AUTOMATED OPTIMIZATION OF MULTISENSOR - MULTITARGET TRACKERS

AUTOMATED OPTIMIZATION OF MULTI SENSOR – MULTI TARGET TRACKERS

By

NIMRA IFTIKHAR, B.Sc. (Eng.)
National University of Science and Technology, Pakistan (2009)

A Thesis
Submitted To The Department Of Electrical & Computer Engineering
And The School Of Graduate Studies
Of Mcmaster University
In Partial Fulfilment Of The Requirements
For The Degree Of
Master Of Applied Science

MASTER OF APPLIED SCIENCE (2015)
(Electrical And Computer Engineering)

MCMASTER UNIVERSITY
Hamilton, Ontario, Canada

TITLE	Automated Optimization Of Multi Sensor – Multi Target Trackers For Multi-Target Tracking
AUTHOR	Nimra Iftikhar National University Of Science And Technology Pakistan (2009)
SUPERVISOR	Dr. Thia Kirubarajan
NUMBER OF PAGES	x, 80

Abstract

Almost every module or project needs to be optimized to get the best results and reduce costs. Multi-sensor, multi-input trackers require a huge number of parameters to run, which have an undefined or unknown to the output of the tracker. It becomes very difficult to manually initialize these parameters to get a good output and there was a need to automate the process of selecting the parameters, validating them and initialing the tracker. The optimizer built to cater for these issues uses heuristic genetic algorithms – Particle Swarm Optimization and Gravitational Search Algorithm to find the best solutions for the problem. The optimizer works with the help of a Parameter Evaluator (developed earlier) to study the output of the tracker and incorporate the multi objective (Pareto) aspect of the problem. The Optimizer can find solutions to any optimization problem if hooked to a corresponding evaluator or fitness function calculator. This feature makes the Optimizer not just another module to the tracker but an independent application that could be used for general purpose optimization solutions.

Dedicated to Farah Khala, who showered me with her relentless support and unconditional love my eccentricity notwithstanding.

Acknowledgements

Praise be to Allah, who gave me the strength to undertake this challenging assignment and granted me wisdom to complete it.

While the topic was interesting per se, I could not have completed this in the desired manner without the painstaking efforts and guidance of Dr Thia Kirubarajan. He exposed multiple dimensions of the problem statement and nudged me on to find an optimal solution.

I am equally indebted to Dr R. Tharmarasa who helped bridge the gap between the very high expectations for this project and my initial input. He was always forthcoming to steer me out of these bottlenecks, to meet the goals within the stipulated time frame.

I would also like to thank my fellow graduate students in the ECE department, in particular all my friends in labs 302 and 202 for their help and support. Further, my sincere thanks are due to the ECE department staff, especially Cheryl Gies.

My family especially Farah Khala was the scarlet thread for this humble accomplishment; without her emotional backup, this all would not have been possible.

Contents

Table of Figures

Chapter 1

INTRODUCTION

1.1 Evolutionary Algorithms

Optimization is the process of finding the input parameters that will give the best output of a required problem. Mathematical optimization refers to either minimization or maximization of a function. It is practiced in many professions as they strive to achieve the best outcomes of their problems. For example when a doctor prescribes a medication to a patient he has to balance multiple parameters to cure the illness. These parameters include speed of recovery, effectiveness of the medicine, comfort of the patient and side effects from the medicine itself. The standard way of prescribing a dose is by the weight of the patient; medicines have a dosage factor which dictates the amount to be prescribed per Kg, so the total dose for a patient is the product of dosage factor and the total weight of the patient. In some cases where the patient's liver or renal system is compromised the doctors have to optimise their medication to cater their slow excretion and metabolism. If the dose is too high, it reaches a receptor saturation point and

the excess is not metabolised, which is either passed out as it is or leads to toxicity. Therefore it is vital to optimize the dose of the drug so that the patient recovers fast and the drug performs well without causing damage to any other system in the body. Similarly in manufacturing plants, optimization is carried out to increase productivity and reduce costs.

Some of these problems have a linear model and require optimization techniques that are simple and efficient; however in many cases the problem is not so simple. Most of these optimization problems have constraints that the parameters have to follow, which increases the complexity of the problem. If the problem cannot be written as a function of the parameters it becomes even more difficult to optimize it. Normally there are a number of possible solutions to a problem in a search algorithm, and finding the best solution in a limited time is challenging. If the number of valid solutions is small then less time is required to find the optimal solution, but if the search space is large (as is the case in many real world problems) searching for an optimal solution becomes difficult and may not always be discovered by regular methods if the resources are limited. Evolutionary Computation is efficient to a great extent in such cases. This thesis discusses optimizing such problems using algorithms from Evolutionary Computation.

In tracking a target, a good tracker-system and tracking algorithm do not essentially guarantee a good output. Initializing the tracker and choosing the best algorithm and its supporting variables for the given scenario are also essential. There are guidelines and theories relating to choosing these values but there is no specific law that will guarantee an optimal solution. Therefore there is a need to work on optimizing these values for the tracker under a specific scenario. Since the number of parameters for initialization is quite big and their ranges combined give an immensely large search space to find optimum values in, one has to look for techniques other than exhaustive searches.

The tracker system in place needs to be setup for each specific scenario and target. The system is initialized by giving all details of the scenario, targets, sensors and tracking algorithm in multiple files. The tracker-system then begins tracking after initialization generates an evaluation report on the performance of the tracker. In most situations one has no control over scenario and sensors, and the only way to get the best results is to choose a suitable tracker and optimal parameters of initialization. Up till now the initialization process was carried out manually by choosing the parameter values by trial and error and a high level of skill was required to tune

these parameters, creating an urgent need to automate this process and make it faster, simpler and more efficient. Since the optimization of this tracker is a multi-dimensional, multi-objective problem with no discernible relations between inputs and outputs, very specific algorithms were required to deal with it. The tracker optimization is a complex problem where the function to optimize and its derivatives are unknown.

This application automates the process of tuning the initialization of the tracker. The optimization of this tracker is a multi-dimensional, multi-objective problem; i.e. there are multiple variables that are to be tuned and multiple objectives to be achieved. The variables are tuned using the Gravitational Search Algorithm (GSA) and Particle Swarm Optimization (PSO), which are optimized based on a list of objectives given by the user, with each objective being given a certain weight to calculate the total 'fitness' of the solution.

Two flavours of swarm based algorithms have been implemented in the application to solve this multi-dimension, multi-objective problem. The optimization techniques treat the tracker as an unknown function and then modify the inputs according to their algorithm to optimize the output. These algorithms include parallel searches from multiple starting points (solutions), where each point evaluates itself and then according to the algorithm explores and exploits the search area using the information provided by other points and the area surrounding its own position, respectively. These algorithms are computationally expensive but since this optimizer would be run once before starting the tracker and is not used during the run time, the expense of time can be sustained. Similar sequential methods are also available which are faster but they do not provide the efficiency one can get with parallel search algorithms , and since speed is not a critical factor the selected methods provide better results. This application will save a significant amount of time and effort by automating the initialization and optimizing.

Many deterministic optimization algorithms require gradient information; however due to the lack of such gradients, it was necessary to look for another approach. The Particle Swarm Optimization and the Gravitational Search Algorithms are population based stochastic optimization techniques which do not require gradient information derived from the error function. This makes them useful to optimize problems where these gradients are unknown or not easy to obtain, as is the case with this tracker optimization problem.

The PSO algorithm was initially introduced by Kennedy and Eberhart [1] as a means to simulate the social behavior of birds and fish in swarms. They then discovered that the algorithm had an optimizing behaviour and could be used for solving optimization problems. This method follows the principal that when birds (agents) are looking for an optimal location they start by scattering into the search space randomly and then communicate their positions and the fitness of their locations to the rest of the swarm. These agents then move around in the search space looking for better locations, based on their own best findings and the best location discovered yet by the entire group.

The Gravitational Search Algorithm follows a similar principal as the PSO. In this method the agents are considered as masses, following Newton's law, that every point mass attracts every other mass with a force proportional to the product of their masses and inversely proportional to the square of the distance between them. The mass of each agent is calculated based on its fitness.

Regular optimization modules are designed to achieve a single objective, but since the quality of tracking is decided by multiple parameters, a multi-objective optimizer was required. For multi-objective optimization the ETFLab's Performance Evaluator (PE) was used, which takes in track information from the tracker and evaluates its performance by calculating performance metrics of the tracker. The Optimizer uses the output of the PE to calculate the total score of the given tracking instance and then incorporates the information for its next iteration. The optimizer can be customized to improving certain performance metrics only and changing the weight of each metric.

1.2 Organization of the Thesis

The thesis is divided into eight chapters; here is the organization of the thesis. Chapter 2 introduces Evolutionary Computation, with a focus on Evolutionary Algorithms and their applications. Chapter 3 discusses Target Tracking and the types of tracking algorithms. Chapter 4 explains the Performance Evaluator used for the Optimizer and the performance metrics it supports. The main Optimizer module and its workings are explained in Chapter 5, it also explains how to use the Optimizer. The optimization techniques (Particle Swarm Optimization and Gravitational Search Algorithm) used in the program are described in Chapter 6. The results are contained in Chapter 7 while the Summary and future work are covered in Chapter 8.

Chapter 2

EVOLUTIONARY COMPUTATION (EC)

Evolutionary Computation is a field of artificial intelligence based on Darwinian principles, since evolution is also an optimization process that aims to improve the ability of an organism to survive a continuously changing environment, the phenomena of natural selection, recombination, reproduction and mutation. EC uses algorithms inspired by biological evolution to solve mathematical optimization problems. The theory of 'Natural Selection' or 'Survival of the Fittest' suggests that all organisms that exist today are a result of continuous adaptation and evolution over the ages. The fitness of an organism suggests how well it has adapted to its environment; the organisms that survive are the ones that are able to adapt to, and bear the environment. The individuals that are more fit have a bigger chance to live longer and reproduce more, thus propagating their genotype to future generations. These processes of finding ways to survive better are the basis of algorithms used in Evolutionary Computation. [2]

Evolutionary Computation has two main branches:

2.1 Evolutionary Algorithms

Evolutionary Computation has a subset of algorithms called Evolutionary Algorithms, which are inspired by biological evolution such as reproduction, mutation, recombination and selection. These algorithms are generic population based metaheuristic and stochastic processes. All the algorithms in this field follow the same theme where candidate solutions for the optimization problem play the role of individuals or agents in a population and the quality of a solution is calculated by a 'fitness' function. These individuals evolve according to some criterion in an iterative process until a given satisfaction or limit is approached. [3]

Evolutionary Algorithms provide good results in almost all types of problems as they ideally do not make any assumptions about the function to optimize. Therefore they have been successful in a variety of fields such as biology, genetics, engineering, economics, chemistry, physics etc.

2.1.1 General Algorithm of Evolutionary Processes

The general steps of the algorithm are as follows:

1. Generate a random initial population (candidate solutions) of the swarm. These will be the first generation of individuals.
2. Calculate the fitness of all the individuals in the population
3. Repeat the following steps until a stopping criterion is met. The stopping criterion could be based on time, quality of solution etc.
    a. Select the best individuals (solutions with the best fitness values) for reproduction – these would be the parents of the next generation
    b. Breed new individuals through crossover and mutation to get the next generation
    c. Evaluate the fitness of the new offspring
    d. Replace the least fit population with new individuals

In some algorithms the least fit populations are used for mutation and not replaced.

2.1.2 Types of Evolutionary Algorithms

The different types of evolutionary algorithms are given below. All these techniques follow the same evolutionary steps differing in only the way the next generation evolves.

- Genetics Expressing Programming [4]

- Genetic Algorithm [5]
- Evolutionary Programming [6]
- Evolutionary Strategy [7]
- Differential Evolution [8]
- Differential Search Algorithm [9]

2.2 Swarm Intelligence

Swarm Intelligence (SI) deals with the behaviour of unsophisticated individuals in a self organized decentralized system, interacting locally with each other and the environment. The individuals in SI while working without a centralized control or global model cause coherent functional global patterns to emerge. These patterns of coordination without control can be used for problem solving in numerous fields [10]. Swarm Intelligence can be defined as:

"Swarm Intelligence is a property of systems of non-intelligent robots exhibiting collectively intelligent behavior." [11]

Swarms are behaviour based systems with algorithms inspired by the social behaviour of insects like ants and bees. These insects are simple individuals but have intelligent group behaviour. One of the books on Swarm Intelligence defines it as phenomenon that emerges from the behaviour of rule based particles. [12]

2.2.1 Swarm Intelligence Models

Swarm Intelligence Models are the types of SI algorithms that are used for different kinds of problem. The following are the few models inspired by the swarm systems in nature:

- Particle Swarm Optimization [1]
- Gravitational Search Algorithm [13]
- Ant Colony Optimization [14]
- Artificial Bee Colony Algorithm [15]
- Bacterial Colony Optimization [16]
- Differential Evolution [8]
- Artificial Immune Systems [17]
- Grey Wolf Optimizer [18]

- Bat Algorithm [19]
- Altruism Algorithm [20]

Some of the popular algorithms have been explained below.

Particle Swarm Optimization

Particle Swarm Optimization [21] [22] [1] is an optimization model which is evaluated by representing candidate solutions as particles in n-dimensional space and finding the fittest point in space. This model has been successfully applied in thesis for optimizing the tracker output where each parameter for optimization is modeled as a dimension in the search space. This model is discussed in detail in a later chapter.

Ant Colony Optimization

Ant Colony Optimization [23] [14] was first developed by Marco Dorigo and his colleagues for solving combinatorial optimization problems in the early 90's. This technique is inspired by the search algorithm used in an ant colony to find the shortest path in a graph. In an ant colony, ants randomly explore the search space for food, when they find a source they return to the colony leaving trails of pheromones on their path. When other ants find such a path they tend to follow the trail and reinforce it if they eventually find food. These trails of pheromones evaporate with time and reduce the attractiveness of the path. Ants take more time following a longer path to home, giving the pheromones more time to evaporate. For a relatively short path, the ants get to make more trips on it in the same time as the long path, and they keep on adding their trails resulting in a higher density of pheromones on the path thus attracting more ants. The evaporation of pheromones saves the colony from converging to a local optima.

Gravitational Search Algorithm

The Gravitational Search Algorithm [13] is based on the Newtonian Law of universal gravitation, which states that any two bodies in the universe attract each other with a force that is directly proportional to the product of their masses and inversely proportional to the square of distance between them. GSA is an optimization technique that considers candidate solutions (agents) as bodies in an isolated system, where the mass of each agent represents the fitness of the solution and the position of the agent represents the solution of the problem in n-dimensional space. Agents with good fitness values have higher masses and they attract other agents toward

themselves while agents with weaker masses gravitate towards the heavier agents. This algorithm is also applied in the optimization of the tracker and is discussed in detail in a later chapter.

2.2.2 Swarm Intelligence Principles

In 2011 Alex Kutsenok developed a 'General purpose swarm intelligence technique' [24] to help engineers design new SI algorithms for their specific problems. He has identified and outlined the basic principles of modeling a new algorithm; this has also helped in classifying algorithms in Swarm Intelligence. Following are the basic principles:

Principle 1: Create a system of agents that work individually on a common problem

The first principle of Swarm Intelligence states that a multi-agent system is used to provide a solution to the problem. The problem is then disintegrated into parts that are given to each independent agent in the system. Agents cannot be directly controlled by any other agent or central component and cannot be allowed to distribute tasks. The nature of the problem helps in dividing the problem in a logical way. Individual agents work on the tasks (sub-problems) assigned to them thus contributing to the global solution.

Principle 2: Agents are simple, fast, and have a limited perspective

In Swarm AI, agents are given a limited or local perspective of their environment. The less information an agent is given, the simpler it is to design that agent and the faster it will carry out the given task to reach a decision. For a good design the aim is to maximize the speed of an agent to perform the simple tasks assigned to it without worrying about the big picture. Agents do not think ahead or use heuristics to make decisions that take a significant amount of time.

Principle 3: Indirect Simple Inter-Agent Communication

Since the first two rules imply that agents are limited in their functionality because they focus on local data instead of the big picture, the third rule provides a way for the system as a whole to act globally. The final requirement of the system is that agents communicate with each other in a simple indirect method. These methods have to be simple as agents cannot interpret complex messages. One way of communication is the 'stigmergy' method in which agents communicate through the environment [24]. This happens when an agent changes the environment so that some other agent notices the change and alters its behaviour accordingly. Thus an indirect

communication is carried out by affecting the environment. The other type of communication is when an agent changes its own state like position or velocity so that other agents might notice and change their behaviour. Direct communication is not allowed in SI algorithms as they tend to make the agents and the system more complex and also slow down the system.

