topology of the search space instead of just an understanding of the local area. This means that the agent can move along the overall gradient instead. An analogy of this is trying to climb up a boulder strewn hill. If you only climb up to the highest peak in the local area you have chosen a poor summit. However if you look around and understand that overall the hill points up in such a direction, you can follow that to the true summit.



7.4 Expansion

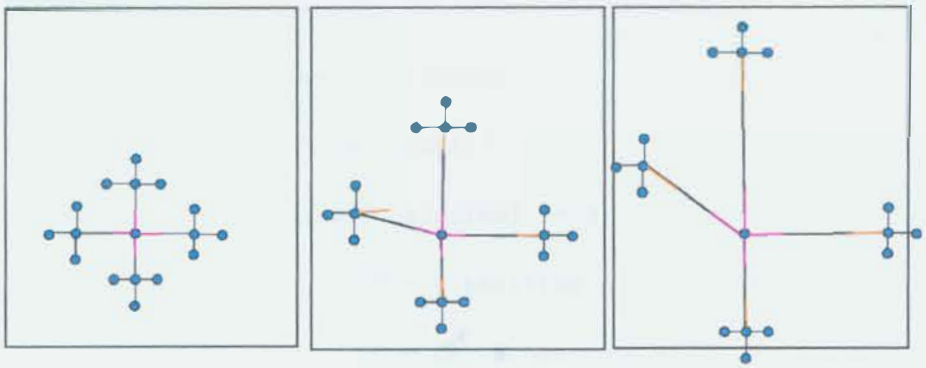
Expansion while sounding similar to growth, is a vastly different process from it. The expansion behavior was specifically written for use in clustering

procedures. The purpose of this behavior was to grow the agent in a similar manner as the growth behavior, while keeping all children of the root within the cluster. In essence the expansion behavior was written to be an edge detector within a clustering problem.

To do this, a method for allowing child nodes of the root move freely while not allowing them to venture out of the cluster. In order to get them to move away from the root node, a force was applied to the nodes in the direction opposite that of the root. If this force were left unchecked, the children would simply slowly move further and further away from the root node. Two factors prevent this from happening and mimic edge detection. The first of which is that each child node, once the behavior begins, is told to follow its own LG vector. The root node stays stationary and all of the children nodes move independently of one another.

The second part of the equation is that the force vector moving nodes away from the root falls off as the fitness of the point decreases. With this behavior, as a node moves towards the edge of a cluster where there are less points and more open space or even a hard line, the fitness falls off dramatically. As the fitness falls off, the force applied to the node reduces as well.

Furthermore in this instance the LG of a point on the edge of a cluster almost exclusively points inwards, to the center of the cluster. With this kind of evaluation we get a balancing of forces moving the point away from the root and along the LG. Due to this the nodes expand towards the edges of the cluster and then stop as they get closer towards it. In this manner the bounds of a cluster are found and then used for classification of the data.



```
//N = current child node
//F = force of expansion
```

```
Expand(N, F)
        for each child of N

                C = N.child

                if C has children

                        CP = child's position

                        VC = vector from child to parent, N
                        NVC = normalized VC

                        CLG = child's LG

                        FIT = childs fitness

                        V = NVC * (F / FIT) + CLG

                        C.position += V

                        C.Expand(C, F)
                end
        end
end


Expand(N, F)

        for i = 1:size(N.children)

                C = N.children(i)

                if size(C.children) != 0

                        CP = C.position

                        VC = CP  N.position
                        NVC = VC / norm(VC)

                        CLG = C.LG
```

```
                    //Cannot drop below 1, otherwise F ends up too large
                    FIT = max( C.fitness, 1)

                    V = NVC * (F / FIT) + CLG

                    C.position += V

                    C.Expand(C, F)
                end
        end
end
```

7.5 Split

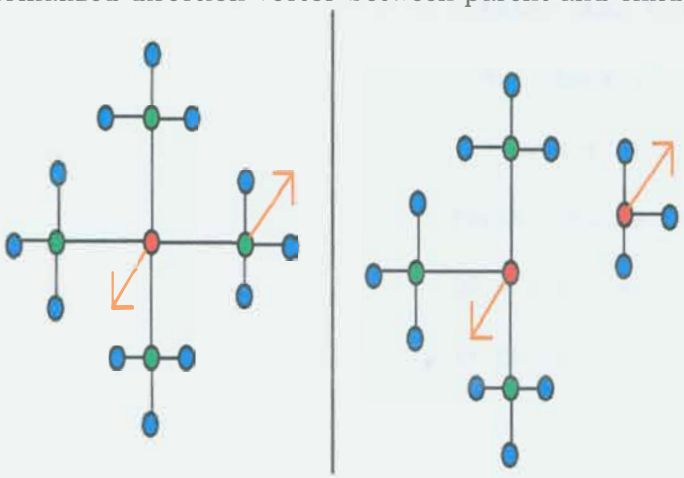
After the LG calculations, splitting is the other cornerstone of the Starfish Algorithm. Splitting allows an agent to break off a child from a parent and use it to create an entirely new agent in the population. These two halves then compete for survival by comparing which has the most fit solution. The agent that is more fit ends up regrowing its missing children, while the loser wastes away and is eventually removed from the population. Developing this form of behavior for the algorithm allows agents to explore two different path choices. The competition aspect prevents this split from permanently altering the size of the population. Three major things need to be completed for a successful split. The LG must be calculated for every node in an agent. A comparison of direction and magnitude of parent and child LG vectors must be made. Finally a new agent must be created while also removing the link from parent to child of the original agent.

The calculation of LG for each node is the catalyst for the split behavior. Using the insight into the topology that the LG gives the algorithm, it can be determined in what direction and with what force each node is being pulled and pushed in. Generally this results in a uniform decision at the root node of what direction to move in. This is not always the case though. Sometimes different children nodes are pulling the root in very different directions. When this occurs the agent does not get a clear picture of where to go next because the opposite pull tends to nearly zero out the LG magnitude. It is very similar to approaching an intersection while driving a car and having two passengers yelling out opposite directions. The root node has no idea at this

point which direction will lead to a better solution, so instead it decides to do both. By splitting the agent at the location where the disagreement occurs, the algorithm can allow both halves to explore the direction they wish to. In this way the agent can explore both options without needing to make a decision and miss a possible solution.

The calculation to decide if a split is appropriate for a given situation relies on an understanding of the direction of both parent and child LG vectors. The method looks for an example of these vectors pointing at least ninety degrees in different directions. When this instance occurs the agent generally begins to cancel out the LG at the root when summing the children vectors. It is at this time the algorithm recognizes that a split needs to occur.

A more technical explanation of this decision is as follows. The method searches through all parent child parings in the agent. If the magnitude of both vectors is great enough, for there is no good reason to split if the vectors are very small, the pairing is considered for a split. Each vector is then normalized. The dot product of these two vectors is calculated and then divided by their product. This calculation gives the cosine of the angle between the vectors. If this cosine is less than or equal to zero then the angle between them is greater than or equal to ninety. At this point in time one last check needs to occur. The algorithm needs to make sure that the child is not pointing towards the parent. If this check was not completed then once the parent or the root finds an optimum, it would trigger a split because all of the other nodes would be less fit. This calculation is done in the same fashion as the one above but the parent's LG vector is replaced with the normalized direction vector between parent and child.



