

# **COOPERATIVE CONTROL AND OPTIMIZATION OF ROBOTIC AGENTS**

By

**OBADAN SAMUEL OHIFEMEN**

Submitted in accordance with the requirements  
for the degree of

**DOCTOR PHILOSOPHIAE**

in the subject

COMPUTER SCIENCE

at the

UNIVERSITY OF SOUTH AFRICA

SUPERVISOR: PROF. ZENGHUI WANG

JUNE 2022

## ABSTRACT

The cooperate behavior that emerges from the interactions among simple multi-agent robots along with the solution possibilities these interactions provide, has formed part of the growing research areas in recent years within the confines of artificial intelligence. In this thesis, we explore these possibilities leveraging single and multi-objective optimization on a machine-learning algorithm: the artificial neural network. The rationale is to achieve goal oriented collaborative control (group behavior) with regard to localizing multiple gradient sources during search operations. With a view to solving the multisource localization problem, we develop an ingenious hybrid metaheuristics algorithm for optimizing exploration (search-oriented) and exploitation (goal-oriented) in both fully observable and partially observable domains. We compared the performance of our model with the existing state of the art metaheuristic algorithms such as Simulated Annealing (SA), Cuckoo-search (CK), Particle Swarm Optimization (PSO) and Genetic Algorithm (GA) using 30 standard benchmark complex objective functions. Results showed significant improvement. The homing (goal-oriented) operation introduced novel concepts for accelerating off-policy reinforcement learning algorithm for Partially Observable Markov Decision Processes (POMDP) via dynamic programming on a multi-agent framework. Finally, we demonstrated an ingenious approach to the resampling phase of Monte Carlo's particle filter (for robot localization) which showed relatively significant improvement in the belief state estimation accuracy with respect to ground truth within POMDP domains. The contribution of this research is twofold: firstly, it presents a framework for search optimization while localizing multiple emission sources. Secondly, it presents ingenious concepts for foraging, gradient source localization along with the potential for search and rescue operations within POMDP environments.

Key terms: Genetic algorithm, Dynamic programming, Evolutionary Neural networks, Cooperation, Search optimization, Hybridization, Reinforcement Learning, Particle filters, Markov decision processes, POMDPs.

## DECLARATION

Name: Obadan Samuel Ohifemen

Student number: 55769438

Degree: PhD (Computer Science)

Exact wording of the title of the dissertation or thesis as appearing on the copies submitted for examination:

Cooperative control and optimization of Robotic agents

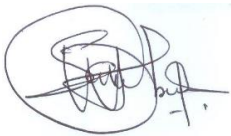
---

---

---

---

I declare that the above dissertation/thesis is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.



\_\_\_\_\_  
SIGNATURE

\_\_\_\_\_  
30/3/2022  
DATE

## **ACKNOWLEDGEMENTS**

My sincere gratitude goes to God Almighty, the giver of life and health, the only wise God, without whom I am nothing.

To my darling spouse Diana whom encouraged and supported me all the way, and my loving daughters Sapphire, Sharon, and Shiloh from whom I draw strength.

To my mother and Siblings for your support and prayers which I consider priceless.

To the best supervisor in the world, Prof. Zenghui Wang. For your fatherly role and mentorship, professionalism and dedication, all I can say is THANK YOU!

## LIST OF PUBLICATIONS

1. Samuel Obadan, Zenghui Wang, A HYBRID OPTIMIZATION APPROACH FOR COMPLEX NONLINEAR OBJECTIVE FUNCTIONS. International Journal of Computing, 17(2), pp. 102-112, 2018.
2. Samuel Obadan, Zenghui Wang, A MULTI-OBJECTIVE OPTIMIZATION APPROACH TO ROBOT LOCALIZATION OF SINGLE AND MULTIPLE EMISSION SOURCES. Procedia Manufacturing, 35, pp. 755-761, 2019.
3. Samuel Obadan, Zenghui Wang A MULTI-AGENT APPROACH TO POMDPS USING OFF-POLICY REINFORCEMENT LEARNING AND GENETIC ALGORITHMS. International Journal of Computing, 19(3), pp. 377-386, 2020.

# TABLE OF CONTENTS

## 1 INTRODUCTION

1.1 Overview -----	14
1.2 Problem Statement -----	15
1.3 Solution Approach -----	16
1.4 Research Context -----	16
1.5 Scope of Study -----	17
1.6 Synopsis -----	18

## 2 LITERATURE REVIEW

2.1 Introduction -----	19
2.2 Hill Climbing Algorithm (The successful single source localization) -----	20
2.3 Biologically Inspired Algorithms -----	21
2.3.1 Glowworm Swarm Optimization (GSO) -----	21
2.3.2 Biasing Expansion Swarm Approach (BESA) -----	22
2.3.3 Particle Swarm Optimization (PSO) -----	24
2.3.4 Biased Random Walk Algorithm (BRW) -----	25
2.3.5 Attractant-Repellent Swarm -----	26
2.4 Probabilistic Algorithms -----	27
2.4.1 Occupancy Grid Mapping using Bayesian Techniques -----	27
2.5 Comparative Analysis -----	30
2.6 Conclusion -----	34

### **3 FORMAL BACKGROUND**

3.1 Introduction	35
3.2 Metaheuristic Algorithms	40
3.2.1 Evolution computation	41
3.2.2 Simulated Annealing (SA)	41
3.2.3 Particle Swam Optimization (PSO)	42
3.2.4 Cuckoos Search algorithm (CS)	43
3.2.5 Hybrid Optimization	44
3.2.6 Evolution Neural Networks	44
3.3 Multi-Objective Optimization	46
3.3.1 Multi-Objective domination	46
3.3.2 The Non-dominated Sorting Genetic Algorithm (NSGA-II)	47
3.4 Reinforcement Learning	47
3.5 MDP and POMDPs	48
3.6 Particle Filters algorithm	50
3.7 Conclusion	52

### **4 A HYBRID METAHEURISTIC OPTIMIZATION PARADIGM**

4.1 Introduction	53
4.2 Hybrid Frame Work Methodology	53
4.3 Polygamy as an Exploitative Strategy for GAs	56
4.4 Experiments	57
4.5 Summary of Results	62
4.5.1 Hypothesis 1 (H1)	66
4.5.2 Hypothesis 2 (H2)	66
4.6 Discussion	70
4.7 Conclusion	71

## **5 MULTI-OBJECTIVE OPTIMIZATION APPROACH TO LOCALIZING MULTIPLE EMISSION SOURCES**

5.1 Introduction	72
5.2 Methodology	73
5.2.1 ENNs (Evolutional Neural Networks) using Discrete Variables	74
5.2.2 Discrete variable encoding and decoding	74
5.2.3 Objective Function	74
5.2.4 The population	75
5.2.5 Constraints	75
5.2.6 Natural selection	76
5.2.7 Crossover (Mating)	77
5.2.8 Mutation:	77
5.2.9 The next Generation	78
5.3 ENNs (Evolutional Neural Networks) using Continuous Variables	80
5.3.1 Polygamy: a Mating Strategy	80
5.3.2 Choosing Best Practice	82
5.3.3 Impact of hydrodynamics on the Robot's search performance	85
5.3.4 Experiment and results	86
5.3.5 Implementing Particle filters: based on TDOA	90
5.4 Multi-Objective Optimization Approach to Multi-Source Localization	92
5.4.1 Sensory feed backs	93
5.4.2 Dynamic programming (DP) for modeling multiple gradient sources	94
5.4.3 Calculating Fitness values (F1 and F2)	95
5.5 Multi-Objective Results	97
5.6 A Comparative Analysis with two Existing Models	100
5.7 Conclusion	102



## **6 AGENT AND SOURCE LOCALIZATION IN POMDP ENVIRONMENTS**

6.1 Introduction -----	103
6.2 Methodology -----	104
6.3 Experiments and Results -----	106
6.4 Monte Carlos Resampling Model -----	109
6.5 The AMCL (Adaptive Monte Carlos Localization) approach -----	113
6.6 Discussion of Results -----	115
6.7 Conclusion -----	115

## **7 SUMMARY AND FUTURE WORK**

7.1 Introduction -----	117
7.2 Adapted hybrid model -----	117
7.3 A Comparative Analysis with Existing Models -----	120
7.4 Conclusion -----	122
7.5 Practical implementations -----	123
7.6 Future research -----	123

<b>REFERENCES</b>	<b>125</b>
<b>ADDENDUM A</b>	<b>136</b>
<b>ADDENDUM B</b>	<b>169</b>

## LIST OF FIGURES

Fig. 2.1 the Biasing Expansion Swarm Approach. -----	22
Fig. 3.1 Flowchart for genetic algorithm. -----	41
Fig. 3.2 A two input multi-layer evolutionary neural network structure -----	45
Fig. 3.3 Schematic for NSGA-II algorithm -----	47
Fig. 4.1 High level abstraction of the hybrid frame work -----	54
Fig. 4.2. Hybrid frame work performance chart for all objective functions evaluations. -----	57
Fig. 5.1 Instantiation of a spool of 100 evolving robots within the obstacle oriented grid map --	75
Fig. 5.2 Double brain feed forward neural network. -----	78
Fig. 5.3 comparing fitness performances for the traditional ENNs with Double Brain -----	79
Fig. 5.4 comparing fitness performances for the traditional ENNs with Double Brain ENNs ----	80
Fig. 5.5 comparing fitness performances for the polygamy enhanced ENNs Roulette wheel ----	82
Fig. 5.6 comparing performances of polygamy with double brain ENN (db poly 5:5), Boltzmann Scaling -----	83
Fig. 5.7 Graphical representations of fitness scores -----	85
Fig. 5.8 Revised neural network architecture for maneuvering in a hydrodynamic environment.-	85
Fig. 5.9 (a) Flows emanating from different directions: North->South, West -> East, East-> West, South->North, and South-west ->North-East at a 45 <sup>o</sup> angle. (b) Robot equipped with 5 sensors separated at 45 degrees interval -----	86
Fig. 5.10 Graphical representation of fitness scores of the best evolving robot for low, medium, high, and random tide, over 50 generations in a hydrodynamic environment -----	88
Fig. 5.11 Graphical representation of fitness scores of the best evolving robot for diagonal low, medium, high, and random tide, over 50 generations in a hydrodynamic environment -----	90
Fig. 5.12 Scenario 1 the robotic agent equipped with 5 sensor inputs into its feed-forward neural network (brain) with 2 gradient sources in RED -----	92
Fig. 5.13 Optimization architecture pipeline -----	93
Fig. 5.14 Scenario 2 the robotic agent equipped with 5 sensor inputs into its feed-forward neural network (brain) with 2 gradient sources in RED. -----	94
Fig. 5.15 modeling multiple emission sources on a grid world. -----	95

Fig. 5.16 Scenario 3 the robotic agent equipped with 5 sensor inputs into its feed-forward neural network (brain) with 2 gradient sources in RED. -----	96
Fig. 5.17 Scenario 4 the robotic agent equipped with 5 sensor inputs into its feed-forward neural network (brain) with 2 gradient sources in RED. -----	97
Fig. 5.18 the Pareto-optimal front for Scenario 1 -----	97
Fig 5.19 the Pareto-optimal front for Scenario 2 -----	98
Fig. 5.20 the Pareto-optimal front for Scenario 3 -----	98
Fig.5.21 the Pareto-optimal front for Scenario 4 -----	99
Fig. 6.1 Multi-agent RL environment. With walls (white cells), absorbing states (red cells), dynamic door (blue cell) and goal node (green cell). -----	106
Fig. 6.2 Single agent reinforcement learning graph with respect to CPU-time -----	107
Fig. 6.3 Multi-agent (size of 4) reinforcement learning graph with respect to CPU-time -----	107
Fig. 6.4 Multi-agent (size of 5) reinforcement learning graph with respect to CPU-time -----	108
Fig. 6.5 Multi-agent (size of 4) feedforward neural network (with GA) learning graph with respect to CPU-time, and Epochs. -----	109
Fig. 6.6 Agent motion model for POMDPs -----	109
Fig. 6.7 Process flow diagram for traditional resampling and localization of belief state using particle filters. Capital ‘A’ (Initial Random sample), Lowercase ‘a’ (resampled) -----	110
Fig. 6.8 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for traditional resampling. -----	110
Fig. 6.9 Extended Process flow diagram for traditional resampling and localization of belief state using particle filters -----	111
Fig. 6.10 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for double phased resampling. -----	111
Fig. 6.11 Extended Process flow diagram for traditional resampling and localization of belief state using particle filters. -----	111
Fig. 6.12 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for triple phased resampling -----	112
Fig. 6.13 Modified Process flow diagram for traditional resampling and localization of belief state using particle filters. Capital ‘(A, B, C D)’ (Initial Random sample), Capital A (selected sample), Lowercase ‘a, b, c’ (resampled) with triple phased resampling. -----	112

Fig. 6.14 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for preprocessed initialization with triple phased resampling. -----113

Fig. 6.15 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for the AMCL (KLD) -----114

Fig. 6.16 (a) True negatives (the agent believes it's in a wall (belief in RED) when it's actually not), (b) False positives (the agent believes it's not in a wall, when it actually is). (c) True positives (agents position and belief are approximately same). -----114

Fig. 7.1 Adapted hybrid model -----119

## LIST OF TABLES

Table 2.1 Multisource algorithms summary -----	30
Table 2.2. Other comparisons on odor-source localization -----	31
Table 4.1. Algorithm for Hybridized optimization Model -----	55
Table 4.2 Minimum optimal values for each meta-heuristic $f(X^*) =$ cross validating global optimal Values. -----	59
Table 4.3 ANOVA analysis of mean function calls of each meta-heuristic for all objective functions. (* indicates “no significant difference”) -----	60
Table 4.4. ANOVA analysis of mean global optimal values of each meta-heuristic for all objective functions (* indicates “no significant difference”) -----	61
Table 4.5. (a) Box plot representation of ANOVA on the mean number of function calls (H1). ANOVA on hypothesis 1(H1)- the mean number of function calls -----	67
4.5. (b.) ANOVA on hypothesis 2(H2)- the mean best optimal value obtained for each meta-heuristic algorithm. -----	67
Table 5.1 Effect of polygamy on ENN using Roulette wheel (R001 and R01) Tournament selection (T001 and T01). Were 001 represents a mutation rate of 0.001 and 01 a mutation rate of 0.01.--	81
Table 5.2 Comparing performances of polygamy on double brain ENN, Boltzmann Scaling, polygamy on ENN with tournament selection method and a fixed mutation rate of 0.01 -----	83
Table 5.3 Comparing performances of polygamy with double brain ENN (db poly 5:5), Boltzmann Scaling (Boltzmann), with four other meta-heuristic optimizers. -----	84
Table 5.4 Summary Results for Trial 1 to 4 -----	87
Table 5.5 Results for Trial 5 -----	89
Table 5.6 Adapted Summary of multi-search options and their characteristics. -----	100
Table 7.1 Algorithm for the adapted hybrid model -----	118
Table 7.2 (a) Adapted Summary of multi-search options and proposed model -----	121
Table 7.2 (b) Adapted Summary of multi-search options and proposed model -----	122

# CHAPTER 1

## INTRODUCTION

### 1.1 Overview

Early implementations of collaborative robotics date back to the robot soccer tournaments where teams of robots equipped with sensors, cameras, and internal processors, simultaneously recognize the ball, sidelines, goal post as well as differentiate between team mates. Most of the components that were originally designed for robot soccer tournaments have made their way into other applications, such as localization, search and rescue operations, reconnaissance, lawn mowing and domestic vacuum cleaners. The same concept used in Robot soccer are present even in more advanced applications of autonomous robots. Roboticists are constantly in the art and science of evolving algorithms to improve robots perception and navigation within disparate environments. Some robot designs are constrained to operate within familiar environments such as can be found with the lawn mowing robots, however on the flip side, an office-mowing robot may require a more complex map building algorithm to aid navigation. More advanced models could analyze and adapt to new environments via simultaneous mapping and localization algorithms even within complex terrains. An example can be seen with the Rover Robot which builds a map using its onboard sensors there by guiding the trajectory of the robot by avoiding unpleasant locations or terrains. This is the underlying principle for the planet Mars exploratory robot.

Cooperation among robotic agents could emerge for the purpose of accomplishing a given goal. There exist two types of cooperation: active and the non-active. In the former, robotic agents exhibit collaborative behavior via an acknowledgement of each other, while in the later, cooperate behavior emerges as each robotic agent embarks on its immediate goal. The phenomenon is also referred to as oblivious cooperation.

The little or no sensing requirement for the design of robotics agents [1-4] in order to achieve cooperate behavior has made this field a focal point of interest within the robotic research community. In this thesis, we investigate the potentials of cooperate behavior among multiple robotic agents for localizing multiple emission sources. In addition, we identify important areas within the enclave of cooperative robots in order to propose a methodology for oblivious collaboration among robotic agents when simultaneously localizing multiple gradient sources.

With recent advances in the field of artificial intelligence (AI), there has emerged a wide range of efficient algorithms [5] which when skillfully hybridized could result in a plausible model for solving some of the problems in the field. This thesis also leverages on some of the existing AI optimization techniques such as Genetic Algorithms, Particle Swarm Optimization (PSO), Cuckoos Search algorithm (CS), and Simulated Annealing (SA) [6-9]. The learning paradigm leveraged on models such as ANNs (artificial Neural Networks) and RL (Reinforcement learning), and finally for state estimation; the Particle filter algorithm. Since Markov Decision Processes (MDPs) do not provide an accurate representation of the real world domain [10-12], due to the uncertainty or rather incomplete knowledge of the state of the environment, we adopted the POMDPs- Partially Observable Markov Decision Processes model for our final simulation.

## **1.2 Problem Statement**

Among the existing algorithms for multisource localization, we do not find clear recommended techniques for robotic agents to proceed with a search after a source is found. The missing piece for the multisource localization problem is to find theoretic frameworks that would guarantee progress (after a source has been localized), while driving the algorithm towards convergence, and a proper termination of the search algorithm. Gazi and Passino [4] works on “attractant/repellent swarm” has come close to solving this problem but however, the work was limited to single-source searches.

When solving the multisource localization problems, some factors are taken into consideration; such as: the complexity of the solution, the type of source/target and the predictability of the environmental variables. The Bayesian Occupancy grid algorithm [13] is one commendable attempt in providing solution to the above stated factors. Although this algorithm provides a near optimal solution to the multisource problems, it however does not provide a clear path towards progress after a gradient source has been localized. In addition, the literature lacked adequate comparative analysis with other biological models/algorithms.

Finally, the question on the modalities for choosing the dependent and independent variables for the purpose of a standardized comparative analysis has been left unanswered. Consequently, there is a need for the validation of a variables such as: the initial distribution of gradient sources, the

location and presence of obstacles, etc. Such standards could help in comparing and contrasting different models and algorithms against the back drop of their merits and demerits for disparate domains. In this thesis, we provide an array of groundbreaking paradigms while addressing the problems of the multisource localization.

### **1.3 Aim and Research Objectives**

The objective of this study is to propose a framework for optimizing the exploration and exploitation of multiple agents using a combination of algorithms to achieve oblivious collaboration for solving the multisource localization problem. There have been campaigns in recent years toward creating robots whose primary goal revolves around the need for robotic agents to sample the distribution of gradient sources in order to localize them. These sources could be in any of the following forms: physical, chemical, biological, and sometimes electromagnetic.

Although localizing single gradient sources have received great attention by researchers, the use of multi-agent for localizing multiple gradient sources is still a very potent research axis which has received far less attention comparatively. The challenges of the multi-source localization using multiple agents are in 3 folds: Firstly, there exist the challenge of partitioning the robots during a search in order to optimize the search duration, Secondly is the need for obstacle avoidance along with the avoidance of other robots during the procedure, and Thirdly, the ability to proceed after each source is found. In addition to these primary challenges, there is the quest for a theoretical foundation with a guarantee for progress, convergence and termination of the search procedure. Our objective is to propose a framework that leverages novel machine learning and optimization paradigms to proffer solution to the above mentioned multisource localization problems

The research methodologies involved, literature reviews, [14-17] propositions, simulations, and evaluations. Using a case study of multiple emission source localization problem, our research encapsulated single objective optimization, multi-objective optimization, robot localization, swarm intelligence, POMDP domains, dynamic programming, genetic algorithm with other related meta-heuristics algorithms, particle filters, reinforcement learning and artificial neural networks.

### **1.4 Research methodology**

It is important to note that localization of multiple emission sources differs from that of observation of multiple sources (often referred to as target locations). While the later involves the tracking



multiple target locations simultaneously [18], [19] with locations that are known prior to the mission, the former on the contrary is involved with finding targets with unknown prior locations. The general approach used in this thesis was both quantitative and experimental. Data were collected via simulated iterations on complex multimodal objective functions. In keeping with the tradition of existing literature for optimization problems, Analysis of Variance (ANOVA) was adopted for the statistical evaluation. Computer programs such as C++ for windows 7 and higher, was used for our simulation.

The proposed framework towards the multi-source localization utilized standard measurement matrices in literature such as Swarm size, Source Number, Source type, Source mobility, Variable source intensity, Dead space, Communication range, Agent deployment, Computational complexity, Obstacle avoidance, Sensing requirement etc. for measuring the performance of the proposed algorithm. The overall methodology was aimed at sustaining best practices in answering our research questions and satisfying our research objectives.

Consequently, we adapt a model for the multisource localization problem which leverages on ingenious artificial intelligence techniques such as; a novel hybrid optimization, MDPs and POMDPs, particle filter, dynamic programming, and the multi-objective optimization. The multi-objective optimization uses the machine learning qualities of the artificial neural networks (ANNs) for a group of robots by leveraging on their basic yaw and thrust actuators in order to achieve collaborative control (group behavior) amidst localization of multiple emission sources.

The final model incorporates an additional agent which we refer to in this context as the MDP-agent. While the preliminary test utilized only two agents, the incorporation of the third agent (MDP-agent) brought robustness and a relatively stable solution to the multisource localization problem.

## **1.5 Scope of the Study**

This thesis demonstrates via software simulations the dynamics of Multi-objective evolutionary algorithms combined with artificial neural networks in solving the multisource localization problems of progress (after a source is found), convergence, and termination of the search operation. Consequently, the hardware dynamics and physics of a typical robotic agent in the real world were not accounted for. In addition, we assumed the emission sources were fixed for both

fully and partially observable 2 Dimensional worlds. Finally, the robot motion dynamics were restricted to oblivious collaboration among multiple agents towards a predefined goal.

## **1.6 Structure of the thesis**

- Chapter 2 presents a background theory on which the paradigm for optimizing control of multi-agents is based.
- Chapter 3 covers different implementations proposed by researchers for multi-source localization, along with their merits and demerits.
- Chapter 4 provides a novel optimization metaheuristic algorithm with a comparative analysis of 30 different benchmark multi-modal objective functions.
- In chapter 5, the performance of metaheuristic optimization on artificial neural networks for single and multiple objectives are investigated.
- Novel concepts for localization in a POMDP environment are considered in chapter 6.
- In chapter 7 (summary and future work) we piece together the innovations and algorithms from previous chapters into a holistic model for resolving the multisource problem along with the results from experiments, and thereafter concluded with our recommendations and future directions.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Introduction

There have been campaigns in recent years toward creating robots that can replace humans in a vast domain of applications [20], especially in fields that place humans in harm's way. Voinov and Nosikov [21] demonstrated a typical example of such domains in the handling of nuclear waste. Algorithms investigated in these domains incorporate applications in reconnaissance and surveillance, object identification and localization, localization of chemical, biological, and radiological materials [22, 23]. In cases of natural disasters, search and rescue operations have also gained popularity. Nanorobotics has not been left out of the campaign as interest in detecting infectious cells in human tissue has gained momentum [24].

The primary focus in these applications revolves around the need for robotic agents to sample the distribution of gradient sources in order to localize them. These sources could be in any of the following forms: physical, chemical, biological, and sometimes electromagnetic.

Although localizing single gradient sources has received great attention by researchers, the use of multi-agent for localizing multiple gradient sources is still a very potent research axis which has received far less attention comparatively. The challenges of the multi-source localization using multiple agents are in 3 folds: Firstly, there exists the challenge of partitioning the robots during a Search in order to optimize the search duration; Secondly is the need for obstacle avoidance along with the avoidance of other robots during the procedure [25] and Thirdly, the ability to proceed after each source is found. In addition to these primary challenges, there is the quest for a theoretical foundation with a guarantee for progress, convergence, and termination of the search procedure.

There exist three main paradigms that have attempted to provide general solutions to the multi-source localization problem. The least complicated of them was derived from the gradient ascent techniques that Rozas *et al.* [26] referred to as hill climbing. The other paradigm which is relatively more complicated is inspired by biological systems such as the biological swarms by Krishnanand and Ghose [27]: Gazi and Passino [3] and the E.coli bacteria proposed by Dhariwal *et al.* [28 ].

The third paradigm utilizes probabilistic methods such as Bayesian occupancy grids, proposed by Pang and Farrell [29], and was squealed the following year by Ferri *et al.* [30].

This chapter investigates the contributions of each of these paradigms along with their merits and demerits in solving the multi-source localization problem.

## **2.2 HILL CLIMBING ALGORITHM (Single source localization)**

Some of the earliest success stories for single-source localization can be attributed to the works of Rozas *et al.* [26] which involved robots that could trace a gas concentration gradient to its source using an electronic nose. Subsequent researches showed that localizing a single gradient source could be optimized when more robots are deployed [31]. Some other successful stories involve burrowing robots tracing an underground chemical concentration up its gradient to the source location. The use of mobile sensors has not been left out in the pursuit of the solution to the source localization problem. Bachmayer and Leonard [32], demonstrated a hill-climbing algorithm for a vehicle network that could localize a gradient source via its artificial inter-vehicle ability.

The domain of Nanorobotics finds expression in the hill-climbing algorithm. Hogg [33] implemented a technique such that with the aid of short-range acoustic sensors, Nanorobots could detect and communicate chemical signals produced by the injured cells while migrating towards the cell or tissue. The success of this model can be attributed to a large amount of Nanorobots that are deployed into the bloodstream which aids a downstream communication among the robots.

The success of the hill-climbing so far has been limited to the single source localization. This underpins one of its major drawbacks. The algorithms based on hill-climbing lacks the capacity to guarantee multi-source localization. This is as a result of the algorithm getting stuck at local optima. Several modifications have been adapted in an attempt to circumvent this drawback such as the incorporation of a random walk, a collaborative swarm, or a probabilistic technique.

Although simulated annealing has been shown to be a decent and classical method for escaping local optima, [34] it still lacks the capacity for localizing multiple emission sources.

## **2.3 BIOLOGICALLY INSPIRED ALGORITHMS**

Organisms in nature have inspired a couple of algorithms that leverage on concentration gradients as depicted in the hill-climbing approach discussed previously. The behavior of an organism such

as the male silkworm moth as demonstrated by Lilienthal *et al.* [35] used pheromone signal to localize a mate. Almost concurrently, Hayes *et al.*, [36] implemented a similar algorithm with the advantage of a distributed and cooperative search based on background flow of odor and wind measurements.

The most promising biological models have come from swarming algorithms. Racy swarming algorithms have been inspired by the foraging behavior of the E.coli bacteria in the presence of multiple gradient sources. In the same vein, there exists a swarm algorithm called "fluxtaxis" which combines fluid dynamics with swarm control to trace chemical plume [37]. The ability of this swarm to converge and diverge based on the local chemical concentration makes it a promising paradigm for swarm algorithms. In the subsequent sections, we discuss some of these biological models in nature that attempt to solve some of the multi-source localization problems.

### **2.3.1 Glowworm Swarm Optimization (GSO)**

The glowworm swarm optimization (GSO) makes use of an adaptive domain which helps in the creation of subgroups in the swarm while localizing multiple gradient sources simultaneously [38]. The algorithm begins with a uniform distribution of the robotic agents in the search space. Having each agent equipped with a local sensory range, the agent closest to an emission source would have the brightest glow. Neighboring agents with a relatively high probability choose this agent as their leader and navigates towards them as the search progresses.

This local decision domain among neighboring agents is largely responsible for the swarm partitioning during the search phase [39]. Central to this algorithm is the separation of the local decision domain from the sensory domain for each agent. This ingenuity helps create a decision domain that is smaller than its sensor range. Since the algorithm is initialized with a uniform distribution, local decision triggers the formation of a swarm which separates into smaller clusters that migrate towards the closest peak, rather than a global optima. It is important to note that the agent is also equipped with low-level collision avoidance sensors.

The GSO algorithm has been tested extensively on multimodal environments of about 100 gradient sources. Three Gaussian odor sources have also been demonstrated as a testbed for the algorithm. In order to avoid bias towards a particular source, the agents were initialized in the middle of the 3 gradient sources. In 15 trials, each source was localized with an average of 7 agents converging

on the source. However, when repeated with random initial states, only 2 of the 3 gradient sources were localized in 9 out of the 15 trials. One notable contribution of the GSO to the multi-source localization challenge; is the clever partitioning of the swarm. However, the algorithm does not address the problem of progress after a source is localized.

### 2.3.2 Biasing Expansion Swarm Approach (BESA)

The BESA algorithm was developed by Cui *et al.* [1]. The algorithm simulates the localization of an unspecified number of chemical gradient sources. Like the GSO algorithm, the model forms a global ad hoc network while maintaining local communication. With this algorithm, agents are able to disperse, cohere, and form a kind of alignment while migrating towards chemical gradient sources.

The algorithm implements an occupancy grid map with which each agent is capable of storing and sharing information about their sensed concentrations and locations. It is important to note that the local communication range is restricted to the adjacent occupancy grid cells. When an agent gets out of communication range, a broadcast is sent throughout the swarm via a mechanism referred to as “multi-hop” which helps to reconnect drifting agents.

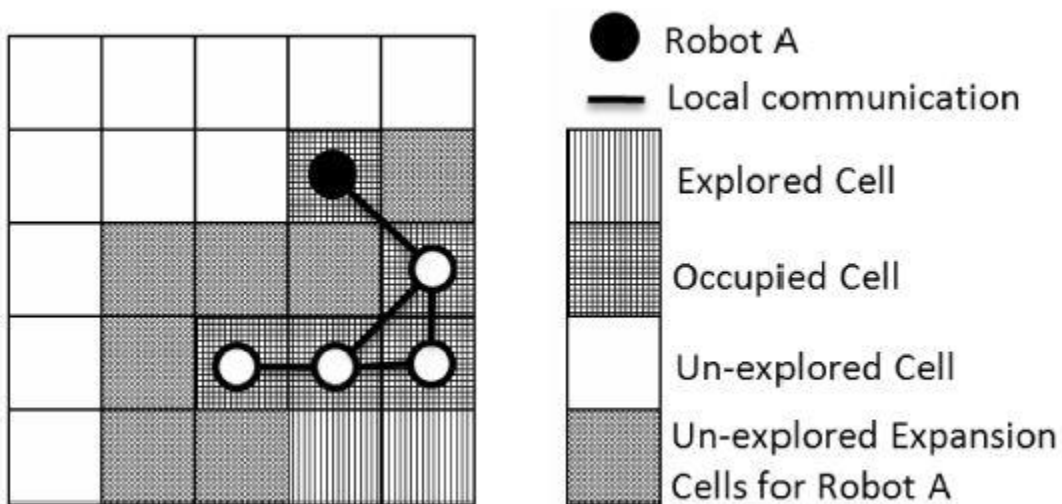


Fig. 2.1 the Biasing Expansion Swarm Approach  
Source: (Cui *et al.* [1]).