2.2.3 Swarm Intelligence Applications

The following are a few of the applications of swarm intelligence as researched by Kevin Roebuck in his book 'User generated content' [25]

- The U.S. military is working on Swarm Techniques for controlling unmanned vehicles
- The European Space Agency is considering to use an orbital swarm for self-assembly and interferometry.
- Its use is also being considered by NASA for planetary mapping.
- Swarm Intelligence is used to select location for transmission infrastructure for wireless communication networks for maximum coverage.
- M. Anthony Lewis and George A. Bekey in their paper proposed using swarm intelligent nano-robots inside the body to kill cancer tumours [26].
- Stochastic Diffusion Search was used by Al-Rifaie and Aber to locate tumours inside the body. [27] [28]

Chapter 3

TARGET TRACKING

In various field applications one faces a problem of locating and/or estimating the state of an object in an unknown environment. When the information available is not enough to exactly calculate the state of the object of interest (target), estimation and probability are used to track the target. Estimation is the process of calculating a parameter from indirect, inaccurate and uncertain observations; thus tracking can be described as the estimation of the current or future state of a dynamic system from uncertain and noisy data. It is defined as:

"Target tracking is the estimation of the current state and prediction of the future states of a target based on measurement received from a sensor that is observing it." [29]

Blackman and Popoli have traced the following essential steps for tracking a target [29]

1. Collect sensor data from the search space containing the target(s).

2. Partition the sensor data into sets of observations or tracks produced by the same sources, eliminating the background and reducing false targets.
3. Estimate the number of targets and their kinematics such as future position, velocity etc.

Tracking is becoming increasingly popular in a wide range of applications. In medicine it can be used to track abnormal cells or bacteria in blood flow. In video or image processing applications target tracking is used to track vehicles on the road or people in an environment. Militarily application include tracking ground vehicles, ships, submarines, airplanes, rockets etc.

There are several techniques available for estimation generally known as tracking filters. These filters get information from the sensors and predict the state of the target using some recursive estimation algorithm. Tracking filters are part of the Data Processing step in a general tracker system. The system can be divided into the following parts.

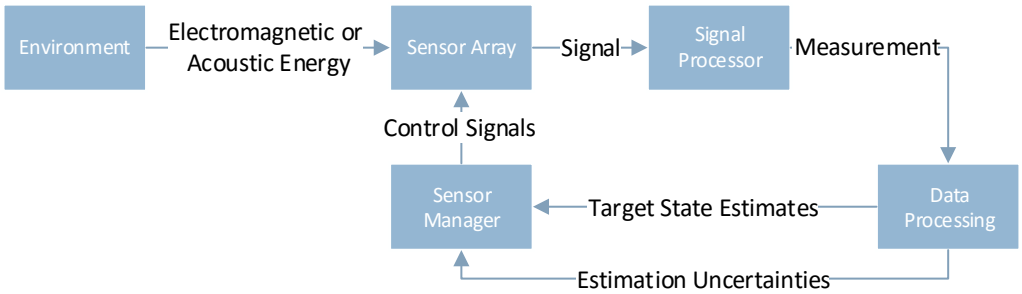

Figure 1 Tracker System

3.1 Tracking Filters

Filtering is the process of estimating the state of a dynamic target from noisy data. To analyze a target (dynamic system) at least two models are required. The book chapter Sensor Data Fusion with Application to Multi-target Tracking summaries the tracking filters as follows [30].

System Model: A system model describes the evolution of the state of the target with time

$$x(k+1) = f\big(k, x(k)\big) + v(k)$$

Measurement Model: A measurement model relates the noisy measurements to the state of the target

$$z(k) = h\big(k, x(k)\big) + \omega(k)$$

where f and h are functions (generally nonlinear), $x(k)$ is the state of the target, $z(k)$ is the measurement vector, $v(k)$ is the process noise and $\omega(k)$ is the measurement noise at measurement time k.

When using the Bayesian approach to estimate the state of the dynamic system the aim is to construct a posterior probability density function (pdf) of the state using all the measurements received till that time. If all the noises are white, the pdf is the complete solution for the estimation of the problem since it contains all the statistical information. In recursive filtering the received measurements are processed sequentially rather than as a batch, thus eliminating the need to store the complete measurement set or to process the existing measurements again if a new measurement becomes available. This kind of filtering has two stages: prediction and update [30].

Prediction: In the prediction stage the filter uses the system model to predict the state of pdf from one measurement time step to the next time step. If the pdf at time k is $p(x(k)|Z^k)$, where $Z^k = [z(1) \quad z(2) \quad \cdots \quad z(k)]$, this stage predicts the pdf at measurement $k+1$ as

$$p(x(k+1)|Z^k) = \int p\big(x(k+1)|x(k)\big) \cdot p(x(k)|Z^k)\ \mathrm{d}x(k)$$

Update: This stage updates the prior using Bayes' formula with the latest measurement $z(k+1)$

$$p(x(k+1)|Z^{k+1}) = \frac{p\big(z(k+1)\big|x(k+1)\big) \cdot p(x(k+1)|Z^k)}{p(z(k+1)|Z^k)}$$

In general the posterior cannot be determined analytically and this given recursive propagation is only a conceptual solution, which can only be applied in some limited cases.

3.1.1 Kalman Filter

The Kalman filter is an optimizing filter that predicts the future state of a dynamic system from inaccurate, indirect and uncertain measurements of the past states of the system. It is a recursive

process that estimates the internal state of the system by learning to differentiate between the noise and inaccuracies of the measurements and the truth. A Kalman filter is also known as an 'optimal estimator' or a 'linear quadratic estimator' [31] [32] [30].

The basic assumption of the Kalman filter is that the state and measurement models of the system are linear i.e. $f\big(k, x(k)\big) = F(k) \cdot x(k)$ where $h\big(k, x(k)\big) = H(k) \cdot x(k)$. It is also generally assumed that the initial state error and all the noises entering the system are Gaussian. The observation noise $v(k)$ is assumed to be white and Gaussian with covariance $Q(k)$ and zero mean while the process noise $\omega(k)$ is assumed to be white and Gaussian with zero mean and covariance $R(k)$. Considering these assumptions if $p(x(k)|Z^k)$ is Gaussian, it can be proved that $p(x(k+1)|Z^{k+1})$ is also Gaussian and can be parameterized by a mean covariance [33].

The algorithm recursively process the information for the state of the system in the following manner [30]:

$$\hat{x}(k+1|k) = F(k) \cdot \hat{x}(k|k)$$

$$P(k+1|k) = F(k) \cdot P(k|k) \cdot F(k)' + Q(k)$$

$$\hat{z}(k+1|k) = H(k+1) \cdot \hat{x}(k+1|k)$$

$$S(k+1) = H(k+1) \cdot P(k+1|k) \cdot H(k+1)' + R(k+1)$$

$$\hat{x}(k+1|k+1) = \hat{x}(k+1|k) + W(k+1)[z(k+1) - \hat{z}(k+1|k)]$$

$$P(k+1|k+1) = P(k+1|k) - W(k+1) \cdot S(k+1) \cdot W(k+1)'$$

where

$$W(k+1) = P(k+1|k) \cdot H(k+1)' \cdot S(k+1)^{-1}$$

If the above assumptions hold, the Kalman filter performs better than any other filter. Many different varieties of the filter have been developed for different problems.

3.1.2 Extended Kalman Filter (EKF)

Most real world problems are nonlinear but the simple Kalman Filter only caters for linear systems, therefore the Extended Kalman Filter was developed to estimate the state of nonlinear

systems. The EKF does not require the state transition and observation models to be linear but instead requires them to be differentiable. It defines the nonlinearity of a model by local linearization of the equations [30].

$$\hat{F}(k) = \frac{df\ (k)}{dx}\Big|_{x=\hat{x}(k|k)}$$

$$\hat{H}(k) = \frac{dh\ (k+1)}{dx}\Big|_{x=\hat{x}(k+1|k)}$$

In the EKF the $p(x(k)|Z^k)$ function is approximated by a Gaussian and then the Kalman Filter is used for the linearized functions, while the state and measurement predictions are calculated using the original nonlinear functions.

$$\hat{x}(k+1|k) = f\big(k, \hat{x}(k|k)\big)$$

$$\hat{z}(k+1|k) = h\big(k+1, \hat{x}(k+1|k)\big)$$

These equations are for a first order expansion of nonlinear systems, there are other EKFs for higher order systems but they are complex and do not perform as well as this one [30].

3.1.3 Probability Hypothesis Density (PHD) Method

When tracking multiple targets, target detection and data association are complicated by the uncertainties in the dynamics of maneuver and clutter, and it is a challenge to associate measurements with the targets that have generated them. The number of targets change also with time and therefore ordinary Bayesian statistics cannot be used to compare states. This problem can be solved using Finite Sate Statics (FISST) [34]. This algorithm gives good results but the problem is that its computational load increases exponentially with the number of targets making it computationally expensive when a large number of targets are present [30] [35].

The book chapter [30] explains the PHD and its calculation as a multi target filter that recursively estimates the number of targets and their states from given observations in an

environment where uncertainties, noise and false detections are present. The PHD is defined over the state space of one target instead of the full posterior distribution reducing the computational cost of propagation in time.

The Probability Hypothesis Density $D_{k|k}(x_k|Z^k)$ is the density which gives out the number of targets $N_{k|k}$ contained in a search space S when integrated over that space.

$$N_{k|k} = \int_S D_{k|k}(x_k|Z^k)\,\mathrm{d}x_k$$

The first order statistical moment of the full target posterior can be recovered from the PHD as

$$D_{k|k}(x_k|Z^k) = \int_{X_k \ni x_k} f_{k|k}(X_k|Z^k)\,\delta X_k$$

where X_k is the multi-target state. The 'maxima' of the PHD gives the approximate expected target states.

Prediction:

In a general multi target environment the number of targets changes due to target disappearances, target spawning and entry of new targets, therefore the predicted PHD incorporates these factors as:

$$D_{k|k}(x_k|Z_{1:k-1}) = \gamma_k(x_k) + \int \begin{bmatrix} e_{k|k-1}(x_{k-1}) f_{k|k-1}(x_k|x_{k-1}) \\ + b_{k|k-1}(x_k|x_{k-1}) \end{bmatrix} D_{k-1|k-1}(x_{k-1}|Z_{1:k-1})\,\mathrm{d}x_{k-1}$$

where $e_{k|k-1}(x_{k-1})$ is the probability that a target with state x_{k-1} at time $k-1$ will be alive at time (k), $b_{k|k-1}(x_k|x_{k-1})$ is the probability of a target with state x_{k-1} spawning at time k, $\gamma_k(x_k)$ is the probability of new born targets at time k and $f_{k|k-1}(x_k|x_{k-1})$ is the single target Markov transition density.

Update:

The predicted PHD is corrected with the available measurements Z_k at time k to get the updated PHD. It is assumed that the number of false alarms is Poisson distributed with average spatial density λ_k . The probability density of the spatial distribution of false alarms is given by $c_k(z_k)$, and the detection probability of a target with state x_k is given by $p_D(x_k)$. The updated PHD at time k is given by

$$D_{k|k}(x_k|\mathrm{Z}^k) \cong \left[\sum_{z_k \in Z^k} \frac{p_D(x_k) f_{k|k}(\mathrm{z}_k|x_k)}{\lambda c_k(z_k) + \psi_k(\mathrm{z}_k|Z_{1:k-1})} + \left(1 - p_D(x_k)\right)\right] D_{k|k-1}(x_k|Z_{1:k-1})$$

where $f_{k|k}(\mathrm{z}_k|x_k)$ denotes the single-sensor/single-target likelihood and $\psi(\cdot)$ is the likelihood operator given by

$$\psi_k(\mathrm{z}_k|Z_{1:k-1}) = \int p_D(x_k) f_{k|k}(\mathrm{z}_k|x_k)\, D_{k|k-1}(x_k|Z_{1:k-1})\, \mathrm{d}x_k$$

3.1.4 IMM Filters

A multiple model estimator is a filtering approach that realizes the different models of target manoeuvres. Targets may change their manoeuvres with time thus requiring a need to model and recognize all the manoeuvres inside a filter. In this approach a number of filters act in parallel for each target maneuver. The Interactive Multiple Model (IMM) [33] [36] [37] estimator uses a bank of different hypothetical target motions to estimate a dynamic system. This filter provides better results than a regular Kalman Filter, which works with a single model. The following is the algorithm for the IMM [30]:

Assumptions:

The base state model of IMM is given by:

$$x(k) = F[M(k)]x(k-1) + v[k-1, M(k)]$$

$$z(k) = H[M(k)]x(k) + w[k, M(k)]s$$

where $M(k)$ is the mode of the system at time (k) among the possible r modes

$$M(k) \in \{M_j\}_{j=1}^{r}$$

The structure of the system and statistics are different for every mode and are given by

$$F\left[M_j\right] = F_j$$

$$v\left(k-1, M_j\right) \sim \mathcal{N}(u_j, Q_j)$$

The Mode jump process is characterized as a Markov chain with known transition probabilities

$$P\left\{M(k) = M_j | M(k-1) = M_i\right\} = p_{ij}$$

Algorithm

The IMM filter has the following stages:

Interaction: At this stage the previous cycle's mode-conditioned state estimates and covariance are used to initialize the current cycle of each mode-conditioned filter using the mixing probabilities.

The estimate and covariance for each model ($\mathrm{j} = 1, \ldots, \mathrm{r}$) are given by

$$\hat{\mathrm{x}}^{0\mathrm{j}}(\mathrm{k}-1|\mathrm{k}-1) = \sum_{\mathrm{i}=1}^{\mathrm{r}} \hat{\mathrm{x}}^{\mathrm{i}}(\mathrm{k}-1|\mathrm{k}-1)\mu_{\mathrm{i}|\mathrm{j}}(\mathrm{k}-1|\mathrm{k}-1)$$

$$\mathrm{P}^{0\mathrm{j}}(\mathrm{k}-1|\mathrm{k}-1) = \sum_{\mathrm{i}=1}^{\mathrm{r}} \mu_{\mathrm{i}|\mathrm{j}}(\mathrm{k}-1|\mathrm{k}-1) \left\{ \begin{array}{c} \mathrm{P}^{\mathrm{i}}(\mathrm{k}-1|\mathrm{k}-1) \\ +\left[\hat{\mathrm{x}}^{\mathrm{i}}(\mathrm{k}-1|\mathrm{k}-1) - \hat{\mathrm{x}}^{0\mathrm{j}}(\mathrm{k}-1|\mathrm{k}-1)\right] \\ \cdot \left[\hat{\mathrm{x}}^{\mathrm{i}}(\mathrm{k}-1|\mathrm{k}-1) - \hat{\mathrm{x}}^{0\mathrm{j}}(\mathrm{k}-1|\mathrm{k}-1)\right]' \end{array} \right\}$$

Mode-conditioned filtering: At this stage the state estimates and covariances conditioned on the mode being in effect are calculated, and the likelihood function for the modes are evaluated. The Kalman Filter matched to each mode $\mathrm{M_j(k)}$ uses the measurements $\mathrm{z(k)}$ to calculate $\hat{\mathrm{x}}^{\mathrm{j}}(\mathrm{k}|\mathrm{k})$ and $\mathrm{P}^{\mathrm{j}}(\mathrm{k}|\mathrm{k})$. The likelihood function is calculated as

$$\Lambda_{\mathrm{j}}(\mathrm{k}) = \mathcal{N}\left[\mathrm{z(k)}; \hat{\mathrm{z}}^{\mathrm{j}}(\mathrm{k}|\mathrm{k}-1), \mathrm{S_j(k)}\right]$$

Probability evaluation: The mixing and update probabilities are calculated in this stage

Mixing probabilities for mode i and j ($\mathrm{i,j} = 1, \ldots, \mathrm{r}$)

$$\mu_{i|j}(k-1|k-1) = \frac{1}{\bar{c}_j} p_{ij}\mu_i(k-1)$$

where

$$\bar{c}_j \triangleq \sum_{i=1}^{r} p_{ij}\mu_i(k-1)$$

Mode probability for each mode $(j = 1, \ldots, r)$

$$\mu_j(k) = \frac{1}{c}\Lambda_j(k)\bar{c}_j$$

where

$$c \triangleq \sum_{j=1}^{r} \Lambda_j(k)\bar{c}_j$$

Overall state estimate and covariance: The model-conditioned estimates and covariances are combined at this stage.

$$\hat{x}(k|k) = \sum_{j=1}^{r} \hat{x}^i(k|k)\mu_j(k)$$

$$P(k|k) = \sum_{j=1}^{r} \mu_j(k)\left\{P^j(k|k) + \left[\hat{x}^j(k|k) - \hat{x}(k|k)\right] \cdot \left[\hat{x}^j(k|k) - \hat{x}(k|k)\right]'\right\}$$

Chapter 4

PERFORMANCE EVALUATOR

As there are several kinds of tracking algorithms available and many developing with time, it is necessary to evaluate the efficiency of each method on a standard scale, here referenced as Measures of Performance (MOP) or Metrics. The MOPs are generally defined according to the type of performance that has to be evaluated or the availability of input data.

MOPs can be related to the characteristics of the measurements of sensors or to the accuracy and quality of tracker estimates. Sensor related MOPs are independent of the tracking algorithm used. These might be helpful for evaluating scenarios with multiple measurement sensors but cannot be used to evaluate the type of tracker used. In most cases the main goal is to evaluate the tracking algorithm rather than the sensors. Consequently the tracker related evaluation consists of a large number of measures, which are divided into two classes: the measures specific to individual trackers and the measures that can be applied to any tracker.