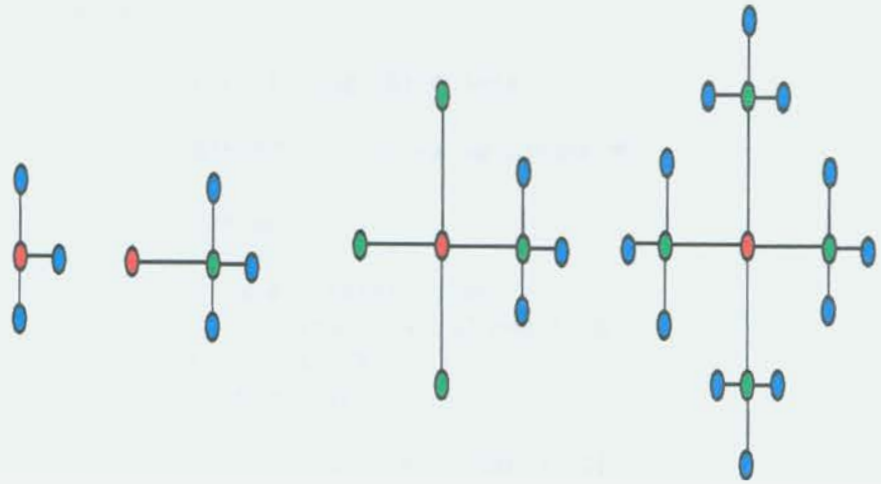
```
//N = current node
//Pop = Starfish population
//PopSize = size of population
//ReqSplitMag = min magnitude of vectors require for split
function[Pop, PopSize] = ConsiderSplit(N, Pop, PopSize, ReqSplitMag)

        NLG = LG of N

        for each child of N

                C = Next child of N

                CLG = LG of C

                MagCLG = Magnitude of CLG
                MagNLG = Magnitude of NLG

                if MagCLG is large enough

                        NormCLG = Normalized CLG
                        NormNLG = Normalized NLG

                        D = dot product of NormCLG and NormNLG

                        //Note there is no reason to take the acos of D, just see if le

                        If D is greater than the split angle

                                //Now check if the vector does not point inwards

                                V = vector between parent and child

                                NormV = Normalized V

                                DP = dot product of NormCLG and NormV

                                if DP is greater than the split angle

                                        //Split the agent into two halves
```

```
                                        PopSize += 1

                                        Newagent = make a new agent from C

                                        Pop(PopSize) = Newagent

                                        N.children(C) = null  //Remove the child from N

                                    end
                                end
                            end
                        end
                    end
```

7.6 Competition

Competition occurs after a split takes place. Both the old agent and the newly created agent keep a reference to each other in order to compete. During each epoch after a split the two halves compare against each other to determine which is has found the best solution. The agent that wins this comparison begins to regenerate its missing portions. The agent that is loses this begins to lose nodes from itself until it finally has no nodes left and is removed from the population.

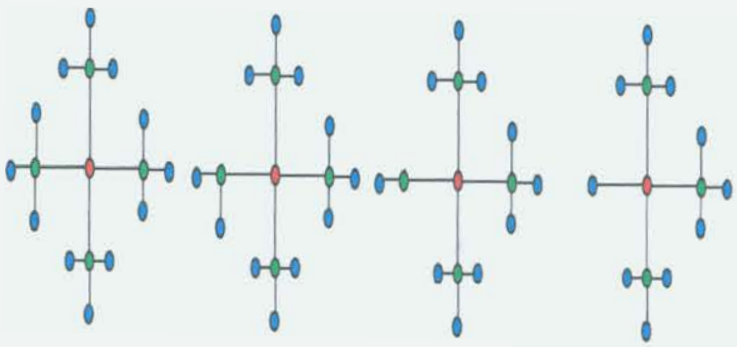
Regrowth is the trickier of the two processes. Both halves must understand what nodes it is missing and replace them to become a complete agent again. This means that for the agent that lost a child, the child, and all of its children, is regrown. For the newly created agent, regrowth means regrowing its parent, the parent's children, and the parent's parent, etc. In this manner the better half is grown back into a full agent, missing no children. To do this a comparison is made between the better half, and a model of a compete agent. This model is created during the creation of the population at the start of the Starfish Algorithm. The model is not used for any solution finding, only for a reference for this behavior. By comparing the nodes present in the winning half to the model, it can be determined which nodes are missing and therefore replaced.



```
//N = current node
//M = model node
function [Parent] = Regrow(N, M)

        if N and M don't have the same root

                //Need to regrow a parent node

                Match = M.FindMatch(N) //find matching nodes

                MatchPos = Match.position  Match.parent.position

                Parent = node()
                Parent.children = N
                Parent.parent = []
                Parent.LG = [0,0,0]
                Parent.name =  Match.parent.name

                D = N.position  MatchPos

                Parent.position = D

                N.parent = Parent
```

```
        else

                //Regrowing child node

                MatchC = FindMissingChild(N,M)

                C = node()
                C.children = []
                C.name = MatchC.name
                C.position = MatchC.position
                C.parent = N
                C.LG = [0,0,0]

                N.children = [N.children: C]

        end

end
```

Decay is a much simpler process and requires no comparison to a model. When an agent begins to decay the algorithm needs to decide which of its nodes it is going to lose. This selection needs to be done carefully. Getting rid of a node with children would also immediately remove all of its children as well. To avoid this a requirement is made that nodes selected for removal must be leaf nodes. The leaf node that has the least fit solution is set to null in its parent's children list. Thereby the node is removed from the agent and the decay occurs.



```
//E = agent being decayed
```

```
Decay(E)

        //WL = worst leaf node object
        //WI = index of WL in parent's children list

        [WL,WI] = E.FindWorstLeaf()

        WL.parent.children(WI) = []
end
```

8 Applications

8.1 Search Space Optimization

Global optimization problems, such as the Michaelwicz and Ackley functions, were the first problems that the Starfish Algorithm was evaluated on. In these functions, a bounded search space is explored in an attempt to find the minimum of the function. In this particular problem, it is difficult to find the minimum of the function in a highly accurate manner. Algorithms such as PSO and Firefly have been used to attempt to find these absolute minimums.

It was decided to use Starfish to try to improve on the results found for these functions. A list of twenty one mathematical equations were used for testing and comparison between the three search algorithms. Each algorithm was allowed to run for a set number of epochs and the best solution at each epoch was recorded and reported on. An important note to make about this testing is that the number of times the fitness function was called during each algorithm was closely monitored. By keeping track of the number of times each function had to use the fitness function, a form of comparison of efficiency could be made.

The goal for the Starfish Algorithm was to follow the slopes and curves of the equations and use them to lead agents to optimal solutions. Since these functions were three dimensional, and the algorithm needed to look for the lowest point on a specific axis, the fitness function was simply the mathematical equation. Giving this equation an x,y pairing as position of a node, it would return the height of the function at that location. Using this gave a clear understanding of the gradient of the function to the population members.

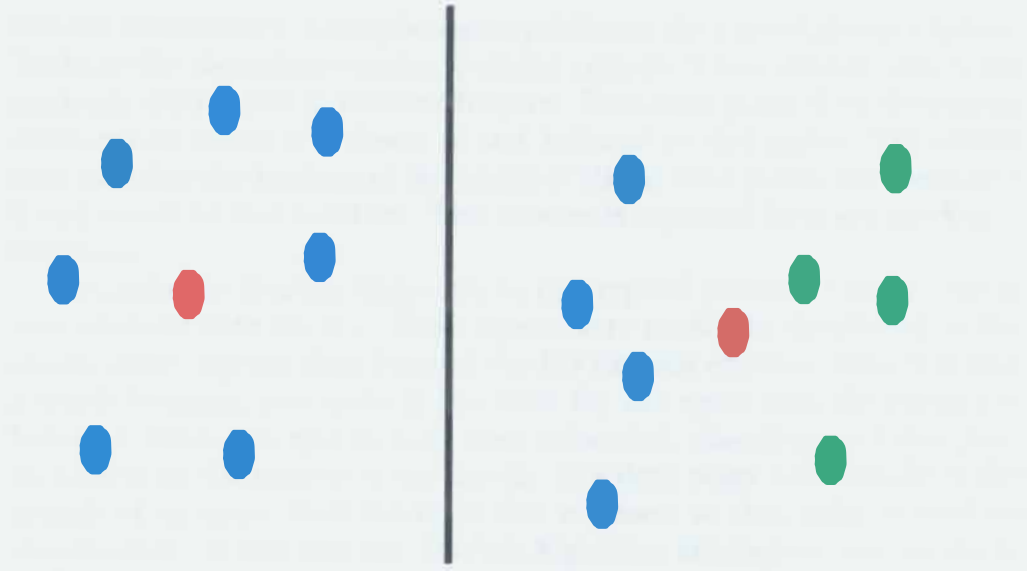
Behaviors for this particular problem set were used to improve results. The behaviors used were splitting, attraction, and growth/shrinking. Attraction was included to mimic the functionality of both PSO and Firefly which both use some kind of attraction. Splitting allowed the agent to travel down two different slopes in the search space. If an agent were to find itself on a ridge that fell off to either side, a split could occur and allow both sides to be explored. Shrinking the agent when a possible optima was found centered the agent in the optimal area. Growing the agent allowed it to ensure that the answer it had found during shrinking was indeed the best in the immediate area.