With reference to fig 2.1, the algorithm restricts the agents’ motion to unexplored, and unoccupied cells, along with the existence of an adjacent cell with at least one agent member of the swarm.

This unique property helps the swarm to maintain cohesion in tandem with the ad hoc network connection requirement for local communication.

Consequently, steering the swarm towards an emission source requires the assignment of a biasing parameter given by the equation (2.1). The rationale is to make each agent pick a cell with the highest concentration (bias) as the next location to occupy as long as the cell is empty while obeying the rules of motion. This algorithm also obeys the rules of obstacle avoidance.

$$B(x, y) = \frac{K}{n} \times \sum_{i=0}^n \frac{C_i}{r_i^2} \tag{2.1}$$

From the above equation (2.1),  $n$  is the total number of agents that communicate their sensed concentrations,  $C_i$  is the concentration sensed by agent  $i$ ,  $K$  is a constant, and  $r_i$  is the distance between expansion cell  $(x, y)$  and the cell in which agent  $i$  is located.

One notable uniqueness of the BESA algorithm is the utilization of the occupancy grid as a control mechanism for the swarm. However, it does not encourage swarm partitioning as they tend to migrate as a single whole in the direction of the gradient source.

A major problem of the multi-source localization is addressed by the BESA which is demonstrated by its ability to avoid other agents during search via swarm separation and progressive expansion.

The drawback, however, is in its inability to progress after a source is found such that other gradient sources are efficiently localized. The authors modeled the BESA algorithm on a swarm of 20 robots using inverse square law for source distribution on two gradient sources. When simulated under certain conditions, the BESA algorithm converged on both sources approximately one half the time it takes the hill-climbing algorithm.

### 2.3.3 Particle Swarm Robots

The PSO (Particle Swarm Optimization) was first introduced by Kennedy and Eberhart in 1995 [7]. The PSO is a swarm-based model for migrating agents within a virtual environment towards a global optima. Just like in evolutionary algorithms, the performance of a swarm is determined via a fitness value or score. This fitness value is dependent on how close the swarm is to the source location [40].

The algorithm assumes communication among neighboring agents in the swarm while evaluating the fitness of each agent in the swarm.

$$v_i = \varphi v_i(t - 1) + p1(xpbest_i - x_i(t - 1)) + p2(xgbest - x_i(t - 1)) \quad (2.2)$$

$$x_i = x_i(t - 1) + v_i(t) \quad (2.3)$$

In Equations (2.2) and (2.3),  $v_i(t)$  and  $x_i(t)$  are the velocity and position vectors of agent  $i$  at time  $t$ , respectively.

$xpbest_i$  is the previous position at which agent  $i$  had the best value of fitness,

$xgbest$  is the previous position of the best value of any neighbor of agent  $i$ .

$p1$  and  $p2$  are vectors containing random positive values for each dimension of the state space

$\varphi$  is a constant that controls the magnitude of velocity [13].

Using the Equation (2.2), the algorithm steers the swarm in the direction of the agent with the best previous position by treating the amount of concentration perceived from gradient sources as fitness. The algorithm has been implemented in the localization of multiple emission sources.

A distributed PSO-based algorithm proposed by Marques *et al.* [2] demonstrated the most promising solution. This modification divided the search into 2 phases: the global and local search. During the global search, the absence of any form of chemical information forces the swarm to explore the search space, during which a field-based model is used to control the motion of the swarm. Once a chemical source is detected, the algorithm switches to a local search using the PSO



algorithm. A simulation test with a swarm of 10 robots and 5 plums (gradient sources) in 3 different environments performed comparatively well when analyzed along with other related algorithms.

Two major contributions of the PSO algorithm to the multi-source localization problem are: the use of sensory-based model for collision avoidance, and also proceeding with the search for other sources after a source is found.

Proceeding with a search is however with the assumption that the source is collected from the environment after it has been found. This concept was inspired by biological foraging behavior found among biological agents. The approach removes the impact of the source on the local field concentration thereby speeding up the localization of the other sources.

#### **2.3.4 Biased Random Walk Algorithm (BRW)**

Dhariwal *et al.*, [28] implemented the Biased Random Walk (BRW) algorithm, inspired by the chemotaxis of *E. coli* bacteria. On a large network, the algorithm was used to localize multiple generic gradient sources. The algorithm implements 2 basic actions: the run action and the tumble action. During the run action phase, the robot executes a linear motion in its current orientation, while the tumbling phase alters the robot's orientation in a randomized pattern. The algorithm extends the length of the run if the run is in the orientation of a gradient source using a source bias. This clever approach helps propagate the robot towards the source location.

The BRW algorithm was simulated using 100 robots within an environment with gradient sources modeled with inverse square law and exponential profiles. With the random deployment of the robots in the presence of multiple gradient sources, it was observed that, even though all sources were tracked, the majority of robots were biased towards sources with higher gradient intensities. This behavior caused an emergent behavior of robots switching from tracing a particular source to another in the presence of a higher gradient.

A major contribution of the BRW algorithm to the multisource problem is the ability to track all gradient sources simultaneously. However, with the absence of local communication among the robots, it becomes difficult to prove that a situation where all agents would converge to a particular

gradient source and thereafter terminating the search process would ever occur. Furthermore, the algorithm lacks the ability to progress after converging on a gradient source.

### 2.3.5 Attractant-Repellent Swarm

The attractant -repellent swarm algorithm which was developed by Gazi and Passino [3] models each agent in a swarm such that the swarm moves in a centralized order. Each agent can sense the relative position of other agents. The model owes its name to the motion pattern which has the ability to attract agents further away to a central point and at the same time repel the agents in order to prevent the agents from converging on top of each other thereby maintaining a safe distance while migrating the swarm towards a gradient source. The equation below shows the motion model for M agents in the swarm.

$$\dot{x}^i = -\nabla_{x^i} \sigma x^i + \sum_{j=1, j \neq i}^M g(x^i - x^j) \quad i = 1, \dots, M \quad (2.4)$$

In equation (2.4),  $x^i$  and  $\dot{x}^i$  are the position and velocity of agent i, respectively. The term  $-\nabla_{x^i} \sigma x^i$  represents motion toward areas of higher concentration, and  $g(x^i - x^j)$  represents a function of long-range attraction and short-range repulsion between individual agents.

The attractant-repellent was metaphorically modeled using nutrient-rich areas of the search space to represent local minima. By modeling the search space as a potential field, the motion model attempts to minimize this potential energy. The velocity center of the swarm is used for the analysis of the group behavior, thus moving the center of the swarm in the direction of the negative gradient source.

When implemented on a multi-source problem with the environment being modeled using a non-uniform Multi-Gaussian profile, it was observed that the algorithm was limited to convergence in areas of minimum gradient or divergence away from the regions gradient maxima. Consequently, swarm convergence ability remained inconclusive.

## 2.4 Probabilistic Algorithms

Probabilistic algorithms belong to the class of algorithms enshrined in mathematical models. These families of algorithms address the problem of source localization via probabilistic methods. Pang and Farrell [29] have used probabilistic models such as Hidden Markov Methods (HMM) coupled with chemical gradients and the dynamics of fluid particles for the prediction of the likelihood of odor presence in a given location. These methods were successfully implemented in single-source localization and tracking.

The probabilistic model is no different from most mathematical models whose contributions may seem less intuitive yet very relevant to the solution of the multi-source problem. In biological models, contributions are explicit mechanisms that address swarm partitioning, obstacle avoidance, and proceeding after a source has been localized. In contrast, probabilistic models integrate multiple sources into a probability distribution grid map which results in an iterative optimization process.

Using these distribution maps, robots then decide on paths that would maximize the progress of the swarm. A significant contribution of probabilistic models can be attributed to the dexterous integration of multiple sources into the robots' world for the prediction of source locations.

### 2.4.1 Occupancy Grid Mapping using Bayesian Techniques

The Bayes theorem which calculates the probability of a hypothesis from a given fact and observation data has become pivotal to modern practices in artificial intelligence. Below is the equation for the theorem:

$$p(h/D) = \frac{p(D/h)P(h)}{p(D)} \tag{2.5}$$

Where  $h$  is the hypothesis,

$D$  is a set of observed data

$p(D)$  is the probability of observing a set of data  $D$

$P(h)$  is the prior probability that hypothesis  $h$  is true prior to observance of any data

$p(D/h)$  is the probability of observing  $D$  given hypothesis  $h$  is true

$p(h/D)$  is the posterior probability of  $h$  being true given the observed data  $D$ .

It is important to note that an occupancy grid is often used for simulating multi-source search problems. This occupancy grid represents a probability distribution of likely positions of an emission source. The distribution is usually uniform prior to any observation. The likelihood that a cell  $i$  is occupied by a gradient source  $e$  at a given time  $t$ , is expressed in log odds as shown below:

$$l_{t,i} = \log \frac{p(e_i/z_1 \dots z_t)}{1 - p(e_i/z_1 \dots z_t)} \quad (2.6)$$

In Equation (2.6),  $l_{t,i}$  is the log odds of occupancy and  $p(e_i/z_1 \dots z_t)$  is the posterior probability that cell  $i$  contains an emitter given sensor measurements at each time step,  $z_1 \dots z_t$  [41]

with a time step update given by the equation:

$$l_{t,i} = l_{t-1,i} + \log \frac{p(e_i/z_1 \dots z_t)}{1 - p(e_i/z_1 \dots z_t)} - l_{t,0} \quad (2.7)$$

Consequently, through an iterative process, robots are capable of deciding on the next most promising cell to advance to via the updated output of the occupancy probability at the previous time step, the current sensor reading, and the prior probability.

One of the raciest and most promising implementations of the probabilistic model was proposed by Scerri *et al.* [42]. The model was implemented on a team of unmanned aerial vehicles which use received signal strength indicator (RSSI) to localize multiple radio frequency (RF) signals emitters over a large distributed Bayesian Binary Filter Grid (BBFG).

The algorithm permits each UAV to maintain a uniquely assigned BBFG while sharing sensor data with other vehicles. Each BBFG is thereafter converted into a map and then to a decision tree with which all UAVs leverage for path planning via regions of maximum information gain. The

planning phase is implemented with an RRT (Rapidly expanding Random Tree) planner which integrates the information gain along with the planned paths of the other vehicles and converts it into a cost map.

The algorithm is instantiated with an initial prior probability distribution which is equal across all grid cells. This standard procedure depicts an unexplored environment. Using sensor readings from the emitter, each UAV is able to maintain a posterior distribution of emitters in the grid. The goal, therefore, is to explore the grid in order to localize an unknown number of emitters.

It is noteworthy that occupancy probabilities in an unexplored area is based on the prior distribution due to the fact that such area may contain an additional emitter. Consequently, each UAV is required to explore the occupancy grid in a deliberate attempt to maximize information gain. The algorithm measures optimal performance by minimizing the following factors: the cost of vehicle flight routes, the difference between the predicted and actual states of the emitters, and finally the overhead in the amount of message sharing among robots.

The implementation of the RRT planner and cost function are very significant to the solution of the multisource problem. The RRT decides the path for each UAV and thus maximizes information gain. By keeping a priority list, the RRT selects nodes with the highest priority and expands them in random directions, to determine a new target node (the best node) with minimal cost via node expansion. The UAV path is then set in the direction of the best target node. Through simulation and live flight tests, the authors demonstrated the localization of 3 emitter sources using a testbed of 20 UAVs.

When an emitter source has been localized during the search, the entropy map is immediately updated. Consequently, UAVs tend to avoid traveling in the direction of the localized emitters during the remaining part of the search.

While this approach does not explicitly partition the swarm, it successfully caters for the localization of multiple emitter sources. However, there exist no theoretical foundations that guarantee progress, convergence, and termination.

## 2.5 COMPARATIVE ANALYSIS

Table 2.1, shows a summary of the different algorithms discussed in this chapter along with how they attempt to solve the multisource localization problem.

**Table 2.1** Multisource algorithms summary **Source:** Kathleen McGill, Stephen Taylor 2011 [13]

	BESA	Bayesian occupancy	BRW	PSO	Attractant/repellant	GSO
Swarm size	20	20	100	10	11	1000
Source number	2	3	2	5	5	100
Type of source	Chemical	RF	Generic	Chemical	Chemical	Generic
Source Model	Inverse square law	Inverse square law	Inverse square law, exponential, etc.	Plume model	Gaussian, quadratic, planar	Gaussian, Rastrigin, etc.
Source Mobility	Fixed	Mobile	Fixed	Fixed	Fixed	Mobile
Source time variance	Constant	Intermittent	Constant, decaying	Constant	Constant	Constant
Variable source configuration	Yes	Yes	Yes	No	No	Yes
Variable source intensity	—	—	Yes	No	Yes	Yes
Dead space	No	--	No	Yes	No	No
Sensing requirements	Concentration, location	RF signal strength, location	Signal intensity	Concentration, wind, location of all robots and obstacles	Concentration, location	Signal intensity
Communication range	Local	Local	None	Local	Global	Local
Robot deployment	Near each other	—	Random, single location	Corner	—	Random, center, corners, right edge
Implementation	Simulation only	Simulation and tests	Simulation and tests	Simulation only	Simulation only	Simulation and tests
Robust to signal noise	Yes	Yes	Yes	Yes	—	Yes
Computational complexity	Medium	High	Low	Medium	Medium	Low
Obstacle avoidance	Swarm control	Minimum path cost	None	Artificial repulsion	Artificial repulsion	Sensory based
Partitioning	None	Maximum entropy	Random	None	None	Local subgroups
Mechanism to proceed after source found	None	None	None	Source “collection”	None	None
Theoretical foundations	None	None	None	None	Single source profiles	Clustering behavior

**Table 2.2** Other comparisons on odor-source localization [43, 44, 49]. The ‘/’ symbol means “or” and 'N' means more than one gradient source.

	Gradient-based		Bio-inspired			Multi-robot		Probabilistic & Map-based		
	Chemotaxis	Biased Random walk	Surge-cast	Surge-Spiral	Pure Casting	Formation-based	PSO-based	Kernel methods	Infotaxis	Probability mapping
<b>Environment</b>										
Laminar flow	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Turbulent flow	NO	NO	NO	YES	NO	NO	YES	YES	YES	YES
Number of sources	1		1			1	1/N	1	1/N	1/N
<b>Information Required</b>										
Explicit plume model	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO
Odor concentration	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES
Wind direction	BOTH	NO	YES	YES	YES	YES	YES	YES	YES	YES
<b>Performance</b>										
Computational load	LOW		LOW			LOW	FAIR	HIGH		
Cost	LOW		LOW			HIGH		FAIR/HIGH		
Distance overhead	LARGE	FAIR	FAIR	FAIR	LARGE	SMALL	SMALL	SMALL	SMALL	SMALL
Time to find the source(s)	LONG		FAIR	FAIR	LONG	SHORT		SHORT		

From table 2.1, the swarm sizes and the source numbers depict the maximum number of agents and gradient sources used in the simulation. The variable source configurations tells if source positions were altered during the simulations. Variable source intensities were indicated if sources had different magnitudes at various locations.

Dead space indicated whether simulations had locations in the environment without any gradient signal. Wherever empty cells exist on the table, it indicates missing details from literature.

Cells with "simulations" indicates that software only was used for the experiments. The columns with "test" depict live demonstrations with robotic artifacts, while those with "experiments" indicates the combination of both tests and simulations.

The models used for gradient sources range from plume models usually suited for chemical sources to generic Gaussian models and RF emission models that conforms to the inverse square law model. Whatever the model adopted, they all show robustness to signal noises. Introducing noise into the sensor signal is a crucial step in simulated models to demonstrate the algorithm's potential of adapting to the real world. Noises used in simulations for the emission profiles were either random or Gaussian.

The computational complexity represents the amount of CPU processing overhead needed to implement the algorithms. The measures for complexity include: amount of data storage required, local or global communication overhead and algorithmic computation.

Table 2.2 shows a more recent comparison of algorithms, based on multiple odor source localization. This time, a closer attention is given to probabilistic models. The first three algorithms rely on gradient source information. The effectiveness of these algorithms depends on the intensity of these gradient sources. The concept of infotaxis (a typical Bayes inference model) as a probability based odor source localization algorithm was first proposed by Vergassola *et al.*, [44]. The algorithm guides a robot to the source location using plume distribution, and continuous distribution update on a grid world.

An improvement on the Infotaxis proposed by E.M. Moraud, D. Martinez [45], considers the fact that odor plume could consist of sporadic odor patches rather than a continuous distribution. In such scenario, the authors recommended the explorative and exploitative methodology.

Bayesian Inference Methods, where also implemented [46-48] on the assumption that the chemical plume is spread along a down flow temperature change. By leveraging on the wind field, the robot is able to predict the best path toward the odor source.

Farrell *et al.*, [49] proposed a model based on the Hidden Markov Models (HMM). This model builds an HMM:  $\lambda = [\pi, A(t), b]$ , where ' $\pi$ ' is the source probability vector and set to be equally distributed over the grid; the matrix ' $A(t)$ ' represents the transition probabilities (of odor from one cell to another) which was calculated using the measured flow velocity; ' $b$ ' is the detection



probability vector. Consequently, the robot could plan its path using the source likelihood map, constructed with the HMM algorithm.

The BRW can be considered the simplest of all algorithms due to the simplicity of its behavior of run and tumble, along with the absence of local nor global communication among agents. In contrast, the Bayesian algorithm, in exchange for robustness paid a great price in the computational complexity via mappings, sensor readings, and path planning. Most of the algorithms employed sensor models for robot and obstacle avoidance. The greatest contributor to swarm partitioning was the GSO algorithm which formed clusters via local decision techniques. The BRW exhibited partitioning via independent random search, which caused the robots to spread in different directions. In the same vein, the Bayesian algorithm partitioning technique was attributed to the UAVs selection of different paths due to the quest to maximize information gain.

To date, the Kalman and particle filtering techniques have not been applied to the robotic multisource localization problem, thus the authors in this research thesis investigate some of the contributions such a filtering algorithm could bring to the multisource localization problem. The authors focused on the multimodal particle filters algorithm due to the robustness of the algorithm.

We find this research significant due to the fact that some of the questions posed by the multisource localization problem remain largely unanswered. Questions such as the mechanism to proceed with a search after a target are found. The PSO suggested absorbing or removing the located source from the scene (such as would be appropriate for foraging or rescue operations) before commencing on a global search. While this approach seems reasonable, it cannot be adapted to all multisource models.

Furthermore, theoretical foundations for progress, convergence, and termination are not clearly stated. Although the attractant-repellant model provided a theoretical foundation for convergence, it was only limited to a single-source localization searches.

## **2.6 Conclusion**

This chapter surveys the major challenges associated with multisource localization and distinguished three principal families of algorithms that have been implemented in the literature to resolving the multisource problems. The algorithms include; hill-climbing algorithm, biologically inspired algorithm, and probabilistic algorithms.

While the localization of a single gradient source has been addressed elaborately in the literature, there has been comparatively low emphasis on the localization of multiple emission sources. In view of the challenges faced by the multi-source localization, we attempt to provide solutions to the major challenges of the multisource problem while presenting an axis for future directions.

## CHAPTER 3

### FORMAL BACKGROUND

#### 3.1 Introduction

Some of the famous meta-heuristic algorithms include: Genetic algorithm (GA), Particle Swarm Optimization (PSO), Simulated Annealing (SA), Differential Evolution Algorithm (DE), Cuckoo Search Algorithm (CK), etc. [50], [51]. These algorithms leverage on a model matrix that evolves random solutions to the predefined objective function. In addition, some of these meta-heuristic algorithms use the variant of the basic genetic algorithm schema (selection, Mutation, and crossover) while evolving solutions [52], [53]. These variants can be summarized into two basic strategies namely; exploration and exploitation. While exploitation targets the best local solution within the search space, exploration attempts to leverage diversification in an attempt to incur the best solution which in most cases lies around one of the local solutions. A good meta-heuristic algorithm can be characterized by the rate at which it finds the global optimal solution to the objective function.

In this chapter, we present a background theory on which our proposed paradigm for optimizing control of multi-agents is based, thereby revealing the intricate parts of the model adapted for corporative control of multi-agents. Consequently, we develop a framework for the hybridization of multiple meta-heuristic algorithms. This framework leverages the strengths of each meta-heuristic algorithm in order to rapidly converge a search process to an optimal or suboptimal solution with minimal computational complexity.

Some artificial multifarious problems also referred to as test functions were chosen to evaluate the robustness of our proposed optimization algorithm. These artificial problems have the advantage of ease in the modification and manipulation of the test algorithm within diverse scenarios. The objective functions could be qualified or grouped into categories such that they are either continuous functions or discontinuous functions, linear functions or polynomial, differentiable or non-differentiable, [54] uni-modal or multi-modal, separable, or non-separable. These objective functions can be sorted or grouped by their modality, basins, valleys, separability, and dimensionality.

Modality: This represents the number of peaks in the function’s topology. When an algorithm comes across such peaks during a search cycle, there exists the likelihood of the algorithm to asymptote at local optima or minima depending on the predefined search criteria.

Basins: unlike peaks, these are steep decline around a large area. The presence of basins could have a significant impact on the success of an algorithm due to insufficient information to guide the algorithm towards the global minima.

Valleys: These occur when narrow domains of minimal difference are surrounded by multiple basins. The floor of a valley could have a significant impact on the success of a search algorithm.

Separable: This measures the difficulty of a function. It is easier for a search algorithm to transverse a separable function than a non-separable function. When the variables of a function are independent of each other, the derivative of the function can be decomposed into sub-functions. This separable feature makes it easier for an algorithm to solve. On the other hand, if the variables are dependent on each other, the function becomes non-separable thus making it more difficult for an algorithm to solve.

Dimensionality: the magnitude of the parametric variables defines the dimensionality of the objective function. Every one-step increase in the number of parameters has an exponential overhead in the amount of computational search space. Almost every meta-heuristic algorithm has dimensionality as a major bottleneck.

Other intricate segments of our model discussed in subsequent chapters include the multi-objective optimization, evolutionary neural networks, reinforcement learning, Markov decision processes (MDPs) and the partially Observable Markov decision processes (POMDPs).

The following is a collection of 30 optimization test objective functions we used to compare, analyze and validate the performance of our hybrid optimization algorithm:

- i. Ackleys 2 function [55]

$$f(x) = -200e^{-0.02\sqrt{x_1^2+x_2^2}} \text{ subject to } -32 \leq x_i \leq 32 \quad (3.1)$$

- ii. Bartels Conn Function [54]

$$f(x) = |x_1^2 + x_2^2 + x_1x_2| + |\sin(x_1)| + |\cos(x_2)| \text{ subject to } -500 \leq x_i \leq 500 \quad (3.2)$$

iii. Beale Function [54]

$$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_2^2)^2 \text{ subject to } -4.5 \leq x_i \leq 4.5 \quad (3.3)$$

iv. Bird function [54]

$$f(x) = \sin(x_1) e^{(1-\cos(x_2))^2} + \cos(x_2) e^{(1-\sin(x_1))^2} + (x_1 - x_2)^2 \text{ subject to } -2\pi \leq x_i \leq 2\pi \quad (3.4)$$

v. Bohachevsky 1 Function [54]

$$f(x) = x_1^2 + 2x_2^2 - 0.3 \cos(3\pi x_1) - 0.4 \cos(4\pi x_2) + 0.7 \text{ subject to } -100 \leq x_i \leq 100 \quad (3.5)$$

vi. Bohachevsky 3 Function [56]

$$f(x) = x_1^2 + 2x_2^2 - 0.3 \cos(3\pi x_1 + 4\pi x_2) + 0.3 \text{ subject to } -100 \leq x_i \leq 100 \quad (3.6)$$

vii. Booth Function [54]

$$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \text{ subject to } -10 \leq x_i \leq 10 \quad (3.7)$$

viii. Branin RCOS-2 Function [58]

$$f(x) = (x_2 - \frac{5.1x_1^2}{4\pi^2} + \frac{5x_1}{\pi} - 6)^2 + 10 \left(1 - \frac{1}{8\pi}\right) \cos(x_1) \cos(x_2) \ln(x_1 + x_2 + 1) + 10 \text{ subject to } -5 \leq x_i \leq 15 \quad (3.8)$$

ix. Brent Function [58]

$$f(x) = (x_1 + 10)^2 + (x_2 + 10)^2 + e^{-x_1^2 - x_2^2} \text{ subject to } -10 \leq x_i \leq 10 \quad (3.9)$$

x. Camel Function – Six Hump [58]

$$f(x) = \left(4 - 2.1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (4x_2^2 - 4)x_2^2 \text{ subject to } -5 \leq x_i \leq 5 \quad (3.10)$$

xi. Camel Function – Three Hump [58]

$$f(x) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2 \text{ subject to } -5 \leq x_i \leq 5 \quad (3.11)$$

xii. Chichinadze Function [54]

$$f(x) = x_1^2 - 12x_1 + 11 + 10 \cos\left(\frac{\pi x_1}{2}\right) + 8\left(\frac{5\pi x_1}{2}\right) - \left(\frac{1}{5}\right)^{0.5} \exp(-0.5(x_2 - 0.5)^2) \text{ subject to } -30 \leq x_i \leq 30 \quad (3.12)$$

xiii. Cube Function [59]

$$f(x) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2 \text{ subject to } -10 \leq x_i \leq 10 \quad (3.13)$$

xiv. Deckkers-Aarts Function [60]

$$f(x) = 10^5 x_1^2 + x_2^2 - (x_1^2 + x_2^2)^2 + 10^{-5}(x_1^2 + x_2^2)^4 \text{ subject to } -20 \leq x_i \leq 20 \quad (3.14)$$

xv. Easom Function [61]

$$f(x) = -\cos(x_1) \cos(x_2) \exp[-(x_1 - \pi)^2 - (x_2 - \pi)^2] \text{ subject to } -100 \leq x_i \leq 100 \quad (3.15)$$

xvi. Freudenstein Roth Function [62]

$$f(x) = (x_1 - 13 + ((5 - x_2)x_2 - 2)x_2)^2 + (x_1 - 29 + ((x_2 + 1)x_2 - 2)x_2)^2 \text{ subject to } -10 \leq x_i \leq 10 \quad (3.16)$$

xvii. Haupt Function 16 [63]

$$f(x) = -x_1 \sin(\sqrt{|x_1 - (x_2 + 9)|}) - (x_2 + 9) \sin(\sqrt{|x_2 + (0.5x_1 + 9)|}) \text{ subject to } -20 \leq x_i \leq 20 \quad (3.17)$$

xviii. Haupt Function 7 [63]

$$f(x) = x_1 \sin(4x_1) + 1.1x_2 \sin(2x_2) \text{ subject to } 0 \leq x_i \leq 10 \quad (3.18)$$

xix. Haupt Function 15 [63]

$$f(x) = -\exp[-0.2\sqrt{x_1^2 + x_2^2} + 3(\cos 2x_1 + \sin 2x_2)] \text{ subject to } -5 \leq x_i \leq 5 \quad (3.19)$$

xx. Egg Crate Function [54]

$$f(x) = x_1^2 + x_2^2 + 25(\sin^2(x_1) + \sin^2(x_2)) \text{ subject to } -5 \leq x_i \leq 5 \quad (3.20)$$

xxi. Goldstein Price Function [64]

$$f(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 - 48x_2 + 36x_1x_2 + 27x_2^2)] \text{ subject to } -5 \leq x_i \leq 5 \quad (3.21)$$

xxii. Rosenbrock Modified Function [65]

$$f(x) = 74 + 100(x_2 - x_1^2)^2 + (1 - x_1)^2 - 400 \exp\left[-\frac{(x_1+1)^2+(x_2+1)^2}{0.1}\right] \text{ subject to } -2 \leq x_i \leq 2 \quad (3.22)$$

xxiii. Rotated Ellipse Function [54]

$$f(x) = 7x_1^2 - 6\sqrt{3}x_1x_2 + 13x_2^2 \text{ subject to } -500 \leq x_i \leq 500 \quad (3.23)$$

xxiv. Scahffer-1 Function [66]

$$f(x) = 0.5 + \frac{\sin^2(x_1^2+x_2^2)-0.5}{1+0.001(x_1^2+x_2^2)^2} \text{ subject to } -100 \leq x_i \leq 100 \quad (3.24)$$

xxv. Test-tube Holder Function [67]

$$f(x) = -4\left[(\sin(x_1) e^{\cos(x_1^2+x_2^2)/200})\right] \text{ subject to } -10 \leq x_i \leq 10 \quad (3.25)$$

xxvi. Pen-Holder Function [67]

$$f(x) = -\exp\left[|\cos(x_1) \cos(x_2) e^{1-[(x_1^2+x_2^2)]^{0.5/\pi}} - 1\right] \text{ subject to } -11 \leq x_i \leq 11 \quad (3.26)$$

xxvii. Trefethen Function [68]

$$f(x) = e^{\sin(50x_1)} + \sin(60e^{x_2}) + \sin(70 \sin(x_1)) + \sin(\sin(80x_2)) - \sin(10(x_1 + x_2)) + \frac{1}{4}(x_1^2 + x_2^2) \text{ subject to } -10 \leq x_i \leq 10 \quad (3.27)$$

xxviii. Adjiman Function [69]

$$f(x) = \cos(x_1) \sin(x_2) - \frac{x_1}{(x_2^2+1)} \text{ subject to } -1 \leq x_1 \leq 2, -1 \leq x_2 \leq 1 \quad (3.28)$$

xxix. Cross-in-Tray Function [67]

$$f(x) = -0.0001 [|\sin(x_1) \sin(x_2) e^{\left| \frac{100 - [(x_1^2 + x_2^2)]^{0.5}}{\pi} \right|} + 1|^{0.1}] \text{ subject to } -10 \leq x_i \leq 10 \quad (3.29)$$

xxx. Damavandi Function [69]

$$f(x) = \left[ 1 - \left| \frac{\sin[\pi(x_1-2)] \sin[\pi(x_2-2)]}{\pi^2(x_1-2)(x_2-2)} \right|^5 \right] [2 + (x_1 - 7)^2 + 2(x_2 - 7)^2] \text{ subject to } 0 \leq x_i \leq 14 \quad (3.30)$$

## 3.2 METAHEURISTIC ALGORITHMS

### 3.2.1 Evolution computation

Evolution computation is a family of population based optimization algorithms which encapsulates genetic programming, evolution strategies and genetic algorithm. These algorithms incorporates models such as selection and mutation which forms the core of the entire evolution computation algorithm.

Genetic algorithm (GA) is a classical optimization meta-heuristic based on the biological model of natural selection. The algorithm involves a clever manipulation of an objective function, a vector or matrix of objective variables, definition of variable constraints, selection, crossover, and mutation. The total number of iterations (epochs) usually depends on the chosen termination criteria which could either be a predetermined number of epochs or the convergence of the algorithm. The GA is said to have converged when little or no significant improvement is observed in the population.

Two key processes guide the GA towards an optimal solution: the selection and the crossover. The roulette wheel selection model is a typical and intuitive probabilistic approach that favors the best



pair of objective variables (variables with high fitness scores) for participating in the mating process.