There is a lot of literature on metrics defined to evaluate tracking algorithms. However there is no solid classification of MOPs that can be used for all tracking algorithm. A good classification of metrics is important and needed to measure the efficiency of trackers against each other. The evaluator used by the Optimizer is called the ETFLab- Performance Evaluator (PE) [38] and provides a sound and complete evaluation of the tracker. It divides the measures into sensor-related and tracker related metrics based on their dependency on the tracking data. It mainly focuses on the tracker-related metrics, which are further classified into tracker-related and independent metrics.

The ETFLab-PE takes inputs from the tracker and evaluates the track information using suitable MOPs. This PE is used to provide a good evaluation of the "Fitness" of the tracker at different parameters.

The ETFLab-PE provides a common ground to test the performance of different tacking algorithms, helping the user to compare and understand the quality of trackers.

The following are the classifications and types of performance metrics used by the evaluator.

4.1 Measurements

4.1.1 Sensor Related Measurements

When no tracking information is available, the physical characteristics of the system are used to evaluate the tracker, which are basically the sensor related metrics. The type and specifications of sensors are vital to a tracker system. It is necessary to weigh the aspects of all types of sensors when using them to track targets. The evaluator has metrics specific to the sensors used by the tracking system. The following measures are used to evaluate a sensor.

Dwell time per sensor

Dwell time is the time that an antenna beam spends on a target. A radar's performance can be evaluated based on the average time it spends to detect targets in a real scenario.

For a multi-sensor system with N_s sensors and data received in N_{MC} Monte Carlo runs, the metric is [39]

$$\tau_{m.k} = \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} \left[\frac{1}{N_{n,k}^{\mathcal{L}}} \sum_{i \in I_n[m,k-1,k]} \tau_{dwell,n}^{i,m} \right]$$

where $N_{n,k}^{\mathcal{L}}$ denotes the members of $\mathcal{L}$ that are available in the k-th time step and the n-th Monte Carlo run; $I_n[m, k-1, k]$ represents the set of dwell requests at time k and $\tau_{dwell,n}^{i,m}$ is the corresponding i-th dwell time in the n-th run.

Energy emitted per sensor

The average energy emitted per sensor is an important metric to evaluate the efficiency of a radar system. If $P_{n,m}$ is the peak power for the m-th sensor in the n-th run of a total of N_{MC} Monte Carlo runs and $\tau_{pw,n}^{i,m}$ is the pulse width at the n-th run then the energy can be calculated as [39]

$$E_{m.k} = \frac{1}{N_{MC}} \sum_{n=1}^{N_{MC}} \left[\frac{1}{N_{n,k}^{\mathcal{L}}} P_{n,m} \sum_{i \in I_n[m,k-1,k]} \tau_{pw,n}^{i,m} \right]$$

Cumulative transmit dwell time/emitted energy over all sensors

Apart from evaluating the each sensor separately on dwell time and energy emitted, the system can be evaluated based on the cumulative dwell time and energy emitted as [39].

$$\tau_k = \sum_m \tau_{m.k}$$

$$E_k = \sum_m E_{m.k}$$

4.1.2 Tracker Related Measurements

The tracker related metrics evaluate the performance of the tracker. These metrics include algorithm independent and algorithm dependant metrics.

Algorithm Independent

Algorithm independent metrics are the ones that can be used to evaluate the tracker for any tracking algorithm. These metrics are categorized based on the availability of truth and tracks.

- **Available truths and tracks**

When truths and tracks are available, the tracking results can be evaluated with the known truths. The metrics associated with this category can be classified according to the following measures:

Track Cardinality Measures: This class of metrics deals with the statistics of the results, e.g. number of confirmed tracks, number of missed tracks, number of false tracks etc. However these metrics do not provide any information about the performance or time characteristics of individual tracks such as the consistency of tracks and the accuracy of estimation.

Time (Durational) Accuracy Measures: This class of metrics deals with the time performance and persistence of the estimated tracks. They provide more useful information about the duration and persistence of the estimated tracks. Following is an example of time durational measure

Track Probability of detection: It is a measure of trackers detection ability in estimating truth.

Accuracy Measures: This is the most common measure used to evaluate the accuracy of the estimated values. Several measures are defined based on the difference between the estimation and truth, e.g.

Root Mean Squared Error: RMSE is a common metric which uses a Mahanabolis distance to compute the error.

- **Available tracks and unknown truths**

In real scenarios there is often no information available about the truths. In such cases the consistency of tracking results may be checked. Common statistical tests may be made on the information received from innovation, which is considered the main source of information.

Algorithm Specific

The evaluator also has metrics specific to the algorithm of the tracker, which include metrics defined for IMM filters, assignment based tracking algorithms, dynamic programming, MHT trackers and IPDA algorithm. The evaluator also has metrics specific to the application of the tracker e.g. tracking people, sensor networks and vision based tracking.

4.2 Output

The output of the evaluator is an xml file, which lists all the metrics with their respective values. The score of each metric is later calculated in the optimizer. Following is an example of the output.

```
<overall_score_wieghts>
  <root_mean_square_error>1.0</root_mean_square_error>
  <average_euclidean_error>0.0</average_euclidean_error>
  <median_error>0.0</median_error>
  <missed_track>10.0</missed_track>
</overall_score_wieghts>
```

Chapter 5

AUTOMATED TRACKER OPTIMIZER

The goal of the application is to fine tune (and initialize) a scenario specific tracker for optimal results. The tracker receives several variables for its initialization and therefore the aim is to provide the tracker with the best parameters for a specific scenario. The problem with optimization of the tracker is that most of the key variables do not have a defined relationship with the tracker output. In different scenarios the variables have different effects on the output; therefore it is very difficult to initialize the parameters manually or by some generalized formula, to get optimal results.

A mechanism to automatically initialize and optimize the tracker is presented here. The initialization part is done by the class ConfigManager, which is not discussed here in detail as it does not contribute to the optimization algorithm and is only a tool used to communicate with the tracker. The optimization is done by the class Optimizer.

The inputs of an optimizer are variables and their ranges, or specific values at which they should be tested. These are written in an xml file and the ranges are given in a MatLab range format.

The output of the optimizer is a report consisting of the top solutions, and if requested initialization files of these solutions are also created.

The optimizer works by first reading input files that provide information about the variable to optimize and what method to use for optimization. It then carries out the required optimization and saves the results in a Report file. Following steps are carried out:

5.1 Initialization (Input)

The optimizer has to be initialized before running. The initialization is the input part of the Optimizer. The optimizer takes the following inputs:

5.1.1 Evaluator Callback

The optimizer is given the pointer to the callback function, which calls the tracker and the evaluator, and returns the evaluator data.

5.1.2 ConfigManager

Depending on the type of initialization the pointer to the ConfigManager is required for saving and editing project files. If this is not given the optimizer constructs one for itself and hides it. Here is a brief description of the ConfigManager:

The tracker is a complex application, which requires numerous inputs to run. In the past it was difficult to initialize and setup the tracker for a scenario, especially for users not skilled in the algorithms running with the tracker and the specifications of the tracker, since the tracker takes in complex xml files as inputs. There was a need to simplify the initialization process and also check the given inputs for errors and compatibility issues, thus eliminating the need to manually write the file.

The ConfigManager is a class that has been specifically designed to initialize a tracker. It provides a graphical user interface (GUI) to input the initialization parameters then validates all the parameters by a previously defined guide (restrictions) and then generates an output xml containing the initialization parameters for the tracker. The input of the class is also an xml file

that provides a pseudo code to create the GUI, which also provides restrictions for the input variables.

The ConfigManager class is basically used to provide a user-friendly xml file writing application. The ConfigManager allows a user to load, edit and save xml files as well as check the values input by the user.

There are two types of files used with the ConfigManager: there is a Default file, which has all the details about the controls (variables) required to create the GUI and the Output file, which provides the output (values) of the GUI controls. The Default file is used independently, while the Output file needs the corresponding Default file to open.

The Default file is the main file that creates the GUI. All the elements of the GUI i.e. controls, groups, sections and subsections that form and shape the structure of the GUI are defined in this file. It also contains the guide (restrictions) on acceptable values of controls and other effects that elements in the GUI might have on themselves or other elements.

The Output file is the output of the Default file. It can be opened, modified and saved using the ConfigManager. This file contains all the values input by the user and is used to feed inputs to the tracker or any other application it is used with.

The ConfigManager opens an input file which can be the Default or the Output file and displays all the variables as controls in it. The user is allowed to modify the values of these controls according to the restrictions provided by the Default file. The values are then checked for constraints and saved in the output xml, which is then fed to the tracker. The ConfigManager, although specifically designed for tracker initialization, can also be used to write xml files for other applications.

The output of the ConfigManager is an xml file also known as the output file. This file contains the initialization information for the tracker.

5.1.3 Configuration File

The configuration file is an xml that gives the following information:

Method

Default method of optimization for the tracker e.g.

```
<Method>GravitationalSearch</Method>
```

Method Parameters

These are the default parameter values for PSO and GSA.

- **Particle Swarm Optimization**

The particle swarm needs the following parameters for initialization.

Max Runs: Maximum number of iterations

Samples: Population size

$\boldsymbol{\omega}$: Inertial weight; if the attribute 'Reduce' is yes, the value of ω will decay with time.

$\boldsymbol{\varphi_p}$: Cognitive factor

$\boldsymbol{\varphi_g}$: Social factor

```
<ParticleSwarm>
  <MaxRuns>350</MaxRuns>
  <Samples>25</Samples>
  <w Reduce="yes">1</w>
  <phiP>1</phiP>
  <phiG>2</phiG>
</ParticleSwarm>
```

- **Gravitational Search Algorithm**

The gravitational Search Algorithm needs the following parameters for initialization

Max Runs: Maximum number of iterations

Samples: Population size

$\boldsymbol{\alpha}$: Gravitational decay constant

$\boldsymbol{G_0}$: Gravitational Constant

```
<GravitationalSearch>
  <MaxRuns>350</MaxRuns>
  <Samples>25</Samples>
  <G0>100</G0>
  <Alpha>20</Alpha>
</GravitationalSearch>
```

Language File

This string gives the file path of the Language File that holds the message and error strings for user interaction.

```
<LanguageFile Path_Type="RelativeToSelf">
common/Optimizer.lang</LanguageFile>
```

Warnings

This flag indicates whether to enable or disable warnings.

```
<ShowWarnings>Yes</ShowWarnings>
```

Save Top Files

Flag to indicate whether to save the top files in the results folder or not

```
<SaveTopFiles>no</SaveTopFiles>
```

History Number

Variable indicating the number of fitness tests to keep in history to avoid running the test on the same solution again.

```
<HistoryNumber>1000</HistoryNumber>
```

Original Files

Information about handling the last saved (original files) of the project.

```
<OrignalFiles>
  <DeleteBackup>1</DeleteBackup>
  <ReplaceWithOptimized>1</ReplaceWithOptimized>
</OrignalFiles>
```

- **Delete Backup**:

This flag indicates whether to delete the backup of the original files at the end of optimization or not.

- **Replace With Optimized**:

This flag indicates whether to replace the original files with the optimized ones or not.

5.1.4 Project File

The project file is an xml that gives information about the project to optimize.

Settings

These are the settings for the optimizer

- **Method**

It is the method of optimization to use for the given project and the initialization parameters for the method.

```
<Method Type="GravitationalSearch">
  <GravitationalSearch>
    <MaxRuns>350</MaxRuns>
    <Samples>25</Samples>
    <G0>100</G0>
    <Alpha>20</Alpha>
  </GravitationalSearch>
</Method>
```

- **Input**

The project for the optimizer needs the following inputs to run:

TopN: It is the number of top solutions to provide at the end of optimization for example, if TopN is 10, the optimizer will provide with solutions with the top 10 scores.

Variables to Optimize: This gives the path to the file that specifies the variables to optimize for the project.

Metrics: This gives the path to the file that specifies the performance metrics that were received from the evaluator.

PE Weights: This gives the path to the file that specifies the weights given to each performance metric.

PV Relations: This gives the path to the file that specifies the relation between the variables and the metrics.

```
<Input>
  <VariablesToOptimize>Variables.xml</VariablesToOptimize>
  <Metrics Path_Type="RelativeToSelf">Metrics.xml</Metrics>
  <PE_Weights Path_Type="RelativeToSelf">PE_Weights.xml</PE_Weights>
  <PV_Relations Path_Type="RelativeToSelf">PV_Rel.xml</PV_Relations>
  <TopN>100</TopN>
</Input>
```

Output

This section of the file gives information on how to save the results of the optimization

```
<Output>
  <Report Path_Type="RelativeToSelf">
  ../../results/optimizer/report.xml</Report>
  <ResultsFolder Path_Type="RelativeToSelf">
  ../../results/optimizer</ResultsFolder>
</Output>
```

- **Report**

This gives the path to the report file where the results are saved

- **Results Folder**

This gives the path to the folder where results are saved i.e. the report file and the TopN project files.

5.2 Preparation

This part of the program deals with preparing the inputs for the optimizer and cleaning up the parameters. Following are the steps taken for the preparation of the data.

5.2.1 Read Input Files

In this step the following input files are read, for which the paths have previously been provided.

Variables File

This file gives the variables to optimize and the range or values at which they are to be tested. The ranges are given in MatLab range format i.e. [start]: [step]: [end] or [start]: [end]. If the step is not given it is considered as 1. The multiple ranges mentioned for a single variable may not be continuous. The value for the variable is check for all the given constraints, e.g. if an integer range is 3:1:5 and 7:1:9 the valid solution for this variable would be $\{v|v \in \{3,4,5,7,8,9\}\}$. If the variable supports a double value and PSO or GSA techniques are used, the step size does not matter and the valid range for v would be $\{v|v \in \mathbb{R} \wedge[(3 \leq v \leq 5) \vee (7 \leq v \leq 9)]\}$.

```
<File Name="../../projects/3d_spherical/tracker.xml"
Path_Type="RelativeToSelf">
  <Control Name="track_confirmation_for_display_m">
    <Ranges>
      <Range Name="Range_1">2:1:4</Range>
    </Ranges>
  </Control>
  <Control Name="track_confirmation_for_display_n">
    <Ranges>
      <Range Name="Range_1">2:1:4</Range>
    </Ranges>
  </Control>
<Control Name="process_noise_scaling_factors">
    <Values>
      <Value Name="Value_1">0.003:0.0005:0.006</Range>
      <Value Name="Value_2">0.004:0.0005:0.005</Range>
    </Values>
</File>
```

Weights File

This file gives the weights of the performance metrics for score calculation.

```
<Weights>
  <Metric Name="RMSE" Weight="1.0" />
  <Metric Name="FalseTrackRate" Weight="10" />
  <Metric Name="NumFalseTrack" Weight="0.0" />
  <Metric Name="CumBroken" Weight="0.0" />
  <Metric Name="Completeness" Weight="0.0" />
  <Metric Name="TrackFragment" Weight="0.0" />
  <Metric Name="ConfLatency" Weight="10" />
  <Metric Name="TrackPD" Weight="0" />
</Weights>
```

5.2.2 Quiet Mode

The ConfigManager is switched into Quiet Mode i.e. the user will not be able to see any messages from the ConfigManager.

5.2.3 Load Project Files

When the variables file is read, the parent files of all these variables are opened with the ConfigManager. The backup of these files are also created as the optimizer will try to change these files.

5.2.4 Cleanup

At the cleanup stage variables and parameters are cleaned up to make the optimization efficient, i.e. variables that have no effect on the output or that are not available in the Default file are removed from the ‘variables to optimize’ list.

Metrics

The following metrics are removed from the evaluation list:

- Metrics with zero weight.
- Metrics that have no effect on the score.

Variables

The following variables are removed from the optimization list.

- Variables whose parent files could not be opened.
- Variables that were not found in their parent files
- Variables that did not affect any of the performance metric.

5.2.5 Choose Best Method

This step evaluates the variables and their ranges to decide which method of optimization would be best. If the number-variables are all integers and their ranges are not very wide so that the total number of required fitness tests is less than the default max runs for PSO and GSA, the optimizer chooses the exhaustive search, otherwise it optimizes with PSO or GSA.

5.3 Optimization (Process)

This is the part where optimization takes place. It has four steps:

5.3.1 Calculate Solution

Here the optimizer calculates a solution for the problem depending on the method selected. These methods are discussed below.

5.3.2 Apply Solution

The values calculated for all the variables by the above step are then written into their respective files using the ConfigManager. If the ConfigManager is successful in validating and saving the files the program moves forward otherwise it goes back to step one to find another solution.

5.3.3 Run Tracker and Evaluator

This part deals with running the tracker and getting the results from the evaluator. There are two ways to get the results:

Callback Function

In this method he results are obtained from the callback function. The pointer to this function is given during the initialization of the optimizer. This function is calls the tracker and then the evaluator, returning the evaluation metrics.