8.2 K Nearest Neighbors

K nearest neighbors, KNN, is a classification technique in which data of unknown class is compared to data of known classes. The algorithm strives to correctly classify the unknown data by looking at the closest K data points to it. Looking at these neighbors of know class, a popular vote is taken to classify the unknown point. In this manner unknown data is compared to classified data to attempt to correctly guess its class.

The Starfish Algorithm was used in this problem to try to give a higher correct classification rate for this problem type. It was hypothesized that by creating a well-defined fitness scheme, an agent could use the LG information to make better classification decisions. The agent structure needed to be fundamentally changed for this problem type. In order to keep the Starfish Algorithm on par with KNN, the parent child structure of nodes had to be discarded. Because KNN specifically tells how many points can be checked with the K value, the Starfish agents were also required to only check those points. If each algorithm were given the same data to try to classify, they would at least initially check against the exact same nearest neighbors. The algorithms differ in what they do with that information and their subsequent actions.

In order to describe the fitness function for KNN using Starfish, the concept of how well a given piece of data is classified. A point is considered well classified, and thereby has a better fitness value, if it's K nearest neighbors are all of the same class. Points that have many different classes or an even ratio of classes surrounding it are poorly classified and have a worse fitness value. This means that data points that have only one type of class surrounding them as the K nearest neighbors will attract towards it. Data points that are surrounded by an even ratio will repel away from them. This makes sense for the fitness function. Points can make a pretty good guess about their class if there is only one type of class around it. Points that have a more even ratio are basically forced to guess which way they want to go.



```
//P = data point
function [fitness] = Fitness(P)

        CLSS = FindKClasses(P, K) // get a list of the classes of the K nearest points

        PC = PopularCount(CLSS) //find how many of each class there are

        PC = sort(PC, 'descend') //sort it

        //Only grab the first two counts, the ratio of these two is good enough for a fi

        m1 = PC(1)
        m2 = PC(2)

        fitness = m2 / m1 //0 if really good, 1 if really poor

end
```

8.3 K Means Clustering

In K means clustering, an algorithm is tasked with classifying a set of data into K clusters. The algorithm is tasked with deciding which data point belongs to which cluster. The goal of the algorithm is to place similar data

into the same cluster, while placing very different data into different clusters. To do so the algorithms creates K cluster centers. These cluster centers are randomly distributed in the search space. Each data point then determines which cluster center it is closest to and 'belongs' to that center. The centers then calculate the location of the center of all the data points that belong to it and moves to that location. This process is repeated for a set number of iterations.

To apply the Starfish Algorithm to this type of problem, cluster centers were replaced with agents. These agents were randomly distributed in the search space. Agents then followed the LG towards clusters. After reaching a stable location, root node is the most fit, the agent uses the expansion behavior. Once the epochs have been exhausted, classification takes place by looking at the bounds of the agents. If a data point falls outside of the bounds of an agent, then the node that is closest to that point is used for classification. In this way the Starfish Algorithm attempts to use agents to find cluster centers, expand towards the edges of the cluster, and use the location of the nodes in the cluster for classification.

Average Euclidean distance was used for the fitness function of this problem. A set number of N closest neighbors were looked at and their distances recorded. The average distance of these points were used as the fitness value. The lower the average distance is, the closer the points are, and the more likely the point is in a cluster and therefore has a better fitness value.

9 Testing

The goal of this research was to determine if the way that information was used in order to calculate the LG for a particular problem impacted the accuracy of results generated. To compare the accuracy of the Starfish Algorithm in each of its three applications, standard models of the comparison algorithms were found and used. Each standard algorithm was then run against the Starfish Algorithm using the same population sizes and number of epochs. In this way the outcomes of the comparison were tried to be given absolute level playing fields. Outcomes of the comparison could thereby be used to draw conclusions about the effectiveness of the Starfish in each of its problems.

While in each of the tree applications, the accuracy of results generated was the primary focus, there are certain subtle differences that need to be taken into consideration. For the optimization problem, the minimum fitness solution found by the algorithm was what was being compared. Each algorithm, PSO, Firefly, Starfish was trying to find the lowest point in the solution space. The one that did this job the best was considered superior. One other factor needed to be considered for this problem, the number of checks against the fitness function each algorithm had to perform. By comparing the accuracy of results generated to the amount of data that had to be gathered, a result was generated to show which algorithm used information in the most efficient manner.

For the KNN testing, results were determined by comparing the accuracy of classification. By starting with a set of data who classes were known, and splitting that data into classified and unclassified sets KNN could be run on it. This means that each data point initially had a class associated with it but was then artificially forgotten. Then the classes chosen by KNN and Starfish were compared to the correct classification of the data before it was forgotten. This generated a percentage that represented how many of the data points were correctly classified by each algorithm. The algorithm that managed to classify the most points correctly was considered superior.

In the K means clustering algorithm, each algorithm had to separate data into distinct clusters. This data was initially grouped into correct clusters, but once again these clusters were forgotten for testing purposes. Each algorithm was then allowed to run over a set number of epochs in order to generate clusters it thought were appropriate. Then these clusters were compared to the initial clusters, looking for similarities to what data was placed

together in the same clusters. The winning algorithm was considered to place data points into similar groupings as the initial correct clusters. This means that if points x and y were in the same clusters, the algorithm that then placed them into a cluster was better than one that put them into different clusters.

9.1 Search Space Optimization

For the testing of search space optimization, each algorithm was tested against every test function. These tests were repeated across multiple population sizes and number of allowed epochs. Each test would be run and a final, most optimal solution and fitness would be provided to the user after execution. These numbers were compared against each search algorithms. By comparing these numbers and noting what test function, population size, and number of epochs were used conclusions could be drawn as to which performed best in a given circumstance.

It needs to be noted that in order to evenly evaluate each algorithm population sizes needed to be made slightly different. This was done in order to keep the number of checks against the fitness function between each algorithm during execution nearly identical. Because the number of checks needed to be identical, the Starfish Algorithm had a dramatically smaller population size than PSO or Firefly. This was due to the fact that a Starfish agent consists of many nodes, each of which needs to check against the fitness function during each epoch. In order to accurately determine which search function made better use of the available information this was a consideration that needed to be taken into account when running the testing procedures.

Testing for this section can be divided into two different categories. These categories pertain to the run time search parameters used by each of the algorithms. The authors of each algorithm defined and used certain parameters that controlled the functionality of the algorithm. These parameters pertained to the vectors and magnitudes used for procedure decisions and actions. This testing will be called Author Testing, AT. The other testing that occurred involved using a genetic algorithm to determine what parameter set yielded optimal results for a search algorithm. This form of testing will be called Genetic Algorithm Testing, GAT. While the testing procedures for each were the same it is important to note this difference because the results will be analyzed in separate sections.

During actual testing of a search algorithm four actions were performed.

The test algorithm, three dimensional function, was loaded into the fitness function. The search algorithm was prepared and passed parameters for execution including population size, number of epochs, bounds and execution parameter lists. The search algorithm executed its routines for the allotted number of epochs and reported on the most optimal solution encountered. This test was then repeated many times and an average result of all of these runs was created. The average results of the search were recorded and appropriately scaled for evaluation. All of these actions were wrapped up in a single control function that also had logic to appropriately scale the population sizes based on which search function was being used. For GAT, the methodology remains the same but parameters passed for execution was selected from a list that the genetic algorithm determined to perform best on the given test function.