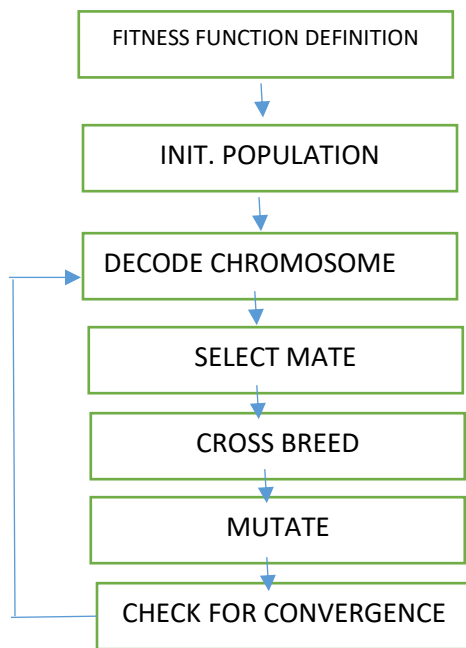


Fig. 3.1 Flowchart for genetic algorithm.

The mating process is implemented by the crossover model. A typical procedure is binary fission of binary encoded objective variables [70-72]. A random split point is chosen for both selected string of binary coded bits, thereafter an exchange (crossover) is conducted.

A subtle but very efficient model is the mutation process. Given a predetermined probabilistic mutation rate, the resulting string from the crossover process may undergo an alteration in one or more of its bits. This procedure helps the algorithm's converging process to progress towards the global optima.

### 3.2.2 Simulated Annealing (SA)

The simulated annealing algorithm as a meta-heuristic optimizer was the 'brainchild' of Kirkpatrick *et. al.*, in the year 1983 [6]. The algorithm mimics the process of a crystal-like lattice via a quick heating and slow cooling process. Like other standard procedures, the algorithm begins with generating a random number of objective variables that are modified via some parametric turning before assigning them to a fitness function which then outputs a fitness score for each pair/ vector of variables.

$$v(new) = d.v(old) \quad (3.31)$$

where,

$v(new)$  = new variables

$v(old)$  = old variables

$d$  = control variable

It's important to note that most literature refers to a set of objective variables as a chromosome. In the Simulated annealing iterative algorithm, a new set of objective values replaces the old ones if there was an improvement in their corresponding fitness scores, however some of the less fit pairs may proceed to the next generation even if their fitness scores worsened as long as they satisfy the following conditions:

$$r \leq e^{\frac{[f(old)-f(new)]}{T}} \quad (3.32)$$

Otherwise, they are rejected.

where,

$r$  = uniform stochastic variable

$T$  = temperature

The algorithm slowly reduces the  $T$  value until it gets close to zero before terminating. During this cooling process, the algorithm does a percentage-wise reduction of the  $d$  value in an attempt to enhance the fitness scores of the population.

### 3.2.3 Particle Swarm Optimization (PSO)

The PSO algorithm is a relatively less involved algorithm with few parameters to manipulate, first proposed by Kennedy and Eberhart [7]. Like the Genetic algorithm, the PSO meta-heuristic mimics a biological model. However, it exempts the crossover and mutation procedures of the Genetic algorithm.

The PSO algorithm iteratively updates the objective variables via a velocity vector. For brevity, the algorithm modifies each set of objective values updating their vector velocities via a clever manipulation of the global best and local best solutions. While the global best is indicative of the best fitness score so far in the iterative process, the local best is indicative of the best fitness score within the current run. The equation below shows the simplicity of this elegant algorithm:

$$vel_{new} = vel_{old} + \gamma \cdot r1 \cdot (P_{local\ best} - P_{old}) + \gamma \cdot r2 \cdot (P_{global\ best} - P_{old}) \quad (3.33)$$

$$P_{new} = P_{old} + vel_{new} \quad (3.34)$$

where,

$vel$  = velocity of each particle

$P$  = particle variables

$P_{local\ best}$  = best local fitness for each particle

$P_{global\ best}$  = global local fitness for each particle

$\gamma$  = learning rate (constant)

$r1, r2$  = stochastic variables

The ease of implementation is another significant advantage of this algorithm.

### 3.2.4 Cuckoos Search algorithm (CS)

The cuckoos search meta-heuristic algorithm was first proposed by Deb and yang [8]. The algorithm was inspired by the interesting reproductive characteristics of the cuckoos' bird. The cuckoos' is an opportunistic bird that lays its eggs among other eggs in a host nest. On the return of the host bird, the host bird may or may not detect the presence of the cuckoos' egg. If the cuckoo egg is undetected, all eggs are hatched otherwise, the nest is completely abandoned or ruined.

The CS meta-heuristic combines the behavior of the host bird and the cuckoos' bird. Intuitively, each nest is a representation of a set of objective variables [73]. The algorithm first generates an  $N$  – population pair or vector of objective variables usually referred to as candidate solutions in the literature. Next, the cuckoos' egg is laid in a randomly chosen nest using a typical random walk 'levy flight' approach:

$$xk' = xk + rand \cdot (levy\ flight)xk \quad (3.35)$$

$$yk' = yk + rand \cdot (levy\ flight)yk \quad (3.36)$$

Next, the fitness of the nest with the cuckoos' egg is compared with the host nest. The host nest is replaced if it has a worse fitness score when compared with the cuckoo's nest. However, if the host bird notices the presence of the cuckoo's egg, the nest is discarded usually with a probability  $p < 0.25$  consequently creating a new nest.

### 3.2.5 Hybrid Optimization

A typical hybrid algorithm blends the strengths of genetic algorithms with the converging speed of any local optimizer [74]. A couple of authors such as Kazarlis *et al.* [9] implemented a scaled-down genetic algorithm with a relatively small population size as a local optimizing strategy. The rationale is to optimize the delicate balance between explorative and exploitative paradigm of the model. When the GA seems to gradually asymptote, it is assumed to at least be in the domain of the global solution, thereafter the local optimizing algorithm seizes the search process in an attempt to obtain an optimal solution. Hybridization could be in any of the forms below:

- (1) Beginning with a GA until it decelerates before seeding a local optimizer
- (2) Start the GA with some local minima obtained from random starting points in the population
- (3) After a predefined number of iterations, seed a local optimizer on a selected elite population using elitism and incorporate the resulting chromosome into the population. Haupt [63] demonstrated finding the global optima by combining a continuous GA with Nelder-Mead downhill simplex algorithm.

### 3.2.6 Evolutional Neural Networks

An evolutionary neural network also referred to as neuro-evolution has its application mainly as a combination of two powerful AI algorithms: the genetic algorithm and the artificial neural networks having its application ranging from artificial life, general game playing and evolutionary robotics.

They are both biologically inspired and are often designed as feed-forward ENNs [75] when combined. This combination is achieved by evolving the weights in a fixed sized neural network while providing the network with a set of inputs.

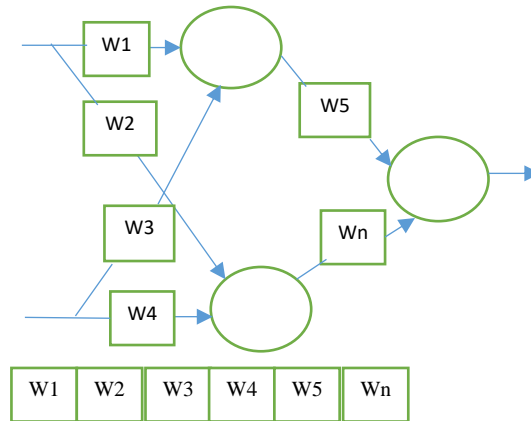


Figure 3.2 A two input multi-layer evolutionary neural network structure

While neural networks are inspired by ideas from neuroscience, genetic algorithms are inspired by the theory of evolution and natural selection. When implementing ENNs, it's imperative to first of all represent the problem domain as a chromosome. In other to find an optimal set of weights  $[w_1, \dots, w_n]$ , for the feed forward multilayer ENN as shown in Fig. 3.2 above, each set of weight would represent a chromosome [76-78]. These initial weights are usually chosen randomly within a minute interval of  $[-1, 1]$ . An array of chromosomes could be thought of as a solution set whose fitness is updated from generation to generation towards an optimal solution. Artificial neural networks in its simplest form, is a collection of connected neurons with each connection having a weighted value.

When the network is presented with an input pattern, (e.g. Numbers representing some imagery features), a pattern of activation (switching on and off) spreads in a forward direction over weighted connections via the hidden layer to the output layer [79], [80-83]. This process mimics the way activation spreads through the network of neurons in the human brain.

One way of applying genetic algorithms to neural networks is by evolving weights in a fixed network. Montana and Davis [82] took this first approach. They implemented network training using GAs instead of back propagation for finding a good set of weights for a fixed set of connections. Because back propagation has the tendency of getting stuck in local optima within its weight space, the use of GAs seemed a desirable approach for overcoming this limitation. They tested back propagation method against GA technique for classifying underwater sonic "Lofargrams" in two classes: 'interesting' and 'not interesting'. The results showed that the GA

optimizer significantly outperformed the back propagation algorithm on this particular task by obtaining better weight vectors at a faster rate.

### **3.3 MUTLIOBJECTIVE OPTIMIZATION**

Multi-objective optimization attempts to resolve more than one objective function which are either minimized or maximized. In single objective optimization problems, the goal is to find a single objective also known as the fitness value [84-86] that optimizes the objective function. However, in multi-objective problems, the goal is to find a set of solutions as close as possible to the Pareto-optimal front and also these solutions should be as diverse as possible.

Another significant difference between single and multi-objective optimization is the presence of two search spaces. In single objective, there is only one search space (the decision variable space) while in multi-objective, there exist in addition, the objective or criterion space.

Although there exists a relationship between the two search spaces, the mapping between them is usually nonlinear with their properties being dissimilar.

#### **3.3.1 Multi-objective Domination**

Most multi-objective optimization algorithms use the construct of domination. Solutions progress towards the Pareto-optimal front based on whether one dominates the other solution or not using the following stated rules:

A solution  $x_1$  is said to dominate another solution  $x_2$  if the following two conditions are true:

- (1) The solution  $x_1$  is no worse than  $x_2$  in all objectives and
- (2) The solution  $x_1$  is strictly better than  $x_2$  in at least one objective

### 3.3.2 The Non-dominated Sorting Genetic Algorithm (NSGA-II)

The NSGA-II is recognized for its simplicity and elegance in preserving diversity via an explicit diversity-preserving mechanism while guiding the objective values towards the Pareto-optimal solution.

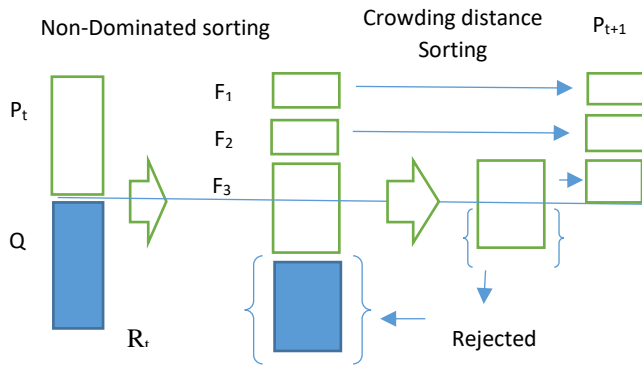


Fig. 3.3 Schematic for NSGA-II algorithm

From fig. 3.3 above, the first two populations  $P_t$  and  $Q_t$  are combined into  $R_t$  of size  $2N$ . Thereafter, a Non-dominated sorting procedure is used to classify the entire  $R_t$  population. The new population is then filled with solutions of different non-dominated fronts ( $F_1$ ,  $F_2$ ,  $F_3$ , etc.) beginning with the best non-dominated front ( $F_1$ ). With an initial population of  $2N$ , only  $N$  (half) is needed for the new population. Consequently rejecting the rest lower fronts.

A crowding distance methodology is thereafter used to maintain diversity among the new population ( $P_{t+1}$ ) before deploying the tournament selection with an output of  $2N$ . This iterative cycle gradually guides the algorithm towards a Pareto-optimal solution.

### 3.4 Reinforcement Learning (RL)

Reinforcement learning is the science of sequential decision making. For grid world agents, it is characterized by an agent's ability to maximize long term rewards leveraging on past experiences obtained via interaction with a stochastic environment. Because the environment is initially unknown to the agent, the agent has to surmount the challenge of handling the delicate balance between exploring and exploiting the environment while maximizing the expected long term reward. Consequently, RL agents usually combine online learning and planning simultaneously via policy optimization [87-89]. The utilities of each state in RL is often referred to as state-valued

function. Analogous to this, is the action valued function often referred to as Q-value function. The process of learning with Q-valued functions is referred to as Q-learning

$$Q(s_t, a)_{new} = Q(s_t, a) + \alpha (r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a)) \quad (3.37)$$

where,

$Q(s_t, a)$  is the current value of the state under a specific action policy  $a$

$r_{t+1}$  is the received reward

$\max_a Q(s_{t+1}, a)$  is the maximum Q-value of the subsequent state under a specific action policy  $a$

$\alpha$  is proportional to the learning rate weighted by  $\gamma$  the discount factor

In this thesis, we investigate a Q-learning RL for our implementation because it learns considerably faster than the state value function. However, reinforcement learning is generally slow, thus the introduction of multi-agents in the training process.

### **3.5 Markov Decision Processes (MDP) and Partially Observable Markov Decision Processes (POMDPs)**

Markov Decision Processes (MDPs) are best suited for robotic navigations in a known environment. The environment is assumed to be Markovian (i.e., the effects of an action stochastically depends on the current state of the world and the executed action). Because the resulting state from the action is not deterministic, the subsequent state of the agent may be unintended. Amidst this stochasticity, the robot must navigate from its current location to a goal location with the minimum possible steps. Thus MDPs create a policy for every possible node in the grid world that is fully observable and stochastic [90-91]. MDPs are usually defined as a tuple  $\langle S, A, T, R \rangle$  where:

S- set of environment states (which must encapsulate all relevant information for taking correct decisions – e.g Map, exact location within the map, state of the world (open or closed door).

A- All actions that the agent can execute. A simplified example would be UP, DOWN, LEFT, RIGHT



T- the stochastic transition function  $T(S, A, S') = P(S'_{t+1} = S_o | S_t = s, A_t = a)$  – the probability of executing an action ‘a’ from state ‘S’ at the time ‘t’ and arriving at state S’ at time ‘t+1’.

R- the reward function which models the utility of the current state as well as the cost of taking a particular action  $R(S, a)$ . A negative living reward (non-zero cost) is usually associated with grid world implementations.

In this thesis, our simulation was tested on planning problems which have a finite and discrete state and action space. The purpose of planning is to find a policy (set of optimal actions) that describes the agent’s behavior in order to maximize the sum of expected rewards

$$U(s) = \sum_{t=0}^{\infty} E[\gamma^t R(S_t)] \tag{3.38}$$

where,  $\gamma$  is the discounted reward as ‘t’ tends towards infinity  $0 \leq \gamma < 1$ . This keeps the solution bounded. However, since our horizon is finite, (i.e. has an absorbing or goal state) we set  $\gamma = 1$

For every state S, we can compute a utility function with the following equation:

$$U(s) = R(s) + \gamma \sum_{s'} T(S, a, S')U(S') \tag{3.39}$$

The optimal utility for each state is given by the Bellman equation

$$U(s) = R(s) + \gamma \text{Max}_a \sum_{s'} T(S, a, S')U(S') \tag{3.40}$$

The optimal policy is given by the equation.

$$\pi^*(s) = \text{argMax}_a \sum_{s'} T(S, a, S')U(S') \tag{3.41}$$

In real-world domains, most of the assumptions behind the implementation of MDPs fall apart because the agent cannot directly observe the state of the environment.

Partially Observable Markov Decision Processes (POMDPs) present us with a more efficient alternative to modeling real-world problems via probability distribution over states also referred

to a belief states. This is because the actual state of the world cannot be fully observed due to inaccurate sensor readings. Alternatively in POMDP environments, beliefs provides a sufficient statistic for history [11] thereby availing sufficient information for the optimal policy per state with the assumption that the underlying MDP is also Markovian.

POMDPs, therefore, can be defined as belief-space MDP with the tuple  $\langle B, A, T, R_B \rangle$  such that:

-B is the set of possible states over beliefs over state S

-A is the set of possible actions

-T is the belief transition function  $T(B, a, B')$ ; representing the transition probability of starting a belief B, take an action a, and arriving at a new belief state B'.

-R<sub>B</sub> is the reward at each belief state.

Just like in the MDP model, we define the Bellman update operator [92] for the Belief-Space MDP (POMDP) as:

$$U(b) = \text{Max}_a \left( R(b) + \gamma \sum_{b' \in B'} T(b, a, b') U(b') \right) \quad (3.42)$$

Consequently, like in MDPs the goal of POMDPs is to find the policy for action selection that maximizes the reward ( $b$ ).

### 3.6 Particle filters algorithm

The particle filter is an elegant algorithm with the potential of mapping trajectory history into belief states which consequently aid agents to learn a mapping from belief states to action in POMDPs [93-95]. Particle filters are an implementation of recursive Bayesian filtering used for modeling non-Gaussian distributions [96-97]. Using the motion and sensor observation model, the algorithm iteratively updates the belief-states via a sequence of prediction steps and correction steps usually referred to as belief updates [98-99].

Predictor step is given by:

$$\overline{Bel}(x_t) = \int P(x_t | U_t, x_{t-1}) Bel(x_{t-1}) dx_{t-1} \quad (3.43)$$

While the Correction step is given by:

$$Bel(x_t) = \eta P(Z_t | x_t) \overline{Bel}(x_{t-1}) \quad (3.44)$$

Combining both equations, we get the Bayes particle filter equation as follows:

$$Bel(x_t) = \eta P(Z_t | x_t) \int P(x_t | U_{t-1}, x_{t-1}) Bel(x_{t-1}) dx_{t-1} \quad (3.45)$$

where,

$\eta$  is the normalization factor

$Bel(x_t)$  is the belief of being in state  $x$  at time  $t$ .

$P(Z_t | x_t)$  is the probability of sensing  $Z_t$  given a state location  $x_t$  at time  $t$ .

$U_t$  is the action or motion step at time  $t$

The particle filter algorithm is given as follows:

$$\{S_{t-1} = \langle x_{t-1}^j, w_{t-1}^i \rangle, U_t, Z_t\}$$

1.  $S_t = \emptyset, \eta = 0$
2. *for*  $i$  *in*  $1 \dots n$
3. Sample index  $j(i)$  from discrete distribution given by  $w_{t-1}^i$
4. Sample  $x_t^i$  from  $P(x_t | U_t, x_{t-1})$  using  $x_{t-1}^{(j)(i)}$  and  $U_t$
5.  $w_t^i = P(Z_t | x_t^i)$
6.  $\eta = \eta + w_t^i$
7.  $S_t = \{S_t \cup \langle x_t^j, w_t^i \rangle\}$
8. *for*  $i$  *in*  $1 \dots n$

$$w_t^i = \frac{w_t^i}{\eta}$$

Step (1) initializes an empty set of particles with the normalization factor set to zero, step (2-3) generates a probabilistic distribution of particles indexed by  $j(i)$  and weighted by  $w^i$ . step (4) generates a sample of posterior probabilities characterized by action step  $U_t$ . Step (5, 6, & 7) computes the important weights of each particle at a position  $x_t^j$  characterized by sensor readings, updates the normalization factor and generates a new set of particles  $S_t$  via a resampling algorithm. Finally, the weights are normalized using the normalization factor in step (8).

### **3.7 Conclusion**

This chapter described the background theory on which our proposed paradigm for optimizing control of multi-agents is based, thereby revealing the intricate parts of the model adapted for investigating corporative control of multi-agents. In the next chapter, we take the first step towards our proposed model by demonstrating the superiority of a novel hybrid optimization metaheuristic algorithm using 30 different benchmark multi-modal objective functions.

## CHAPTER 4

### A HYBRID METAHEURISTIC OPTIMIZATION PARADIGM

#### 4.1 Introduction

The popular 'no free lunch' theorem suggest that all algorithms' performance are at par when averaged across all possible objective functions. Consequently, choosing the most appropriate algorithm is often more of an art than science. There exist certain features such as the modality, the basins, the valleys, the separability, and the dimensionality which contributes to the complexity of an objective function. While the separability and modality could affect the complexity of the function, the dimensionality and the domain range of the function has an exponential impact on function's search space. These features poses a great problem to every meta-heuristic optimizer. Consequently, we leverage the concept of hybridization such that the strengths and peculiarity of each meta-heuristic are harmonized and synergized into a more powerful and robust optimizer.

Hybrid algorithms usually combines the strengths of genetic algorithms along with the converging speed of any local optimizer. A couple of authors such as Kazarlis *et al.*, [9] implemented a scaled down genetic algorithm with a minute population size as a local optimizing strategy. The rationale behind hybridization is often to combine the explorative capabilities of a meta-heuristic algorithm such as GA with the speed at which a local optimizer converges. When the GA seem to gradually asymptotes, its assumed to at least be in the domain of the global solution, thereafter the local optimizing algorithm seizes the search process in an attempt to obtain an optimal solution.

Hybridization processes could begin with a Genetic algorithm until it decelerates before seeding a local optimizer. Alternatively, the genetic algorithm could begin with some local minima obtained from random starting points in the population and thereafter seed a local optimizer on a selected elite population in response to a predefined number of iterations. The resulting chromosome is then incorporated into the next generation.

#### 4.2 Hybrid Framework Methodology

Preliminary analysis of the reviewed meta-heuristic algorithms (appendix A) revealed the strengths of each algorithm on the 30 different benchmark objective functions. The classical GA employs a moderately balanced explorative and exploitative strategy while the cuckoos search algorithm is highly explorative. This quality of the cuckoo's algorithm gives it an edge over the classical GA when deployed on complex objective functions. The polygamy induced GA provides a highly

exploitative strategy. These diverse capabilities informed our choice of algorithms for the creation of the hybridized model.

The proposed model leverages on three different meta-heuristics combinations for solving the optimization problem. The three meta-heuristic of choice are the GA, CK, and POLY (i.e. GA-with polygamy). Using 3 meta-heuristics avails us with 3 factorial (3!) possible unique combinations of the meta-heuristics. For example [CK, GA, POLY] with [CK<sup>2</sup>, GA<sup>4</sup>, POLY<sup>5</sup>] where the superscripts represent the duration of sub epoch assigned to each meta-heuristic. The algorithm (Table 4.1) begins by first sampling a random population of 50 ( $M$ ) chromosomes with scalable dimension size of 2 ( $x_1, x_2$ ) continuous variables. Simultaneously, a subpopulation ( $p \leq M$ ) of random combinations of meta-heuristics are spurned to evolve or optimize the matrix of chromosomes towards an optimal solution to the given objective function. It is important to note that each meta-heuristic randomly obtains a duration ( $f_i^{k(i)}$ ) of range [0, 5]. Thus the maximum number of sub epoch for each  $p \leq N(\max)$  where  $N(\max) = 15$ .

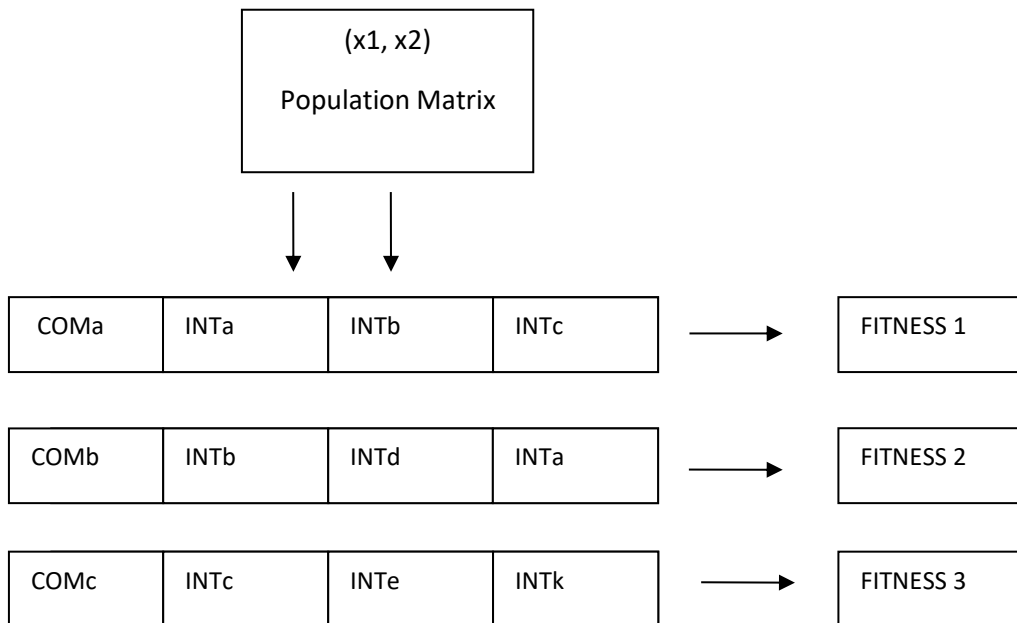


Fig. 4.1 High level abstraction of the hybrid frame work

From Fig. 4.1, the COMa represents the stochastic combination or order of meta-heuristic implementation. A typical initial order for COMa = 0, could be [CK<sup>2</sup>, GA<sup>4</sup>,POLY<sup>5</sup>], for [INTa, INTb, INTc] respectively, with the supper scripts representing the number of sub epochs (duration) each algorithm is permitted to run.

At the end of the first main epoch, a typical GA style algorithm (without the crossover operator) is used to select and mutate the  $p$  chromosomes. Exempting the crossover operator reduces the computational complexity and helps the hybrid algorithm evolve faster.

Table 4.1. Algorithm for Hybridized optimization Model

<p><math>X^* \rightarrow 0</math>, <math>M \rightarrow</math> population size, <math>N \rightarrow</math> duration of sub epochs</p> $k = \text{duration}(\text{randint}[0\ 5])$ $p = \{f_1^{k(1)}, f_2^{k(2)}, \dots \dots f_n^{k(n)}\} \leq M$ $f_i^{k(i)} = \text{metaheuristic}(i) \text{ with duration}(k(i))$ $N = \sum_{i=0}^n f_i^{k(i)} \forall f_i \in P, s.t \ n \leq N(\text{max})$ <ol style="list-style-type: none"> <li>1. Sample New Random population (<math>p</math>) of size <math>M</math></li> <li>2. While (<math>M</math>)</li> <li>3.     For each (<math>p(i) &lt; N</math>)</li> <li>4.     Evaluate <math>p(i) = \{f_1^{k(1)}, f_2^{k(2)}, \dots \dots f_n^{k(n)}\}</math></li> <li>5.     <math>X^* \leftarrow</math> update best local optima for each <math>p(i)</math></li> <li>6.     End (for loop)</li> <li>7. Preserve elite schema</li> <li>8. Evaluate mutation condition = (TRUE)</li> <li>9. Mutate (<math>f_i^{k(i)}</math>) order of combinations, mutate (<math>k(i)</math>) duration</li> <li>10. Return best <math>p(i)</math> schema order, <math>X^{best} \rightarrow</math> parameters, <math>X^{best} \rightarrow</math> global optima</li> </ol>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Elitism is used to preserve the best 5 performing chromosomes to the next generation. The mutation process alters the combination (or order) of meta-heuristics along with their respective superscript durations.

Leveraging tournament selection, as a selection strategy, the fittest chromosomes are passed to the next generation. The  $X^*$  (global minima) is updated at the end of each main epoch. For the research, the termination criteria was set at 50 epochs (MAX) or when the optimal solution has been found.

### **4.3 Polygamy (POLY) as an Exploitative Strategy for GAs**

The concept of diversity and exploitation are two paradigms that have contributed immensely to the success of the GA. While diversity attempts to prevent the algorithm from stagnating within a local optimum, exploitation on the other hand attempts to achieve faster convergence of the algorithm.

One approach towards maintaining diversity within a population is by replacing existing identical solution strains with newly formed strains especially in cases where they exist multiple similar strains in the population [100].

Other methods include a mechanism for favoring dissimilar strains while similar ones are discouraged leading to convergence on multiple peaks [101]. Another related approach is to restrict mating among similar strains while encouraging mating among dissimilar ones thereby increasing diversity [102]. Similarly, a tag stamping mechanism has been used to indicate strains that are eligible to mate as they pass from one generation to another [103].

Polygamy on the other hand attempts to explore the power of exploitation. Using this approach, every strain within the population is forced to mate with the fittest strain within each generation. A clever implementation of this strategy helped the algorithm converge faster to the global optima. Polygamy behavior was allowed when only little improvement was observed within five



consecutive generations’ thus fine-tuning the population towards the global optima.

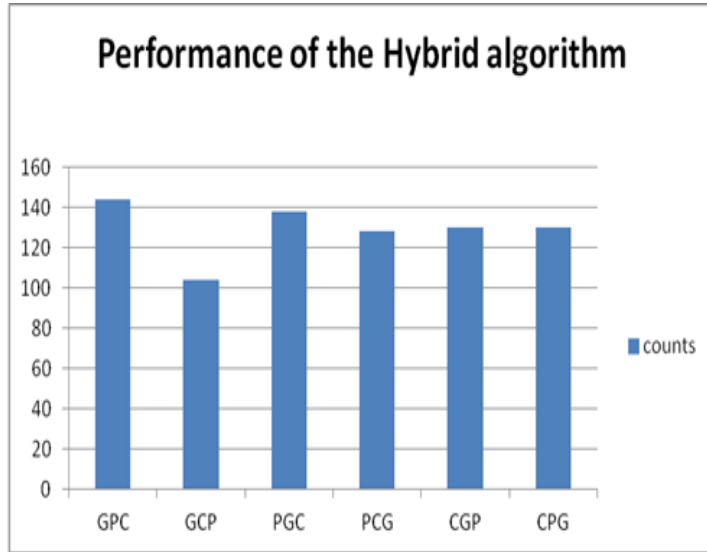


Fig. 4.2. Hybrid frame work performance chart for all objective functions evaluations. (Height indicative of the number of times each combination was responsible for the optimal solution.)

#### 4.4 Experiments

On the basis of ANOVA, we reject the NULL Hypothesis ( $H_0$ ) on the premise of a significant difference between the mean optimal values of our tested meta-heuristics. Consequently, we proceed with our analysis to discover the pairs of meta-heuristics that differ significantly. For this purpose, we use SHEFFÉ’s test. The test states that “the critical difference (CD) for each pair of meta-heuristic can be obtained using the equation below:”

$$CD_{ith,Jth} = \sqrt{MS_{within} \left( \frac{1}{n_i} + \frac{1}{n_j} \right) * (k - 1) * f_{k-1,n-k} * \alpha * n} \quad (4.1)$$

where,

$\alpha$  = critical value at 5% significance;

$k$  = number of meta-heuristics;

$n$  = number of test bench mark functions;

$MS_{within}$  = ANOVA mean square within samples.

In this research, we used 30 benchmark functions to test the success of the CK, PSO, SA, POLY (a proposed exploitative strategy), and HYB (our proposed hybrid framework). Table 4.2 Shows the best optimal values for each meta-heuristic on the benchmark functions compared against standard expected optimal solutions  $f(x^*)$ . A fixed dimension size of 2 was implemented for all benchmark functions. The population size was preset at 50 while the maximum number of function evaluations for each iteration was set to 500. For the purpose of cohesion, the global minimal values below  $-10^{-15}$  were considered as zero (0) in all experiments.

The global optima (minima ( $X^{best}$ )) of each benchmark function was evaluated 20 times using random initial population at every instance (see addendum A for detailed tables). The mean performance of optimal solutions and durations (number of function calls) during each phase of the experiments were recorded for further analysis.

Also, two ANOVA tests were conducted for multiple comparisons of the performance of each meta-heuristic algorithm.