Exe Paths

The results from the fitness function are read from a file in this method. The optimizer first runs the exe for the tracker and then for the evaluator. When the evaluator gives its output in the form of a file, the Optimizer reads the file for the evaluation metrics.

5.3.4 Score Calculation

The score is calculated by adding the weighted sum of all metric values translated to score. The individual score of each metric is calculated and then summed up based on their weights. The individual score of each metric is calculated by different methods based on the type of the metric. Below are the seven types of value to score relations and their calculation methods.

- Linear Direct

  $score = weight * value$

- Linear Inverse

  $score = weight/value$

- Direct

  $score = weight * value$

- Inverse

  $score = weight * (IdealValue - value)$

- Log

  $score = weight * log\ (value)$

- No Effect

  $score = 0$

- Unknown

  $score = 0$

Example:

For the following three metrics:

Name	Relation	Weight	Value
A	Linear Direct	w_A	v_A
B	Linear Inverse	w_B	v_B
C	Log	w_C	v_C

The total score would be:

$$Score_{Total} = w_A * v_A + w_B/v_B + w_C * log(v_C)$$

5.4 Results (Output)

The output of the optimizer (saved in the results folder) consists of the following two components:

5.4.1 Report

The report is the final output of the optimizer. It lists in descending order the Top N solutions discovered; i.e. the solutions with the Top N scores.

5.4.2 Project Files

If requested the optimizer saves the project files for the tracker, with Top N solutions in separate folders. These can be later copied to the original folder for testing by the user.

5.5 Modules:

The application has been divided into the following modules:

1. Tracker
2. Optimizer
3. ConfigManager
4. Performance Evaluator

The following flow charts show how the algorithm runs (Figure 3 Optimizer Flow Chart) and what part of algorithm is handled by each module (Figure 2 Modules).

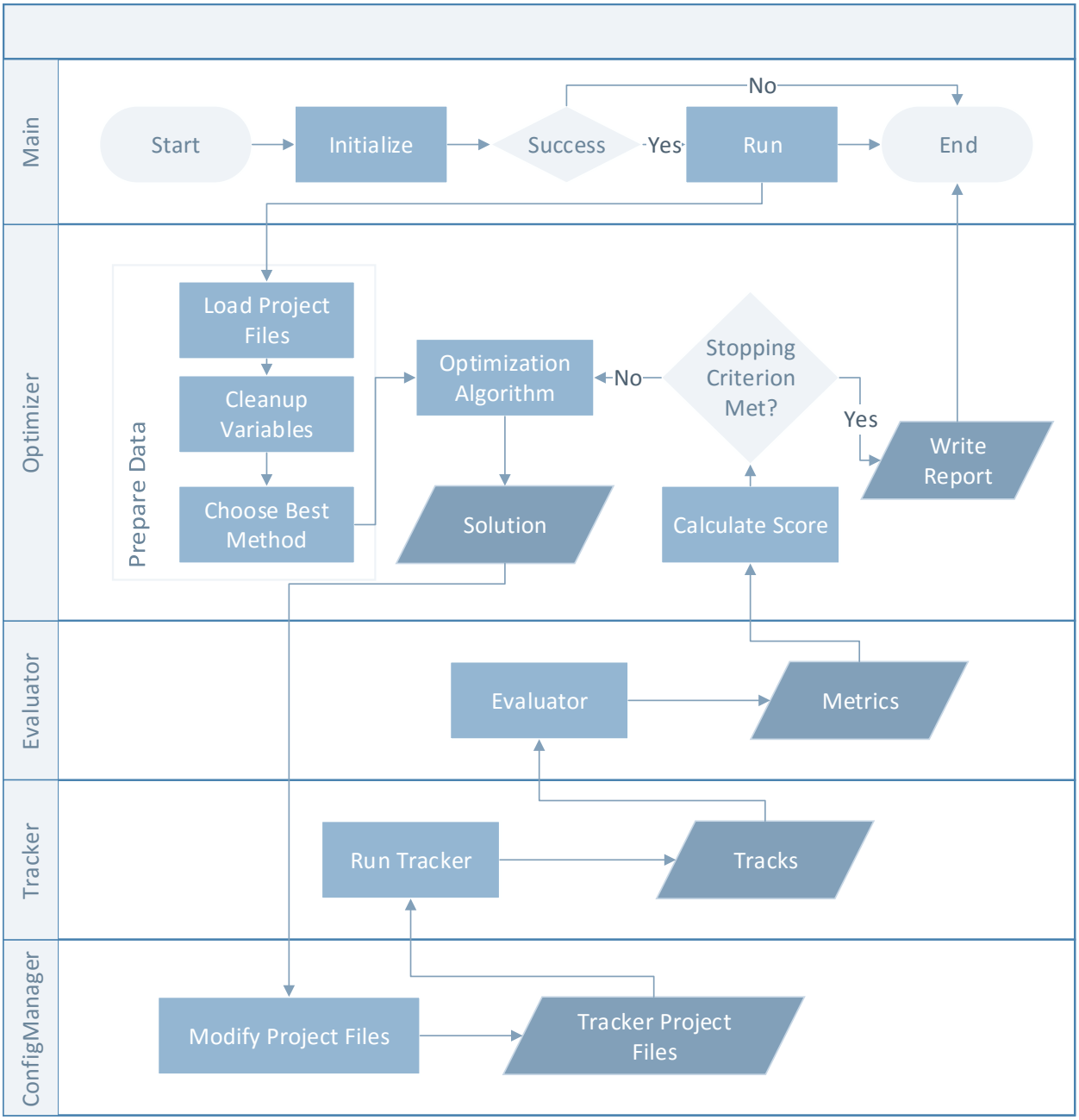

Figure 2 Modules

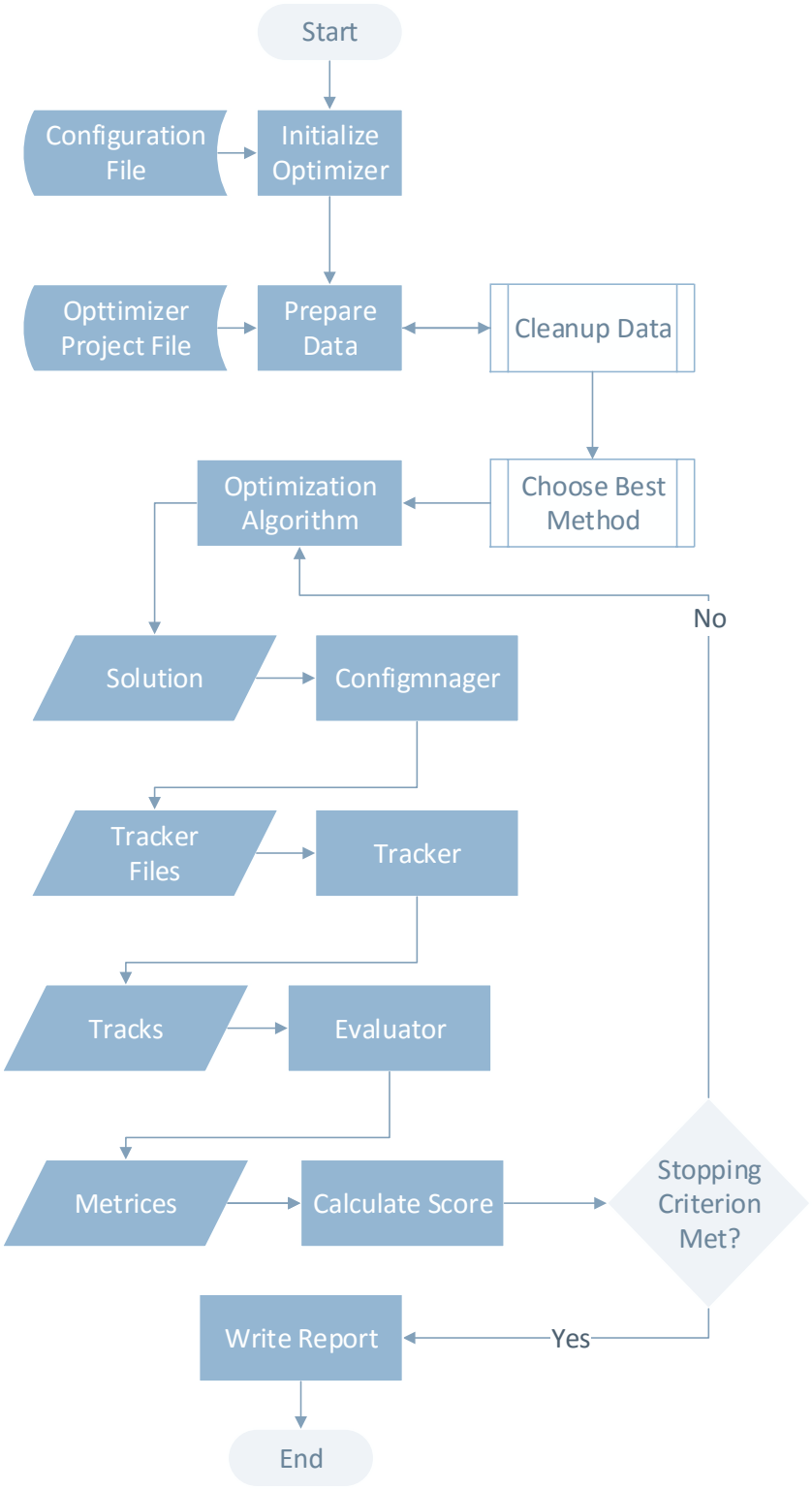

Figure 3 Optimizer Flow Chart

Chapter 6

OPTIMIZATION TECHNIQUES

Regular optimization algorithms are not feasible with this tracker optimization problem since it has to deal with a higher dimensional search space where the variables involved have an irregular, nonlinear effect on the output and are also not differentiable functions. Also the search space increases exponentially with the increase in dimensions or range making exhaustive search very impractical.

Heuristic methods are best applied for problems with multiple dimensions and large search spaces. There are many nature inspired algorithm like the Genetic Algorithm, Ant Colony Search Algorithm, Particle Swarm Optimization, Gravitational Search Algorithm etc. However these algorithms do not guarantee the optimal solution in all cases; different algorithms work best for different problems.

This Optimizer provides two of these heuristic methods and one exhaustive search algorithm. For smaller scale problems (smaller number of variables and short ranges) the Optimizer uses 'All Combinations' (exhaustive search), otherwise one of the heuristic algorithms is deployed.

The following are the three available methods.

6.1 Exhaustive

6.1.1 All Combinations

For this method the tracker is evaluated for every combination of values associated with each variable. If there is a set of n variables $X = \{X_0, X_1, X_2 \dots X_n\}$ and each X_i has n_i number of values, then the total number of combinations would be $\prod_{i=0}^{n} n_i$. Each of the solution is evaluated and the ones with the top N scores are saved and written into the report.

For example if there are the following values for the variables A, B and C:

$A = \{a_0, a_1, a_2\}$

$B = \{b_0, b_1, b_2\}$

$C = \{c_0, c_1\}$

The tracker will be evaluated for the following 18 combinations:

$$\text{Combinations} = \begin{Bmatrix} a_0 & b_0 & c_0 \\ a_0 & b_0 & c_1 \\ a_0 & b_1 & c_0 \\ a_0 & b_1 & c_1 \\ a_0 & b_2 & c_0 \\ a_0 & b_2 & c_1 \\ a_1 & b_0 & c_0 \\ a_1 & b_0 & c_1 \\ a_1 & b_1 & c_0 \\ a_1 & b_1 & c_1 \\ a_1 & b_2 & c_0 \\ a_1 & b_2 & c_1 \\ a_2 & b_0 & c_0 \\ a_2 & b_0 & c_1 \\ a_2 & b_1 & c_0 \\ a_2 & b_1 & c_1 \\ a_2 & b_2 & c_0 \\ a_2 & b_2 & c_1 \end{Bmatrix}$$

The total number of combinations is $n_A \cdot n_B \cdot n_C = 3 \cdot 3 \cdot 2 = 18$

6.2 Heuristic Methods

Heuristic methods are experience based programming methods that involve learning and discovery to find a solution that is not guaranteed to be optimal but adequate for a given set of goals. These methods are used where exhaustive search is not feasible.

Many of the heuristic methods are nature-inspired, of which the Particle Swarm Optimization (PSO) and the Gravitational Search Algorithm (GSA) have been implemented in this application. These methods have a population of candidate solutions (multiple starting points) in the search space that work in parallel and communicate with each other to get an optimal position. There are two patterns of search involved: exploration and exploitation. Exploration is searching a broader region of the search space while exploitation is finding the optima around a good solution. Exploration increases the velocity of the agents and prevents from the optimizer from getting trapped in local optima. These methods facilitate exploration in the beginning of the algorithm and with time fade it out and fade in exploitation. The balance between the two is very essential for a good performance.

The members of population called particles in PSO and agents or masses in GSA go through the following steps:

- Self-Adaptation: members try to improve their performance
- Cooperation: members communicate and collaborate with each other
- Competition. Members compete to survive

Both of the mentioned algorithms have stochastic properties. All the heuristic methods provide satisfactory results but there is no method that is superior to all methods for all optimization problems. Different methods perform best for different solutions.

6.2.1 Particle swarm

Background

The Particle Swarm Optimization (PSO) is used to optimize multi dimensional problems. The optimizer is inspired by the behaviour of social animals such as insects (ants, bees, and termites), fish, birds etc. It works the way these animals interact in swarms or colonies when looking for food. They spread out in the given space and look for good positions; they communicate the better positions with each other and also keep in memory the best ones they have discovered themselves. Based on this information all the individuals adjust their positions and velocities. The PSO is mostly applied in search engines and optimizers. The PSO was originally discovered by Kennedy and Eberhart [1] who were trying to simulate the social model of these creatures in their swarms, flocks or schools. They observed that this method was performing optimization.

Introduction

The PSO is an iterative, computational method that optimizes a problem by creating random feasible solutions in a search space and improves them iteratively using the quality information of each candidate.

The PSO is useful for handling multi-dimensional, nonlinear problems as it is a pattern search method and does not require the problem to be differentiable. It is meta-heuristic as it makes no assumptions about the problem and can search very large spaces. The optimal solution is not guaranteed but the probability of finding one is high if the parameters for the optimizer are

correctly initialized. In conclusion PSO is suitable for problems that are noisy, partially irregular and change over time.

Algorithm

The PSO works by evaluating a population (swarm) of randomly chosen solutions (particles) for the tracker. It moves the particles around in the search space, with reference to the particle with the best position and the particle's own best position through time, to find an optimal solution. These best positions (solutions with the highest evaluator score) act as guide for the particles and are updated if a position with a higher score is found. This process is repeated several times until the stopping criterion is met.

The variables act as dimensions of the particle and the value of each variable is considered as the position of the particle in that particular dimension.

The score (quality) of a particle at a certain position (candidate solution) is called the fitness of that particle and is denoted by the function f. The fitness function takes in a vector and gives out a scalar. The gradient of this function is not known and can be anything. Since the goal is to find a solution that gives the maximum evaluator score, the goal of the optimizer would be to find a solution a for which the fitness is greater than the fitness of all the positions in the search space (U) i.e.

$f(a) \geq f(b)$ where $b \in U$

Let N be the number of particles in the swarm, and for each particle i has position vector x_i and velocity vector $v_i$in the search space S. Let p_i be the best known position of the particle and g be the best known position of the entire swarm.

- **Initialization**
  1. The optimizer starts with initializing the position of each particle i in each dimension with a random uniform distribution based on the range of that particular dimension:
     $x_i = U(b_l, b_u)$
     where b_l and b_u are the upper and lower boundaries of the search space.
  2. The best known position $\mathrm{p_i}$ of each particle is initialized to its initial position:
     $p_i \leftarrow x_i$

3. If the fitness of the particle is greater than the global best(best one discovered yet by all particles) , the global best position is updated:
   If $\{f(p_i) > f(g)\},\ g \leftarrow p_i$
4. Initialize the particles velocity
   $v_i \leftarrow U(-|b_u - b_l|, |b_u - b_l|)$

- **Iteration**

These steps are repeated for each particle until a stopping criterion is met:

1. Pick random numbers r_p and r_g
   $\mathrm{r_p} \leftarrow \mathrm{U}(0,1)$
   $r_g \leftarrow U(0,1)$
2. For each dimension d update the velocity
   $\mathrm{v_{i,d}} \leftarrow \omega * \mathrm{v_{i,d}} + \ \varphi_\mathrm{p} * \mathrm{r_p} * (\mathrm{p_{i,d}} - \mathrm{x_{i,d}}) + \varphi_\mathrm{g} * \mathrm{r_g} * (\mathrm{g_g} - \mathrm{x_{i,d}})$
   where $\omega, \varphi_\mathrm{p}$and $\varphi_\mathrm{g}$are constants discussed below.
3. Update the particles position
   $\mathrm{x_i} \leftarrow \mathrm{x_i} + \mathrm{v_i}$
4. If $\{\mathrm{f}(\mathrm{x_i}) > f(\mathrm{p_i})\}$ update the particle's best known position:
   $\mathrm{p_i} \leftarrow \mathrm{x_i}$
5. And if $\{\mathrm{f}(\mathrm{p_i}) > f(\mathrm{g})\}$, update the swarms best known position
   $\mathrm{g} \leftarrow \mathrm{p_i}$

- **Velocity**

The velocity update equation has three parts: the momentum part$[\omega * v_{i,d}]$, cognitive part $[\varphi_p * r_p * (p_{i,d} - x_{i,d})]$ and the social part$[\varphi_g * r_g * (g_g - x_{i,d})]$.