Scaling of the search results was done in order to easily show how well each search performed on each test function. By knowing the, relative, upper and lower bounds of the fitness of the function, results could be scaled between the values of zero and one. Doing this allows for a function that has a fitness range from zero to ten to be easily compared to a function that ranges from zero to one hundred. In this way it could be easily seen how close to the minimum of the function a search found and therefore see which search algorithms did best against which test algorithms.

9.1.1 Genetic Algorithm Parameter Selection

The genetic algorithm parameter selection was designed to discover combinations of parameters used by a search algorithm that performed best. While each algorithm was designed to use parameters defined by the author, there was no guarantee that these parameters would produce the best result possible. Quite possibly the parameters could actually hinder performance in a given situation. This was the reason for using a genetic algorithm to find parameter sets.

The genetic algorithm works by creating a population of parameter sets. Each member of the population, a parameter set, contains a list of all the parameters the search function needs in order to execute. The values of these parameters are randomly chosen but are bounded by user input. Population members then compete against each other to see which is able to generate the most fit solution. The winners are retained in the population and go through a process called breeding. The breeding process takes two winning

members from the population and randomly chooses parameters from each to create a new population member. This is done by randomly selecting which parent contributes each parameter to the child. The child then replaces a losers from the competition process. This is repeated for a set amount of time in attempts to improve the population's performance in testing. By retaining and breeding winners while shedding losers, the algorithm tends to lead towards better and better parameter sets.

Using a genetic algorithm for parameter tuning will hopefully lead to more optimal solutions being found by all algorithms. It has been shown in [2] and [7] that using algorithms to fine tune parameters, more desirable solutions were found during experimentation. Furthermore, [1], [3] and [4] discuss the impact that different parameters had an impact on the solution sets. These papers give hope that more optimal solutions will be discovered by applying an evolutionary strategy to parameter selections. Due to the complex nature of the Starfish Algorithm, parameter interaction is not well understood nor its impact on solutions' optimality. Therefore these tests are being run to tune all algorithms to elicit their best performance.

9.2 K Nearest Neighbors

To test the effectiveness of the Starfish Algorithm against the KNN algorithm, it had to be determine which algorithm was more able to correctly classify data. A classify set and known set were provided to each algorithm. The classify set represented data points of unknown class. The known set are points that retained their classification. It is the job of each algorithm to classify the data points of the classify set by looking at the data points in the known set. These sets were created by picking a certain percentage of points from the original data set to be used in the known set. The percentage of the original data set that was used for the known set is a variable set by the user. For this particular application the method for choosing known data ensured that an equal number of data was selected from each different class type.

Each algorithm was given the same data sets to work with. Each also used the same value for k, representing how many neighbors could be looked at a given time. The Starfish Algorithm also needed to be told how many epochs it was allowed to determine a classification. These algorithms then returned an array that represented the chosen class for each point in the classify set. By comparing this array to the original classification of these

points, an accuracy rating could be calculated. This accuracy rating simply counted the number of points that were correctly classified and divided it by the number of total points. A rating of one meant perfect classification, while a rating closer to zero meant poor classification. By repeating these tests many times an average value for accuracy could be created. Comparing the accuracy ratings between the two algorithms gives the final result for which one performs better in a given circumstance.

9.3 K Means Clustering

Testing the K means clustering algorithm against the Starfish Algorithm occurred in a very similar manner to KNN testing. Once again a data set of clustered data points is used. This data is stripped of its cluster value which is used for comparison later. Each algorithm is then passed the data set along with how many clusters to attempt to create. The number of clusters can be user defined, however for this testing this number is defined to be the correct number of clusters in the data set. This means that if a data set is defined as having three clusters, each algorithm is told to try to find three clusters.

Once testing begins, each algorithm returns an array representing which cluster each data point belongs to. Since clusters for each algorithm are given random numbers as identification, it is impossible to simply compare these numbers to the correct cluster list. This means that originally a cluster of data may have been assigned the number one as its id. The algorithms being tested could give this same cluster the number two as an id but would still be considered accurate.

To overcome this problem, software was written to compare the data in a cluster. By looking at what the actual data was in a cluster and comparing it to the original clustering data, an accuracy score could be determined. Clusters that had very similar data had very high accuracy and vice versa. Each cluster provided by either Starfish or K means was compared against each cluster of the original data. The cluster that scored the highest in accuracy against the original data was considered to be the paired cluster. This means that if cluster one had a score of ninety five while cluster two had a score of twenty five against cluster four in the original data, cluster one was considered the correct match. After correct matches were found for each cluster from each algorithm, their accuracies were stored for evaluation. In this way the two clustering algorithms could be compared to one another

to determine which had done better.

10 Testing Sets

For each of the problem sets that the Starfish Algorithm was applied to, data for the tests was required. To ensure that results from testing procedures were accurate and representative, it was important to choose good, reliable and well used data sets. This means that the data sets chosen had to be acceptable by other scientists in similar fields of work.

10.1 Search Space Optimization

For the search space optimization tests, a list of twenty one functions were chosen for evaluation. These test functions included the Michaelwicz and Ackley functions among others. A complete listing of these test function can be found in Appendix A. These functions are well known in the computing world and have been used in the evaluation of PSO and Firefly search.[2][4][5][6] The Starfish Algorithm makes use of the different sizes, shapes and roughness or each function.

- Size The boundaries of the test. Some are very small, while other are very large.
- Shape The way in which the test is curved overall. Either towards one side, the center or in several locations.
- Roughness How hilly or bumpy the test is.

By using these different, well known tests, many different situations could be explored to evaluate the effectiveness of the algorithms. This gave a clear understanding of the performance of each algorithm.

10.2 K Nearest Neighbors

In the testing for KNN, four data sets from the UCI data repository were chosen. These data sets were the Iris, Breast Cancer, Liver and Wine classification sets. A list and links to these data sets can be found in Appendix B. These data sets come from a well-known source, UCI, and have been used extensively in papers that involve classification.[1][18] Therefore they are a good choice for testing both KNN and the Starfish Algorithm.

10.3 K Means Clustering

For testing of K means clustering, the Fundamental Clustering Problems Suite was chosen for use. It has been the standard, go to data set for evaluation of clustering algorithms. It has been used to test K means, single-linkage and other clustering algorithms.[19][20] Each data sets represents a certain problem that is solved by known clustering algorithms with varying success. This is done in order to reveal benefits and shortcomings of algorithms in question.[UCI] In this way the Starfish Algorithm can be compared to other clustering algorithms for evaluation using this data set.

11 Results

In this section the results generated from the testing procedures will be explored. This section will walk through the results from different experiments, explain their meaning and provide information as to where they can be found in the appendix. This section will not attempt to draw conclusions from these results, only provide the evidence from those conclusions drawn later.

11.1 Search Space Optimization

In the SSO testing, two different starfish types were used. The first batch of tests used a starfish that consisted of seventeen nodes. The second batch of testing used a starfish of only five nodes. Population sizes were changed accordingly for each test to equalize the number of checks made to the fitness function by all algorithms. The number of epochs each algorithm was allowed to use was held at a constant thirty for all tests. In this section, the statement that a one algorithm out performed another means that its answers were an order of magnitude more accurate.

11.1.1 Seventeen Nodes AT

For the starfish with seventeen nodes, tests were run with populations of size 1, 2, 3 and 6 against populations for PSO and Firefly that used the same number of fitness function checks.

Starfish Population	PSO/Firefly Population	Starfish Better	Starfish Worse
1	10	Bohachevsky: Sphere Sum	D&P G&P Perm
2	30	Bohachevsky: Sum	Perm
3	50	Bohachevsky	Perm
6	100	Ackley Bohachevsky	Booth Perm

The Starfish Algorithm also consistently did better than PSO and Firefly, and consistently worse against them on other functions. This means that

results were not dramatically different, but did tend to stay consistent in one direction or another.