### **Hypothesis 1 (H1)**

The Null hypothesis for the first ANOVA test is stated as follows: ***“There is no difference in the speed of convergence to the global optima among test meta-heuristic algorithms”*** (Table 4.3)

### **Hypothesis 2 (H2)**

The Null hypothesis for the second ANOVA test is stated as follows: ***“There is no difference in the mean global solutions between the tested meta-heuristics”*** (Table 4.4)

Both hypothesis were tested with 95% confidence ( $\alpha = 0.05$ ).

Considering the relative complexity of the Hybrid framework, the algorithm was allowed to run for  $1/10^{\text{th}}$  of the max allowed epoch of 500. The rationale was to give all algorithms a level playing ground as each one ran for approximately equal CPU time. Fig. 4.2 shows the Hybrid framework performance chart for all objective functions evaluations. The height of the bar chart is indicative of the number of times each combination was responsible for the optimal solution.

Table 4.2 Minimum optimal values for each meta-heuristic  $f(X^*)$  = cross validating global optimal Values.

	CK	SA	GA	PSO	POLY	HYB	$f(X^*)$
F1	-199.445	-198.373	-199.95	-192.542	-199.934	-199.99985	-200
F2	12.41074	15.92034	1.698666	1153.498	2.91809	1.0000125	1
F3	0.004353	0.005751	0.038877	1.553515	0.137703	0.00000445	0
F4	-106.459	-106.357	-106.725	-78.6656	-106.746	-106.7638	-106.765
F5	1.122465	1.652505	0.071357	42.05747	0.097529	1E-10	0
F6	0.658878	1.619049	0.092741	43.84189	0.067781	3.7165E-06	0
F7	0.013956	0.034721	0.037903	2.301233	0.022482	1.9705E-06	0
F8	-37.6089	-32.8167	-33.1681	-25.4354	-29.5683	-39.189025	-39.1956
F9	0	0	6.066888	9.16517	4.978992	0	0
F10	-1.02875	-1.02601	-1.03125	-0.81944	-1.0314	-1.03163	-1.03163
F11	0.001254	0.003813	6.42E-05	0.277253	0.00022	5.02E-08	0
F12	-42.716	-42.6382	-42.8708	-38.6017	-42.938	-42.9444	-42.9444
F13	0.035444	0.108808	0.230391	3.343998	0.220859	2.7823E-05	0
F14	-24565.6	-24381.5	-24743.6	-17984.4	-24730.8	-24776.5	-24776.5
F15	-0.74984	-0.4442	-0.80794	-1.8E-05	-0.85862	-1	-1
F16	0.449115	0.528326	1.094281	13.30241	1.731257	0.0002405	0
F17	-25.1967	-25.2065	-25.1565	-22.9869	-25.045	-25.2305	-25.2305
F18	-18.4309	-14.8903	-18.1753	-12.6669	-18.2862	-18.5547	-18.5547
F19	-343.423	-337.944	-345.357	-256.668	-345.313	-345.35985	-345.36
F20	0.020221	0.070733	4.35E-05	1.820651	0.000651	3.208E-07	0
F21	3.065329	3.082658	3.018892	4.676115	3.057481	3.0000135	3
F22	34.76376	36.27151	68.25507	75.53155	72.95737	34.086535	34.0412
F23	65.07671	81.28288	2.818507	3631.894	11.20961	3.8552E-05	0
F24	0.010749	0.065113	0.00069	0.45203	0.004565	0	0
F25	-10.8489	-10.837	-10.8603	-10.3874	-10.8607	-10.8723	-10.8723
F26	-0.96325	-0.96342	-0.95512	-0.86909	-0.9603	-0.963535	-0.963535
F27	-2.82207	-2.69868	-3.17427	-1.65882	-3.08633	-3.381076	-3.38814
F28	-2.02177	-2.00677	-1.77877	-1.44159	-1.78244	-2.02181	-2.02181
F29	-2.06159	-2.06093	-2.06261	-1.93824	-2.06228	-2.06261	-2.06261
F30	1.518037	2.736039	2.000005	3.508812	2.000301	0.21866506	0

Table 4.3 ANOVA analysis of mean function calls of each meta-heuristic for all objective functions. (\* indicates “no significant difference”)

The significantly different algorithms based on Mean Function calls  $\leq$  (500 EPOCHS)

Function	P-value	Algorithms					
		CK	SA	GA	PSO	POLY	HYB
F1	1.82898E-17	*	*	CK,SA, POLY, PSO	*	*	CK, SA, PSO, POLY
F2	1.07148E-10	*	*	CK	*	*	CK, POLY
F3	2.08378E-43	*	*	*	*	*	CK, SA, PSO, POLY,GA
F4	2.49674E-29	*	*	CK	*	CK	CK, SA, PSO, POLY,GA
F5	2.20353E-16	*	*	CK,POLY	*	CK	CK, POLY
F6	1.73217E-14	*	*	*	*	*	CK, SA, PSO, POLY,GA
F7	3.20673E-09	*	*	*	*	*	CK, SA, PSO, POLY,GA
F8	1.23486E-29	*	*	*	*	*	CK, SA, PSO, POLY,GA
F9	2.75464E-98	GA,PSO, POLY	*	*	*	*	GA, PSO, POLY
F10	9.70E-27	*	*	CK, SA, PSO	*	*	CK, SA, PSO, POLY,GA
F11	1.6558E-07	*	*	CK, SA, PSO, POLY	*	*	CK, SA, PSO
F12	1.0131E-27	*	*	CK, SA, PSO, POLY	*	*	CK, SA, PSO, POLY,GA
F13	-----	*	*	*	*	*	*
F14	9.59387E-26	*	*	CK, SA, PSO	*	*	CK, SA, PSO, POLY,GA
F15	1.59751E-16	*	*	CK, SA, PSO	*	*	CK, SA, PSO, POLY,GA
F16	-----	*	*	*	*	*	*
F17	2.50691E-48	*	*	*	*	*	CK, SA, PSO, POLY,GA
F18	9.84919E-28	*	*	*	*	*	CK, SA, PSO, POLY,GA
F19	7.07297E-21	*	*	CK, SA, PSO	*	*	CK, SA, PSO, POLY
F20	5.20026E-08	*	*	CK, SA, PSO, POLY	*	*	*
F21	1.48091E-11	*	*	*	*	*	CK, SA, PSO, POLY,GA
F22	-----	*	*	*	*	*	*
F23	-----	*	*	*	*	*	*
F24	1.8132E-15	*	*	CK, SA, PSO, POLY	*	*	CK, SA, PSO, POLY
F25	4.61192E-27	*	*	CK, SA, PSO	*	*	CK, SA, PSO, POLY,GA
F26	1.65387E-32	*	*	*	*	*	CK, SA, PSO, POLY,GA
F27	-----	*	*	*	*	*	*
F28	1.34422E-45	GA, PSO, POLY	*	*	*	*	CK, SA, PSO, POLY,GA
F29	1.52471E-39	*	*	CK, SA, PSO, POLY	*	*	CK, SA, PSO, POLY
F30	-----	*	*	*	*	*	*

Table 4.4. ANOVA analysis of mean global optimal values of each meta-heuristic for all objective functions (\* indicates “no significant difference”)

The significantly different algorithms based on minimum optimal values (X\*) @ 500 EPOCHS

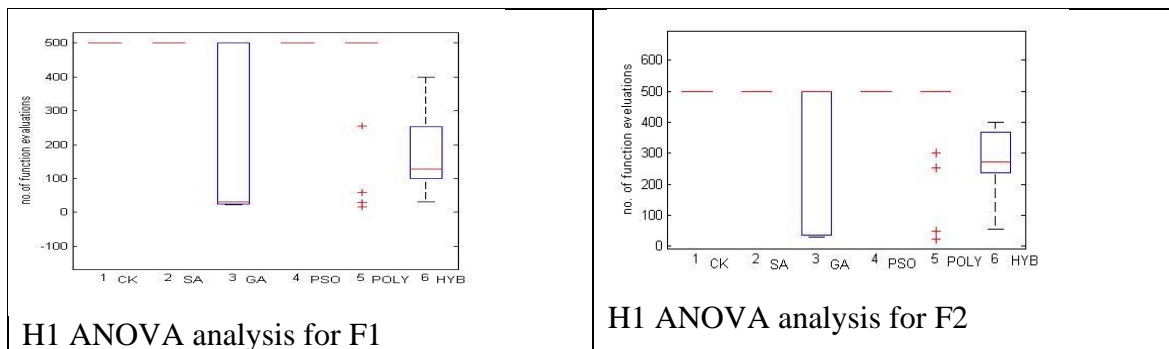
Function	P-value	Algorithms					
		CK	SA	GA	PSO	POLY	HYB
F1	4.42439E-45	PSO	PSO	SA, PSO	*	PSO	SA, PSO
F2	5.73609E-20	PSO	PSO	PSO	*	PSO	PSO
F3	4.2247E-13	PSO	PSO	PSO	*	PSO	PSO
F4	5.09685E-19	PSO	PSO	PSO	*	PSO	PSO
F5	9.08241E-22	PSO	PSO	PSO	*	PSO	PSO
F6	7.49381E-37	PSO	PSO	PSO	*	PSO	PSO
F7	3.49218E-15	PSO	PSO	PSO	*	PSO	PSO
F8	3.2879E-33	GA, SA, POLY, PSO	POLY, PSO	POLY, PSO	*	PSO	GA, SA, POLY, PSO
F9	1.10236E-14	POLY, PSO	POLY, PSO	SA, CK	*	PSO	GA, POLY, PSO
F10	1.51277E-22	PSO	PSO	PSO	*	PSO	PSO
F11	9.35908E-16	PSO	PSO	PSO	*	PSO	PSO
F12	1.68892E-14	PSO	PSO	PSO	*	PSO	PSO
F13	1.62595E-19	PSO	PSO	PSO	*	PSO	PSO
F14	9.59387E-26	PSO	PSO	PSO	*	PSO	PSO
F15	1.39766E-26	SA, PSO	PSO	SA, PSO	*	PSO, SA	CK, SA, PSO
F16	4.45976E-18	PSO	PSO	PSO	*	PSO	PSO
F17	1.26316E-19	PSO	PSO	PSO	*	PSO	PSO
F18	9.84919E-28	SA, PSO	PSO	SA, PSO	*	PSO, SA	SA, PSO
F19	3.62808E-37	PSO	PSO	PSO	*	PSO	PSO
F20	1.55505E-10	PSO	PSO	PSO	*	PSO	PSO
F21	2.68003E-11	PSO	PSO	PSO	*	PSO	PSO
F22	1.30234E-57	GA, POLY, PSO	GA, POLY, PSO	PSO	*	*	GA, POLY, PSO
F23	2.8735E-16	PSO	PSO	PSO	*	PSO	PSO
F24	4.33902E-60	SA, PSO	PSO	SA, PSO	*	SA, PSO	SA, PSO
F25	1.47518E-34	PSO	PSO	PSO	*	PSO	PSO
F26	3.7071E-35	PSO	PSO	PSO	*	PSO	PSO
F27	3.70578E-41	PSO	PSO	CK, SA, PSO	*	CK, SA, PSO	CK, SA, POLY, PSO
F28	2.02948E-40	GA, POLY, PSO	GA, POLY, PSO	PSO	*	PSO	GA, POLY, PSO
F29	6.21067E-30	PSO	PSO	PSO	*	PSO	PSO
F30	1.0275E-27	SA, PSO	PSO	SA, PSO	*	SA, PSO	GA, CK, SA, POLY, PSO

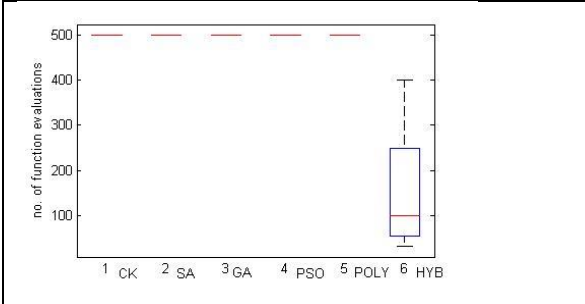
## 4.5 Summary of Results

### 4.5.1 Hypothesis 1 (H1)

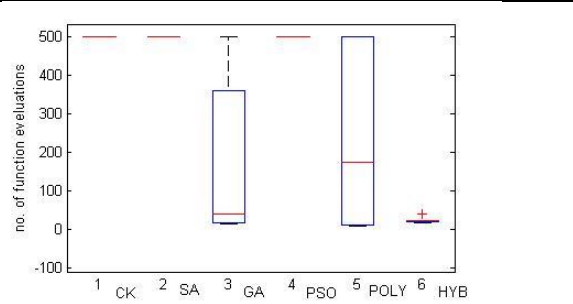
The success of the proposed Hybrid model, CK, GA, SA, and a modified GA (polygamy induced) has been statistically compared. When compared over hypothesis 1 (Table 4.5(a.)) which is based on the mean function calls (speed of convergence), the proposed hybrid framework displayed superiority over all other models. Next to the Hybridized model was the GA algorithm, thereafter the CK search algorithm. It was observed that the CK model outperformed the GA (with a statistically significant difference) in just 2 out of the 30 benchmark functions (the F9 – ‘brent function’ and the F28- ‘Adjiman Function’) which are relatively complex functions in terms of their differentiability, modality, separability, and modality. The PSO owes its performance to its stability problem and also, the amount of permitted epochs for evaluation used in most PSO implementations is usually high (approx. 2 million) [36], [37], [38] for each benchmark function, as against the 500 epochs used in this research. There exists no significant difference in the speed of convergence (in cases where convergence occurred) between the SA, PSO, and the polygamy induced GA. However, the polygamy induced GA performed better than the CK in two of the benchmark functions (F4-bird Function, F5- Bohachevsky 1 Function).

Table 4.5. (a) Box plot representation of Anova on the mean number of function calls (H1). Anova on hypothesis 1(H1)- the mean number of function calls