Momentum Part*:* The momentum part of the velocity controls the local exploitation and the global exploration of the search. The constant ω is called the inertia weight which determines the ratio between exploitation and exploration. A higher value of ω would increase the velocity and thus provide with more exploration while a lower value would slow the particle down and make it exploit its local surroundings. The inertial weight ω can be set to a higher value in the beginning for global exploration and then gradually decreased over time to refine the search.

***Cognitive Part**:* The cognitive part of the velocity conveys the confidence of the particle i.e. how much the particle trusts its own findings. The coefficient φ_p called the cognitive factor determines how much effect the particle's own experience has on the velocity. It is also known as an acceleration coefficient.

***Social Part**:* The social part of the velocity guides the particles to the global best position. The social coefficient φ_g determines how much the particle trusts the success of other particles. It is also known as an acceleration coefficient.

If the calculated velocity of a particle in some dimension makes it go out of the search space it is adjusted to bring the next step right on the boundary of the space.

PSO Applications

The first practical application of PSO was in 1995 in the field of neural networks. The application was able to train and adjust the weights of a feed-forward multilayer perceptron neural network. The results of the algorithm were as effective as the conventional error back-propagation approach. After that PSO became rapidly popular and due to its simple and efficient nature its applications were explored in several fields. Since then a lot of work has been done on the algorithm and many improvements and varieties have been introduced. These varieties have made PSO applicable to many different optimization problems from unconstrained, single-objective or static problems to constrained multi-objective or dynamic problems. Following are a few applications of the PSO [40]:

- Combinatorial optimization problems
- Computational intelligence applications
- Electrical and Electromagnetic applications
- Signal processing
- Graphics
- Image analysis: IRIS recognition, face detection and recognition, image segmentation , image classification, defect detection, image retrieval, image registration , pixel classification , detection of objects, texture synthesis , microwave imaging , scene matching, , character recognition , shape matching , image noise cancellation .

- Video Analysis: MPEG optimization, motion estimation, object tracking, body posture tracking, traffic incident detection etc
- Robotics
- Bioinformatics
- Medical applications.

Flow Chart

The following flow chart shows the algorithm of the PSO

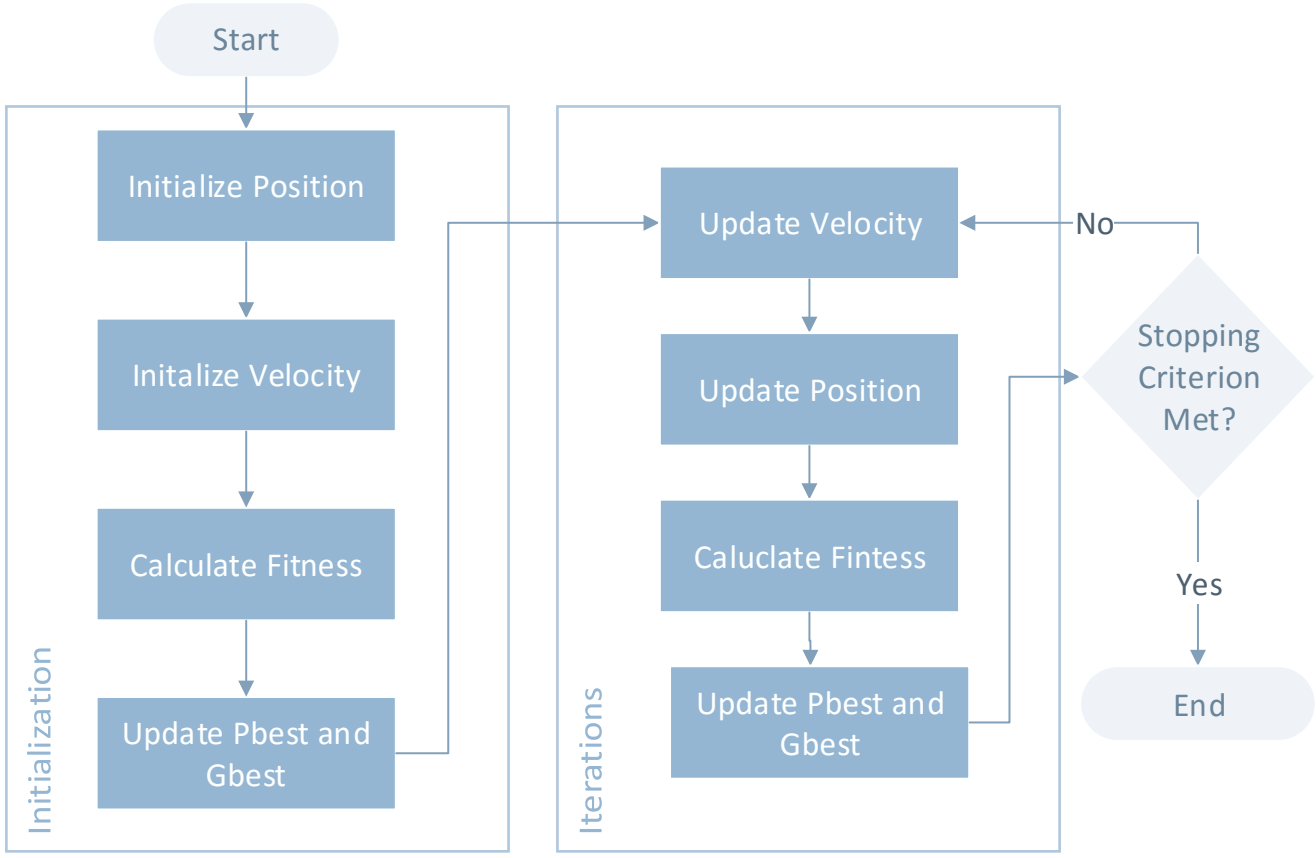

Figure 4 PSO Flow Chart

6.2.2 Gravitational Field Search

Background

The Gravitational Search Algorithm (GSA) developed by Rashedi et.al in [13] is based on Newton's law of universal gravitation, which states that any two masses in the universe attract each other with a force directly proportional to the product of their masses and inversely proportional to the square of the distance between them. The GSA is formulated on the principals of gravitation and it is assumed that the candidate solutions interact with each other the way masses in the universe attract each other.

Let there be two masses M_1 and M_2 , separated by distance R , and the gravitational force by which they attract each other be *F,* then:

$$F = G\frac{M_1 M_2}{R^2}$$

Where G is the gravitational constant.

Since the actual value of G decreases with the age of the universe, the following function is used to calculate the value of G with time.

$$G(t) = G(t_0) * \left(\frac{t_0}{t}\right)^{\alpha}, \qquad \alpha < 1$$

where $G(t)$ is the value of the gravitational constant at time t, and $G(t_0)$ is the value at the first cosmic quantum-interval of time t_0.

According to Newton's second law of motion, the acceleration of each particle will be

$$a = \frac{F}{M}$$

There are three kinds of masses:

Active gravitational mass (M_a) is the measure of an object's gravitational field. Objects with small active gravitational mass have a weaker gravitational field than objects with greater active gravitational mass.

Passive gravitational mass (M_p) is the measure of the gravitational field's strength for a particular object. Objects with small passive gravitational experience a smaller force from the gravitational field than objects with greater passive gravitational mass.

Inertial mass (M_i) is the measure of an object's inertia. Objects with large inertial masses resist more to forces, and change their states of motion slowly than objects with smaller inertial masses.

For a collection of masses, the above equations can be re-written, as

$$F_{ij} = G\frac{M_{aj}M_{pi}}{R^2} \text{ and } a_i = \frac{F_{ij}}{M_{ii}}$$

where F_{ij} is the force that acts on mass i by mass j, and $a_i$is the acceleration of mass i due to the overall force acting on it. M_{aj}, M_{pi} and M_{ii} represent the active mass of i, passive mass of j and the inertial mass of i.

Introduction

The Gravitational Search Algorithm considers all particles as masses communicating through the gravitational field. All the masses attract each other causing a global movement of all of them. The mass of each object is the measure of the quality of its solution. As acceleration is inversely proportional to the masses, the heavier masses move slowly giving us exploitation and the lighter masses move fast giving us exploration.

Each mass have four aspects: position, inertial mass, active gravitational mass, and passive gravitational mass. The position of a mass provides the solution and the masses correspond to the quality of the solution. The algorithm adjusts the masses of all particles to find the optimal solution with all of them converging toward the heaviest mass.

Algorithm

Let N be the number of agents in this system and d be the number of dimensions (variables). Then $X_i$and V_i represent the position and velocity vector of the agent i respectively, and x_i^d and

v_i^d are the position and velocity (respectively) of the i-th agent in dimension d. f is the fitness function.

- **Initialization**

1. The optimizer starts with initializing the position x_i of each agent i in each dimension with a random uniform distribution based on the range of that particular dimension.

   $x_i = U(b_l, b_u)$

   where b_l and b_u are the upper and lower boundaries of the search space.
2. Initialize the velocity of all particles:
   $v_i \leftarrow U(-|b_u - b_l|, |b_u - b_l|)$
3. Initialize the acceleration of all particles with zero:
   $a_i \leftarrow 0_d$
4. Initialize the force of all particles with zero:
   $F_i \leftarrow 0_d$
5. Calculate the fitness $f(X_i)$of each particle.
6. If the fitness of the particle is greater than the global best(best one discovered by all particles at the current time) , the global best position is updated:
   If $\{f(X_i) > f(\text{Best})\}$, $\text{Best} \leftarrow X_i$
7. If the fitness of the particle is less than the global worst (worst one discovered by all particles at the current time), the global worst position is updated:
   If $\{f(X_i) > f(\text{Worst})\}$, $\text{Worst} \leftarrow X_i$
8. Update the all-time-global-best $\text{Best}_{\text{AT}} = \max_{i\in\{1,\ldots,N\},t\in\{0,\ldots,T\},} f_i(t)$ position with the current global best calculated at the end of initialization. This is not part of the original algorithm but an addition that was brought in for this application. Its purpose is discussed later in the article.

- **Iteration**

These steps are repeated until a stopping criterion is met:

1. **<u>Update the best and worst positions:</u>**

$\text{Best}(t)$, $\text{Worst}(t)$ and All-time best Best_{AT} position in the search space are calculated as:

$$\text{Best}(t) = \max_{j\in\{1,\ldots,N\}} f_j(t)$$

$$\text{Worst}(t) = \min_{j\in\{1,\ldots,N\}} f_j(t)$$

$$\text{Best}_{AT} = \max_{i\in\{1,\ldots,N\},t\in\{0,\ldots,T\},} f_i(t)$$

2. **Calculate G(t)**

$G(t) = G(G_0, t)$ decreases with time, it is calculated as

$G(t) = G_0 * e^{-(\propto\frac{t}{T})}$ where T is the total number of iterations and $\propto$ is normally taken as 20.

3. **Calculate the mass $M_i(t)$ for all agents:**

It is assumed that all the three kinds of masses are equal for each particle i, i.e.:

$M_{ai} = M_{pi} = M_{ii} = M_i$

M_i is calculated by the fitness evaluation of the agent. If an agent has a higher mass then this means that it has a higher fitness value. Heavier agents walk more slowly than lighter ones.

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^{N} m_j(t)}$$

where $m_i(t) = \frac{f(X_i(t)) - f(\text{Worst}(t))}{f(\text{Best}(t)) - f(\text{Worst}(t))}$

If $f\big(\text{Best}(t)\big)$ is equal to $(\text{Worst}(t))$, $m_i(t)$ is assumed as 1.

If $M_i(t)$ is equal to zero, and the agent is accelerated toward the best position in that dimension with max acceleration.

4. **Calculate the force $F_i(t)$ for all agents :**

At time t, the force acting on mass i from mass j will be:

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t)M_{aj}(t)}{R_{ij}(t) + \varepsilon}\left(x_j^d(t) - x_i^d(t)\right)$$

$$R_{ij}(t) = \left\|X_i(t), X_j(t)\right\|_2$$

where $R_{ij}(t)$ is the Euclidian distance between the two agents i and j, and ε is a small constant preventing the denominator from becoming zero.

To give a stochastic characteristic to the algorithm, the total force that acts on agent i in a dimension d is calculated as a randomly weighted sum of the d-th components of the forces from all other agents.

$$F_i^d(t) = \sum_{j=1\ j\neq i}^{N} r_j * F_{ij}^d(t)$$

Where r_j is a uniform random variable in the interval $[0,1]$

5. **Calculate the acceleration $a_i(t)$ for all agents**

The acceleration can then be calculated as

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)}$$

Calculate the velocity $v_i(t)$ and position $X_i(t)$ for all agents

The velocity an agent is a fraction of the current velocity plus the acceleration

$$v_i^d(t+1) = r_i * v_i^d(t) + a_i^d(t)$$

where r_i is a uniform random variable in the interval $[0,1]$, which is used to give a randomized characteristic. If the velocity is large enough to make the particle exceed bounds it is reinitialized with a uniform random value inside the bounds.

$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1)$$

- **K-Best**

To save computational costs and to exploit the best masses the number of agents is reduced with time. Then only a set of agents with bigger masses apply forces to all the agents, and guide the weaker ones towards themselves. This method should be activated gradually and for the last few iterations so that the optimizer may not get trapped in local maximas.

$$F_i^d(t) = \sum_{j\in Kbest, j\neq i}^{N} r_j * F_{ij}^d(t)$$

- **All Time Best**

Since the global best gives the best position of the current time, it is possible that the best position from the past is replaced by a lower one in the next iteration. So the all time best position in kept in memory and at the end of the algorithm it is presented as the optimal position.

Flow Chart

The following flow chart shows the algorithm of the GSA

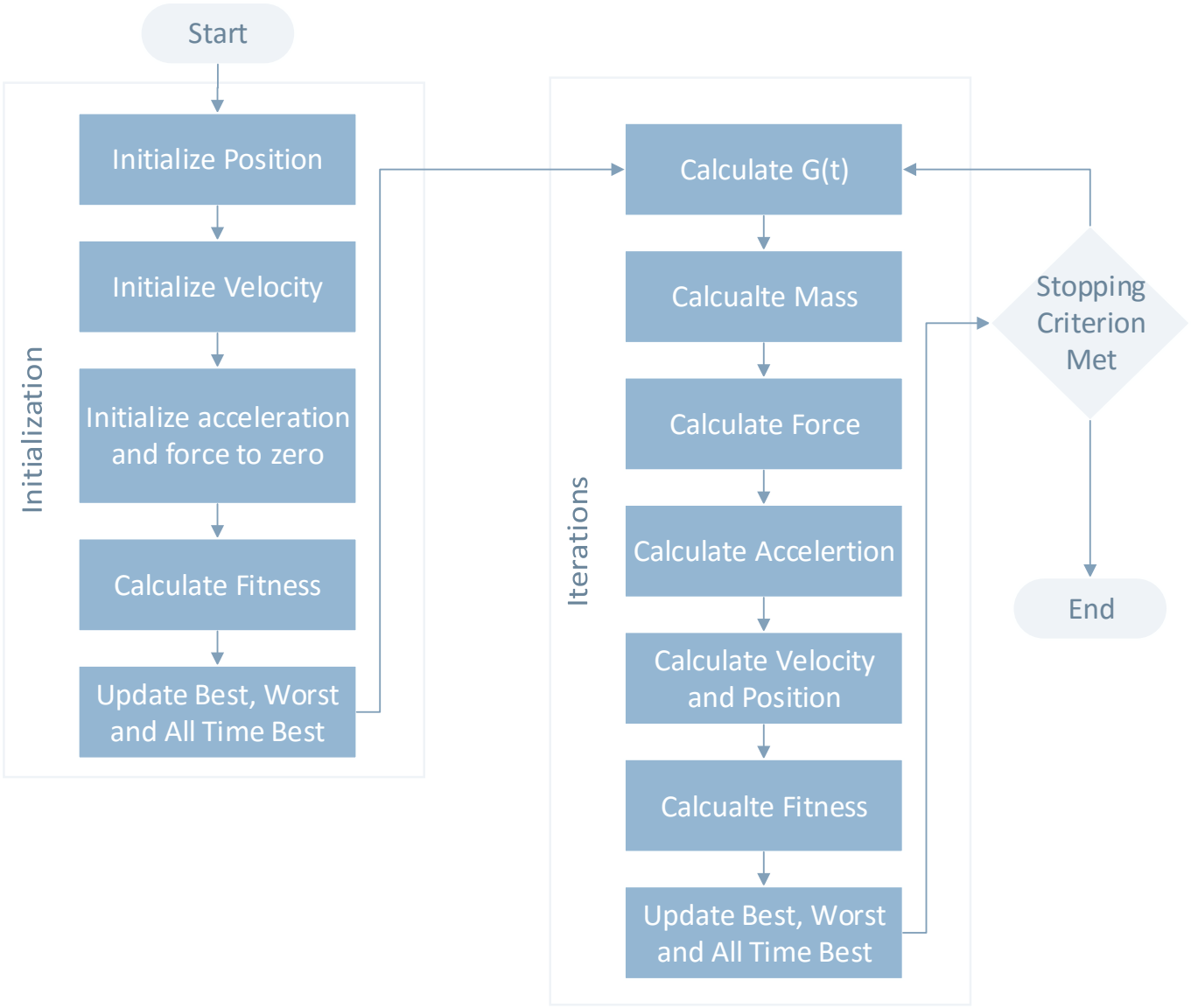

Figure 5 GSA Flow Chart

6.2.3 Discrete Variables

Variables that have discrete values or variables with non-float values are rounded up before being used but they remain as float values in their algorithms.