Consistently Better	Consistently Worse
	Rastrigin Spiral

11.1.2 Five Nodes AT

For the starfish with five nodes, tests were run with populations of size 2, 4, 10 and 20 against populations for PSO and Firefly that used the same number of fitness function checks.

Starfish Population	PSO/Firefly Population	Starfish Better	Starfish Worse
2	10	Bohachevsky Matyas Ridge Sphere Sum	D&P G&P
4	20	Bohachevsky Easom Matyas Sum	Perm
10	50	Ackley Bohachevsky Easom Schwefel	Perm
20	100	Ackley Bohachevsky Easom	

The Starfish Algorithm also consistently did better than PSO and Firefly, and consistently worse against them on other functions. This means that results were not dramatically different, but did tend to stay consistent in one direction or another.

Consistently Better	Consistently Worse
Ackley Schwefel Shubert	

11.1.3 Genetic Algorithm Parameter Selection

The same tests above were repeated after allowing a genetic algorithm to chose the run-time parameters for each search function. Particle Swarm Optimization, Firefly Algorithm, and the Starfish Algorithm used these parameter sets to determine certain aspects of their behaviors. The hope of this was to fine tune each search algorithm for each test function. In this way it was hoped that results and patterns in the above section would be confirmed or even improved on by optimizing the parameter lists in this fashion.

By allowing the genetic algorithm to select parameters for each search algorithm, the algorithm was optimized to find the solutions for a given test. If the genetic algorithm was able to select good parameter sets then it would be expected that the solutions they find would be more optimal than the ones in the above results. The results show each search algorithm, and what test function an improvement of at least one magnitude was achieved in the found results. These results had to be present in the majority of tests to be included.

Starfish 17 Nodes	Starfish 5 Nodes	Particle Swarm Optimization	Firefly Algorithm
Booth	Branin	Bohachevsky	Bohachevsky
Branin	Griewank	Michalewics	
Griewank	Michalewics	Sum	
Perm	Perm		
Shubert	Shubert		
Spiral			

There was only one instance of the GA consistently producing worse results. This occurred when the FA was used with the Shubert function. This resulted in a loss of precision between between two and three magnitudes. These kinds of results were seen no where else in the tests.

11.1.4 Seventeen Nodes GAT

For the starfish with seventeen nodes, tests were run with populations of size 1, 2, 3 and 6 against populations for PSO and Firefly that used the same number of fitness function checks.

Starfish Population	PSO/Firefly Population	Starfish Better	Starfish Worse
1	10	Ackley Branin Rastrigin Shubert	Michalewics Perm Ridge
2	30	Ackley Easom Griewank Shubert	
3	50	Ackley	Perm
6	100	Easom	Rastrigin

In the above section it was noted on what tests the starfish and other algorithms did noticeably better across all population sizes. In these tests, no consensus could be reached. The data for the seventeen node starfish showed no trends that were strong enough to be reported on.

11.1.5 Five Nodes GAT

For the starfish with five nodes, tests were run with populations of size 2, 4, 10 and 20 against populations for PSO and Firefly that used the same number of fitness function checks.

Starfish Population	PSO/Firefly Population	Starfish Better	Starfish Worse
2	10		
4	20	Michaelwics Schwefel	
10	50	Ackley Easom	
20	100	Ackley	

The Starfish Algorithm also consistently did better than PSO and Firefly, and consistently worse against them on other functions. This means that results were not dramatically different, but did tend to stay consistent in one direction or another.

Consistently Better	Consistently Worse
Griewank Shubert	

11.2 K Nearest Neighbors

Upon reviewing the results from KNN testing, it was noted that one of two outcomes had been achieved. Depending on which data set was used, the results were either dramatically worse compared to KNN or on average slightly better under given conditions. The suspected reason for these results will not be covered in this section.

Under very specific circumstances, the results show a trend in the data that leads to the Starfish Algorithm outperforming KNN by up to five percent in correct classification. The rate of occurrence of this performance difference was nearly fifty percent in some tests. This means that as each test set was run one hundred times, in some instances nearly fifty of those tests were outperforming KNN in correct classifications by up to five percent.

This favorable outcome only occurs under very specific circumstances and quickly disappears when these conditions are not met. When the number of polled neighbors, K, is very low, 2,4, the Starfish Algorithm begins to outperform the KNN algorithm. This trend is strengthened when the number of epochs that the Starfish Algorithm is allowed to execute over increases. Furthermore when the size of the classified data set increases, the Starfish Algorithm more consistently out performs KNN even at lower epochs and higher K values. To reiterate, when the K value is low the Starfish Algorithm can be seen to classify data more accurately, up to five percent, than KNN. This trend of performance is more readily noticeable when the number of epochs is increased, or the classified data set used is larger.

In the case where the Starfish Algorithm dramatically underperformed compared to the KNN algorithm the Iris data set was used. By looking at the difference in the correct classification rates, a large skew in favor of the KNN algorithm is clearly seen. In some cases this skewing is only about five percent more correct classifications for KNN. In others however this skew extends out to twenty percent. This means that there was up to twenty percent of the data was incorrectly classified in the Starfish Algorithm compared to the KNN algorithm. Additionally this classification error rate was not uncommon. When using the Iris data set, on average more than eighty percent of the tests performed favored using the KNN approach to the Starfish Algorithm. It is to be noted that the trend seen in the favorable outcomes, while not overcoming the generally poor results with this data set, is still noticeably present in the collected results.

11.3 K Means Clustering

After initial testing for the K Means Clustering application of the Starfish Algorithm, future testing was suspended indefinitely. Results from the initial testing procedures, which explored number of neighbors and number of epochs used, dissuaded the continuation of testing. Looking at the generated clusters and accuracy of those clusters in both the Starfish and K Means results showed a profound difference in the desirability of results. K Means clustering outperformed the Starfish Algorithm in every test, regardless of the parameters of those tests.

In every test case, K Means had more predictable, less overlapped and more accurately generated clusters than did the Starfish Algorithm. The Starfish Algorithm was rarely able to completely classify a cluster correctly. Even when this was achieved though it appeared to be more due to luck than the performance of the algorithm. K Means however was able to correctly classify at least one full cluster in many of the tests. Furthermore the average accuracy of the K Means algorithm was above sixty percent correct classification in nearly all tests. The Starfish Algorithm on the other hand was lucky to achieve an accuracy rating above fifty percent.

Upon visually inspecting the clustering groups generated by each algorithm it became clear why K Means outperformed the Starfish Algorithm. The clusters generated by K Means were well defined with sharp edges, little to know overlap and centers that were clearly within a cluster. The Starfish Algorithm had clusters that were poorly defined. Edges of these clusters seemed to have been poorly chosen and made little sense in connection to the correct answer. This meant that the center of these clusters were often in open areas of the search space rather than in the center of data points. This led to clusters that either spanned more than one correct cluster or did not completely cover a single cluster. Because of this behavior the Starfish Algorithm was unable to correctly classify point accurately and did not perform better than the K Means algorithm.

12 Conclusion

After reviewing the evidence gained from the testing section of this paper, certain conclusions can be drawn in support and refusal of the original hypothesis. Each section of experiment will be covered separately and be given its own individual conclusion. Once that is completed, the sum of the work will be looked at to determine whether the Starfish Algorithm has achieved its goal of higher accuracy by using gathered information more efficiently.

12.1 Search Space Optimization

The conclusions drawn from the results of the application of the Starfish Algorithm in search space optimization will be presented in the same way the results were. There will be two major categories, Author Testing and GAT, which will each have two subcategories, seventeen node and five node starfish. Each section will also draw conclusions about how effective each starfish configuration in each test was in finding optimal solutions.