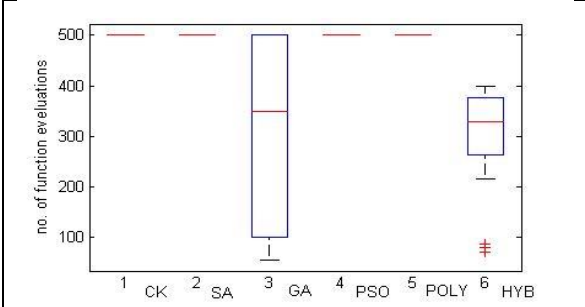




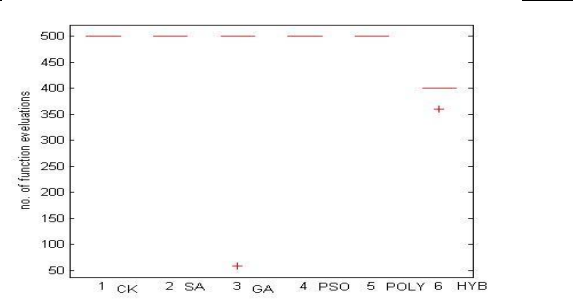
H1 ANOVA analysis for F3



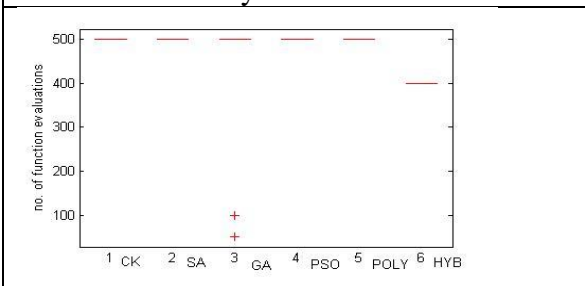
H1 ANOVA analysis for F4



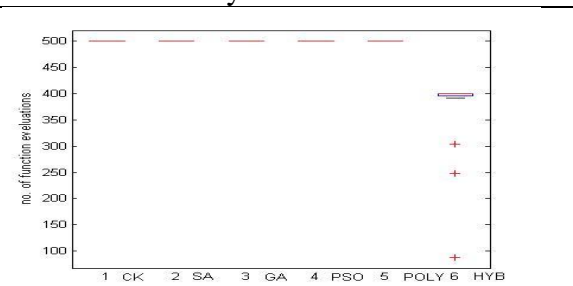
H1 ANOVA analysis for F5



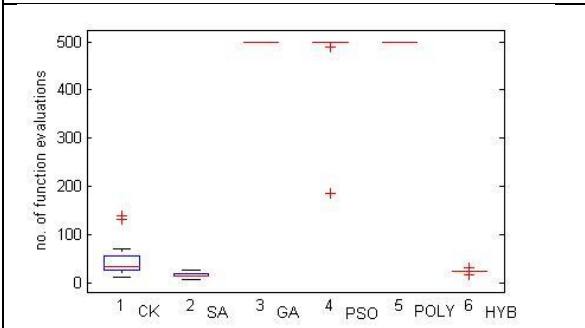
H1 ANOVA analysis for F6



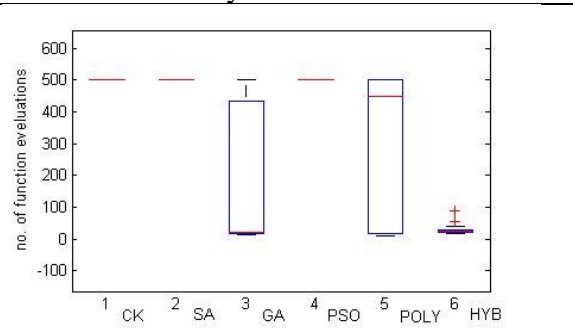
H1 ANOVA analysis for F7



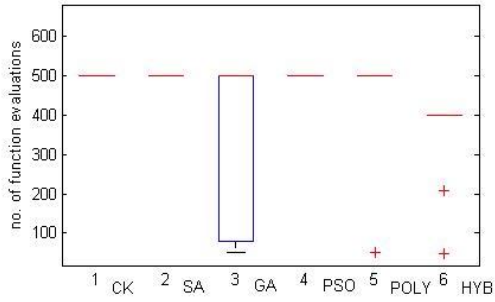
H1 ANOVA analysis for F8



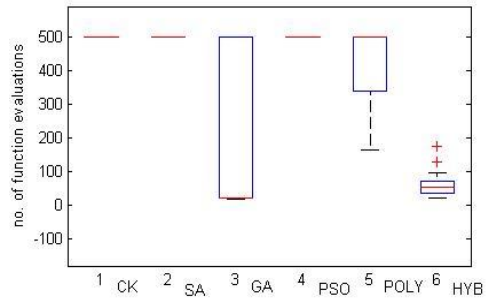
H1 ANOVA analysis for F9



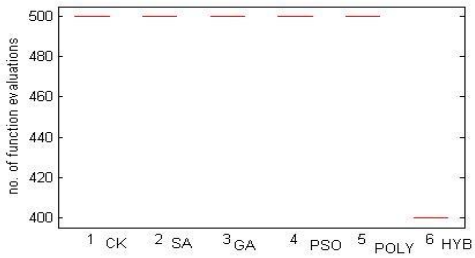
H1 ANOVA analysis for F10



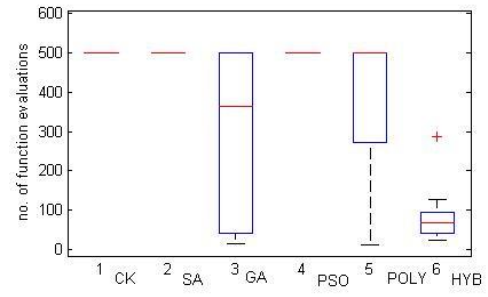
H1 ANOVA analysis for F11



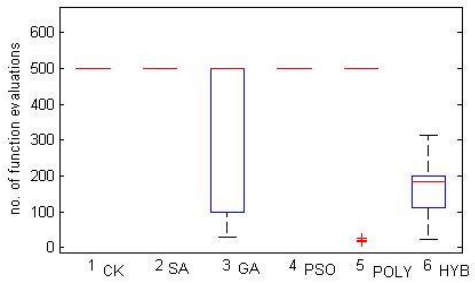
H1 ANOVA analysis for F12



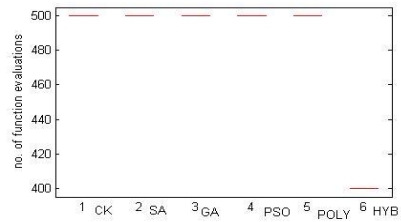
H1 ANOVA analysis for F13



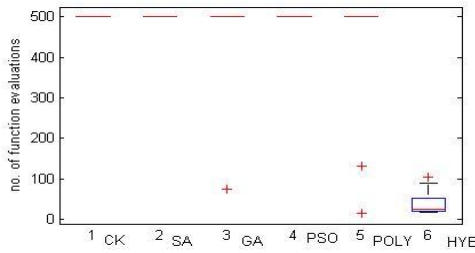
H1 ANOVA analysis for F14



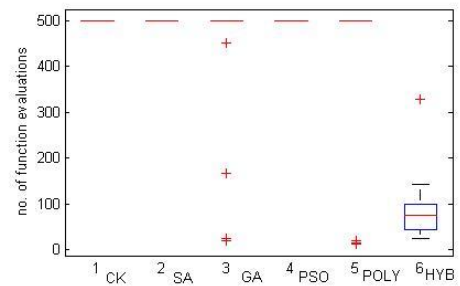
H1 ANOVA analysis for F15



H1 ANOVA analysis for F16

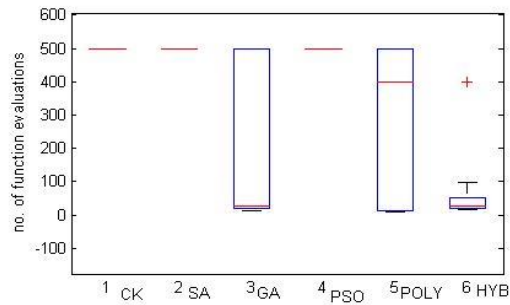


H1 ANOVA analysis for F17

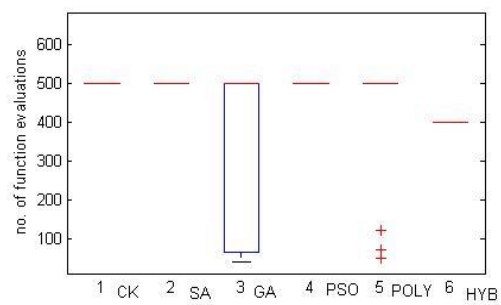


H1 ANOVA analysis for F18

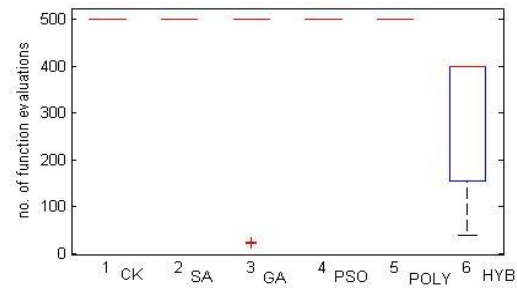




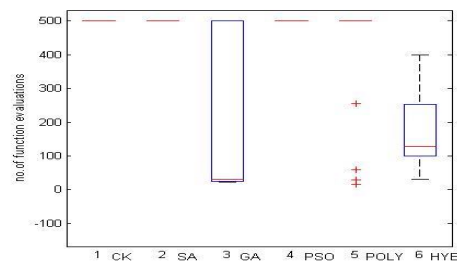
H1 ANOVA analysis for F19



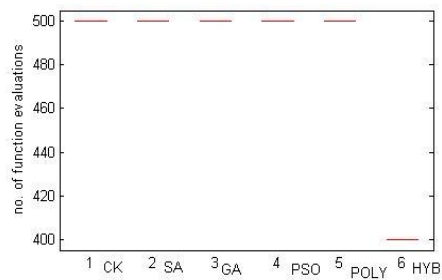
H1 ANOVA analysis for F20



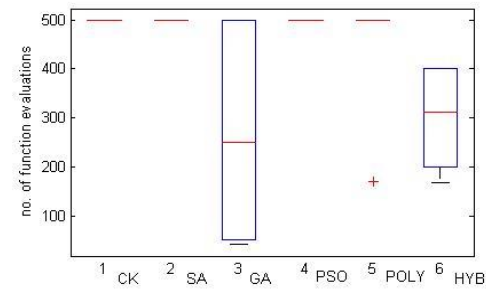
H1 ANOVA analysis for F21



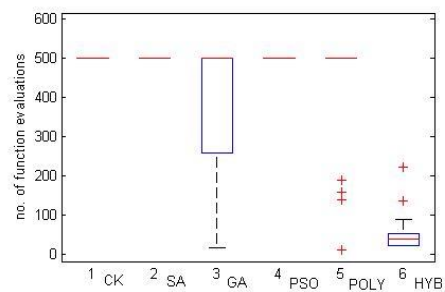
H1 ANOVA analysis for F22



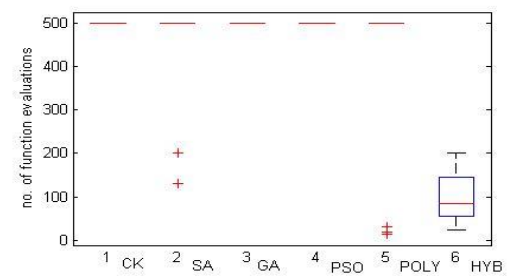
H1 ANOVA analysis for F23



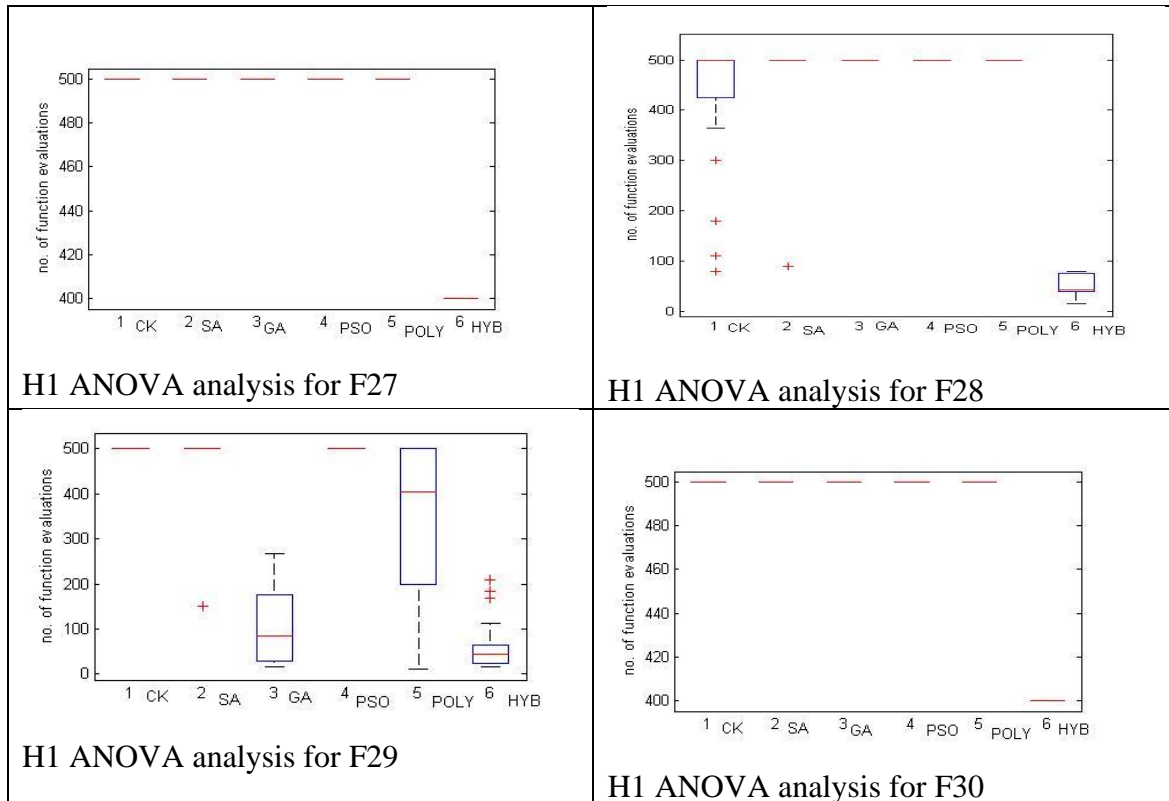
H1 ANOVA analysis for F24



H1 ANOVA analysis for F25



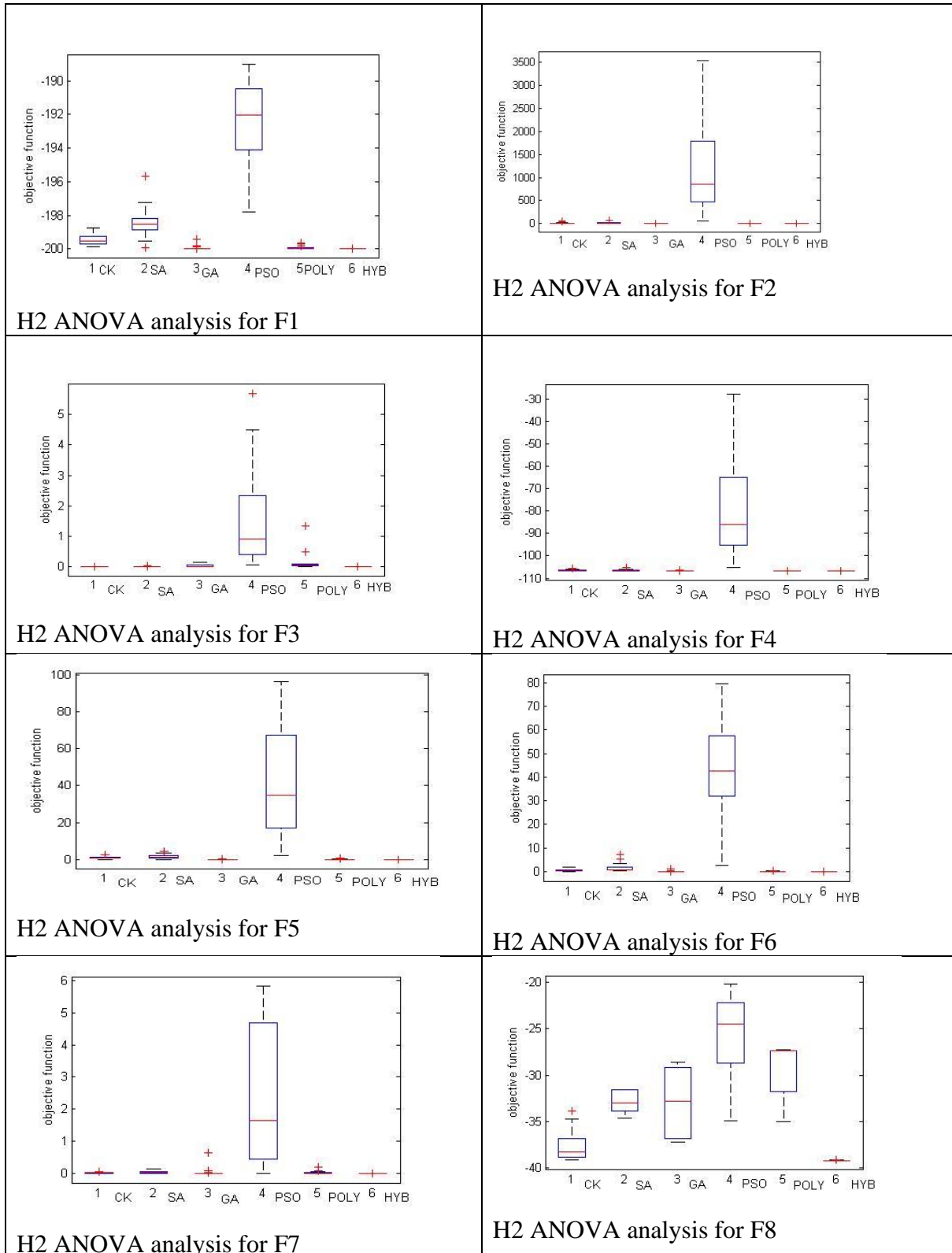
H1 ANOVA analysis for F26

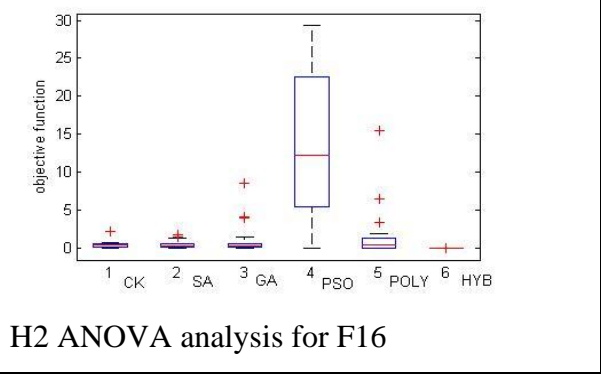
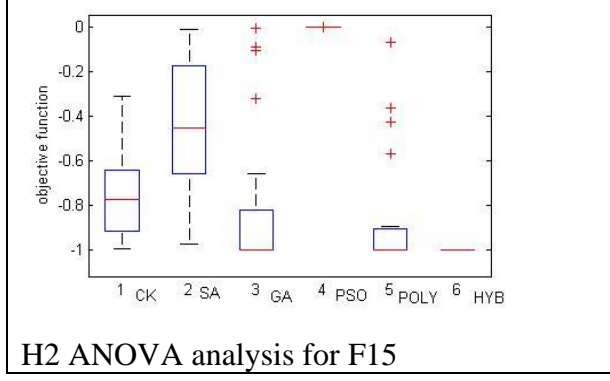
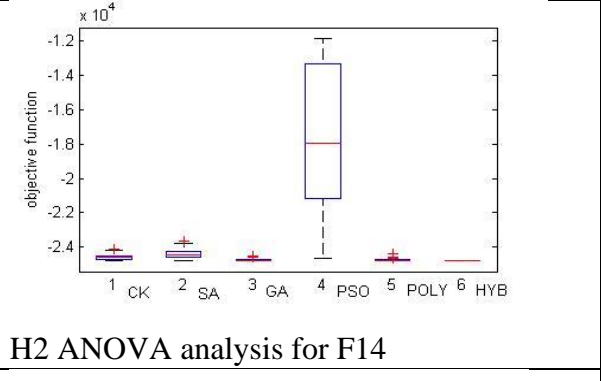
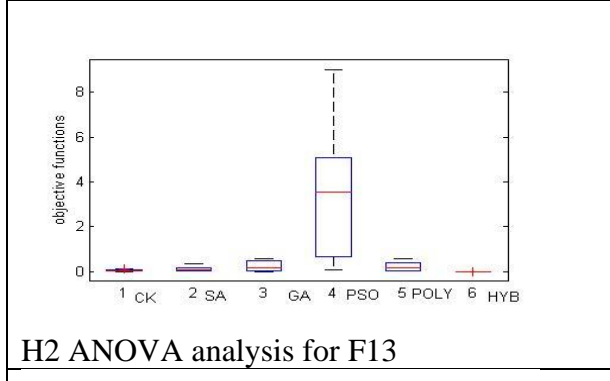
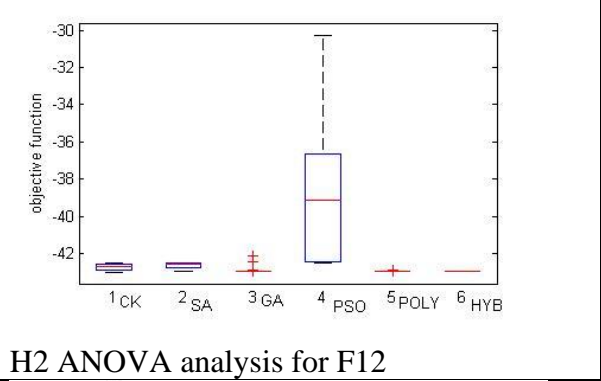
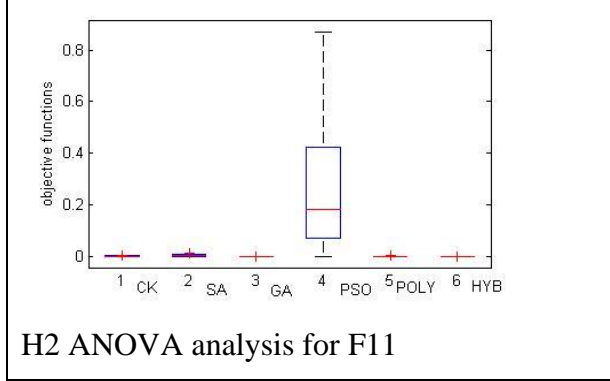
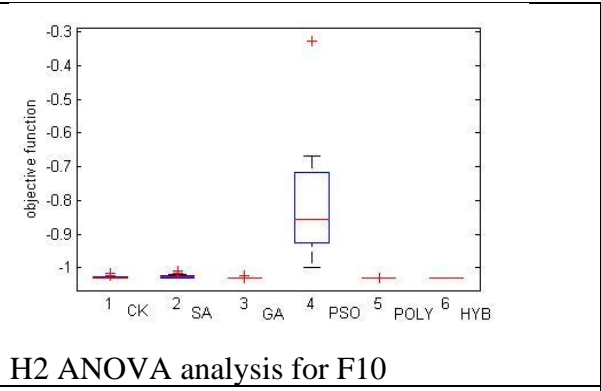
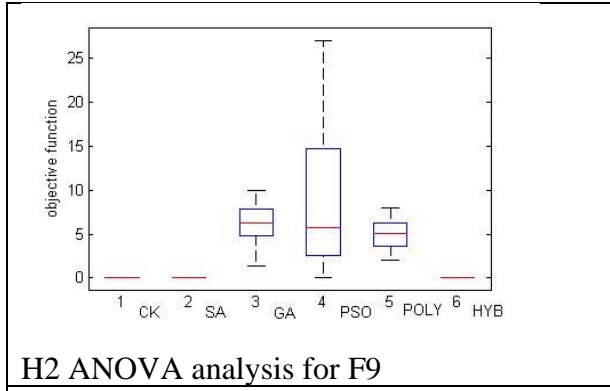


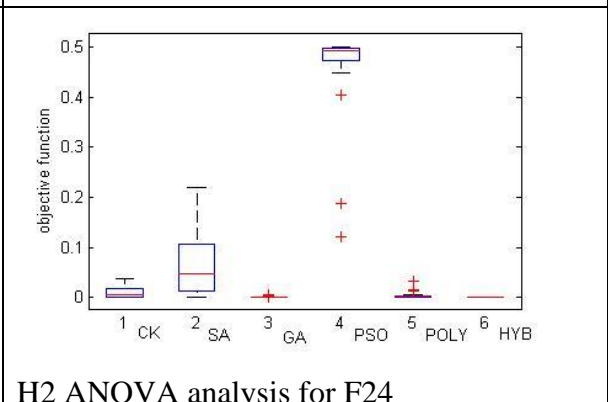
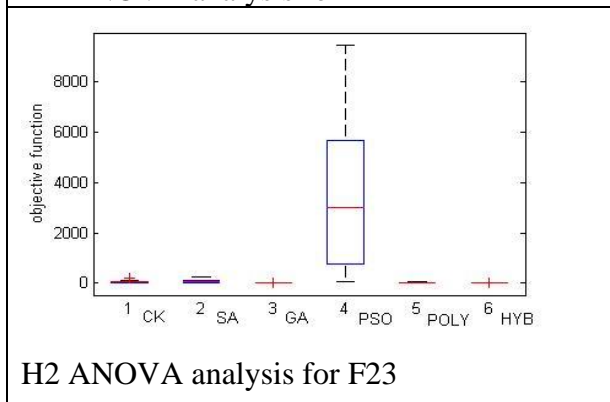
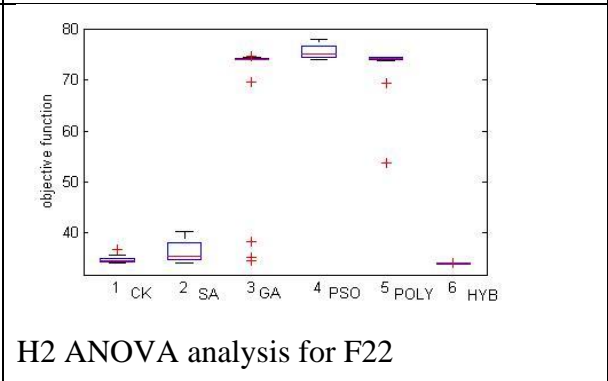
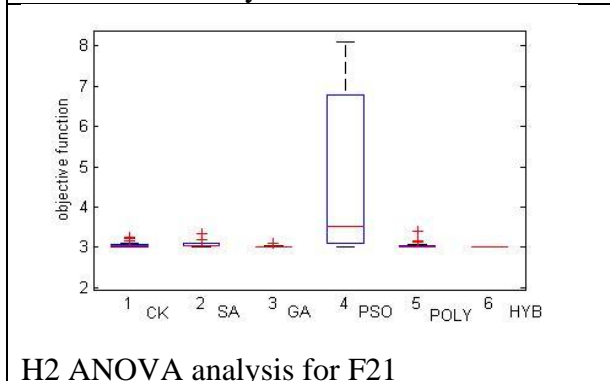
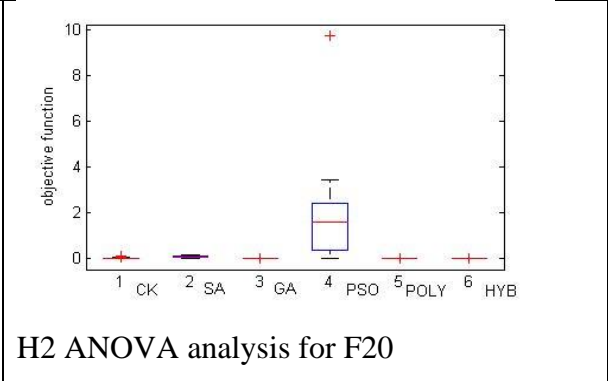
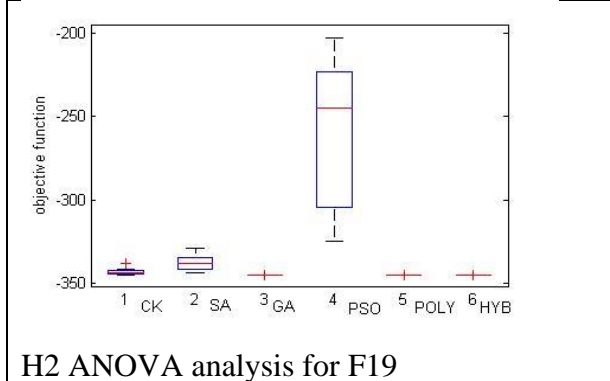
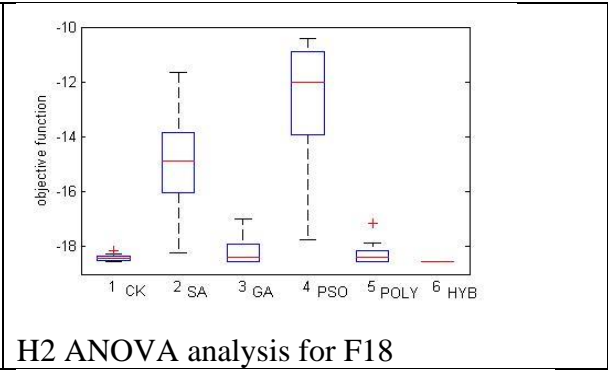
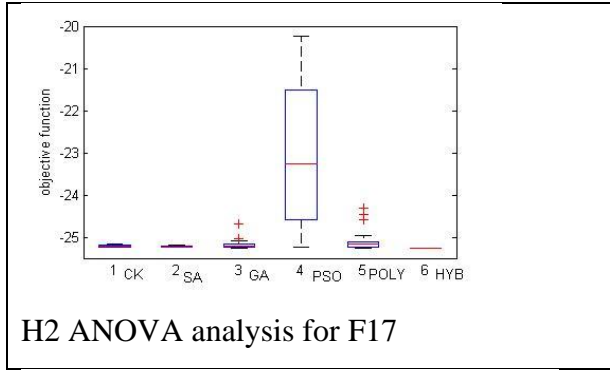
#### 4.5.2 Hypothesis 2 (H2)

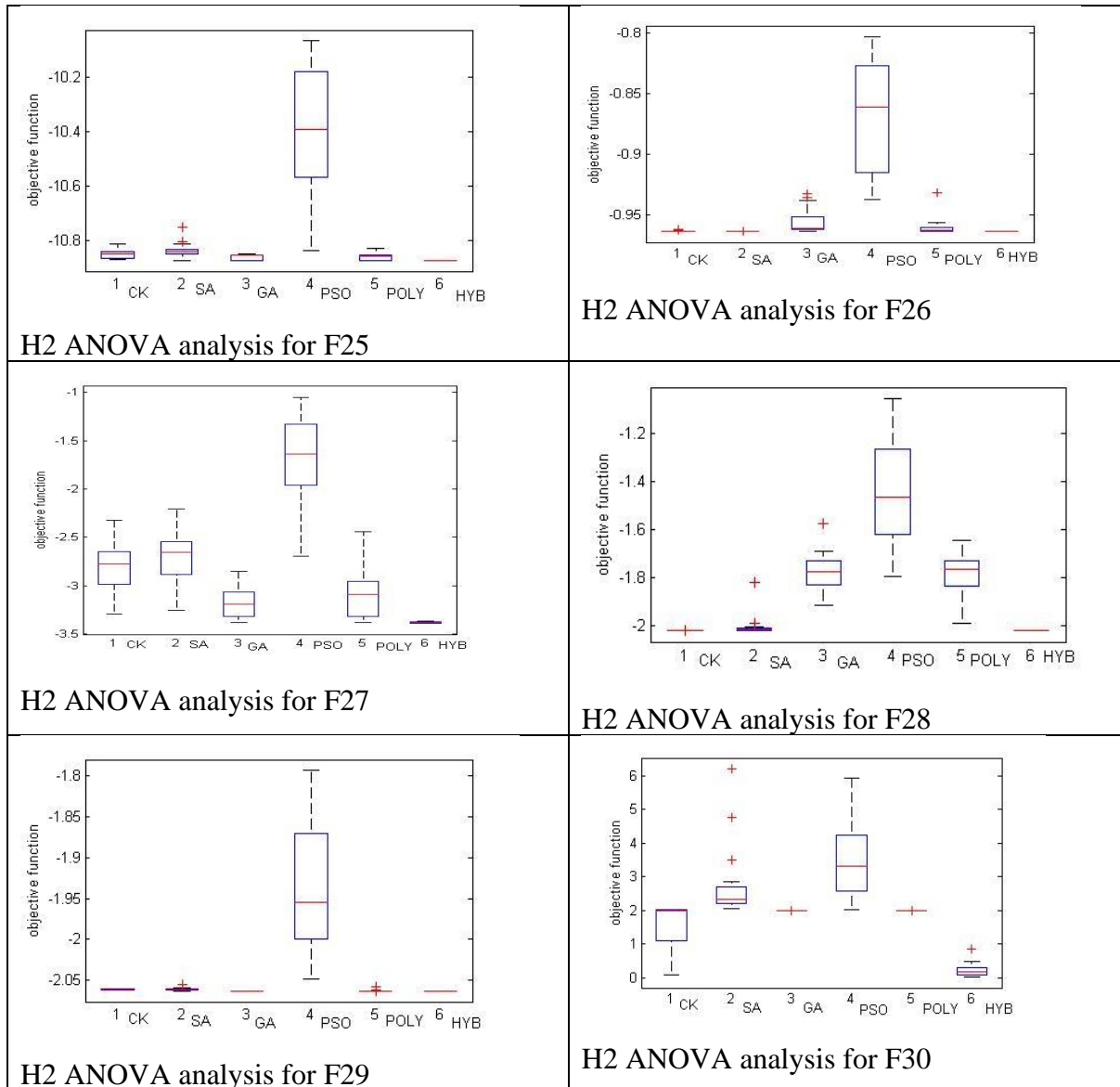
The success of the proposed Hybrid model can also be seen from the mean global optimal value as shown from the ANOVA analysis (Table 4.5(b.)) In addition to the convergence speed advantage, the proposed hybrid model has a significantly better performance when compared with the other meta-heuristics with an impressive advantage over the traditional GA in (F8- ‘Branin RCOS-2 Function’, F9-‘Brent function’, F22- ‘Rosenbrock Modified Function’ F28-‘Adjiman Function’ and F30- ‘Damavandi Function’). These functions are recognized for their complexity in terms of their differentiability, modality, separability, and modality. We can infer from statistical results that the speed of convergence is a major advantage of our proposed model while its ability to consistently converge at the global optima within a short period is an added advantage.

4.5 (b) Anova on hypothesis 2(H2)- the mean best optimal value obtained for each meta-heuristic algorithm.









#### 4.6 Discussion

The global minimum values of each of the benchmark functions used in this research has been solved 20 times with the CA, GA, PSO, SA, and the hybrid model algorithms using the same initial population at every turn. The run-time and the minimum function calls of the best solution, and the final global optimal values (table 4.2) has been documented during experiments and used for further statistical analysis. Subsequently, the ANOVA analysis of mean function calls of each meta-heuristic for all objective functions has been computed along with the ANOVA analysis of mean global optimal values of each meta-heuristic for all objective functions.

As it is seen from Table 4.3, the performances of the Hybrid algorithm discovers the optimal values in all functions, faster than the GA, CK, SA, algorithms. The "\*" indicates "no significant difference". The CK displayed statistical advantage over GA, PSO, and POLY in F9 and F28 benchmark functions. The GA outperformed the CK with a statistical advantage over CK, PSO, SA, and POLY in 14 benchmark functions out of the 30 benchmark functions. Our hybrid model outperformed all metaheuristics used for our experiment in 22 out of the 30 benchmark functions. It's important to note that none of the metaheuristics performed significantly better (faster) than our proposed hybrid model of benchmark functions.

Similarly, in Table 4.4, we show the minimum mean optimal values for the 30 benchmark functions. Again, we see our proposed hybrid model display superior performance over the metaheuristic algorithms with statistical significance in more than 9 benchmark functions.

Table 4.5(a) and 4.5(b) shows a boxplot ANOVA which reveals the median and spread of the number of functions calls (epochs) and the minimum optimal values respectively, on the metaheuristics and the hybrid model over the 30 benchmark functions.

## **4.7 Conclusion**

In this chapter, we present a superior metaheuristic hybrid model when compared empirically with the CK, SA, GA, PSO, and a modified GA in solving optimization problems. The hybrid framework systematically combines the strengths of multiple meta-heuristics leveraging on the traditional GA mutation strategy. The empirical analysis revealed faster convergence to a global optimum with minimal computational complexity. The framework provides an axis for further research work on the scalability (max number of meta-heuristics) and also the efficiency of the algorithm when combined with other meta-heuristics apart from those used in this research such as ABC (artificial bee colony), DE (Differential Evolution), Ant Colony, etc. In the next chapter, we investigate the success of such a framework, on multi-objective optimization problems with respect to proffering solutions to the multisource localization problem.

## CHAPTER 5

### MULTI-OBJECTIVE OPTIMIZATION APPROACH TO LOCALIZING MULTIPLE EMISSION SOURCES

#### 5.1 Introduction

Among the existing algorithms for multisource localization, we do not find clear recommended techniques for robotic agents to proceed with a search after a source is found. The missing piece for the multisource localization problem is to find theoretic frameworks that would guarantee progress (after a source has been localized), while driving the algorithm towards convergence, and a proper termination of the search algorithm. Gazi and Passino [4] works on “attractant/repellent swarm” has come close to solving this problem but however, the work was limited to single-source searches [104].

When solving the multisource localization problems, some factors are taken into consideration; such as: the complexity of the solution, the type of source/target and the predictability of the environmental variables. The Bayesian Occupancy grid algorithm [13] is one commendable attempt in providing solution to the above stated factors. Although this algorithm provides a near optimal solution to the multisource problems, it however does not provide a clear path towards progress after a gradient source has been localized. In addition, the literature lacked adequate comparative analysis with other biological models/algorithms.

Finally, the question on the modalities for choosing the dependent and independent variables for the purpose of a standardized comparative analysis has been left unanswered. Consequently, there is a need for the validation of a variables such as: the initial distribution of gradient sources, the location and presence of obstacles, etc. Such standards could help in comparing and contrasting different models and algorithms against the back drop of their merits and demerits for disparate domains. In this thesis, we provide an array of groundbreaking paradigms while addressing the problems of the multisource localization.

In this chapter, we present an algorithm which combines dynamic programming, NSGA-II optimizer on a feedforward artificial neural network model, for a pair of robots simulated with simple yaw and thrust motions in an attempt to achieve oblivious collaboration and control while



localizing multiple gradient sources. This methodology unveiled a potential solution to the problem of progress and termination of multiple emission source localization.

## 5.2 METHODOLOGY

The missing piece of the multisource search puzzle is the absence of theoretical foundations for progress (after a source is found), convergence, and termination of the search operation algorithm. Motivated by this problem we commence our journey towards a solution to the multi-source localization problems by adapting biological inspired algorithm for optimizing robot trajectory within the search environment based on evolutionary neural networks (ENN) with respect to predefined objective functions. Cooperation among multi-agents could be active (agents acknowledging each other) or inactive (agents oblivious of each other). Motivated by the simplicity and independence of oblivious agents, we propose a solution to the multi-source localization problems by leveraging inactive cooperation amongst robots, with respect to unveiling a “methodology for oblivious collaboration” among agents while localizing multiple gradient sources simultaneously.

### 5.2.1 ENNs (Evolutional Neural Networks) using Discrete Variables

We introduce the section with preliminary experiments based on a grid world which has fixed obstacles within the search space. The GA begins by defining a chromosome or an array of variable values to be optimized. Each chromosome is an array of possible actions the robot can take such as: [Rotate Left (RL), Rotate right (RR), Thrust (T), Back-up (B), Pause (P)]

Consequently, if the chromosome has N variables, then the chromosome could be written as an N element Variable vector:  $chromosome = [RL, RL, RR, P, B, T, T, T, T, RL, RR, RR, RR, \dots N^{th}]$

Each chromosome has a fitness score derived by evaluating the fitness function  $f$  of each chromosome.

$$Fitness = f(chromosome) = f(RL, RL, RR, P, B, T, T, T, T, RL, RR, RR, RR, \dots N^{th}).$$

Since we are trying to find the chromosome with a maximum number of explored cells, the fitness score is thus adapted from a maximization function. Consequently, the optimization algorithm searches for the global maxima. The total number of explored cells by each robot (chromosome) forms the fitness of that specie or instance of the robot.

### 5.2.2 Discrete variable encoding and decoding

The discretized GA implementation works with action encodings. Whenever the fitness function is evaluated, the chromosome must first encode and then decode the variables into a vector of actions.

A chromosome is an array of genes. Each gene is encoded by randomly generating an action from the fixed range of possible actions and also randomly generating a duration 'd' from a clamped range of durations such that  $(0 < d \leq 30)$ .

Thus, for  $action = [RL, RR, T, P]$ ,

The chromosome =  $([RL][d1], [RR][d2], [P][d3], [T][d4], \dots, [action]^{th}[d]^{th})$  such that  $[RL][d1]$  is a represents a typical example of an encoded gene.

#### Decoding the gene:

A two-gene chromosome for example  $([RR][4], [T][2])$ , would cumulate into a chromosome of six vector actions as shown below:

Chromosome =  $[RR, RR, RR, RR, T, T]$

### 5.2.3 Objective Function

The objective function (in this case, the grid world) determines the fitness score. The GA attempts to optimize an exhaustive search of the entire search space. The algorithm randomly generates a set of a hundred chromosomes for a hundred robots with an initial fitness score of zero. At the end of the first generation, the set of a discrete variable (chromosomes) that can navigate the robot through the highest number of grid cells becomes the fittest chromosome (highest fitness score). The fitness score is directly proportional to the total number of explored cells by the robot at the end of each run.

### 5.2.4 The population

The GA starts with a group of chromosomes known as the population. Each chromosome corresponds to a set of discrete action vectors for navigating the grid map (fitness function). Because each chromosome also corresponds to an instance of a robot, a population of 100

robots/chromosomes was randomly generated with each robot starting with a fitness score = 0. After the first epoch, where each robot's vector of actions is tested against the fitness function, the best-fit chromosome would have the highest fitness score while the worst fit chromosome would have the lowest fitness score.

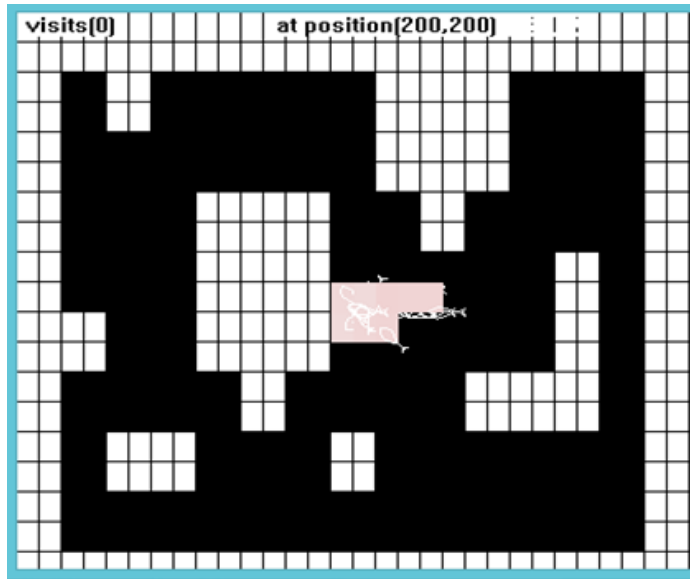


Fig. 5.1 Instantiation of a spool of 100 evolving robots within the obstacle oriented grid map

### 5.2.5 Constraints

In our implementation, the motion of the robot is constrained to forward thrust, backward thrust, and yaw (rotation). Obstacle avoidance is programmed into the robot's motion. In essence, the robot does not need to learn the presence of obstacles but rather mimicking an onboard sensor deployment (in a real-life scenario) the robot is capable of aborting an action that would lead into a brick wall or obstacle within the grid world.

### 5.2.6 Natural selection:

Two chromosomes are selected from the mating pool of 100 chromosomes to produce two new offspring. Before the mating takes place, the best four chromosomes from the population are preserved. This process is known as *elitism* [105], [106]. This procedure helps with better convergence. The pairing continues until:

$$\text{New population} = \text{initial population} - \text{number of elites.}$$

This brings the total number of new offspring to 100 which is the number of the initial population. Unlike the natural biological model, GA maintains a fixed number of chromosomes after each epoch (generation).

One epoch iterates through the entire population of chromosomes while carrying out Selection, crossover, the mutation for each pair of chromosome based on some probability constraints.

The Roulette wheel and tournament selection are standards for most genetic algorithm applications. For this research, we compare both selection methods and analyze their convergence against varying crossover and mutations rates.

### **5.2.7 Crossover (Mating):**

Using roulette wheel as the selection method, the GA randomly selects two chromosomes for mating. The chromosomes with higher fitness scores stand the better chance of being selected for mating than those with lower scores.

With the tournament selection method, the GA randomly picks a small subset of chromosomes (3-5) from the mating pool. The chromosome with the highest fitness from this pool is then chosen as a parent.

Mating is the creation of one or more offspring from the parents selected in the pairing process. Two parents mate to produce offspring, and the resulting offspring is then placed into the population.

Because each action (bit) on the string of chromosomes could make a significant difference in fitness, a multi-point crossover on the chromosome was implemented. That is, for each chromosome where the crossover is to be performed, we determine a *swap rate* and then swap over individual bits where appropriate as we iterate through each chromosome.

$$\text{Swap rate} = [\text{random number (x) such that } (0 < x < 1)] \times [\text{Chromosome length (before decoding)}]$$

Thus crossover would happen at any point within the decoded chromosome where the swap rate is greater than a randomly generated variable between 0 and 1 (Crossover criteria is satisfied).

Example:

Parent 1: [RR][4], [RR][10], [RL][6], [T][3], [T][10], [T][4], [T][8], [B][5], [B][2]

Parent 2: [RL][6], [**RL**][5], [RL][15], [**B**][7], [B][9], [T][2], [T][12], [**P**][16], [P][4]

Crossover criteria satisfied at the gene positions 2, 4, and 8 (in bold)

Child 1: [RR][4], [**RL**][5], [RL][6], [**B**][7], [T][10], [T][4], [T][8], [**P**][16], [B][2]

Child 2: [RL][6], [**RR**][10], [RL][15], [**T**][3], [B][9], [T][2], [T][12], [**B**][5], [P][4]

### 5.2.8 Mutation:

The mutation is central to the reason why GAs would almost always find an optimal solution. It introduces traits, not in the original population and keeps the GA from converging too fast before sampling the entire objective space. A single point mutation toggles a bit. Just like crossover, mutation points are randomly selected from a string of chromosomes.

The mutation is the last phase of the GA process before the emergence of a new generation. This slight perturbation has the potential of enhancing or degrading the fitness of the individual chromosome but on another hand, enhances the overall robustness of the genetic algorithm.

In our implementation, a multipoint mutation enabled the GA makes significant progress towards global optima. Consider the example below:

With a mutation rate of 0.001 for action and duration

Child 1: [RR][4], [**RL**][5], [RL][6], [**B**][7], [T][10], [T][4], [T][8], [**P**][16], [B][2]

Child 2: [RL][6], [**RR**][10], [RL][15], [**T**][3], [B][9], [T][2], [T][12], [**B**][5], [P][4]

The mutated bit in italics:

Child 1: [RR][4], [**RL**][5], [RL][6], [**B**][7], [*B*][5], [T][4], [T][8], [*T*][16], [B][2]

Child 2: [*B*][6], [**RR**][10], [RL][15], [**T**][3], [B][9], [T][2], [T][12], [**B**][5], [*T*][14]

### 5.2.9 The Next Generation:

After the mutation, the next generation is tested on the fitness function which in our case is an experimental grid world with obstacles. The chromosome is decoded into a string of actions only. The duration determines the number of times the action is carried out. By feeding these strings of actions to the actuators of our robot, motion is exhibited and the amount of grid space covered is calculated as fitness for each instance of our robot. This entire process is iterated from generation to generation until the average fitness of the population is equal or near equal to the fitness of the overall best-fit chromosome.

At this point, we can terminate the iterative process and adopt the current status (our best fit chromosome) as our optimal solution. Moreover, subsequent generations would witness little or no improvement in their average fitness score.

The rationale behind choosing the best fit is to maximize the transversal of the search space under discrete variable implementation [107].