6.2.4 Population Size

There is no guaranteed way to calculate the optimum population size based on the number of dimensions and search space but as a rule of thumb it is 3 or four time the number of variables. If the population size is too low, the purpose of using a population based algorithm would be killed. If the number is too high and the number of evaluator runs are fixed or the amount of time is fixed, the number of iterations would become low and the algorithm would not be able to properly explore or exploit the search space.

6.2.5 GSA versus PSO

GSA and PSO are both SI algorithms and follow the same basic process, but some of the principles they follow are different. Here are some of the important differences among the two algorithms observed by the authors of GSA [13].

- PSO calculates the direction of an agent using only the global and personal best positions while GSA's calculation is based on the overall force obtained by all the other agents.
- PSO updates the agents regardless of the fitness of their solutions while the GSA updates the force proportional to the fitness value.
- PSO keeps the personal and global best in memory when updating the velocity while GSA is memoryless and involves only the current positions in the updating process.
- PSO updates without including the distance between solutions into its calculations while the GSA takes distance into account as the force calculated is inversely proportional to the distance between the solutions.

6.2.6 History

The optimizer keeps a history of the last H_n solutions tested by the algorithm which prevents it from evaluating the same solution again. Ideally all the history should be maintained, however memory constraints allow only the last H_n solutions to be stored in memory. The number H_n is given by the configuration file of the optimizer.

Chapter 7

RESULTS

The performance of the optimizer was evaluated with mathematical functions, images and the tracker. Below is a discussion of the results and a comparison of the methods.

7.1 Images

The methods were tested with images by using them as fitness functions for a 2D problem, i.e. the pixel intensity at each point was considered as the fitness of that point and the aim of optimization was to maximize the function by finding the brightest point in the image. Two kinds of images were used, one with only one global optimum (Gradient.jpg) and the other with multiple local and global optimum areas (X-Ray.jpg). The plots have been plotted using data averaged over 50 simulations. The following images were used:

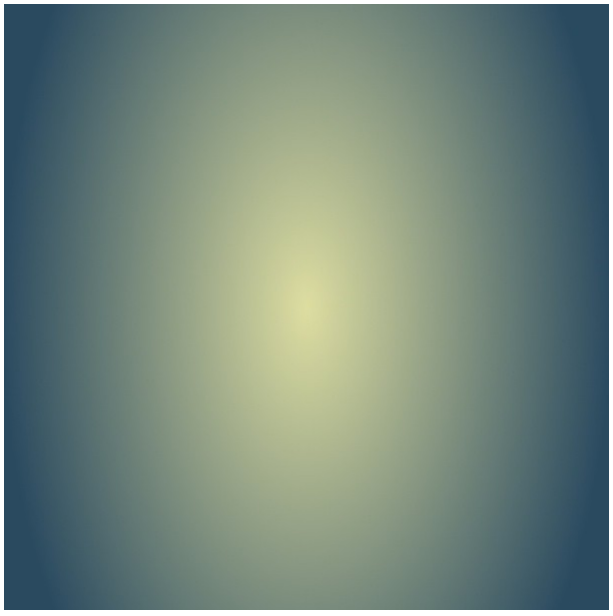

Figure 6 Gradient Image
Single Global Optimum Area. Max Pixel Intensity is 216

Figure 7 X-Ray Image
Multiple local and Global Optimum Areas. Max Pixel Intensity is 255 [41]

If the number of iterations is the same for all tests, the following results are obtained for different sample sizes using Particle Swarm Optimization and Gravitational Search Algorithm on the Gradient Image.[1]

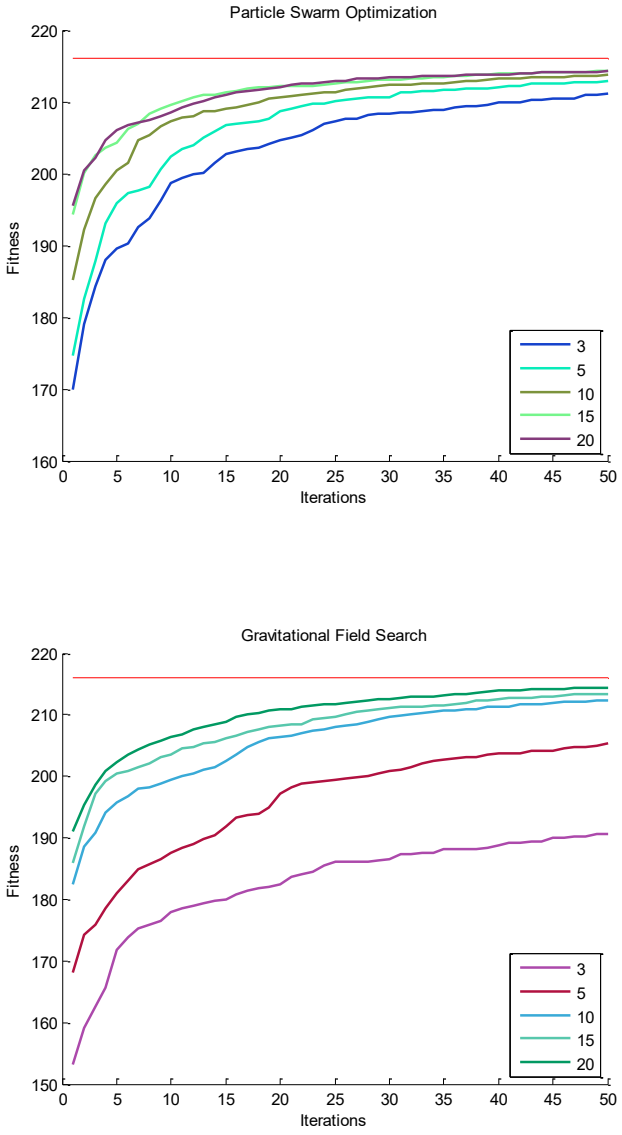

Figure 8 Fixed number of Iterations (Gradient Image)

[1] The legend shows the Sample sizes while the red horizontal line on the top represents the global maximum value (216).

If the number of iterations is the same for all tests, the following results are obtained for different sample sizes using Particle Swarm Optimization and Gravitational Search Algorithm on the X-Ray Image.[2]

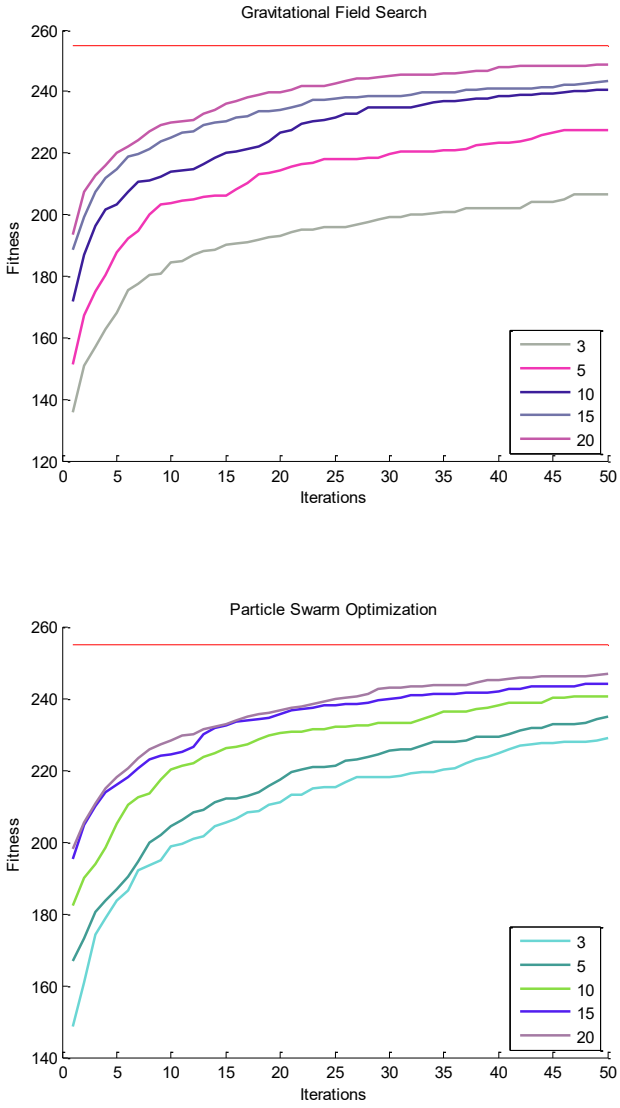

Figure 9 Fixed number of Iterations (X-Ray Image)

[2] The legend shows the Sample sizes while the red horizontal line on the top represents the global maximum value (255).

If the number of fitness tests is kept same, and the number of iterations are changed based on the sample size the following results are obtained for the Gradient Image using PSO and GSA[3].

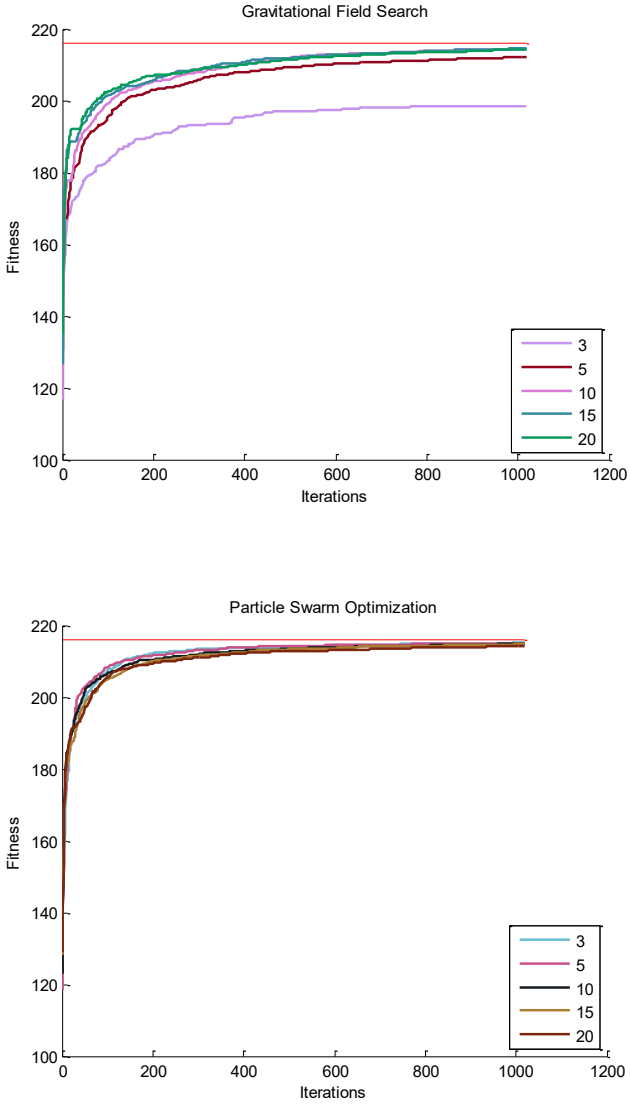

Figure 10 Fixed number of Fitness Evaluations (Gradient Image)

[3] The legend shows the Sample sizes while the red horizontal line on the top represents the global maximum value (216).

If the number of fitness tests is kept same, and the number of iterations are changed based on the sample size the following results are obtained for the X-Ray Image using PSO and GSA[4].

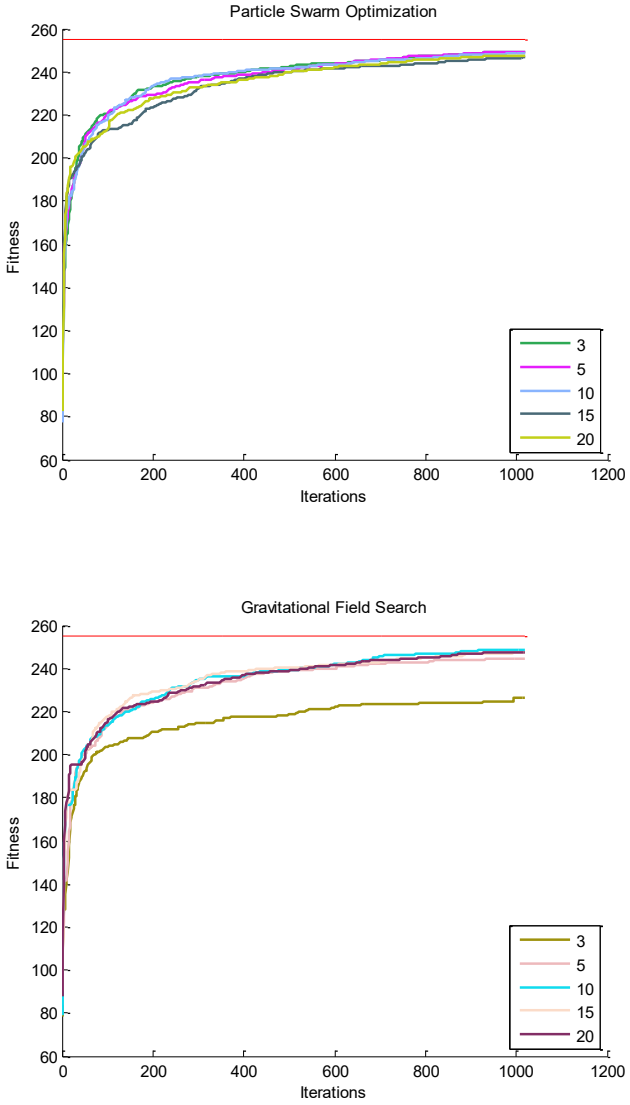

Figure 11 Fixed number of Fitness Evaluations (X-Ray Image)

[4] The legend shows the Sample sizes while the red horizontal line on the top represents the global maximum value (255).

7.2 Tracker

The Optimizer was run with the tracker to find the optimal solution for it, with the fitness evaluation done by the help of the Performance Evaluator. The Optimizer gave considerable improvements in many areas for the Tracker. Below are the results of the few performance parameters it was run to optimize. These results are averaged over multiple Monte Carlo runs. These bar plots show the product of the parameter value and the weight of the parameter used to calculate the scores.

In this example, PSO was used as the optimization method with 10 samples and 10 Maximum runs i.e.10*10=100 fitness evaluations and $\omega = 1, \varphi_p = 1 \; and \; \varphi_g = 2$

The variable to optimize was the "Process Noise Scaling Factor" and the allowed range was from 0.01 to 1.

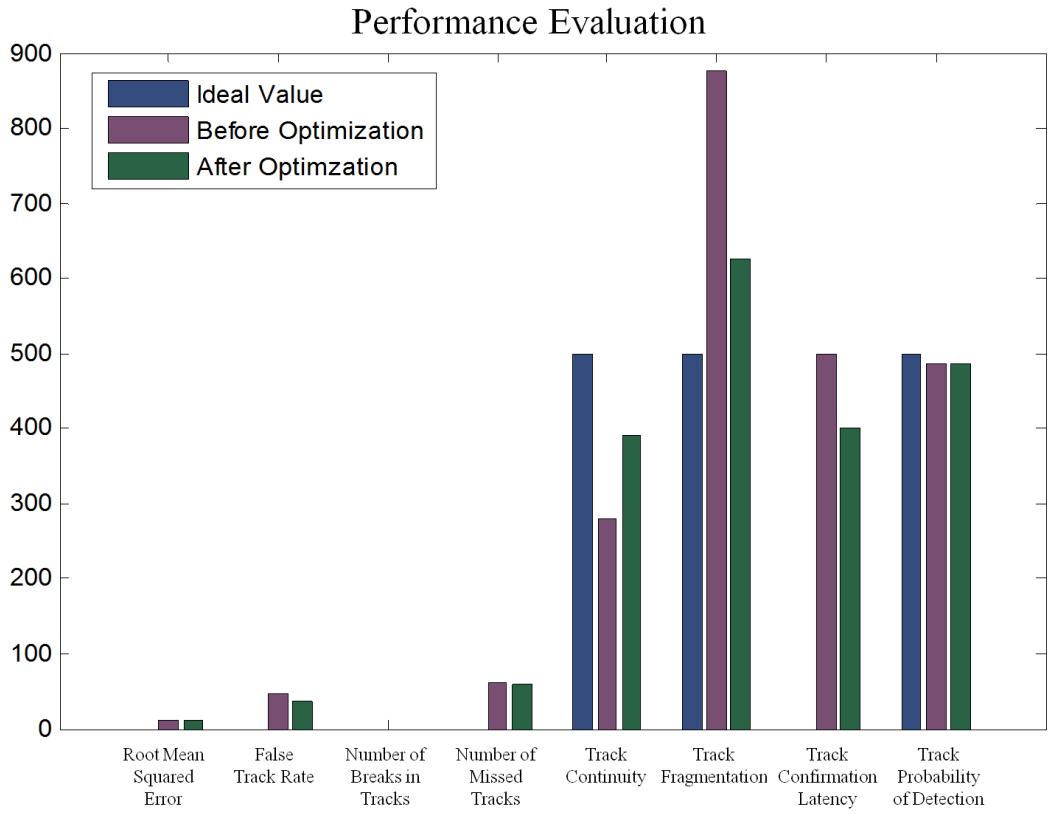

Figure 12 Performance Evaluations

This above chart shows how much the tracker performance improved by using the Optimizer on one variable.