12.1.1 Seventeen Nodes AT

The results of the seventeen node starfish were as the initial hypothesis had theorized. The algorithm was able to use the information it gathered from the fitness scores to make accurate judgments on where to likely find optimal solutions. On many of the tests run, the Starfish Algorithm was able to perform equally as well as both PSO or Firefly. On certain tests the algorithm actually performed quite a bit better than the other two, while in other circumstances it did not fair nearly as well. Specifically the Starfish Algorithm did better on tests like the Bohachevsky and Sum functions. These functions presented clear smooth curves that the algorithm was able to follow down towards a global optima. This is in support of the initial hypothesis, the algorithm was able to use information it gathered to find solutions that are more optimal than other methods. The use of LG calculations was able to help propel population agents to discover better solutions by consuming the same amount of information.

The areas this algorithm did not do well in included tests such as the Perm, D&P and G&P functions. These tests saw a noticeable drop in performance from the Starfish Algorithm under all conditions. The change in

population size and starting positions did little to alter performance. It is suspected that the shape and size of these functions had a large role to play in the function's performance. Each function had very large, very flat surfaces with a very low standard deviation between neighboring points. This means that an optimal solution would only be slightly different than its neighbor. In these circumstances the starfish agents were unable to calculate a reliable LG that would propel it towards the solution. The agents had to fall back to moving towards their best node which was also unable to accurately discover solutions under these circumstances.

Overall the trends for these tests show that when given a generally smooth continuous curve the Starfish Algorithm is likely to outperform other methods such as PSO and Firefly by a significant margin. This is due to the ability of starfish agents to calculate and follow local gradients when searching for solutions. However there are certain functions that have been shown to be difficult for the algorithm to solve. In cases where large, flat planes hide a global solution the agents are unable to effectively move towards optimal solutions. These findings support the notion set out by the hypothesis and sets the conditions for which the Starfish Algorithm prevails over other methods.

12.1.2 Five Nodes AT

The results of the tests with five node starfish showed that these starfish actually did better than the starfish with seventeen nodes. After the experiments were completed and the results tabulated, the five node starfish more consistently produced a more optimal result it a wider variety of tests. This also means that the five node starfish performed better than both PSO and Firefly on more occasions. These starfish were able to increase the number of functions in which a more optimal solution was found than with traditional methods, as well as more consistently follow this trend from test to test. These results therefore conclude that not only do five node starfish perform better than seventeen node starfish, but that when given the choice the smaller of the two should be chosen due to its performance against conventional methods.

This trend of better performance follows the trends set forth by the seventeen node starfish. The Starfish Algorithm generated more optimal solutions in functions that had large, smooth curves as their main features. Functions such as the Matyas and Bohachevsky illustrate the types of problems that the algorithm does best with. The algorithm still failed in cases where large,

flat, uniform areas exist. Like the seventeen node tests, these tests also had trouble with functions like D&P, G&P and Perm. These failures happened less frequently and with less severity in these tests than the previous ones, showing that the five node starfish are more capable of dealing with these problems.

By decreasing the number of nodes in a starfish an increase in performance was noted. This increase is almost certainly due to the growth in the size of the populations used during these tests. Since the number of nodes contained in each starfish was lower, more starfish could be used in a population while maintaining the same number of fitness function calls made by conventional methods. This allows the population to explore a greater area and more possibilities than when the number of nodes is increased. As the number of nodes in the starfish get lower and lower, the algorithm begins to approximate the functionality of PSO. This is because as the starfish shrinks and has fewer nodes, it is less and less able to accurately calculate the LG. Without being able to give an accurate estimate of the LG, the attract behavior of the algorithm controls most of the movement for the population. When the algorithm is almost entirely controlled by this behavior it is nearly identical to PSO. However, it can be concluded from the results that by using a starfish of size five, enough of the functionality associated the LG calculations to gain an advantage over PSO and Firefly. This conclusion can be made based on the results generated by this test.

12.1.3 Seventeen Nodes GAT

The results from experimenting with genetic algorithms and seventeen node starfish showed a significant improvement in optimality of generated solutions. Each of the test algorithms saw this improvement in nearly every test function it was run against. PSO, Firefly and the Starfish Algorithm all delivered more optimal solutions than the tests with Author Testing. There were very few cases where the parameters selected by the genetic algorithm performed worse. None of these cases consistently occurred though and were not outlandishly worse. These nearly ubiquitous results show that using an algorithm to tune search algorithms can lead to much better results at the cost of computing time.

As in the experiments with Author Testing, the Starfish algorithm performed best when test functions had relatively smooth continuous curves. When compared to PSO and FA, the genetic algorithm parameter selection

actually aided the Starfish Algorithm in finding more optimal solutions. This means that in these tests the Starfish Algorithm did better on more test functions than in the test with Author Testing. Specifically it was seen that test like Griewank and Shubert had more optimal solutions found for them by the Starfish Algorithm in GAT than in the same test using Author Testing. The Starfish Algorithm also seemed to continue to preform poorly against PSO and FA in tests involving large flat areas. This shows that while the genetic algorithm did a significant amount of work tuning the function for each test, even it could not make enough changes to overcome this limitation.

With these results it can be concluded that by using a genetic algorithm to fine tune algorithms it can be possible to see a significant increase in the quality of their results. This test shows this principle at work through each search algorithm. Each algorithm generated much better results than in previous tests, with very few exceptions. Furthermore it has been shown that by tuning the Starfish algorithm, further improvements over traditional methods can be seen. The Starfish Algorithm can be seen to be the better choice for optimal solution generation in a wider variety of tests, while also mitigating poor performance in more difficult test functions.

12.1.4 Five Nodes GAT

Like its seventeen node counterpart, the five node starfish saw a similar trend of improved results during GAT. In all of the tests, improvements were made and more optimal solutions were found as compared to test with Author Testing. These improvements in solution generation did not however lead to an improvement against PSO and Firefly. In fact the opposite happened. Unlike in the seventeen node tests, on average these five node starfish did worse against PSO and Firefly than in tests with Author Testing. It can be seen in the results that during GAT, five node starfish outperformed PSO and Firefly less often. The results show this with fewer entries in the Starfish Better column of the genetic algorithm tests than in the standard parameter tests. They also show that the Starfish algorithm was not beaten by conventional methods in any of the tests.

From these results for this test several conclusions can be drawn. The main things that can be learned is that while tuning can help to generate more desirable answers, it is by no means a guarantee of better performance against other methods. Here it has been shown that while the solutions generated by five node starfish are better than with Author Testing, the al-

gorithm actually performs worse than it did previously when compared to PSO and Firefly. This is possibly due to five node starfish acting much more like PSO than the seventeen node starfish. Due to their decreased number of nodes, these starfish act very similarly to PSO with the attraction behavior playing a major role in their interactions. When the genetic algorithm tries to tune these starfish it finds that modifying behaviors has little effect on the generated solutions. Therefore the members that perform the best are the ones that rely less on behaviors and more on the LG and attraction. This could have led to the results that were seen, the LG would lead the starfish to better solutions in certain cases, while also not allowing it to outperform conventional methods in others. Furthermore these results reinforce the notion that tuning algorithms can lead to much better results. The results also continue to give support to the Starfish algorithm being the best choice for solution generation with certain functions.

12.1.5 Speed Considerations

One attribute of the tests that was not empirically evaluated was the speed at which each algorithm ran compared to its Starfish counterpart. From an anecdotal perspective, in all cases the Starfish algorithm ran significantly slower than any competing algorithm. This is due to the amount of extra mathematic computations the Starfish algorithm performs. Due to its more intensive nature the Starfish algorithm took many hours more to finish the most complex of tests compared to conventional counterparts. These calculations are what make up the Starfish Algorithm's behaviors and movement functions. Many of these functions have complicated features that could be could be computed in a more economic fashion. By reevaluating the execution of these algorithms and the mathematics they involve, improvements could be made on the execution time. This was not the focus of these tests and attention was not specifically focused on the speed.

Due to these speed concerns one could raise the argument that conventional methods could use that extra computation time to run more epochs and generate more optimal solutions. This is indeed correct, conventional methods could use the extra time to attempt to find better solutions. Obtaining results for this argument was not a focus of these studies. This testing was specifically aimed at evaluating if gathered information could be used to discover more optimal solutions. Therefore not attempt has been made in answering this question and cannot be commented on.