### 5.3 ENNs (Evolutional Neural Networks) using Continuous Variables

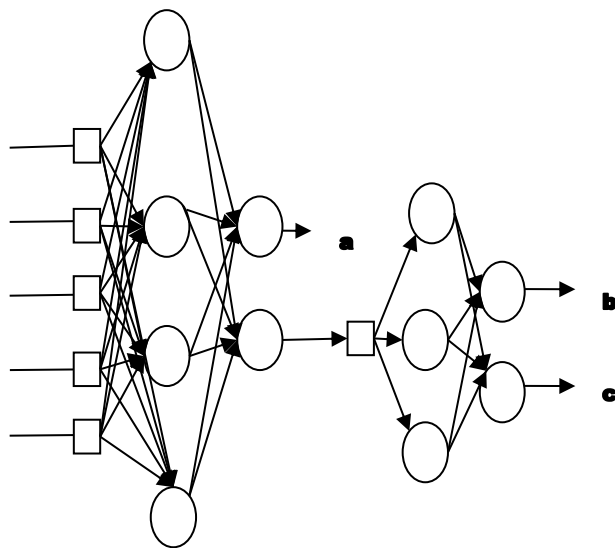


Figure 5.2 Double brain feed forward neural network. a= thrust forward b= left turn, c=right turn

The rationale behind this implementation was to improve the robot's trajectory capability by training the robot to learn the preferred turn (left or right) and also investigate if this would result in a significant improvement in the robot's performance. This time, if output 'a' is the highest, the robot thrust forward one step else the second neural network determines the direction of the turn. If output 'b' is highest, the robot turns left else it turns right by 0.1 radians. The same procedures for crossover and mutation are carried out on both networks simultaneously by keeping the selection, crossover, mutation, and turning rate constant for both networks [108]. When compared with the traditional neural network strategy, the brain expansion technique performed better with tournament selection but was outperformed by the traditional method when tested using the roulette wheel as a selection technique.

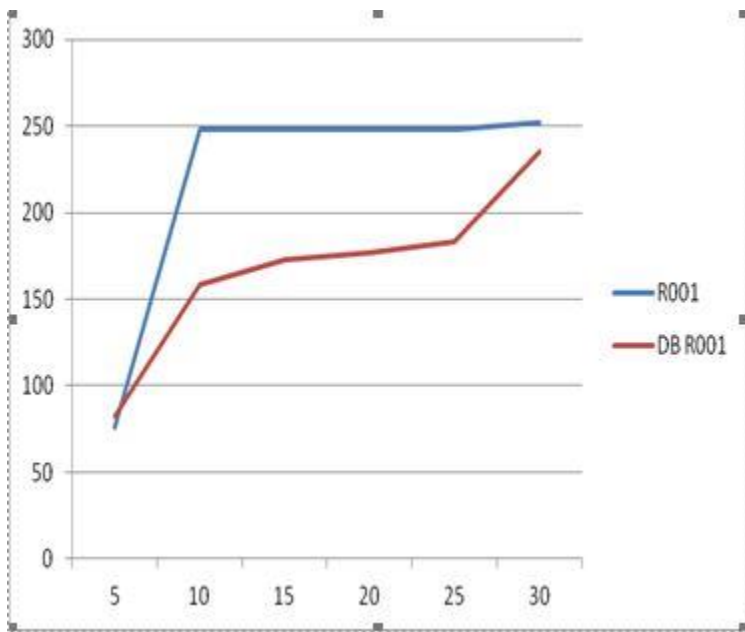


Fig. 5.3 comparing fitness performances for the traditional ENNs (blue R001) with Double Brain ENNs (red DB R001) using roulette wheel selection with mutation rate of 0.001

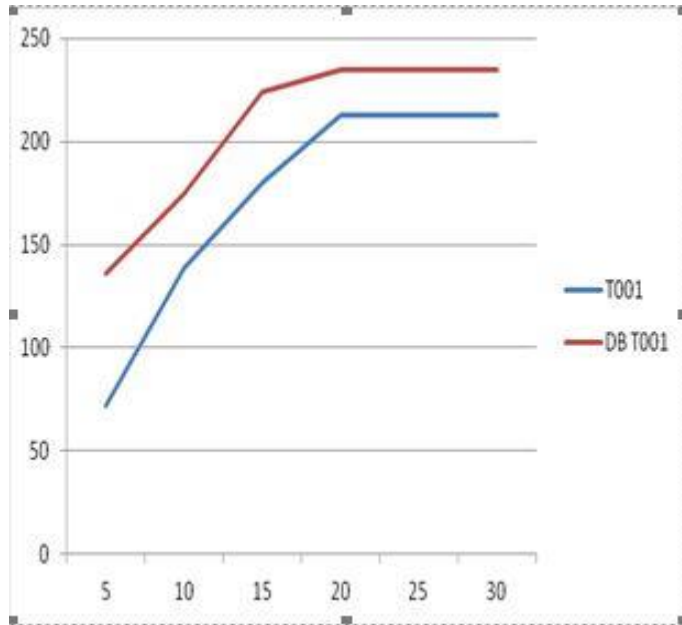


Fig. 5.4 comparing fitness performances for the traditional ENNs (blue T001) with Double Brain ENNs (red DB T001) using Tournament selection with mutation rate of 0.001

### 5.3.1 Polygamy: a Mating Strategy

A couple of strategies for applying crossover and mutation have been explored in most GA literature. For example, De Jong's [98] experiment prevented too many similar individuals from being in the population at the same time. This feat was achieved by enforcing newly formed offspring replacing the existing individual most identical to it. Goldberg and Richardson [99] achieved something similar using a proportional "fitness sharing function". Using this feature, similar individuals were punished while dissimilar ones were rewarded, thus allowing convergence on diverse peaks rather than all converging on the same peak.

Eshelman and Schaffer promoted diversity, putting a restriction on mating by disallowing mating between similar individuals (incest). The rationale was to keep the population as diverse as possible. Holland suggested the use of 'matching tags' as a mating selection strategy. These tags provide restrictions on mating as only those with matching tags are allowed to mate across generations.

The concept of polygamy, on the contrary, does not intend to promote diversity but rather similarity. With polygamy, every other member of the population is forced to mate with the fittest



of that generation. This implementation was effective in finding the global optima (best solution) when hybridized with the traditional selection method. The GA is allowed to run until it slows down and then polygamy sets in thereby fine-tuning the fittest chromosomes towards an optimal or best solution.

In our implementation, the GA slowed down after 30 generations before triggering polygamy. After a total of 50 generations, results show significant improvement for both roulette wheel selection and tournament selection with a mutation rate of 0.01. However, reducing the mutation rate to 0.001 shows little or no improvement in the overall fitness of the population. All experiments were conducted using a fixed standard 0.7 probability to determine the crossover rate.

Table 5.1 Effect of polygamy on ENN using Roulette wheel (R001 and R01) Tournament selection (T001 and T01). Were 001 represents a mutation rate of 0.001 and 01 a mutation rate of 0.01.

Epoch	R001	R01	T001	T01
5	76	74	72	159
10	248	239	139	229
15	248	254	180	240
20	248	255	213	246
25	248	258	213	249
30	252	267	213	262
35	252	270	213	263
40	253	277	213	271
45	253	278	213	272
50	253	278	213	272

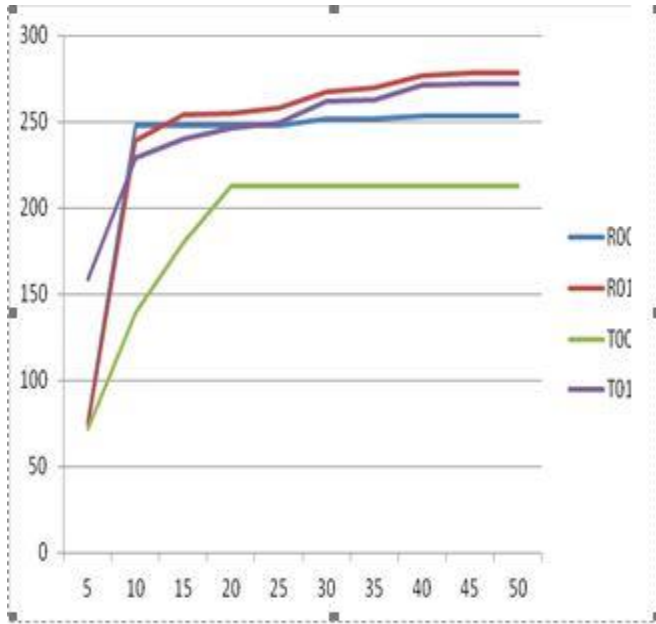


Fig. 5.5 comparing fitness performances for the polygamy enhanced ENNs using Roulette wheel (R001 and R01) Tournament selection (T001 and T01). Where 001 represents a mutation rate of 0.001 and 01 a mutation rate of 0.01.

### 5.3.2 Choosing Best Practice

The Boltzmann scaling method is a popular technique for optimizing GAs. This technique is used when the selection pressure is required to be low at the beginning of the run in order to maintain diversity but as the GA converges toward a solution, the fitter individuals are selected for mating. This method uses a continuously varying temperature to control the selection rate.

The best and most consistent solution was achieved when the concept of polygamy was applied to the double brain technique. When implemented with a 5 ratio 5 (5:5) generation switching the technique (looping the first 5 generations using traditional technique while subsequent 5 generations use polygamy), the resulting fitness score of 278 (90% of the grid environment searched) after 50 epochs provide satisfactory performance. After, multiple trials, the result remain consistent using tournament selection and a mutation rate of 0.01.

Table 5.2 Comparing performances of polygamy on double brain ENN, Boltzmann Scaling, polygamy on ENN with tournament selection method and a fixed mutation rate of 0.01

Epoch	neural poly	boltzmann	db poly 5:5
5	103	144	23
10	114	162	49
15	226	197	60
20	247	235	187
25	248	254	214
30	249	257	220
35	250	262	239
40	253	266	251
45	270	271	278
50	270	271	278

However, the implementation of polygamy on ENNs after (30 epochs – 50 epochs) and the Boltzmann’s scaling method (for 50 epochs) shows identical performance. This result may suggest the inclusion of polygamy to the ENNs as an alternative approach to the Boltzmann’s scaling method for ENNs.

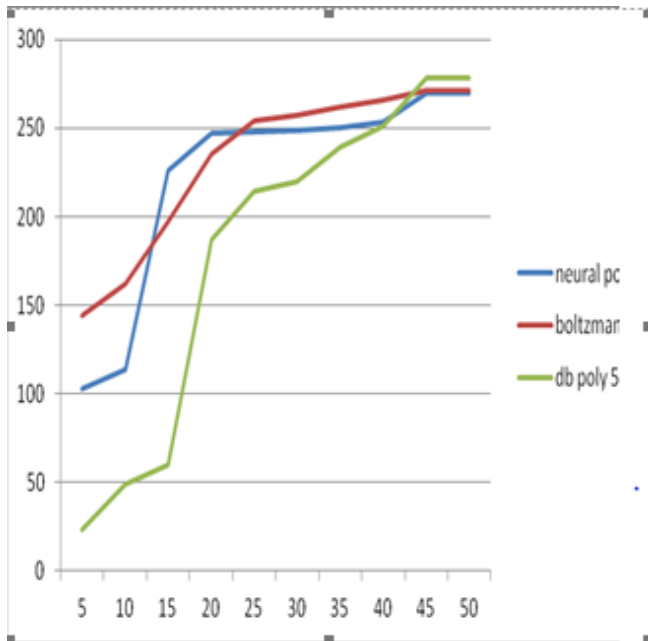


Fig. 5.6 Comparing performances of polygamy with double brain ENN (db poly 5:5), Boltzmann Scaling (Boltzmann), polygamy on ENN (neural poly) using tournament selection method and a fixed mutation rate of 0.01

Choosing the optimization technique that works best for a particular objective function sometimes could be more of an art than science as not all optimization algorithms may perfectly fit into an implementation. Table 5.3 shows results (which is an average over 20 iterations) of the performance of four other natural meta-heuristic algorithms (PSO, SA, Cuckoos search, and the hybrid cuckoo/GA) on our objective function. Among these algorithms, the PSO performed best with a fitness score of 255. Not surprisingly, the hybridized cuckoos/GA was an improvement over the cuckoo’s search algorithm with an addition of 28 grid cells. However, the DB-poly (5:5) methodology outperforms all algorithms with a fitness score of 278 which turns out to be about 90% of the entire searchable (obstacle-free) section of the grid world.

Table 5.3 Comparing performances of polygamy with double brain ENN (db poly 5:5), Boltzmann Scaling (Boltzmann), with four other meta-heuristic optimizers.

Epoch	PSO	SA	Cuckoos	hybrid Cuckoos/ GA	boltzmann	db poly 5:5
5	62	43	102	93	144	23
10	100	123	120	104	162	49
15	133	133	159	114	197	60
20	168	195	177	124	235	187
25	175	226	196	214	254	214
30	225	226	210	214	257	220
35	255	232	210	214	262	239
40	255	245	217	241	266	251
45	255	245	217	241	271	278
50	255	245	217	241	271	278

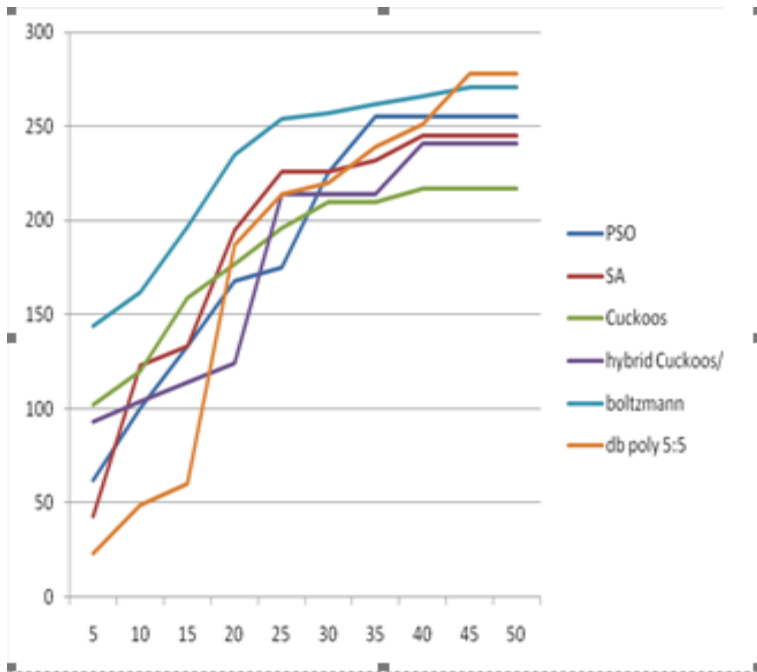


Fig. 5.7 Graphical representations of fitness scores over 50 generations for polygamy with double brain ENN (db poly 5:5), Boltzmann Scaling (Boltzmann), Simulated Annealing (SA), particle swarm optimization (PSO), cuckoos search, and the hybrid cuckoos/GA.

### 5.3.3 Impact of hydrodynamics on the Robot's search performance

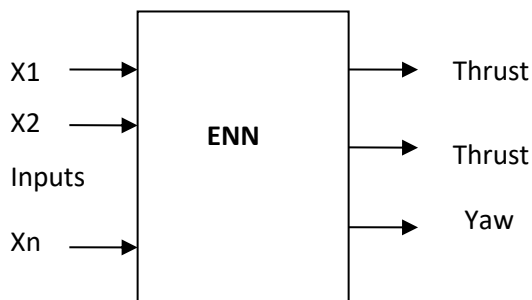


Fig. 5.8 Revised neural network architecture for maneuvering in a hydrodynamic environment.

In this research, we also investigated the impact of hydrodynamic drag into the simulation environment [109]. A constant drag coefficient is applied to the environment which could act in opposition to the linear velocity of the robot depending on the orientation of the robot. Using this method, the robot only experiences drag when traveling in the opposite direction of the current flow. Consequently, the robot experiences increased velocity when traveling in the same direction as the current flow. From figure 5.8, the propulsion of the robot is a function of the output of the evolutionary neural network (ENN). The network is further modified for hydrodynamic environments such that the robot is capable of switching between two linear velocities: thrust 1 and thrust 2, along with a yaw action used for altering the robot's orientation. In our simulation (leveraging C++ for windows), thrust 2 propels the robot with a velocity 3 times the velocity of thrust 1. A more accurate fluid dynamic simulator may provide better accuracy with respect to a real-world scenario, however not without significant computation complexity.

Also, in this initial phase of our experiment, we are more interested in the general behavior of the evolved robots with respect to their fitness scores. An exhaustive search in an aquatic like environment could be challenging due to the environmental dynamics which could make it difficult (or easier as the case may be) for the robot to conduct an exhaustive search on the environment. Since intelligence can also be defined as the ability to recognize the advantage, the robot must learn to take advantage of the conditions of the environment in accomplishing its goal (exhaustive search).

### 5.3.4 Experiment and results

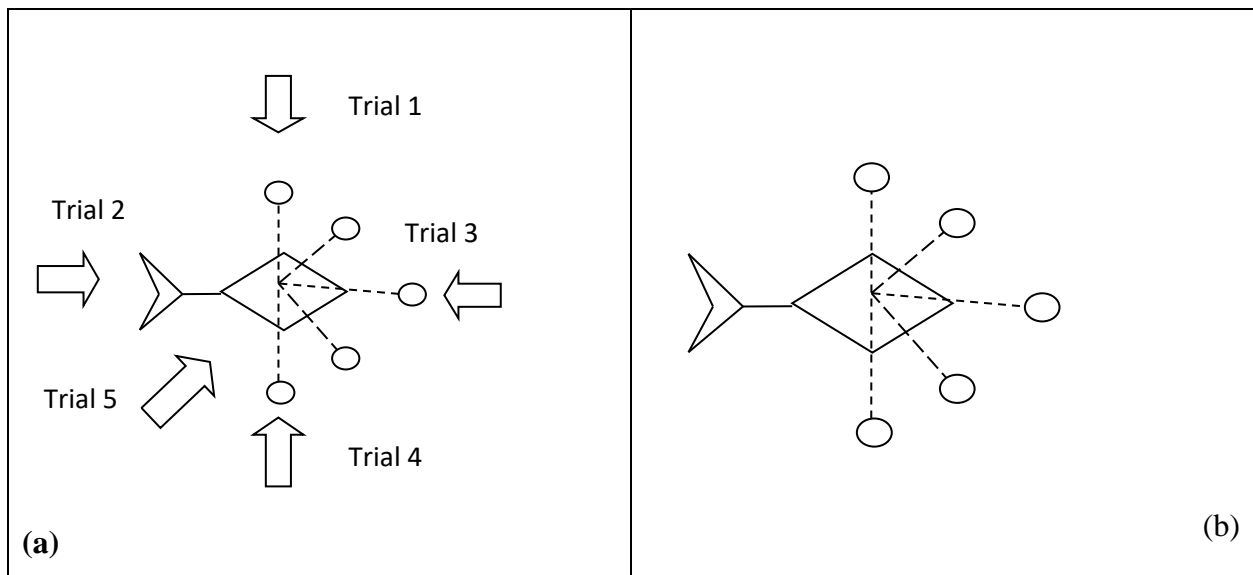


Fig. 5.9 (a)Flows emanating from different directions: North->South, West -> East, East-> West, South->North, and South-west ->North-East at a 45<sup>0</sup> angle. (b) Robot equipped with 5 sensors separated at 45 degrees interval

Five separate trials, as shown in Figure 5.9, were conducted to evolve robots search performance for environments with flows emanating from different directions. Trial 1 simulates a flow from North->South, Trial 2 from West -> East, Trail 3 from East-> West, Trial 4 from South->North, and finally Trial 5 from South-west ->North-East at a 45<sup>0</sup> angle.

The simulation was conducted with 3 categories of tidal flow: Low, Medium, and High. In the Low tide category, the robot propels forward with half of thrust 1 velocity when in opposition to the flow. In the Medium tide category, thrust 1 propels the robot forward with the same velocity as that of the tidal flow, thus the robot should remain stationary when in opposition to the flow. In the High tide category, the robot propels forward with half the velocity of thrust 2 when in opposition to the flow. The trials consist of an averaged summary of 20 replicate runs evolved over 50 generations with a population of 100 robots.

Table 5.4 Summary Results for Trial 1 to 4

Epoch	low tide	medium tide	high tide	Random tide
5	231	122	102	142
10	279	154	165	282
15	286	205	193	299
20	287	213	203	290
25	299	242	208	291
30	302	247	211	306
35	302	269	219	308
40	302	284	225	301
45	302	299	232	300
50	302	300	232	296
Avg. Fitness	210.81	197.6	121	215.85

Table 5.4 shows the fitness scores of the best Evolving robot for direct low, medium, high, and random tide, over 50 generations in a hydrodynamic environment for trails 1-4. Trials 1->4

exhibits similar results for low, medium, high, and random tides as shown in the table above. In the random tide mode, the flow rate switches between Low, Medium, High, and No tide in each time frame while keeping the direction of the flow constant.

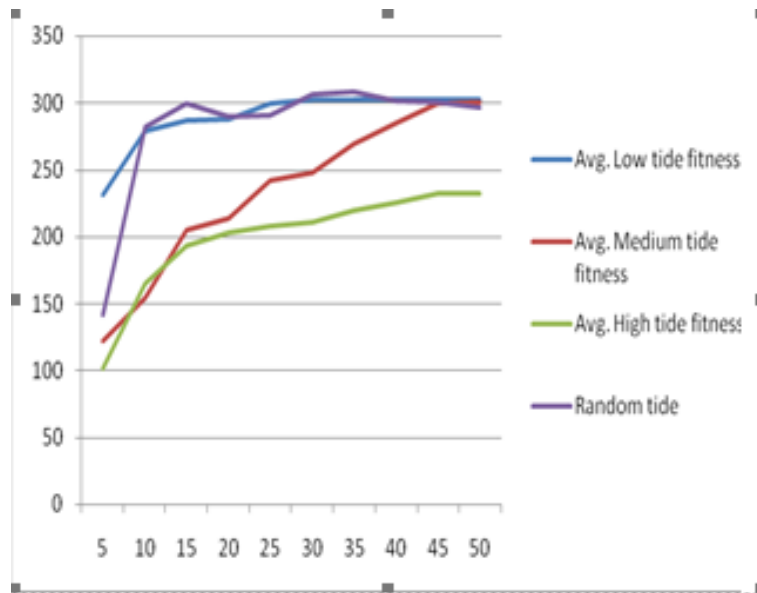


Fig. 5.10 Graphical representation of fitness scores of the best evolving robot for low, medium, high, and random tide, over 50 generations in a hydrodynamic environment for trails 1 - 4

Not surprisingly, the robot was able to perform faster and better in accomplishing its goal by taking advantage of the environment [109-111]. It was observed that the robot performed better in environments with lower tidal flows than the higher ones. Similarly, the robot was able to adjust to the switching flow rates. The best fittest score fluctuated throughout the run but stabilized at the range (295, 308) fitness score, which is a very satisfactory performance covering 95% of the entire search space. The fluctuations were as a result of the stochastic nature of the environment.



Table 5.5 Results for Trial 5

Epoch	low tide	medium tide	high tide	Random tide
5	75	214	122	50
10	129	245	220	107
15	129	245	220	109
20	134	249	220	117
25	148	252	221	173
30	164	252	221	173
35	212	254	221	163
40	232	255	226	171
45	232	255	226	197
50	232	255	226	204
Avg. Fitness	67.9	167.53	98.99	63.17

The summary table above shows the fitness scores of the best Evolving robot for diagonal low, medium, high, and random tide, over 50 generations in a hydrodynamic environment for trial 5. In trial 5, a subtle but significant difference from trials 1- 4 was observed. Apparently, the diagonal tidal flow poses a greater restraint on the fittest robot’s trajectory. The robot performed better (with respect to the objective function) in a medium tide environment than in the high and low tide environment. This could be as a result of the increased constraint of a diagonal collision avoidance mechanism preprogrammed into the robot’s motion model, consequently making the robot to be less adventurous at the lower diagonal tides conditions and also too careful at higher diagonal tide conditions thereby making the medium diagonal tides a more favorable condition while attempting an exhaustive search of the environment.

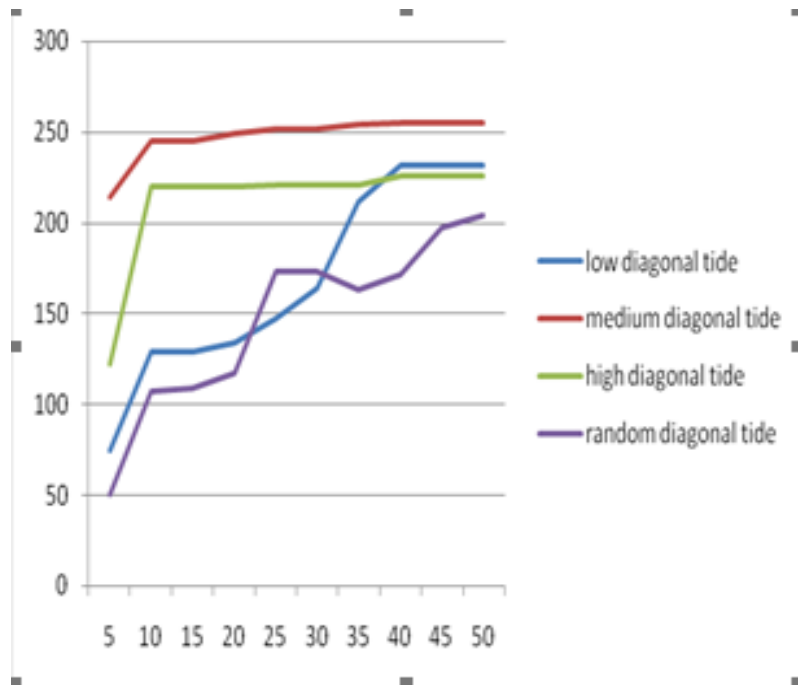


Fig. 5.11 Graphical representation of fitness scores of the best evolving robot for diagonal low, medium, high, and random tide, over 50 generations in a hydrodynamic environment for trail 5

From the above graph (fig. 5.11), it is obvious that randomizing the diagonal tide makes the environment more arduous for the robot to meet up with its objective function. At the end of 50 generations, the fittest robot is barely able to cover about 65% of the entire search space.

### 5.3.5 Implementing Particle filters: based on Time difference of arrival (TDoA)

In a deliberate attempt to demonstrate the practicability of our algorithm in a real-world scenario, we introduce the particle filter algorithm to our dynamic programmed (clean-up robot) for localizing itself within the search environment [112-115]. In a non-deterministic (hydrodynamic) but fully observable environment (grid map), it becomes imperative for a robot to localize itself while simultaneously planning its path to the desired goal node using dynamic programming (DP). The presence of ambient noise and reverberations using acoustic sensors in underwater localization applications pose a hindrance to a single point intersection of acoustic signals [116]. As a result, traditional methods used for acoustic localization rarely succeed even at low reverberation levels. Prior experiments comparing Kalman filters and particle filters in an underwater environment

showed better performance leveraging particle filters with smoother and more robust trajectory estimation [39]. In addition, X.U. Yaosong *et al.*, [116] demonstrated the accuracy and practicability of particle filter implementation in underwater applications with a low RMSE bias of 0.0732.

The particle filter technique for localization involves a representation of a random collection of particles, each of which represents a hypothesis of where the robot might be. Each particle is an abstract replica of the actual robot with 3 or more vector values  $(x_i, y_i, \theta_i)$  representing their positions (within Cartesian coordinates) and orientations [117-120] within the belief space (search environment). Thus, each particle is a possible state of the actual robot.

Over an iterative cycle of motion and measurements by both the actual robot and particles, the particles with measurements that are more consistent with the actual robot measurements are given higher weights (importance) thereby increasing their re-sampling likelihood. These particles survive in proportion to their measurement probability. The point estimate of the actual robot's location and pose is then computed based on the new weighted re-sampled particles using their mean or median values. Particle filter which turns out to be very effective in handling non-Gaussian and non-linear problems were formulated on the concepts of Bayesian theory.

We simulated a search and rescue operation using 4 acoustic sensors, modeled at the corners of the grid world. During the search and rescue procedure, 3 robots were randomly deployed within the search environment along with a certain (pre-determined) number of simulated victims whose locations are unknown.

It's important to note that two of the robots have been trained and optimized on navigating the grid world while the third robot navigated via dynamic programming (path planning) which we hereafter refer to as the DP-robot. In our algorithm, each robot possesses a prior knowledge of the map along with the map's state and subsequently the updated state of the map as the grid cells get visited. As a result, oblivious collaboration emerges as they cooperate in the search and rescue operation while conducting an exhaustive search of the environment. Each robot is assumed to be capable of picking up a located victim within the search space while navigating the environment. The same technique used in foraging robots. It's important to note the algorithm possesses no heuristics with respect to the location of the victims. The success of the algorithm is strictly based on the ability to exhaustively search the entire grid world. This definitely exposes a major

weakness in the algorithm but however opens a gateway for better algorithms as revealed in subsequent sections.

#### 5.4 Multi-Objective Optimization Approach to Multi-Source Localization

Collision avoidance is a very important feature design for mobile robots. With the introduction of ENNs, the robot was simulated to have on board sensors with which it could maintain tags on the intensity of each gradient source. This feature makes the robot capable of perceiving his environment via its input sensors.

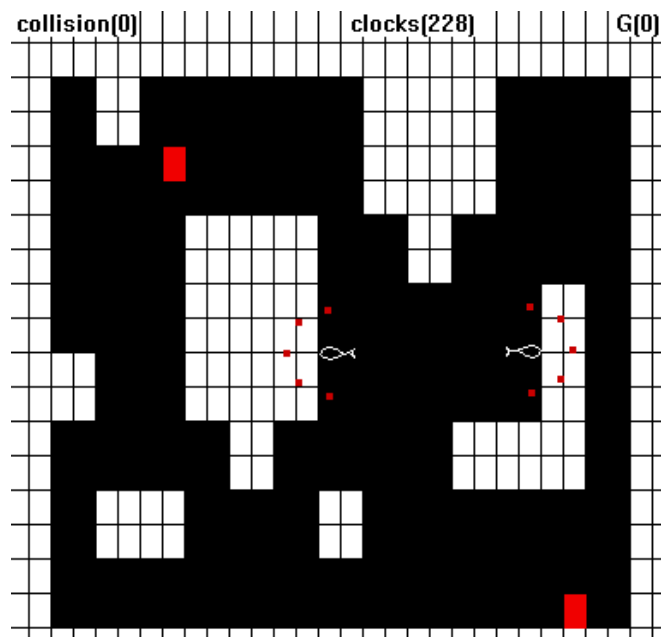


Fig. 5.12 Scenario 1 the robotic agent equipped with 5 sensor inputs into its feed-forward neural network (brain) with 2 gradient sources in RED

The artificial neural network was modeled using 5 sensors inputs, 2 output neurons with the sigmoid activation function along with 20 neurons for the hidden layer. The trajectory of the robot is determined by the neural network outputs (fig. 5.12). Both outputs were used for the controls of the robot. While one output was used for the thrust motion, the other was used for the yaw with a turning radius of 0.1 radians. The neural network weights were stochastically determined within the bounds of  $[-1, 1]$ .

Each Epoch was determined using CPU clock cycles precisely 7500 frames after which the NSGA-II algorithm the best performing robots are preserved via its selection, mutation and crossover model as shown in (fig. 5.13) below.