Tests were performed to compare the two methods using the Tracker. The variable to optimize was the "Process Noise Scaling Factor" and the allowed range was from 0.01 to 1. Both Methods were given 10 samples and 10 Maximum runs i.e.10*10=100 fitness evaluations. For PSO: $\omega = 1, \varphi_p = 1 \; and \; \varphi_g = 2$ and GSA: $\alpha = 1, G_0 = 100$

The tracker gave improved results for both the optimization methods, but the score for GSA was slightly better than PSO. As it is a minimization optimization, lower scores are better. Score before optimization was 1213.73265, while score with GSA: 522.72521, and the score with PSO was 523.43729. These results are an average of multiple Monte Carlo runs.

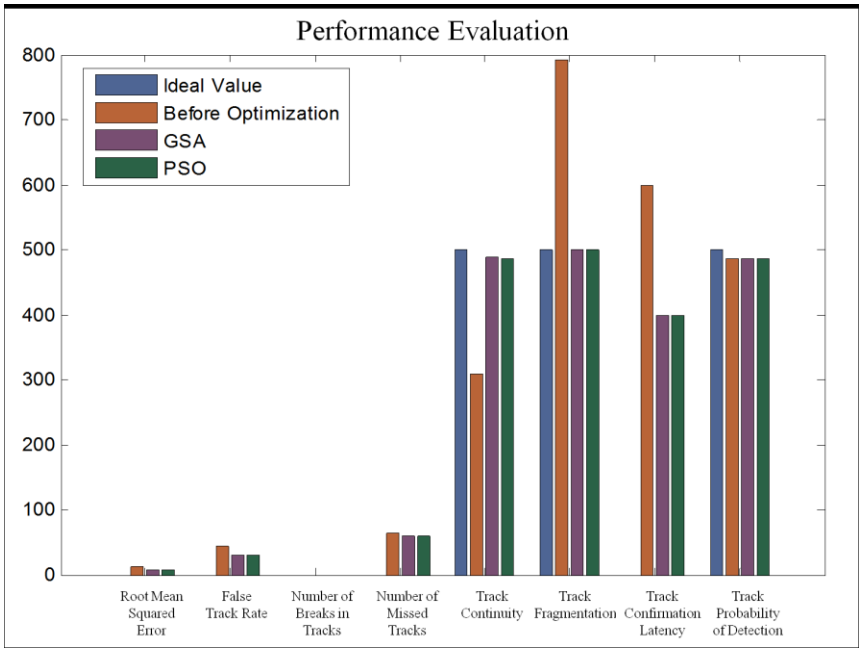

Figure 13 Comparison of PSO and GSA

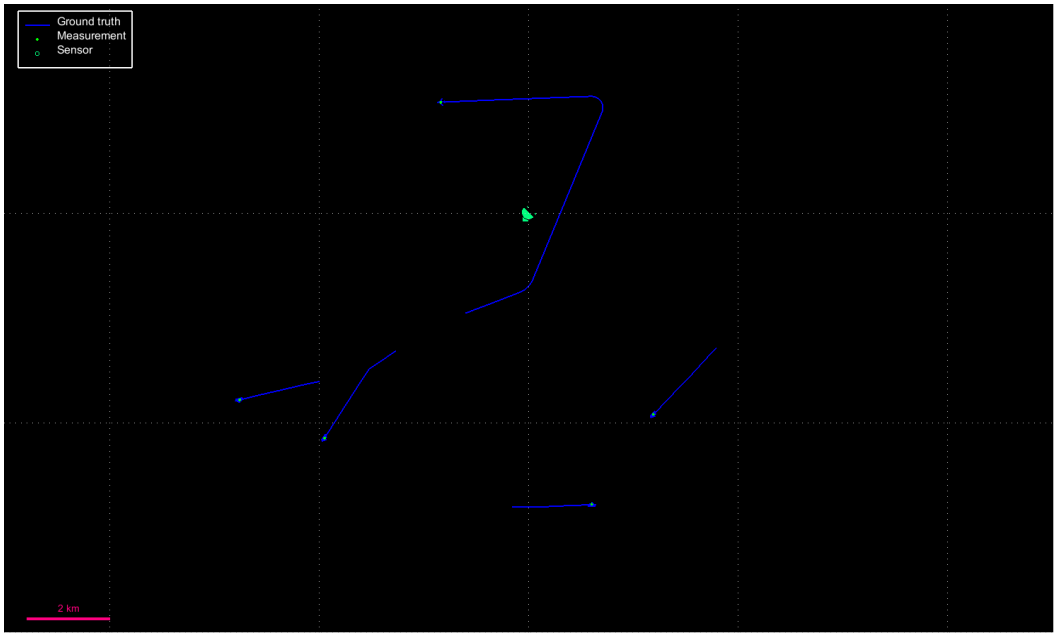

Figure 14 Tracker Ground Truth

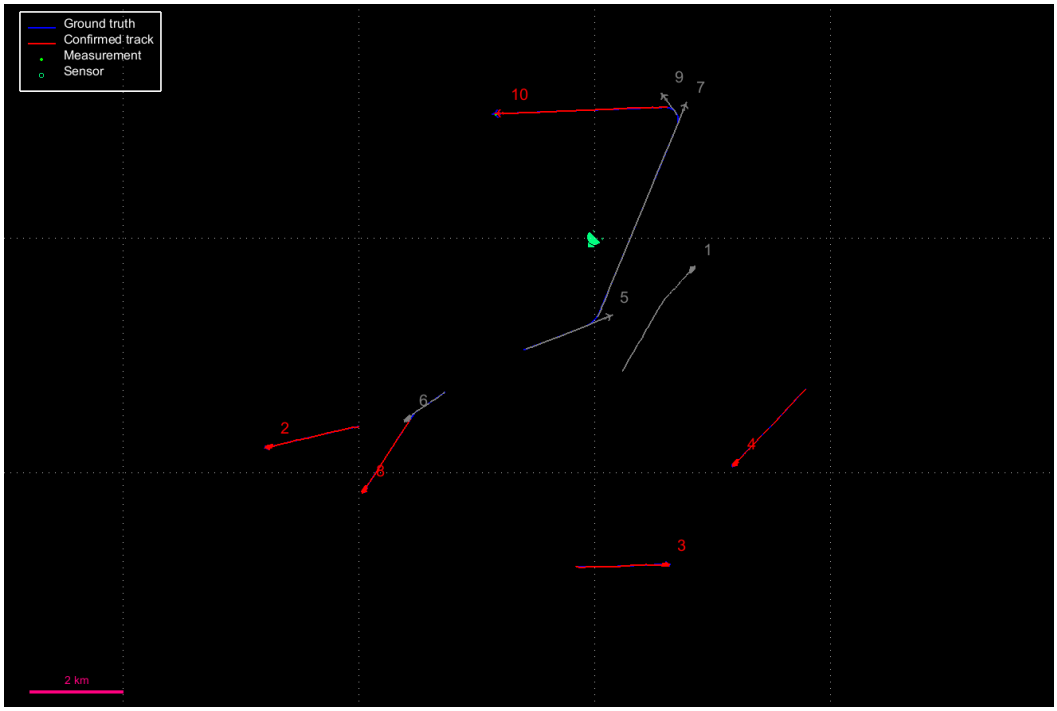

Figure 15 Tracks before Optimization

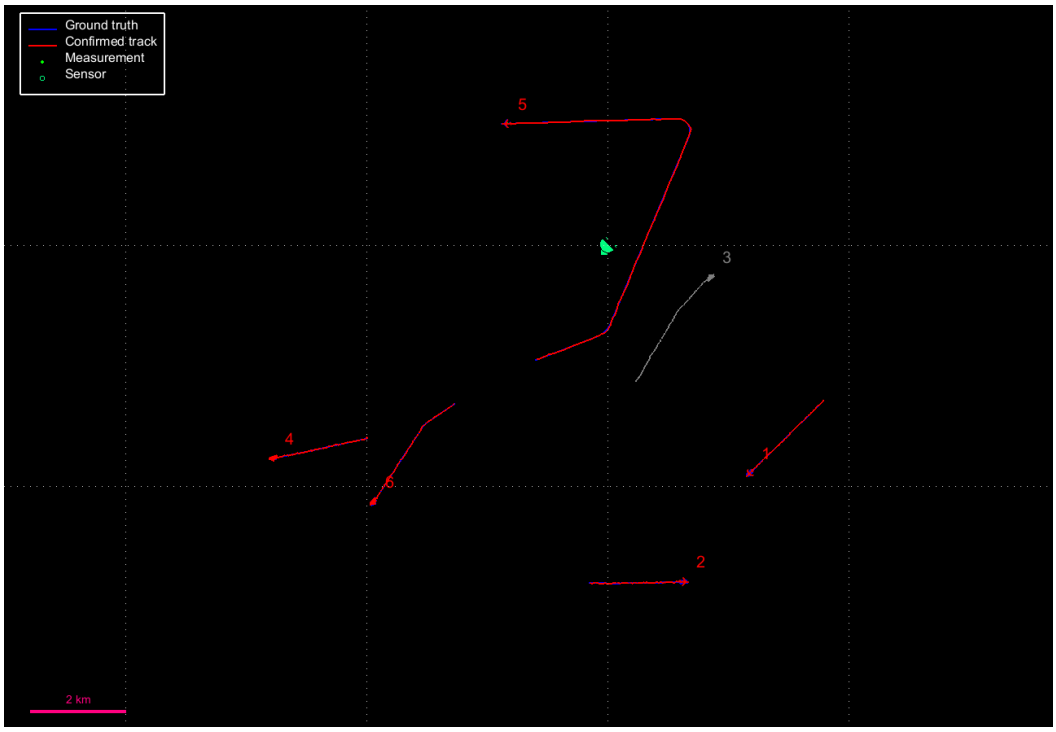

Figure 16 Tracks after Optimization

7.3 Observations

By studying the plots it is observed that the PSO gives better results for simpler problems while the GSA is more effective for complex problems. Bigger population sizes give better fitness curves if the number of Fitness Evaluations (FEs) is not considered and iterations are kept the same. This is because for bigger populations, higher numbers of FEs are carried out. Normally the FEs are costly so if each population size is given a fixed allowance of FEs , it is observed that the populations with a medium size perform the best as they strike a good balance between iterations and population. The performance of GSA drops considerably with population sizes less than 5.

The PSO and GSA perform differently in different circumstances. In some case the PSO performs better than the GSA and in some not. The cost and resource management of both the methods change with the scenarios. It has been observed that GSA is more costly w.r.t time while PSO is considerably faster than GSA.

The fitness percentage is calculated as:

$$\text{Fitness Percentage} = \frac{\text{Maximum Fitness Achived}}{\text{Global Optimum}} \times 100\%$$

Following are the bar plots comparing Fitness Percentages for the two methods over different Population Sizes.

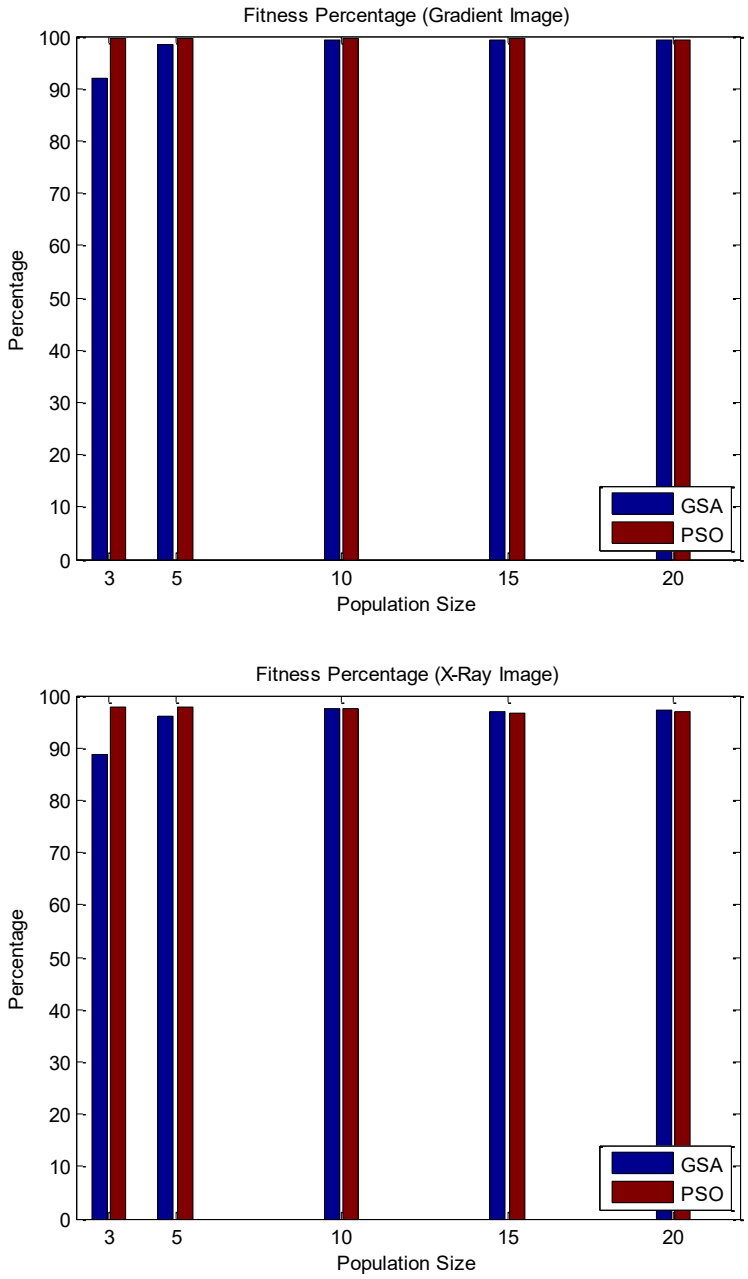

Figure 17 Fitness Percentage

Chapter 8

SUMMARY

8.1 Conclusions

Studying the results of the Optimizer from the Tracker, it can be seen that the Optimizer gives a significant improvement on the results. With the automation of the initialization and optimization of the tracker, and the assurance that the tracker is correctly initialized, the use of the Tracker has been simplified and improved to a great extent. Since this application is going to be used offline without having any issue of time, many parameters can be optimized in combination for the given scenario.

The Optimizer is a separate module, independent of the tracker, therefore it can be easily used with any general optimization problem, and it can be set up to figure out the optimization method automatically if unsure about which one to use.

8.2 Future Work

The optimizer is fully functional and supports all the features mentioned in earlier chapters however we have some future work in mind to improve it further. Following are the additional features we will be working on:

8.2.1 Simulated Annealing (SA)

The Simulated Annealing is a heuristic search algorithm, which starts the search from a single point and continues in a sequential manner. This might significantly reduce the number of fitness calculations carried out to find the optimal solution, since both the algorithms supported in the application are parallel search ones.

8.2.2 MatLab Configurator

There is work in progress on a C++ application that will take in a Default (template) file for a configuration manger and convert it into a MatLab (.m) file that generates a working GUI similar to the ConfigManager. This can then be used with the optimizer to process solutions.

8.2.3 Fitness Approximation

In most real EA applications the evaluation of the fitness function is the most computationally complex and expensive part of the algorithm. To lower the cost of computation fitness, approximation techniques could be applied to make the fitness function simpler. Fitness approximation techniques involve building a simpler fitness model from known fitness values by learning from them and interpolating them. These techniques include lowering the polynomial degree of the original fitness function, regression analysis, artificial neural networks, adaptive fuzzy fitness granulation etc. When working with high dimensional problems with limited number of training samples it is difficult to build an efficient approximate model of the fitness function and using them for evolutionary algorithms may result in the algorithm converging to a local optima. In some cases it is helpful to use a fitness function together with the approximate model but due to the complexity of the tracking problem fitness approximation was not applicable.