12.1.6 General Conclusion

After reviewing all of the evidence from the tests preformed for search space optimization certain trends were noticed and reported on. These trends suggest that the approach taken by the Starfish Algorithm is potentially preferable under certain conditions. These trends support the hypothesis stated by this paper and tested for. This approach has shown to show promise in solving certain kinds of problems over other methods.

This paper has explored how the number of nodes in search agents affects the optimality of generated solutions. It was shown that using Author Testing agents with five nodes did better than agents with seventeen nodes when compared to PSO and Firefly. During testing with genetic algorithms, both agent sizes, nodes, made sizable improvements over their Author Testing counterparts. It was also noted that in GAT that while both sizes generated more accurate solutions, the five node starfish did not fare as well as seventeen node starfish compared to PSO and Firefly.

It has also been shown that genetic algorithms offer a way to optimize algorithms with complex variable interactions. Nearly every test showed significant improvement to their untuned counterparts. This process of optimizing algorithms for their task shows that there is an advantage to undertaking such processes when searching for optimal solutions. The evidence for these claims also shows that trends for these algorithms seem to remain intact after optimization. The Starfish Algorithm performed best under the same conditions before and after tuning occurred.

In conclusion, these results offers evidence supporting the use of the Starfish Algorithm's principles in search spaces that have smooth continuous regions. Furthermore evidence was gathered that indicated that the Starfish Algorithm could have advantages in search spaces with spiked regions i.e. Easom. Evidence was also seen that shows this approach does not fair well when a solutions lies in a low variance plane i.e. Perm. In the cases suggested trends have been seen that suggest the Starfish Algorithm's approach to solution generation has an advantage over basic versions of conventional methods such as Particle Swarm Optimization and Firefly Algorithm. These results support the hypothesis of this paper and offers encouraging evidence for future research.

12.2 K Nearest Neighbors

The use of the Starfish Algorithm in comparison to K Nearest Neighbors showed promising results. While the majority of the tests run were unquestionable in favor of KNN, a small subset were not. This small set of results were shown to have fallen within a specific range of test parameters. It was found that in instances where the value of k was close to 0, the Starfish Algorithm outperformed KNN in accurately classifying test data. While this increase in accuracy was small, between one and five percent, it consistently showed up in tests with varying parameters and across all test sets.

The reason for the described experimental results is hard to determine in this instance. The Iris data set was the worst performer in the test sets, with the Starfish Algorithm being soundly defeated in all test. It was also the smallest in terms of both number of instances and for the number of attributes. This meant that both algorithms were classifying against a smaller known set, and with fewer dimensional data. This trend can be seen in the other data sets, with larger more complex ones being on a more even footing with KNN.

The reason for the emerging performance pattern would seem to be to the way in which the Starfish Algorithm operates. It was designed with the idea to use information more efficiently in order to generate more accurate results. Where this pattern occurs is when KNN has the most trouble gathering information about its surrounding. In these instances the Starfish Algorithm is able to make better use out of the available information in order to more accurately classify data. This is supporting evidence for the hypothesis and a clear indication of when and how performance between these algorithms shift.

Due to these results it is suggested that the application of the Starfish Algorithm should be considered for addition testing. By tweaking and retooling the way the algorithm behaves during these tests it may be possible to increase the gap in accuracy between the two algorithms. It could be shown that the pattern holds for more data sets, or across a wider range of testing parameters. The findings of these experiments provides evidence that a pattern exists and that it may be a possible improvement.

12.3 K Means Clustering

The application of the Starfish Algorithm to cluster determination problems resulted in clear evidence. This evidence showed that the Starfish Algorithm was not, in its current state, suited to solve those kind of clustering problems. The evidence gathered showed that the rules and behaviors governing the algorithm led to poorly defined clusters, a large majority of overlap of clusters, and an inability to define edges to the clusters. K means performed better in all categories and measurements. Without a rewrite of the way the algorithm attempts to solve the problem, (fitness function, behaviors, finite state machine), it is unlikely that these results can be changed.

13 Future Work

Future evaluation of the effectiveness of the Starfish Algorithm could continue on many fronts. The ideas of the algorithm has only been explored and evaluated in a limited framework. The ways in which the algorithm could be changed or improved are nearly endless. These possibilities fall into three main categories, fitness functions, behaviors and applications. Continued evaluation of the different ways in which the Starfish Algorithm could be modified can build on the evidence gathered by this paper. Each category of additional research has the ability to be explored in depth to find if the how they can alter the results found in this paper. The next paragraphs offer suggestions on where the algorithm could be altered and improved in order to gain more desirable results.

The fitness function for the algorithm is the real key piece in the effectiveness of its execution. There are limitless ways in which fitness functions for certain applications could be written. The approaches thus far explored have been simplistic, using basic rules and calculations. Very few factors were taken into account for most of the fitness evaluations, but this can be readily changed in future explorations. Properties and information about other starfish, solution to boundary proximity, previous movements and candidate solutions, etc. were not explored during this research but could be used to evaluate fitness. These additional factors to consider during the fitness evaluation could help starfish to more readily evaluate how well a candidate solves a problem as well as deciding where to evaluate next. Furthermore higher lever mathematics such as primary component analysis or calculus could be used in an attempt to gain further insight into the problem space. Primary component analysis could be dynamically used during fitness calculations to find which dimensions of the data have impacted the fitness the most. This could then be used to help the starfish move across those dimensions for solution discovery. Calculus could be integrated into the calculations to aid in the accurately calculating the LG. There is a large opportunity to explore how the fitness function impacts solution finding and there are many factors that can be modified from the current process to generate better results.

Another area that can be explored further is the behaviors each starfish uses to aid solution discovery. Behaviors can execute a wide range of functionality, limited only by the programmer writing them. The user is able to decide when and how the behaviors are used during execution of the search algorithm. The combinations for behavior functionality and how they are

used in solution finding are nearly endless. Behaviors can be developed for very specific problems encountered during testing and used to overcome those problems. These behaviors could be used to change the way in which starfish interact with each other, how nodes of the starfish move about the search area, the way splitting and competition occurs or how multidimensional problem spaces are explored. This is a great area for future work due to the lack of restrictions on what behaviors can do and how they can be used.

Finally the application areas for the Starfish Algorithm is an area that deserves further exploration. In this paper only three types of problems were explored and evaluated. In truth any area that PSO, FA, KNN or K means have been applied to, the Starfish Algorithm could be structured to be suitable for as well. Several examples were given in the prior work section of this paper which would also be applicable to the Starfish Algorithm. Problems like genetic algorithms, where the location of subsequent candidates are not readily known, could benefit from the use of local gradients for solution discovery. PSO has been used in training neural networks and their weighted connections. This would also be an appropriate are to attempt to apply the Starfish Algorithm and determine if it is able to better learn input output pairings and efficiently train the network. The application of the Starfish Algorithm to these areas of research and many others it not known. Exploration of these areas with the algorithm will help to determine the practicality of its use for further research. It will also provide direct evidence for evaluating the performance of this algorithm in different problem sets. This will show the varied kinds of problems the algorithm is suited for and which ones it is not.

14 Bibliography

[1] J. Kennedy, R. Eberhart, "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks, Vol 4. pp. 1942-1948. 1995.

[2] Y. Dai, L. Liu, Y. Li, "An Intelligent Parameter Selection Method for Particle Swarm Optimization Algorithm," Fourth International Joint Conference on Computational Sciences and Optimization, Vol. , no. , , 2011.

[3] Y. Shi, R. Eberhart. , "A Modified Particle Swarm Optimizer," International Conference on Evolutionary Computation , Vol. , no. , , 1998.

[4] X. Yang, "Firefly algorithm, stochastic test functions and design optimization," International Journal of Bio-inspired Computation, Vol. 2, no. 2, 78-84, 2010.

[5] S. Lukasik, S. Zak. , "Firefly Algorithm for Continuous Constrained Optimization Tasks," 10.1007/978-3-642-04441-08, Vol. , no. , 97-106, 2009.