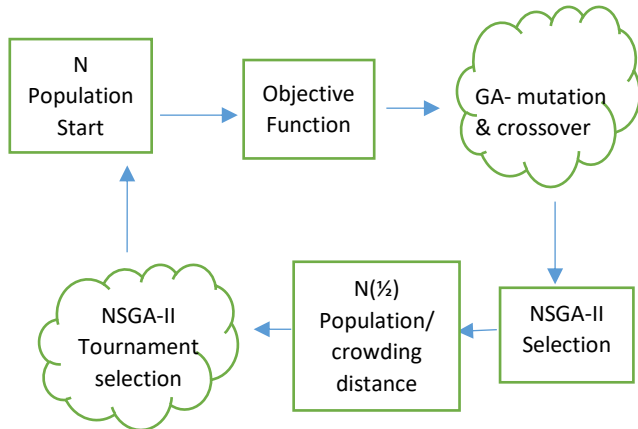


Fig. 5.13 Optimization architecture pipeline

The above figure shows the processes for each clock cycle of about 7500 frames. A single artificial neural network model is used for the training of a hundred pairs of robotic agents. Weights are filtered and returned into the neural network at the beginning of each epoch.

The process is terminated based on any of the following criteria: either there is convergence such that subsequent epochs do not produce better results or the stipulated number of epochs for the training process has been attained.

#### 5.4.1 Sensory feedback

The values returned by each sensor is bounded between  $[-1, 1]$  depending on the location of the end nodes of each input sensor (fig 5.9b). While the return value for an obstacle is -1, a float variable value is returned for other cell nodes. These values varies based on the distance from the gradient source nodes.

The multi-objective optimization model was adopted as a result of the dual conflicting properties of the robots characterized by exploration and homing. A robot biased towards exploration may consequently pay less attention to homing on the target source while on the other hand, a robot biased towards homing may ignore other available emission sources thereby failing to progress

after a gradient sources has been localized. As a result, the NSGA-II model was adopted to find the Pareto-optimal front that best satisfies both objectives by finding the most appropriate balance between exploration and homing.

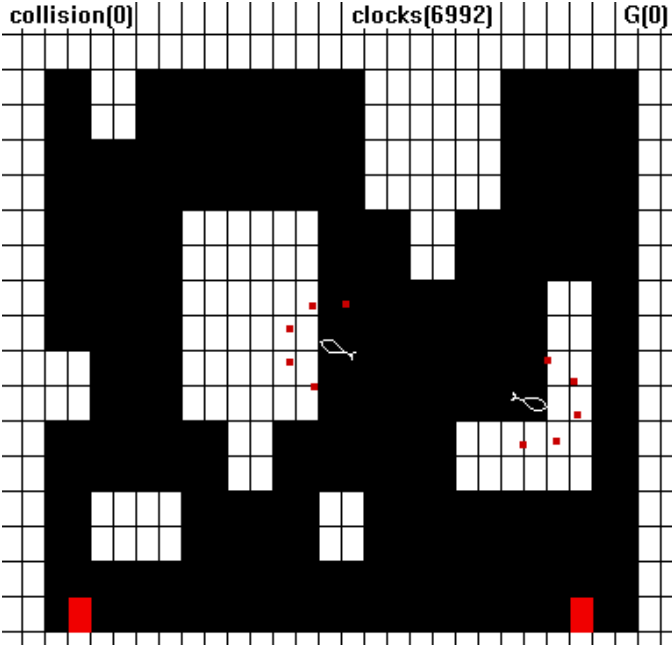


Fig. 5.14 Scenario 2 the robotic agent equipped with 5 sensor inputs into its feed-forward neural network (brain) with 2 gradient sources in RED.

**5.4.2 Dynamic programming (DP) approach for modeling multiple gradient sources**

We found the Dynamic programming (DP) model most appropriate for simulating multiple gradient sources. Although DP is often implemented for path planning by providing policies for each valid node, adapting this same strategy to gradient sources mimics real world scenarios adequately. From the fig. 5.15 below, we see a square grid model. The zero points are used to indicate regions of maximum gradient concentration. Consequently, as we move away for the zero points, we begin to migrate towards regions of lower concentration which is represented by higher values. It is important to note that this model by default assumes no regions with dead space exists.

3	2	1	2	/////	5
4	////	0	1	////////	4
5	////	1	2	////////	3
4	3	2	2	1	2
5	////////	////////	1	0	1
5	4	3	2	////////	2

Fig. 5.15 modeling multiple emission sources on a grid world.

### 5.4.3 Calculating Fitness values (F1 and F2)

The following equations were used for calculating the homing (F1) and explorative fitness score (F2) respectively. It's important to note that in robotic applications, it is intuitive to view the grid world and the task as the objective function. However, the decision space constraints for the weighted values are within the [-1, +1] range.

$$f1 = \sum_{i=1}^N \sum_{j=1}^n \frac{1}{C_j + 1} \quad (K) \tag{5.1}$$

$$f2 = \sum_{i=1}^N \sum_{j=1}^n (V_j) \tag{5.2}$$

where,

$C_j$  is the estimated Euclidian distance from source location derived from the sum of all sensor readings.

$K$  is a large constant (typical value 1000)

$n$  is the number of sensor readings

$N$  is the duration of clock cycles (typical value 7500)

$V_j$  is the sum of all sensor values in visited cells

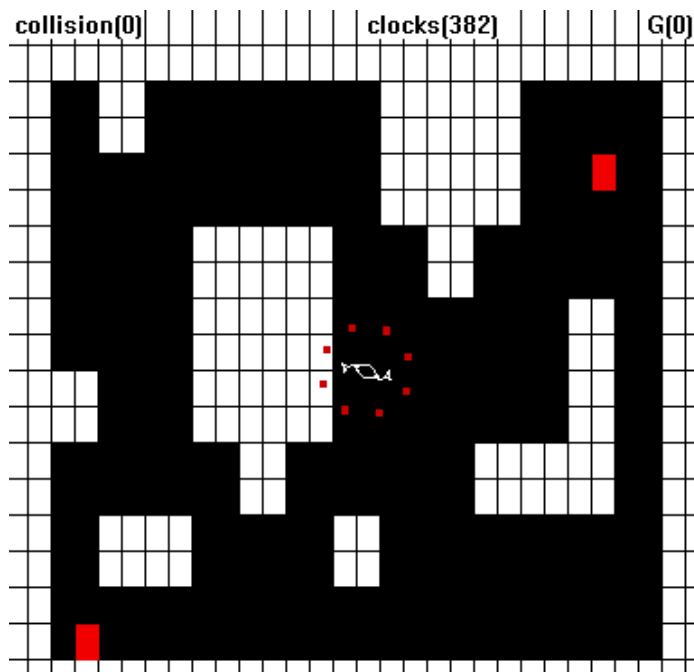


Fig. 5.16 Scenario 3 the robotic agent equipped with 5 sensor inputs into its feed-forward neural network (brain) with 2 gradient sources in RED.



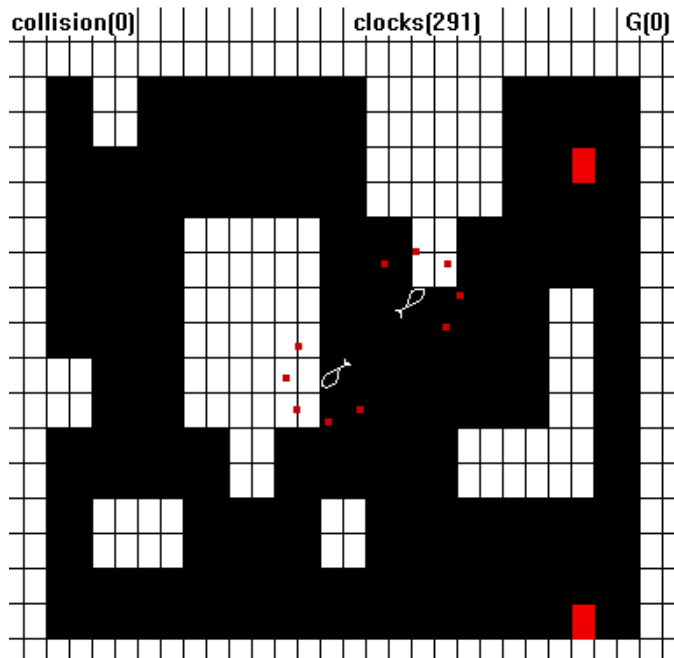


Fig. 5.17 Scenario 4 the robotic agent equipped with 5 sensor inputs into its feed-forward neural network (brain) with 2 gradient sources in RED.

### 5.5 Multi-Objective Results

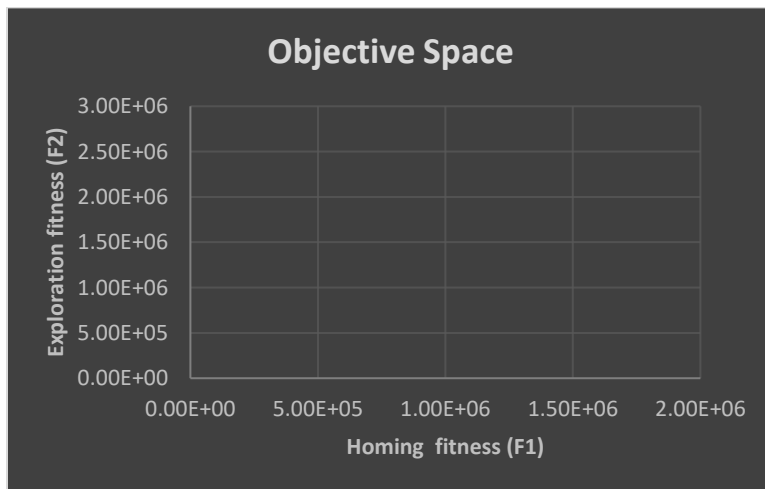


Fig. 5.18 the Pareto-optimal front for Scenario 1

The above scenario figure (5.18) shows the localization of both emission sources by both agents. It was possible to identify best solution via continuous monitoring of the homing score. Both emission sources were localized by the end of 5000 clock cycles which apparently was also the

end of the 3rd generation. The algorithm was terminated by the 5th generation having shown no significant improvement with the missions best completion time at 1250 clock cycles. The agents were observed to have localized both emission sources by splitting apart.

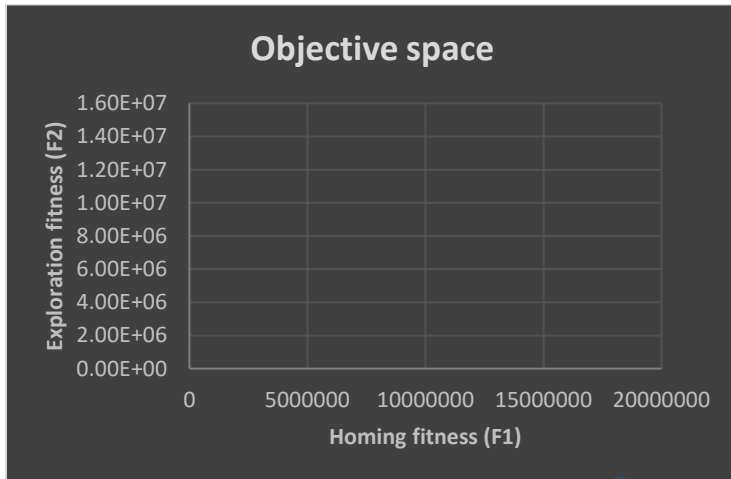


Fig 5.19 the Pareto-optimal front for Scenario 2

The second scenario above (fig. 5.19) also shows the localization of both emission sources by both agents. Both emission sources were localized by the end of 3300 clock cycles which apparently was also the end of the 3rd generation. The algorithm was terminated by the 5th generation having shown no significant improvement with the missions best completion time at 2180 clock cycles. Once again, the agents were observed to have localized both emission sources by splitting apart.

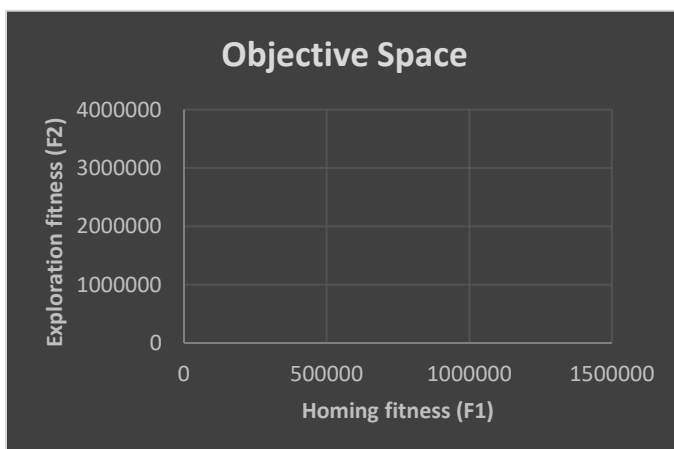


Fig. 5.20 the Pareto-optimal front for Scenario 3

The third scenario above (fig.5.20) also shows the localization of both emission sources by both agents. Both emission sources were localized by the end of 2350 clock cycles which apparently was also the end of the 3rd generation. Again, the agents were observed to have localized both emission sources by splitting apart.

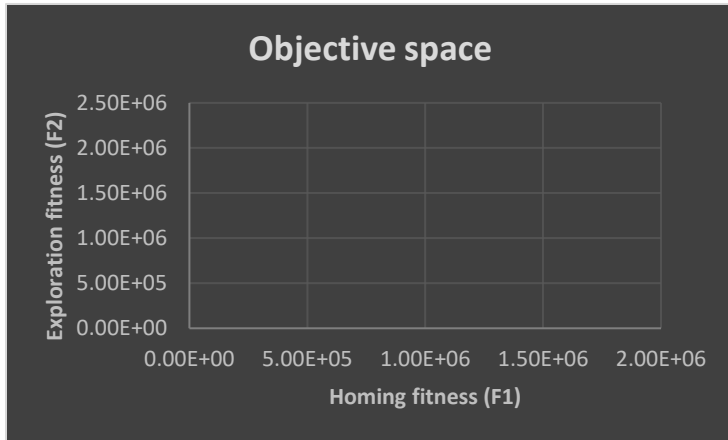


Fig.5.21 the Pareto-optimal front for Scenario 4

In our fourth simulation (scenario 4, fig.5.21), shows the localization of both emission sources by a single agent. This characteristic behavior presents a potential contribution of this algorithm towards solving the problem of progress after localizing an emission source. Both emission sources were localized by the end of 4550 clock cycles which apparently was also the end of the 3rd generation. The algorithm was terminated by the 5th generation having shown no significant improvement with the missions best completion time remaining unchanged.

## 5.6 A Comparative Analysis with two Existing Models

Table 5.6 Adapted Summary of multi-search options and their characteristics.

Source: Kathleen McGill, Stephen Taylor 2011 [13]

	<b>GSO</b>	<b>PSO</b>	<b>Proposed Model</b>
<b>Swarm size</b>	1000	10	2
<b>Source Number</b>	100	5	2
<b>Source type</b>	Generic	Chemical	Generic
<b>Source mobility</b>	Mobile	Fixed	Fixed
<b>Variable source intensity</b>	Yes	No	Yes
<b>Dead space</b>	No	Yes	No
<b>Communication range</b>	Local	Local	Global
<b>Agent deployment</b>	Random, center, corner	Corner	Center
<b>Computational complexity</b>	Low	Medium	Medium
<b>Obstacle avoidance</b>	Sensory based	Artificial repulsion	Artificial repulsion
<b>Sensing requirement</b>	Signal intensity	Other Robots, obstacles	Signal intensity
<b>Proceeding after source is found</b>	None	Source collection	Source collection
<b>Theoretical foundations</b>	Clustering behavior	None	Multiple source profiles

The above table (table 5.6) shows the implementation models for the PSO and the GSO algorithms along with their implementation approach towards solving the multisource localization problem. While the 'swarm size' is indicative of the number of robots used for each trial, the 'source number' is indicates the number of emission sources to b localized by the robotic agents. There are two main types of gradient sources: the Generic and the chemical. The PSO model was implemented

using chemical sources while our proposed model along with the GSO makes use of generic sources. However, it is important to note that our proposed model did not include dead space in the simulation. However, this could easily be added by allowing robots that are instantiated on a dead space to roam until a gradient source is sensed.

Since the PSO was modeled using plume, this made chemical sources for gradient source simulation most appropriate. These gradient sources as simulated such that robots could extrapolate via spatial proximity close they may be to a gradient source. The attribute of each source is defined by the source mobility. Sources could either be fixed or mobile. Our proposed model was implemented using fixed sources. However the model could be adapted to accommodate gradient sources that could drift. There are two communication methods that could be implemented: the local communication and the global communication. The GSO was implemented using the local communication method such that each robot made decisions based on the information shared between themselves. However, our proposed model was implemented using the global communication model. Consequently, each robot updates its inputs from the changing state of the grid world (triggered by each robot's motion) with which the artificial neural network learns the most beneficial trajectory.

Furthermore, there are different strategies for instantiating our robots. They could either be randomly distributed, or centralized, or clustered at the corner of the grid world. This could be subjective based on the model of the grid world. For our proposed model, all simulations were instantiated at the center. Computational complexity denotes the magnitude of hardware and software required with respect to CPU time overhead, Memory consumption and the complexity of the algorithm. Armed with these indicators, we classified complexity into the following: LOW, MEDIUM, and HIGH. Consequently, we classified as HIGH if any of the two mentioned indicators were high, and similarly, we classified as LOW if any of the two indicators were low.

Our proposed model along with the PSO model made use of artificial repulsion for obstacle avoidance implementation. In contrast, the GSO model implemented obstacle avoidance using sensory based model. One major contribution of our model to the multisource localization problem can be traced to the ability of our model to progress with a search after a source is localized and also progress the search process towards convergence and termination.

It is important to note that our model implemented source collection just as could be found with the PSO algorithm. However we observed via simulations, the potential for progress even in

scenarios with no degrading sources. Finally, partitioning indicates the ability to create sub-group of robots from a swarm as could be found in the GSO implementation model. We reserve this paradigm as part of our future research where neural networks could learn to form clusters of robots amidst localizing multiple gradient sources.

## **5.7 Conclusion**

In this chapter, we have progressed from discrete models to evolving Neural network models in achieving corporative control in task oriented missions using a group of robots and then climaxed on the dynamics of Multi-objective evolutionary algorithms combined with artificial neural networks in addressing a fundamental multisource localization problem of progress after a source is found. It also unveiled a guaranteed procedure for finding multiple emission sources. In addition, we witnessed via experiments, the improvements ENNs can bring to optimization of the robot's trajectory for the search process taking advantage of sensor feedback as inputs to the neurons rather than just evolving discrete robot actions. In our implementation, we assumed the emission sources were fixed and the grid a fully observable 2 Dimensional world.

## CHAPTER 6

### AGENT AND SOURCE LOCALIZATION IN POMDP ENVIRONMENTS

#### 6.1 Introduction

Recent advances in the field of artificial intelligence (AI) has made available a wide range of efficient algorithms which when skillfully hybridized could result in a plausible model for solving some of the problems in the field. In this chapter, the authors leverage on some of the existing AI techniques for optimization, such as genetic algorithms, for learning such as ANNs (artificial Neural Networks) and RL (Reinforcement learning), and finally for state estimation such as the Particle filter algorithm. Since Markov Decision Processes (MDPs) do not provide an accurate representation of the real-world domain, due to the uncertainty or rather incomplete knowledge of the state of the environment, we adopt the POMDPs- Partially Observable Markov Decision Processes model for our simulation.

Ever since the introduction of value iteration algorithm for planning [121] in the 1970s, the concept has undergone a couple of refinement by numerous authors with an attempt to adapt it to solving more complex real-world problems. The combinational explosion of linear components (also referred to as the curse of dimensionality in some literature) in the value function is one of the major reasons POMDPs are impractical for most applications [122], [123]. Another related problem with value iteration is the exponential growth of distinct action-observation histories (also referred to as the curse of history). Some ingenious pruning methods have been used to ameliorate the problem but these pruning methods are in themselves computationally expensive to implement and only works for small finite horizon problems [124].

Some better strategies have been implemented such as PBVI (Point-Based Value Iteration) which iteratively updates a subset of representative belief points. Another promising method implemented for both discrete and continuous belief states is the MCMDP (Monte Carlos Markov Decision Process). This method attempts to map POMDPs directly to their underlying MDPs using Bayes Particle filter for belief updates. On a parallel front, the Reinforcement Learning (RL) algorithm is considerably a slow but elegant approach to learning in an unknown environment. Although the action-value (Q-learning) is faster than the state-value, the rate of convergence to an optimal policy or maximum cumulative reward remains a constraint. However, RL has the advantage of learning an underlying MDP [125-127] for dynamic and stochastic environments.

In this section, the authors via experiments investigated the effect, impact or/and contributions of multi-agents to accelerating the rate (thereby shortening the duration) at which the utilities converge to an optimal policy for planning within POMDP environments. The agents' leverage on the greedy strategy for online exploration-exploitation using off-policy model-free algorithm [128]. We then compared this RL model with an ingenious multi-agent framework equipped with a feedforward neural network which is optimized offline via an objective function (based on localization of goal and absorbing nodes) using a genetic algorithm. We then identified the promises and constraints of both paradigms and thus propose future recommendations.

Furthermore, because every POMDP can be mapped directly to its underlying MDP, we examined the effect of multi-layered particle filter [129-132] (weighted resampling) mechanism for minimizing the error between an agent's belief state and actual state. Results show that this simple procedure quickly filters out outliers responsible for large errors in the initial approximate belief state.

## **6.2 Methodology**

At the end of chapter 5, we had developed a model that addressed the theoretical foundations for progress (after a source is found), however, we are left with a problem of convergence and termination of the search and localization operation. Motivated by these problems, we introduced novel concepts for accelerating an off-policy reinforcement learning algorithm for Partially Observable Markov Decision Processes (POMDP) leveraging the multi-agents frame work. Motivated by the need to optimize the learning process, we compared our off-policy reinforcement learning algorithm with an ingenious GA (Genetic Algorithm) approach for multi-agent offline learning using feedforward neural networks. At the end of the trainings (episodes and epochs) for reinforcement learning and genetic algorithm respectively, the convergence rate for both algorithms with respect to creating the underlying MDPs for POMDP problems were compared. Next, we demonstrated the impact of layered resampling of Monte Carlo's particle filter for improving the belief state estimation accuracy with respect to ground truth within POMDP domains. The adoption of this novel approach provided us with a robust algorithm for the convergence and termination for the search and localization of multiple gradient sources.



Similar to the model used in chapter 5, we adopted a multi-layer artificial neural network with 5 inputs, 2 output neurons, and 20 neurons in the hidden layer for each mobile agent. Choosing the structure of a neural network is rather more of an art than science. However, certain factors such as the speed of training the network, could influence the choice on the number of hidden layers (level of complexity). The higher the complexity, the slower the training process. Since we adapted a feedforward model for weight adjustment rather than the backpropagation methodology, we tend to worry less about overfitting the model due to the fact that the network is not presented with a predefined optimal label. The feedforward network improved by advancing its frontiers via an objective function. Consequently, a single hidden layer with about 15-20 neurons acceptable performance with respect to efficiency and minimal complexity for our model design. Each neuron is activated using the sigmoid activation function. The output of the network determines the robot's trajectory. If output 1 is the highest, the robot thrust forward one step else it turns (rotate left or right) by 0.1 radians. The weights of the neurons (chromosome) are stochastically determined at the instantiation of the first run between  $[-1, 1]$  range.

We compared two learning paradigms for a POMDP environment. In the first paradigm, we simulated the learning phase of a POMDP model using online, off-policy reinforcement Q-learning with both single and multi-agents. The rationale is for the agents to learn optimal policy within belief space. In the second paradigm, we simulated an alternative off-line learning approach using feedforward neural networks for multiple agents (with a size of 4) whose weights were optimized using genetic algorithm over multiple epochs. The rationale is for the agent to learn the model of the world by localizing all-absorbing states including the goal node and terminating with an optimal policy with respect to the goal node using dynamic programming. We then compared their merits and demerits with respect to practicability within the scope of the multisource localization problem.

Finally, we investigated via experiments, an ingenious modification to the resampling phase of the Monte Carlos Particle filter algorithm such that an agent navigates with an improved belief state estimation accuracy with respect to the agents' actual state (ground truth). Consequently, the agent would be capable of mapping the POMDP environment to its underlying MDP with high fidelity.

## 6.3 Experiments and Results

The experiments can be divided into two sub sections: section A and Section B.

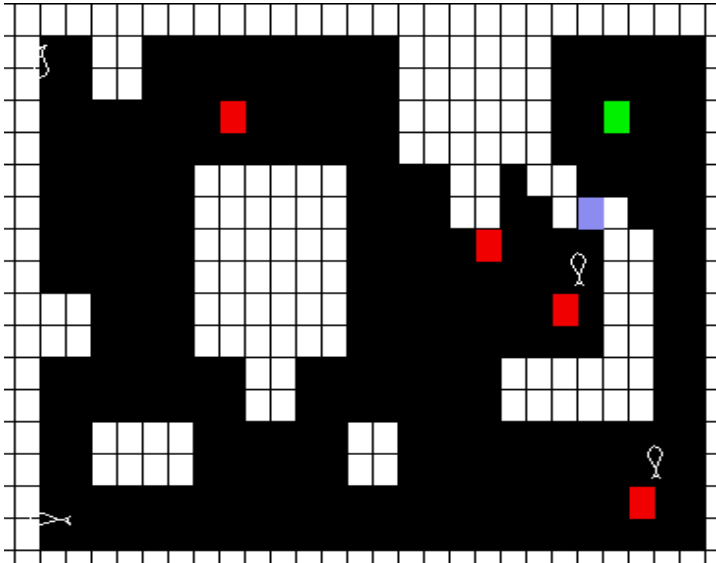


Fig. 6.1 Multi-agent RL environment. With walls (white cells), absorbing states (red cells), dynamic door (blue cell) and goal node (green cell).

The experiments were divided into two sections: Section A and Section B. In this section (section A), we showed how the multi-agent Q-learning RL algorithm [133-137] converges quickly when compared with a single off-policy agent. It is important to note that the learning algorithm created an underlying MDP model for the grid world (fig. 6.1) at convergence.

The first simulation had a single RL agent in a 30 X 20 grid world (fig. 6.2) with obstacles (white cells), absorbing nodes (red cells), a single door (blue cell) which toggles (open/close) between episodes and a single goal node (in green). In the second simulation (fig.6.3), three more agents are added to the single agent. In a deliberate attempt to investigate the significance of the addition of a single agent, we ran a third simulation with 5 agents (fig 6.4).

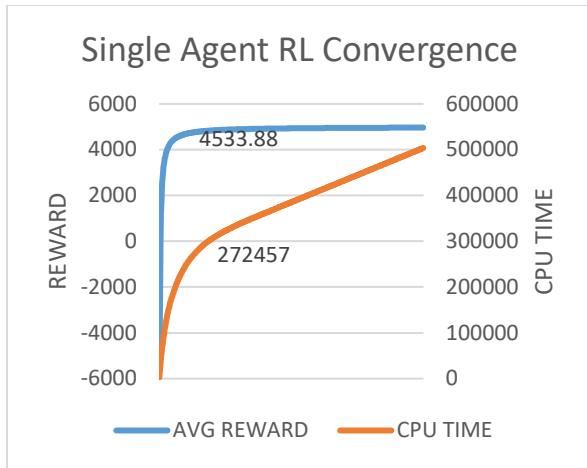


Fig. 6.2 Single agent reinforcement learning graph with respect to CPU-time

The results show a significant difference in the convergence rate. It is interesting to note that multi-agents displayed some emergent behaviors (outside the scope of this research) during the on-line training process while migrating the algorithm towards convergence.

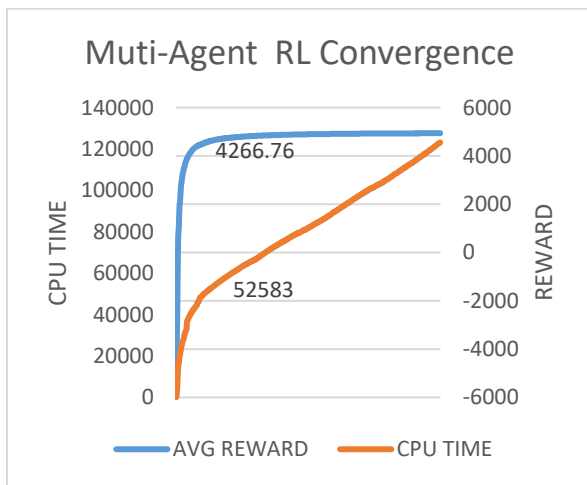


Fig. 6.3 Multi-agent (size of 4) reinforcement learning graph with respect to CPU-time

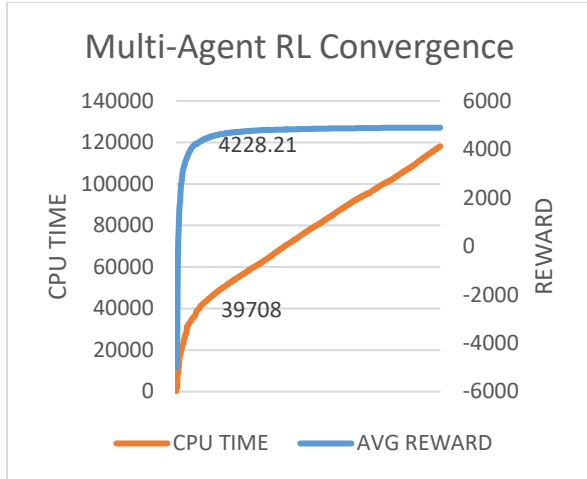


Fig. 6.4 Multi-agent (size of 5) reinforcement learning graph with respect to CPU-time

In comparison, we simulated an alternate approach to creating an underlying MDP model for a grid world using multi-agents (4 agents) each equipped with feedforward neural networks, which are optimized by genetic algorithm. The objective of function of these agents is to learn the model of the world via exploration. Training is done off-line via epochs over multiple generations. The fitness function for each generation of the multi-agents is given by:

$$\sum_{i=1}^m \sum_{j=1}^n R(s)_{i,j} + \beta_{i,j} \tag{6.1}$$

where,

$R(s)$  are the living positive reward for each new explored state (i,j) in the grid world,

$\beta_{i,j}$  are extra rewards assigned to absorbing and goal nodes.

The iteration terminates after a predefined number of epochs or after a predefined minimum sum of rewards is obtained. The simulation terminates by creating an underlying MDP (Optimal policy) using dynamic programming with respect to the goal node. It's important to note that the entire learning procedure is considered to be off-line. Each epoch ran for a fixed duration (3750) CPU-time over 12 epochs (fig. 6.5) before termination.

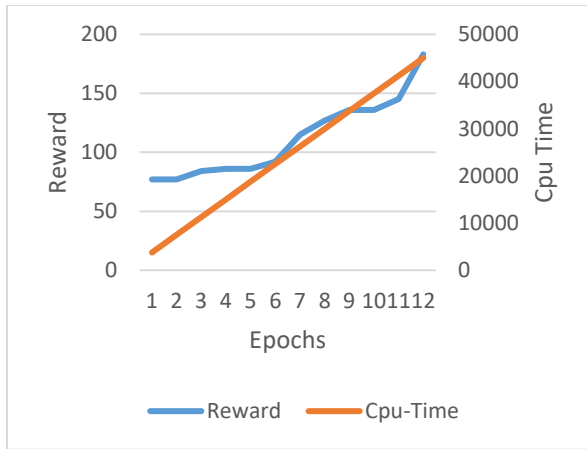


Fig. 6.5 Multi-agent (size of 4) feedforward neural network (with GA) learning graph with respect to CPU-time, and Epochs.