Adaptive Fuzzy Fitness Granulation [42]

Adaptive Fuzzy Fitness Granulation (AFFG) is a technique for approximating a model for the fitness function to reduce computational costs. In AFFG, a pool of solutions, represented by

fuzzy granules with finesses calculated by the original fitness function are maintained. If a new solution (granule) is sufficiently close to an existing fuzzy granule from the pool then the new granule's fitness is approximated with the known granules fitness. Otherwise if the new granule is considerably different from the ones in the pool, it is added as a new fuzzy granule to the pool. The size of the pool as well as the radius of influence of each granule is adaptive and changes with the utility of each granule and the overall population fitness.

To reduce the number of fitness evaluations the radius of influence of each granule is kept large in the beginning and is gradually reduced as time passes. This results in more fitness evaluations when the completion is fierce among similar and converging solutions. Granules that are not used are slowly removed from the pool to prevent it from becoming too big.

This method might prove very helpful with the algorithms in this application and we will work to incorporate this to reduce our computational costs.

Appendix A.

OPTIMIZER FOR GENERAL USE

The optimizer, although designed to optimize the tracker, can also be used for general purpose multi-objective, multi-dimensional optimization. The optimizer can be linked to fitness-evaluator function that can take in input values, test them and give out an array of evaluation parameters and their values. The pointer to the function can be specified in the configuration file. Since nothing in the optimizer is hard coded and all the input variables and evaluation parameters are read through input files, it can easily be used for any purpose. Following are the features and specification for the optimizer for general use.

Fitness Calculator

The function called to check the output value of the given solution is known as the fitness calculator. This function takes in the candidate solution, calculates the value of that solution, then runs an evaluation on it and sends back the evaluation parameters and their values in a list.

By-Passing ConfigManager

The ConfigManager is used by the optimizer to write the candidate solutions in their project xml files. However if a problem does not require the inputs through an xml file, the ConfigManager can easily be bypassed and the values can be sent to the fitness calculator function mentioned above.

Appendix B.

CONFIGURATION FILE

```
<Optimizer>
  <ShowWarnings>Yes</ShowWarnings>
  <DeleteOldResults>Yes</DeleteOldResults>
  <Method>GravitationalSearch</Method>
  <SaveTopFiles>no</SaveTopFiles>
  <OrignalFiles>
    <DeleteBackup>1</DeleteBackup>
    <ReplaceWithOptimized>1</ReplaceWithOptimized>
  </OrignalFiles>
  <GravitationalSearch>
    <MaxRuns>350</MaxRuns>
    <Samples>25</Samples>
    <G0>100</G0>
    <Alpha>20</Alpha>
  </GravitationalSearch>
  <ParticleSwarm>
    <MaxRuns>350</MaxRuns>
    <Samples>25</Samples>
    <w>1</w>
    <phiP>1</phiP>
    <phiG>2</phiG>
  </ParticleSwarm>
  <LanguageFile Path_Type="RelativeToSelf">Optim.lang</LanguageFile>
</Optimizer>
```

Appendix C.

INPUT FILES

Project File

```
<Project>
  <Settings>
    <Method Type="GravitationalSearch">
      <GravitationalSearch>
        <MaxRuns>350</MaxRuns>
        <Samples>25</Samples>
        <G0>100</G0>
        <Alpha>20</Alpha>
      </GravitationalSearch>
    </Method>
  </Settings>
  <Input>
    <ConfigManager>
      <DefaultFiles Path_Type="RelativeToSelf">
        ../config/default_tracker.config</DefaultFiles>
      <MakeRelativePath Path_Type="RelativeToSelf" />
    </ConfigManager>
    <TrackerPath Path_Type="RelativeToSelf">
      MultiTrack.exe</TrackerPath>
    <TrackerProjectFile Path_Type="RelativeToSelf">
      3d_spherical/project.xml</TrackerProjectFile>
    <PEPath Path_Type="RelativeToSelf">
      release/PerfEval_x32_release.exe</PEPath>
    <PEProjectFile Path_Type="RelativeToSelf">
      projects/pe/project.xml</PEProjectFile>
    <PEOutputFile Path_Type="RelativeToSelf">
      results/evaluation_results.xml</PEOutputFile>
    <VariablesToOptimize Path_Type="RelativeToSelf">
      Variables.xml</VariablesToOptimize>
    <Metrics Path_Type="RelativeToSelf">common/Metrics.xml</Metrics>
    <PE_Weights Path_Type="RelativeToSelf">PE_Weights.xml</PE_Weights>
    <PV_Relations Path_Type="RelativeToSelf">PV_Rel.xml</PV_Relations>
    <TopN>10</TopN>
  </Input>
  <Output>
    <Report Path_Type="RelativeToSelf">results/report.xml</Report>
    <ResultsFolder Path_Type="RelativeToSelf">results</ResultsFolder>
  </Output>
</Project>
```

Weights File

```
<Weights>
  <Metric Name="RMSE" Weight="1.0" />
  <Metric Name="AveEucError" Weight="0.0" />
  <Metric Name="AveGeomError" Weight="0.0" />
  <Metric Name="AveHarmError" Weight="0.0" />
  <Metric Name="MedianError" Weight="0.0" />
  <Metric Name="FalseTrackRate" Weight="10" />
  <Metric Name="NumFalseTrack" Weight="0.0" />
  <Metric Name="CumBroken" Weight="0.0" />
  <Metric Name="Completeness" Weight="0.0" />
  <Metric Name="NumMissedTrack" Weight="0.0" />
  <Metric Name="RedundantTrackRatio" Weight="0.0" />
  <Metric Name="SpuriousTrackRatio" Weight="0.0" />
  <Metric Name="NumSpuriousTracks" Weight="0.0" />
  <Metric Name="TrackContinuity" Weight="0.0" />
  <Metric Name="TrackFragment" Weight="0.0" />
  <Metric Name="ConfLatency" Weight="10" />
  <Metric Name="TrackPD" Weight="0" />
  <Metric Name="CumSwap" Weight="0" />
</Weights>
```

Variables File

```
<Variables>
  <File Name="tracker.xml" Path_Type="RelativeToSelf">
    <Control Name="track_confirmation_for_display_m">
      <Ranges>
        <Range Name="Range_1">2:1:4</Range>
      </Ranges>
    </Control>
    <Control Name="track_confirmation_for_display_n">
      <Ranges>
        <Range Name="Range_1">2:1:4</Range>
      </Ranges>
    </Control>
  </File>
  <File Name="sensor.xml" Path_Type="RelativeToSelf">
    <Control Name="target_detection_probability">
      <Ranges>
        <Range Name="Range_1">0.90:0.01:0.98</Range>
      </Ranges>
    </Control>
  </File>
</Variables>
```

Metrics File

```
<Metrics>
  <Metric>
    <Name>RMSE</Name>
      <LongName>root_mean_square_error</LongName>
        <Type>LinearDirect</Type>
        <IdealValue>0</IdealValue>
        <Minimize>1</Minimize>
    </Metric>
    <Metric>
      <Name>AveEucError</Name>
      <LongName>average_euclidean_error</LongName>
      <Type>LinearDirect</Type>
      <IdealValue>0</IdealValue>
      <Minimize>1</Minimize>
    </Metric>
    <Metric>
      <Name>AveGeomError</Name>
      <LongName>average_geometric_error</LongName>
      <Type>LinearDirect</Type>
      <IdealValue>0</IdealValue>
      <Minimize>1</Minimize>
    </Metric>
    <Metric>
      <Name>AveHarmError</Name>
      <LongName>average_harmonic_error</LongName>
      <Type>LinearDirect</Type>
      <IdealValue>0</IdealValue>
      <Minimize>1</Minimize>
    </Metric>
</Metrics>
```

PV Relations File

```
<PV_Relations>
  <RMSE>
    <Unknown>
      <File Name tracker.xml" PathType="RelativetoApplication">
        A;B;C</File>
    </Unknown>
    <Linear_D>
     <File Name tracker.xml" PathType="RelativetoApplication">
        D;E;F</File>
    </Linear_D>
  </RMSE>
  <AveEucError>
    <Unknown>
      <File Name tracker.xml" PathType="RelativetoApplication">
        A;B;C</File>
    </Unknown>
    <Linear_D>
     <File Name tracker.xml" PathType="RelativetoApplication">
        X;Y;Z</File>
    </Linear_D>
    <Inverse>
     <File Name sensor.xml" PathType="RelativetoApplication">
        J;K;L</File>
    </Inverse>
</PV_Relations>
```

Appendix D.

OUTPUT FILE

Report

```
<Optimizer>
  <Combination Number="1" Score="10">
    <File Name="test2.xml" PathType="RelativeToApplication">
      <A>6.957507,6.964889</A>
      <C>two</C>
      <B>1</B>
    </File>
  </Combination>
  <Combination Number="2" Score="9">
    <File Name="test2.xml" PathType="RelativeToApplication">
      <A>6.701281,6.627104</A>
      <C>two</C>
      <B>1</B>
    </File>
  </Combination>
  <Combination Number="3" Score="8">
    <File Name="test2.xml" PathType="RelativeToApplication">
      <A>6.592305,7.000000</A>
      <C>two</C>
      <B>-1</B>
    </File>
  </Combination>
  <Combination Number="4" Score="8">
    <File Name="test2.xml" PathType="RelativeToApplication">
      <A>6.682393,6.686016</A>
      <C>two</C>
      <B>1</B>
    </File>
  </Combination>
    <Combination Number="5" Score="7.5">
     <File Name="test2.xml" PathType="RelativeToApplication">
      <A>6.534048,6.000000</A>
      <C>three</C>
      <B>1</B>
    </File>
  </Combination>
</Optimizer>
```

Bibliography

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

[2] R. B. Devi, E. Barlaskar, O. B. Devi, S. P. Medhi and R. R. Shimray, "Survey on Evolutionary Computation Tech Techniques and its Application in Different Fields," *International Journal on Information Theory (IJIT),* vol. 3, pp. 73-82, July 2014.

[3] W. Contributors, "Wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/Evolutionary_algorithm.

[4] C. Ferreira, "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems," *Complex Systems,* vol. 13, no. 2, pp. 87-129, 2001.

[5] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing,* vol. 4, no. 2, pp. 65-85, 1994.

[6] L. O. A. W. M. Fogel, Artificial Intelligence through Simulated Evolution, John Wiley, 1966.

[7] H.-G. Beyer, The Theory of Evolution Strategies, Springer, 2001.

[8] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization ,* vol. 11, p. 341–359, 1997.

[9] P. Civicioglu, "Backtracking Search Optimization Algorithm for numerical optimization problems," *Applied Mathematics and Computation,* vol. 219, pp. 8121–8144,, 2013.

[10] T. White, "Swarm Intelligence," [Online]. Available:

http://www.sce.carleton.ca/netmanage/tony/swarm.html.

[11] W. J. G. Beni, "Swarm Intelligence in Cellular Robotic Systems," in *Proceed. NATO Advanced Workshop on Robots and Biological Systems*, Tuscany, Italy, June 26–30 (1989).

[12] J. E. R. C. S. Y. Kennedy, Swarm Intelligence, San Francisco: Morgan Kaufmann Publishers, 2001.

[13] E. Rashedi, H. Nezamabadi-pour and S. Saryazdi, "GSA: A Gravitational Search Algorithm," *Information Sciences,* vol. 179, no. 13, p. 2232–2248, 13 June 2009.

[14] M. Dorigo and T. Stützle, Ant Colony Optimization, MIT Press, 2004.

[15] D. D. Karaboga, "An Idea Based On Honey Bee Swarm for Numerical Optimization," Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[16] H. W. Ben Niu, "Bacterial Colony Optimization," *Discrete Dynamics in Nature and Society,* vol. 2012, 2012.

[17] L. N. de Castro and J. Timmis, "Artificial Immune Systems: A New Computational Intelligence Approach," Springer, 2002, p. 57–58.

[18] S. Mirjalili, S. M. Mirjalili and A. Lewis, "Grey Wolf Optimizer," *Advances in Engineering Software,* vol. 69, p. 46–61, 2014.

[19] X. S. Yang, "A New Metaheuristic Bat-Inspired Algorithm, in: Nature Inspired Cooperative Strategies for Optimization," *Studies in Computational Intelligence,* vol. 284, pp. 65-74, 2010.

[20] V. K. Jiří Pospíchal, "A Study of Altruism by Genetic Algorithm," in *Advances in Soft Computing*, Springer, pp. 507-520.

[21] S. Abraham, S. Sanyal and M. Sanglikar, "Particle Swarm Optimization Based Diophantine Equation Solver," *International Journal of Bio-Inspired Computation,* vol. 2, no. 2, pp. 100-

114, 2010.

[22] W. contributors, "Particle swarm optimization," 18 August 2014. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Particle_swarm_optimization&oldid=621827390. [Accessed 18 September 2014].

[23] M. Dorigo, V. Maniezzo and A. Colorni, "Distributed Optimization by Ant Colonies," *ECAL91 - European Conference On Artificial Life,* pp. 134-142, 1991.

[24] A. Kutsenok, "Swarm AI: A General-Purpose Swarm Intelligence Technique," *Design Principles and Practices: An International Journal,* vol. 5, pp. 7-16, 2011.

[25] K. Roebuck, User Generated Content: High-impact Emerging Technology - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors, Emereo Publishing, 2004.

[26] M. A. Lewis and G. A. Bekey, "The Behavioral Self-Organization of Nanorobots Using Local Rules," in *In Proceedings of the 1992 IEEE/RSJ IROS*, Raleigh, North Carolina, 1992.

[27] M. Al-Rifaie and A. Aber, " Identifying metastasis in bone scans with Stochastic Diffusion Search," in *Proc. IEEE Information Technology in Medicine and Education, ITME* , 2012.

[28] M. M. Al-Rifaie, A. Aber and A. M. Oudah, "Utilising Stochastic Diffusion Search to identify metastasis in bone scans and microcalcifications on mammographs," in *IEEE International Conference on Bioinformatics and Biomedicine Workshops (BIBMW)*, 2012.

[29] S. Blackman and R. Popoli, Design and Analysis of Modern Tracking Systems, Artech House Publishers, 1999.

[30] R. Tharmarasa, K. Punithakumar, T. Kirubarajan and Y. Bar-Shalom, "Sensor Data Fusion with Application to Multitarget Tracking," in *Handbook on Array Processing and Sensor Networks*, John Wiley & Sons, 2010 .

[31] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Fluids Engineering :82,* pp. 35-45, 1960.

[32] R. Tharmarasa, "Large-Scale Optimal Sensor Array Management for Multitarget Tracking," Hamilton, Ontario, Canada, 2003.

[33] Y. Bar-Shalom, X. Li and T. Kirubarajan, Estimation with Applications to Tracking and Navigation, Wiley, 2001.

[34] R. Mahler, "An Introduction to Multisource-Multitarget Statistics and its Application," Lockheed Martin, 2000.

[35] M. Pace, B. N. V. (. Melbourne), D. L. (DCNS), P. D. Moral and F. Caron., "Multiple target Tracking with PHD Filters," ALEA, [Online]. Available: https://alea.bordeaux.inria.fr/index.php/phd-filters. [Accessed 14 07 2015].

[36] H. A. P. Blom, "A Sophisticated Tracking Algorithm for ATC Surveillance Data," in *Proceedings of International Radar Conference*, Paris, 1984.

[37] H. A. P. Blom and Y. Bar-Shalom, "The Interacting Multiple Model Algorithm for Systems with Markovian Switching Coefficients," *IEEE Transactions on Automatic Control,* vol. 33 , no. 8, pp. 780- 783, 1988 .

[38] *Performance Evaluator - User Guide,* 2012.

[39] R. A. Coogle and P. F. West, "MIMO radar benchmark: performance metrics," *ONR Workshop on MIMO Radar,* July 2009.

[40] H. Ahmed and J. Glasgow, "Swarm Intelligence: Concepts, Models and Applications," Kingston, Ontario, February 2012.

[41] T. White, B. Pöltl, D. Agrinz, A. Maier-Pilecky, M. Moser and T. Russold, "SFB Research Center," University of Graz , [Online]. Available: http://math.uni-graz.at/mobis. [Accessed 07 09 2015].

[42] M.-R. Akbarzadeh-T, M. Davarynejad and N. Pariz, "Adaptive fuzzy fitness granulation for evolutionary optimization," *International Journal of Approximate Reasoning (Impact Factor: 1.98),* vol. 49, p. 523–538, 2008.

[43] U. Orguner, "Introduction to Target Tracking," Ankara.

[44] N. J. Gordon, D. J. Salmond and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," in *IEE Proceedings-F, Radar and Signal Processing*, 1993.

[45] N. Amjady and H. Soleymanpour, "Daily Hydrothermal Generation Scheduling by a new Modified Adaptive Particle Swarm Optimization technique," *Electric Power Systems Research,* vol. 80(6), pp. 723-732, 2010.

[46] R. C. Eberhart and Y. Shi, "Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization," *Proceedings of IEEE International Congress on Evolutionary Computation,* vol. 1, pp. 84-88, 2000.

[47] N. Nedjah and L. d. Macedo Mourelle, Swarm Intelligent Systems, vol. 26, Rio de Janeiro: Springer-Verlag Berlin Heidelberg, 2006.