[6] N. Chai-ead, P. Aungkulanon, P. Luangpaiboon, "Bees and firefly algorithms for noisy non-linear optimization problems," Prof. Int. Multiconference of Engineers and Computer Scientists, Vol. 2 pp. 1449-1454. 2011

[7] F. V. Bergh, An Analysis of Particle Swarm Optimizers, Ph.D. Dissertation. University of Pretoria Pretoria , South Africa, South Africa, 2002.

[8] J. Laaksonen, E. Oja. , "Classification with learning k-nearest neighbors," International Symposium on Neural Networks, Vol. , no. , , 1996.

[9] T. Fomby, K-Nearest Neighbors Algorithm: Prediction and Classification, 2008.

[10] K. Beyer, "When is nearest neighbor meaningful?,"Database TheoryICDT99, Vol. , no. , 217-235, 1999.

[11] J. Macqueen, "Some methods for classification and analysis of multivariate observations,", Vol. , no. , , 1967.

[12] K. Wagstaff, C. Cardie, S. Rogers, "Constrained K-means Clustering with Background Knowledge," International Conference on Machine Learning, Vol. , no. , 577-584, 2001.

[13] G. Hamerly, C. Elkan, "Alternatives to the k-means algorithm that find better clusterings," International Conference on Information and Knowledge Management, Vol. , no. , 600-607, 2002.

[14] M. Omran, A. Engelbrecht, A. Salman, "A Color Image Quantization Algorithm Based on Particle Swarm Optimization," Informatica (slovenia), Vol. 29, no. 3, 261-270, 2005.

[15] S. Kirkpatrick, M. Vecchi, C. Gelatt, "Optimization by Simulated Annealing," IBM Germany Scientific Symposium Series, Vol. 220, 4598, no. 4598, 671-680, 1983.

[16] X. Hu, R. Shonkwiler, M. Spruill, "Random restarts in global optimization," , Vol. , no. , , 2009.

[17] V. Estivill-castro, "Why so many clustering algorithms: a position paper," ACM SIGKDD Explorations Newsletter, Vol. 4, no. 1, 65-75, 2002.

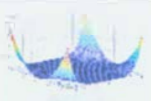
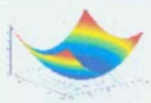
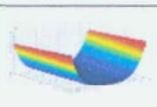
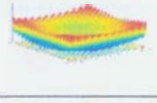
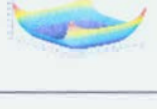
[18] Y. Jiang, Z. Zhou, "Editing training data for kNN classifiers with neural network ensemble," Advances in Neural NetworksISNN, Vol. , no. , 356-361, 2004.

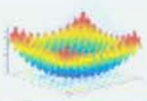
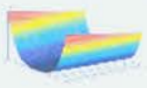
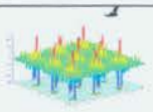
[19] A. Ultsch, L. Herrmann , "Self Organized Swarms for cluster preserving Projections of high-dimensional Data," , Vol. 27, no. , , 2010.

[20] S. Brin, "Dynamic Data Mining: Exploring Large Rule Spaces by Sampling," , Vol. , no. , , 1998.

[21] M. Mitchell, An Introduction to Genetic Algorithms, 5th ed., Cambridge: A Bradford Book The MIT Press, 1999, p. 7 - 10.

15 Appendix A: Optimization Problem Set

Function Name	Image	X Bounds	Y Bounds	Abs. Min
Ackley		-30 → 30	-30 → 30	0
Beale		-4.5 → 4.5	-4.5 → 4.5	0
Bohachevsky		-100 → 100	-100 → 100	0
Booth		-10 → 10	-10 → 10	0
Branin		-5 → 10	0 → 25	0.4
Dixon & Price		-10 → 10	-10 → 10	0
Eason		-30 → 30	-30 → 30	-1
Goldstein & Price		-2 → 2	-2 → 2	0
Griewank		-60 → 60	-60 → 60	0
Hump		-5 → 5	-5 → 5	0

Matyas		-10 → 10	-10 → 10	0
Michalewics		0 → 3.1415	0 → 3.1415	-1.8
Perm		-2 → 2	-2 → 2	0
Rastrigin		-5.12 → 5.12	-5.12 → 5.12	0
Ridge		-5 → 5	-5 → 5	-5
Rosenbrock		-10 → 10	-10 → 10	0
Schwefel		-500 → 500	-500 → 500	0
Shubert		-10 → 10	-10 → 10	-186.73
Sphere		-5.12 → 5.12	-5.12 → 5.12	0
Spiral		-10 → 10	-10 → 10	-0.98

Sum		-10 → 10	-10 → 10	0
Trid		-4 → 4	-4 → 4	-10
Zakharov		-10 → 10	-10 → 10	0

16 Appendix B: KNN Problem Set

Liver - archive.ics.uci.edu/ml/datasets/Liver+Disorders
Breast - archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+
Wine - archive.ics.uci.edu/ml/datasets/Wine
Iris - archive.ics.uci.edu/ml/datasets/Iris

17 Appendix C: Genetic Algorithms

Genetic algorithms are used in computational models to search for optimal solutions in vast search spaces.[21] While they are used by computer scientists, the algorithm finds its inspiration in genetics and natural selection. These processes in nature select which organisms will survive and be given the chance to reproduce. The act of reproduction allows surviving population members to pass on their genetic material to the next generation, which preserves its characteristics and traits.[21] In this way population members will pass on more desirable traits, while less desirable traits are removed from the population. In this way it is hoped that the population becomes more and more capable of preforming the tasks set to it and is the basis of evolution.When these ideas are applied to optimization algorithms, a program can be created that mimics natural selection. These programs generally have four main components, populations of chromosomes, fitness selection, crossover, and random mutation.[21]

A population of of chromosomes represents candidate solutions to the problem being solved. Each chromosome of a population member represents some facet of the problem being solved.[21] As an example imagine trying to follow directions to get to a store. Your directions consist of a list of turns to make or skip, i.e. Left, Right, Straight. By randomly combining some number of directions you would create a population member where each direction represents a chromosome. Trying to write directions like this is not very practical though, because there is only a slim chance that a random combination of directions will lead you to your destination. This leads to the next part of genetic algorithms, the fitness selection.

Fitness selection simply means that population members that complete the task the best are retained for subsequent evaluation, while all others are discarded.[21] In the example about directions, some metric would have to be used to determine which members were better. Therefore we will say that directions are considered more fit the closer they bring you to your destination. Since the task is to arrive at the destination this seems to be an appropriate metric. By then evaluation all the members of the population in this way, they can be ranked from most fit to least fit.[21]

After some kind of ranking is performed, a set number of unfit individuals can be discarded.[21] In order to maintain the size of the population they are then replaced through a process called crossover. Crossover comes in many different flavors so only the general concept will be discussed. To give some

insight into this process, consider when two organisms mate to produce an offspring. The offspring inherits characteristics from both of its parents and more often than not acts in similar manners to them. Two zebras mating will produce another zebra child, with four hooves, stripes and a tail, but their offspring will never be a whale. In this way offspring maintain the characteristics that made their parents so successful.[21]

The final process acting on these population members is called random mutation. When a new offspring is created it inherits characteristics from its parents, but it is not often advantageous to have offspring only have these characteristics. In the directions example imagine two parents the consist of only Right and Straight direction chromosomes. Through the crossover process their children will only ever have Right and Straight direction chromosomes as well. The process of random mutation selects some of these chromosomes and changes them to other directions, such as Left. This means that while the parents only have two direction chromosomes, there is a chance that their offspring could have all three.[21] This is done to inject additional variety into the population in hopes of producing faster convergence at a low cost to the process.[21]

These four components are then repeated over and over again for a set number of epochs or until some criteria is met.[21] During each iteration each population member is evaluated, the top number of members are selected for crossover and their offspring are randomly mutated and replace less desirable members of the population. By repeating these processes many times it is hoped that the algorithm selects a set of chromosomes that completes the task in a most optimal way, or in the example gets you to the store.[21]