In this section (section B), we simulated the planning phase for a single agent in a POMDP environment that leverages on the underlying MDP created in section A. Our methodology incorporated the particle filter algorithm leveraging the roulette wheel selection for the resampling phase [138].

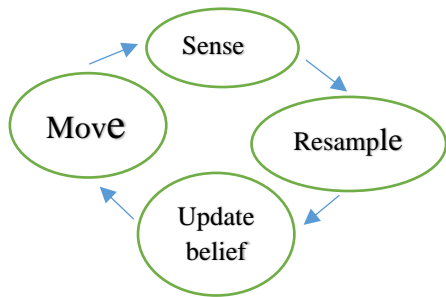


Fig. 6.6 Agent motion model for POMDPs

### 6.4 Monte Carlos Resampling Model

In our simulation, four-sensor nodes are strategically placed at the edges of the grid world with which the agent is able to localize itself with respect to its belief update [139]. Gaussian noise was added to the sensor inputs. For simplicity, we discretized the agent’s motion within the stochastic environment. The key idea is to efficiently map the belief state of the agent (particle filter averaged output) with the actual state of the agent. From fig (6.6), the agent’s policy is mapped directly to its belief which is based on the underlying MDP. Consequently, an accurate mapping would ultimately guide the agent to the goal node.

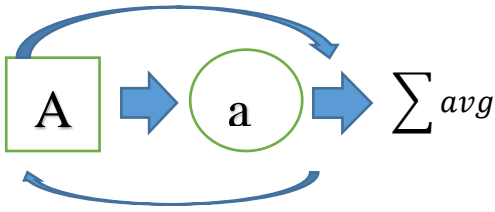


Fig. 6.7 Process flow diagram for traditional resampling and localization of belief state using particle filters. Capital 'A' (Initial Random sample), Lowercase 'a' (resampled)

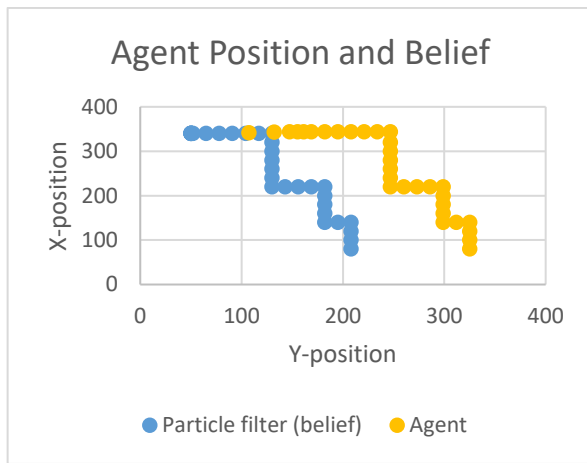


Fig. 6.8 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for traditional resampling.

The resampling model depicted in figure (6.7) is the traditional resampling model where particles are initialized randomly within the entire grid world [140] as depicted with the capital A. thereafter, a new weighted sample-based on important weights is produced (lower case a) via the roulette wheel selection algorithm. The x.y coordinates of the belief state are thereafter obtained by averaging the sum of the particles x.y coordinates. Fig. (6.8) shows the average result of this model. It's important to note that the agent motion model (fig 6.7) is iterated about five times with zero motion at the initialization phase before state transitions commence. The key idea is to minimize the error between the belief state and actual state before any transition begins. It's important to note that the initial state (position) of the agent in the world is unknown. An improved model (fig. 6.9) attempts to eliminate outliers resulting from the weighted samples via passing those samples through roulette wheel a second time to produce better-weighted sample fig (6.10) (lower case b) before averaging.

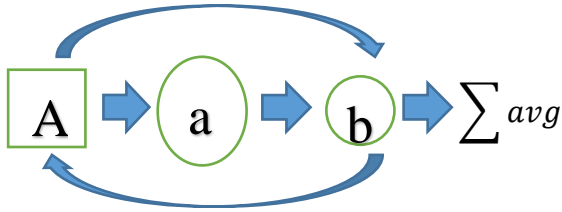


Fig. 6.9 Extended Process flow diagram for traditional resampling and localization of belief state using particle filters. Capital 'A' (Initial Random sample), Lowercase 'a, b' (resampled) with double phased resampling.

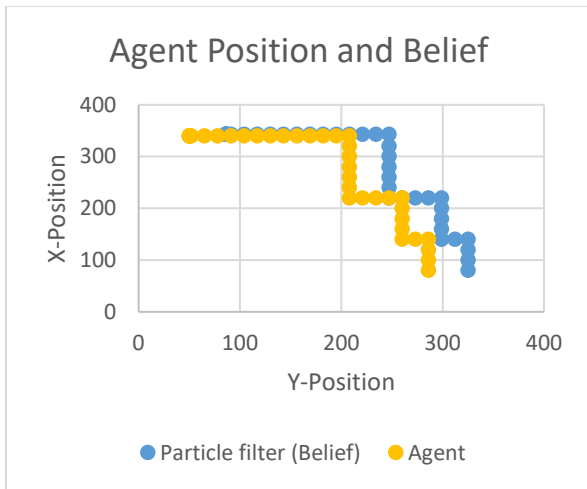


Fig. 6.10 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for double phased resampling.

Introducing a third layer (fig. 6.11) resampling produced even better results on the averages as shown in fig (6.12).

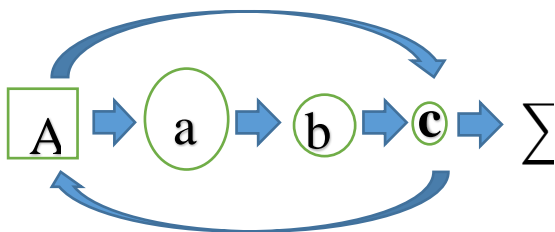


Fig. 6.11 Extended Process flow diagram for traditional resampling and localization of belief state using particle filters. Capital 'A' (Initial Random sample), Lowercase 'a, b, c' (resampled) with triple phased resampling.

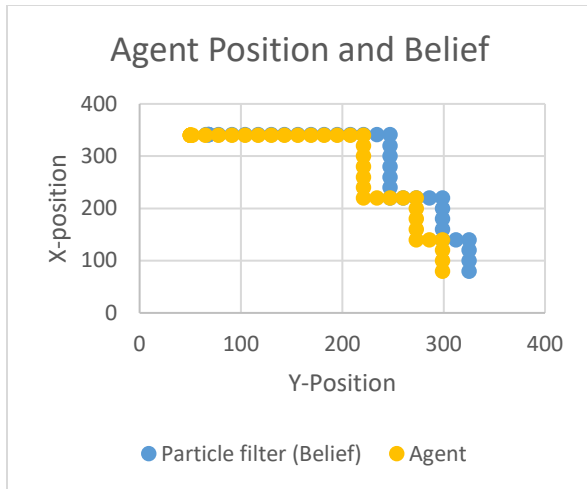


Fig. 6.12 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for triple phased resampling.

In our final model, we include a preprocessing phase such that N number of particles are randomly replicated 4 times in batches over the entire world depicted in the A, B, C, and D segments as shown in figure (6.13 and 6.14)

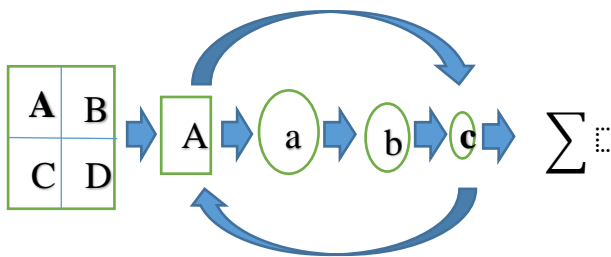


Fig. 6.13 Modified Process flow diagram for traditional resampling and localization of belief state using particle filters. Capital '(A, B, C D)' (Initial Random sample), Capital A (selected sample), Lowercase 'a, b, c' (resampled) with triple phased resampling.



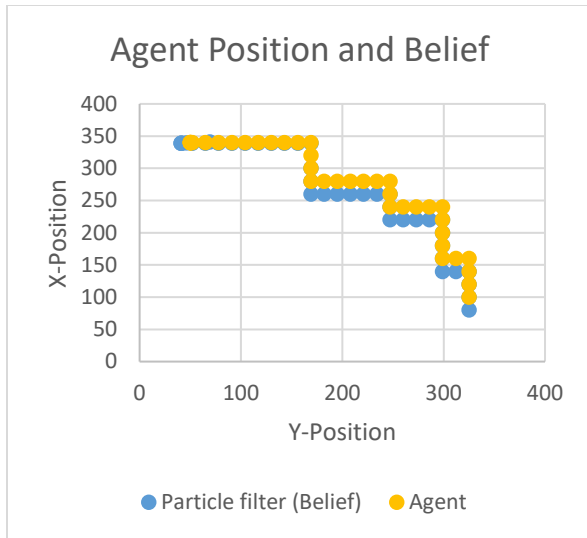


Fig. 6.14 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for preprocessed initialization with triple phased resampling.

The agent intuitively attracts the batch of particles with the highest probabilistic weights into the iterative phase. Thus leaving behind other batches of  $N$  particles. This procedure keeps the computational complexity simple while improving accuracy as shown in fig. (6.14)

## 6.5 The AMCL (Adaptive Monte Carlo Localization) approach

For the purpose of comparisons, we investigate the AMCL model which is a relatively recent state of the art localization algorithm. This algorithm randomly adjusts the number of free particles during the resampling phase based on their weights. By leveraging on the Kullback-Leibler divergence (KLD) algorithm, [141,142] the AMCL adapts a linear relationship with the number of particles in non-empty cells of the state space, and an upper bound on the number of resampled particles throughout the sense and move cycle [143] as shown in fig 6.15.

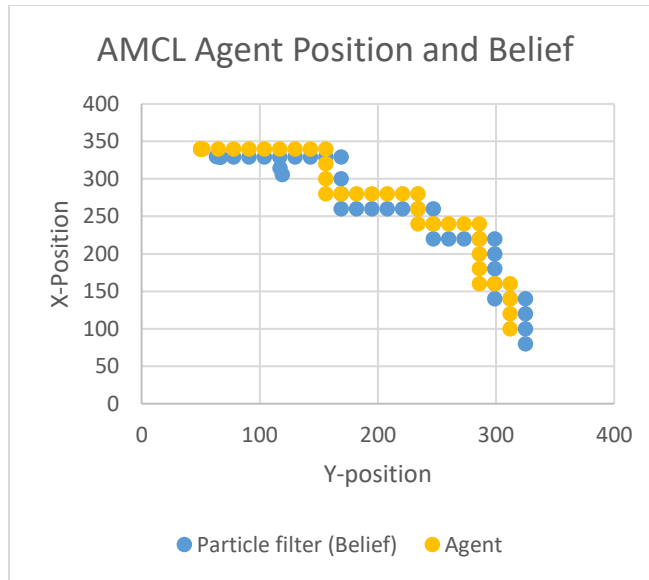


Fig. 6.15 Agent belief state (particle filter) and actual state transition from start position (upper left) to goal position (lower right) for the AMCL (KLD)

Our resampling model clearly shows better performance in comparison. The deployment of our implementation drastically reduced the frequency of occurrence of true negatives (fig 6.16a) the agent believes it's in a wall when it's actually not), and false positives (fig. 6.16b) the agent believes it's not in a wall when it actually is) when observed over multiple runs. The final model maintained true positives over multiple runs as shown in (fig. 16.6c).

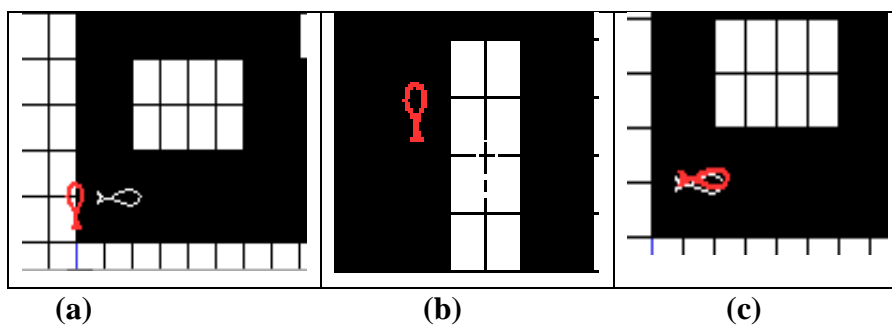


Fig. 6.16 (a) True negatives (the agent believes it's in a wall (belief in RED) when it's actually not), (b) False positives (the agent believes it's not in a wall, when it actually is). (c) True positives (agents position and belief are approximately same).

## 6.6 Discussion of Results

We obtained preliminary results in two phases for a typical learning and planning problem within a POMDP environment. In the first phase, we simulated the learning phase of a POMDP model using online, off-policy reinforcement Q-learning with both single and multi-agents. The rationale is for the agents to learn optimal policy within belief space. The simulation results showed a significant difference in CPU-time over episodes between the single and multi-agent framework. The multi-agent (with a size of 4) converged much faster. With the addition of an extra agent, we witnessed even further improvement in CPU-time.

In contrast, we simulated an alternative off-line learning approach using feedforward neural networks for multiple agents (with a size of 4) whose weights were optimized using genetic algorithm over multiple epochs. The rationale is for the agent to learn the model of the world by localizing all the absorbing states including the goal node and terminating with an optimal policy with respect to the goal node using dynamic programming. This model converged faster than the Q-learning model however not without some drawbacks. The model is not naturally suited for dynamic environments (such as open/closed doors) without a major modification to the algorithm which could impact the computational complexity.

In the second phase (planning phase), results show how segmented initialization of particles combined with multi-layer resampling improved belief state accuracy with respect to the agents' actual state estimation. Consequently, the mapping of the POMDP to the underlying MDP was with high fidelity.

## 6.7 Conclusion

We have compared two paradigms towards the learning phase and also contributed to the planning phase via a clever modification to the resampling stage of the particle filter algorithm within a POMDP environment. The multi-agent Q-learning is more robust for both static and dynamic environments, however, it asymptotes relatively slower when compared with the multi-agent feedforward neural network counterpart. In practical scenarios, the complexity of the world could influence the most appropriate choice of algorithm. Furthermore, we leverage on a classical resampling method (the roulette wheel) to demonstrate how an ingenious adaptation of the

algorithm using particle filters, improved the belief state accuracy with respect to the actual agent position within POMDP environments.

## CHAPTER 7

### SUMMARY AND FUTURE WORK

#### 7.1 Introduction

So far, we have been able to reveal via simulations, how simple multi-agent robots could achieve cooperate behavior while offering solutions to the multi-source localization problem. Localizing single emission sources is relatively trivial when compared to localizing multiple emission sources. While the former has been treated extensively in literature, the latter has comparatively received little emphasis. In addition, while there exists some incomplete solutions to the multi-source problem, the solutions lack adhesive synergy along with the absence of lucid techniques for the agents to proceed with a search after gradient source has been found. The key challenges to the multi-source localization paradigm are: lack of convergence, lack of a clear procedure for terminating search and finally, the absence of theoretical foundations for progress (after gradient source has been localized).

This thesis merges the strengths of multi-objective optimization with artificial neural networks for a group of non-complex robotic agents in order to achieve collaborative control (group behavior) while localizing multiple emission sources.

While an online, off-policy reinforcement Q-learning multi-agents paradigm may be effective for other problem domains, the implementation is not easily adapted to the standards used in literature for the multi-source localization problem. This is due to the fact that the location of the gradient sources should be unknown prior to the search. Consequently, we simulated as alternative, an off-line learning approach using feedforward neural networks for multiple agents whose weights were optimized using the hybrid optimization approach over multiple epochs. The rationale is for the agent to learn offline, an optimized way of navigating the grid world, amidst multiple gradient sources.

#### 7.2 Adapted hybrid model

Our final adapted model for resolving the multisource localization problems leverages on all the paradigms, algorithms and innovations discussed so far in this thesis. They include: the novel hybrid optimization, MDPs, and POMDPs, particle filter, dynamic programming, and the multi-objective optimization. The multi-objective optimization uses machine learning qualities of the artificial neural networks (ANNs) for a group of robots by leveraging on their basic yaw and

thrust actuators in order to achieve collaborative control (group behavior) amidst localization of multiple emission sources.

The final model incorporates an additional robotic agent that we refer to in this context as the MDP-agent. While the preliminary test utilized only two agents, the incorporation of the third agent brought robustness and a relatively stable solution to the multisource localization problem. The Algorithm is shown in the table below:

Table 7.1 Algorithm for the adapted hybrid model

<pre><b>Initalize_training();</b> <b>At convergence, select any best performing team at the pareto optimal front.</b> <b>Deploy all agents</b> <b>while (targets_not_found) {</b> <b>search();</b> <b>if (target_found){</b> <b>Activate policy to target location();</b> <b>move_MDP_agent();</b> <b>search();</b> <b>}</b> <b>}</b></pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

From the above table, the algorithm begins by initializing the training process. This phase is implemented with the multi-objective optimization algorithm, leveraging on the hybrid optimization procedure discussed in chapter 4 for a fast and robust convergence towards the Pareto optimal front. The fitness of the agents is determined by their objective functions: explorative and exploitative (homing) capabilities. While the explorative capability encourages the agents search of the entire grid world, the exploitative capability leverages on the agents bias towards homing in on gradient sources using the source signal trails. Because these capabilities are inversely related,

we opted for a multi-objective optimization algorithmic approach. At convergence, the best performing generations (at the Pareto optimal front) would have optimized (to a satisfactory level) the objectives of exploration and exploitation. Thereafter, all agents are deployed in the simulated world. The explorative and exploitive search (oblivious collaboration) commences and continues until all gradient sources are localized. It is important to note that there exists no local communication between agents, however, global communication is achieved via map update. Each agent keeps in real-time an updated copy of the map.

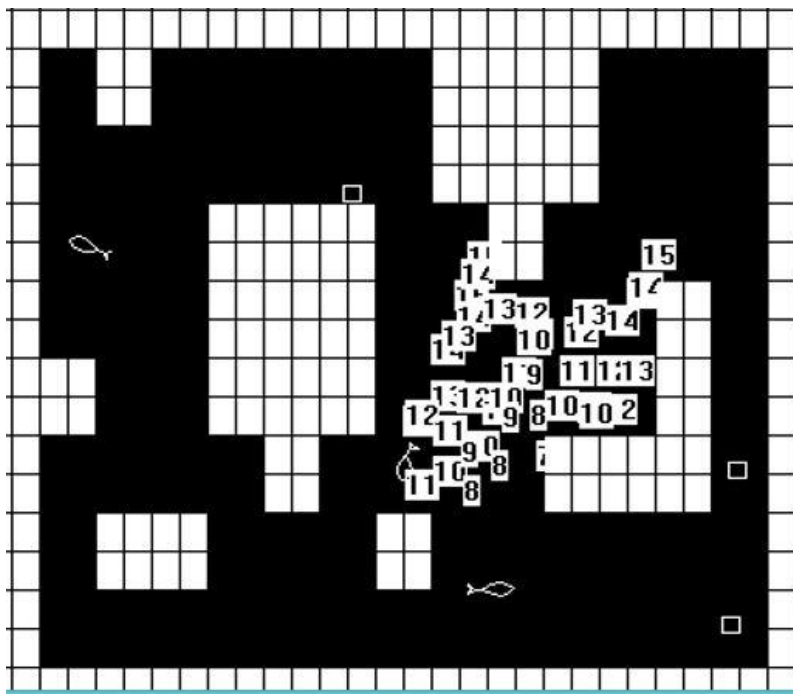


Fig. 7.1 Adapted hybrid model

Consequently, after a source has been localized, the MDP-agent leverages on a policy-based, path planning algorithm using dynamic programming to navigate towards the localized source (via global communication) while the agents unobtrusively proceed with their search. Targets are approached by the MDP-agent in a first find, first serve procedure. The algorithm terminates when the entire search space has been explored or rather when all gradient sources have been localized. By separating the agents for the search tasks from the MDP-agent which is saddled solely with the homing task (fig. 7.1), we implement a clever and robust algorithm for resolving the multisource localization problem.

This hybridized algorithm solves the fundamental problems of the multisource localization by providing progress after a source is found, convergence, and termination. In the presence of range, sensors, the algorithm showed high stability and robustness when simulated in a POMDP environment. Using a novel resampling technique for the Monte Carlo belief state update localization of agents, the MDP-agent was able to navigate to all source targets with significant accuracy over multiple simulated runs.

### **7.3 A Comparative Analysis with Existing Models**

We put in perspective our proposed model with one of the relatively best (with respect to the multisource problem) models in literature: the Bayesian occupancy grid model. The swarm size for the Bayesian model was 20, our hybrid model deploys just 2. The number of target source our implementation was 2 while the Bayesian used 3. It's noteworthy that this lack of standardizations for testing makes it difficult to tell what algorithm is best. However, that is outside the scope of this research thesis. While our simulation made use of a generic gradient source, the Bayesian model deployed radio frequency (RF) signal source. Unlike the Bayesian implementation, our source targets were fixed in real-time, however different source location was used for simulations. In addition, our implementation had no dead space nevertheless, a Gaussian source intensity was implemented such that source signals were highest at source locations and signal attenuation occurred as you move away for source location. While PSO used a foraging technique for a localized source such that the source impact on the world was removed after it was found, our implementation allowed for signal degradation such that lower gradient signals from other non-localized sources are perceivable. This phenomenon helped the search to progress after a gradient source is found. Just like the Bayesian algorithm, we consider the computation complexity for our final hybrid to be "high" due to the incorporation of the MDP-agent along with the training at initialization. We show that our hybrid model possesses a theoretical foundation based on modern machine learning paradigm (artificial neural networks) on POMDP environments.



Table 7.2 (a) Adapted Summary of multi-search options and proposed model

Source: Kathleen McGill, Stephen Taylor 2011 [13].

	<b>BESA</b>	<b>BRW</b>	<b>Proposed Model</b>
<b>Swarm size</b>	20	100	3
<b>Source Number</b>	2	2	2
<b>Source type</b>	Chemical	Generic	Generic
<b>Source mobility</b>	Fixed	Fixed	Fixed
<b>Variable source intensity</b>	-	Yes	Yes
<b>Dead space</b>	No	No	No
<b>Communication range</b>	Local	None	Global
<b>Agent deployment</b>	Near each other	Random, single location	Center
<b>Computational complexity</b>	Medium	Low	High
<b>Obstacle avoidance</b>	Swarm control	None	Artificial repulsion
<b>Sensing requirement</b>	Concentration location	Signal intensity	Signal intensity
<b>Proceeding after source is found</b>	None	None	Gradient attenuation
<b>Theoretical foundations</b>	None	None	ML/ POMDPs

Table 7.2 (b) Adapted Summary of multi-search options and proposed model

Source: Kathleen McGill, Stephen Taylor 2011 [13].

	<b>Bayesian occupancy</b>	<b>Attractant/repellant</b>	<b>Proposed Model</b>
<b>Swarm size</b>	20	11	3
<b>Source Number</b>	3	5	2
<b>Source type</b>	RF	Gaussian quadratic, planar	Generic
<b>Source mobility</b>	Mobile	Fixed	Fixed
<b>Variable source intensity</b>	-	Yes	Yes
<b>Dead space</b>	-	No	No
<b>Communication range</b>	Local	Global	Global
<b>Agent deployment</b>	-	-	Center
<b>Computational complexity</b>	High	Medium	High
<b>Obstacle avoidance</b>	Minimum path cost	Artificial repulsion	Artificial repulsion
<b>Sensing requirement</b>	RF signal strength, location	Concentration location	Signal intensity
<b>Proceeding after source is found</b>	None	None	Gradient attenuation
<b>Theoretical foundations</b>	None	Single source profiles	ML/POMDPs

#### 7.4 Summary and conclusion

Our research began with the hybridization of 3 metaheuristic algorithms and compared the hybrid combination of these algorithms in terms of different performance metrics. These include the number of times each of the hybrid combinations has produced an optimal solution and different other metrics such as the computational complexity and convergence to the global optima.

This contribution to the field of robotic optimization reveal that by leveraging the strengths of multiple metaheuristic algorithms, hybridization can lead to more optimal results.

Building upon the positive results obtained on the hybridization (chapter 4) with the 30 benchmark objective functions, we went further with the hybridization paradigm using dynamic programming, NSGA-II and feedforward artificial neural network to simulate multi-robot collaboration and control in a multi-source localization setting. The objective was to present a more practicable model for proceeding with a search after a gradient source has been localized.

In addition, reinforcement learning was addressed with two paradigms towards the learning phase being compared and a modification to the resampling stage of the particle filter algorithm in order to resolve the convergence and termination of the search algorithm within a partially observable environment.

The hybrid optimizer also provided a scalable framework on which other meta-heuristics (apart from those used in this research) can be tested such as: ABC (artificial bee colony), DE (Differential Evolution), Ant Colony, etc.

Finally, we brought all these concepts together into a unified framework which implements oblivious, yet optimized cooperative control of robotic agents and was demonstrated in resolving the multisource localization problem. Empirical results revealed superior performance over existing counterparts.

## **7.5 Practical implementations**

Our proposed algorithm finds practical value in swarm robotics, mission-oriented operations, reconnaissance, search and rescue operations, foraging, and Nanorobotics. With minor modifications, the failure of one agent doesn't jeopardize the entire operation. Where the cost of producing one robot is significant, optimization of the robot's motions becomes pertinent. This underpins the rationale or significance of an optimized swarm algorithm

## **7.6 Future direction**

In our implementation, we assumed the targets are fixed in a 2-dimensional grid world which could be fully observable or partially observable. Unlike the Bayesian algorithm, our research finding was based strictly on simulations. A practical test may uncover more interesting axis for further

investigations. Furthermore, where targets are mobile, the dynamics of our MDP-robot implementation may scale significantly to include localization of the drifting targets. It would be interesting to know how mobile sources would affect the dynamics of our implementation. Another exciting axis would be the investigation of how our proposed algorithm could learn to partition a larger school or swarm into subgroups as they progress with the search process.

## REFERENCES

1. X. Cui, R. K. Ragade, and A. S. Elmaghraby, "A collaborative search and engage strategy for multiple mobile robots with local communication in largescale hostile area", International Symposium on Collaborative Technologies and Systems, pp 244 –249, San Diego, California, January, 2004.
2. L. Marques, U. Nunes, and A. T. De Almeida, "Particle swarm-based olfactory guided search", *Auton. Robot*, 20, pp. 277–287, 2006.
3. V. Gazi, K.M. Passino, "Stability analysis of social foraging swarms. *IEEE Trans. Syst.Man Cybernet- Cybernetics*", 34, 1, pp. 539–557, 2004.
4. K.M. Passino, "Biomimicry of bacterial foraging for distributed optimization and control", *IEEE Contr. Syst. Mag.*, 22, 3, pp.52–67, 2002.
5. S. Pouyanfar, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Comput. Surv.*, 51,5, pp. 92, 2018.
6. S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, 220, 671–680, 1983.
7. J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks*, Perth, Australia, pp.1942–1948, Nov. 1995.
8. X.S. Yang, S. Deb, "Cuckoo search via Levy flights", in: *Proc. Of World Congress on Nature & Biologically Inspired Computing*, India. IEEE Publications, USA, pp. 210-214, 2009.
9. S. A. Kazarlis, S. E. Papadakis, J. B. Theocharis, and V. Petridis, "Microgenetic algorithms as generalized hill-climbing operators for GA optimization," *IEEE Trans. Evol. Comput.*, 5, pp.204-217, 2001.
10. S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localization for mobile robots," *Artificial Intelligence*, 128(1-2), 2000.
11. J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," in *Proc. Int. Jnt. Conf. on ArtificialIntelligence*, pp. 477–484, 2003.
12. N. Roy, G. Gordon, and S. Thrun, "Finding aproximate POMDP solutions through belief compression," *J. Artificial Intelligence Research*, 23, pp. 1–40, 2005.
13. K. McGill, S. Taylor, "Robot algorithms for localization of multiple emission sources," *ACM Comput. Surv.*, 43(3), pp. 1-15, 2011.

14. T. Kubota, Y. Kuroda, Y. Kunii, I. Natakani, "Micro-planetary rover Micro5," in: Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (ESA SP-440), Noordwijk, Netherlands, pp. 373–378, 1999.
15. R.R. Murphy, "A decade of rescue robots," In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems," Vilamoura, Portugal, pp. 5448–5449, 2012.
16. L. Dunbabin, and M. Marques, "Robotics for environmental monitoring," IEEE Robotics & Automation Magazine, 19 (1), pp. 20-23, 2012.
17. Couceiro, M.S., Portugal, D., Rocha, R.P., "A collective robotic architecture in search and rescue scenarios," In: Proc. of 28th Symposium on Applied Computing (SAC 2013), pp. 64–69, 2013.
18. Barlow, G. J., Choong, K. O., and Smith, S. F., "Evolving cooperative control on sparsely distributed tasks for UAV teams without global communication," In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 177–184, 2008.
19. Luke, S., Sullivan, K., Panait, L., and Balan, G. T., "Decentralized algorithms for cooperative target observation," In Proceedings of the International Conference of Autonomous Agents and Multiagent Systems, pp. 911–917, 2005.
20. H. Bach, I. G. McLean, C. Akerblom, and R. Sargisson, "Improving mine detection dogs: an overview of the GICHD dog program," in Proc. EUDEM2-SCOT conference on requirements and technologies for the detection, removal and neutralization of landmines and UXO, pp.15–18, 2003.
21. I. Voinov, M. Nosikov, "Automatic and Manual Control Algorithms of Radiation-Proof Manipulators." In Proceedings of the IEEE 2018 Global Smart Industry Conference (GloSIC), Chelyabinsk, Russia, pp. 1–6, 2018.
22. M.H. E. Larcombe, J.R. Andhelsall, "Robotics in Nuclear Engineering: Computer Assisted Teleoperation in Hazardous Environments with Particular Reference to Radiation Fields," Kluwer Academic Publishers, Norwell, MA. 1984
23. B. Tribelhorn and Z. Dodds, "Evaluating the roomba: A low-cost, ubiquitous platform for robotics research and education," In Robotics and Automation, 2007 IEEE International Conference on, pp.1393 –1399, 2007.
24. R. A. Freitas, "Current Status of Nanomedicine and Medical Nanorobotics," J. Computational and Theoretical Nanoscience, 2, pp. 1–25, 2005.
25. H. Q. Min, J. H. Zhu, and X. J. Zheng, "Obstacle Avoidance", In Proc. Int. Conf. on Machine Learning and Cyber-netics, pp. 2950-2956, 2005.

26. R. Rozas, J. Morales, D. Vega, "Artificial smell detection for robotic navigation," In Proceedings of the 5th International Conference on Advanced Robotics, pp. 1730–1733, 1991.
27. K. N. Krishnanand and D. Ghose, "A glowworm swarm optimization based multi-robot system for signal source localization," In Design and Control of Intelligent Robotic Systems: SCI 177. D. Liu, L. Wang, and K. C. Tan, Eds. Springer Verlag, Berlin, Germany, pp. 49–68, 2009a.
28. A. Dhariwal, G. S. Sukhatme, A. A. G. Requicha, "Bacterium-inspired robots for environmental monitoring," In Proceedings of the International conference on Robotics and Automation, pp. 1436–1443, 2004.
29. S. Pang, and J.A. Farrell, "Chemical plume source localization," IEEE Trans. Syst. Man Cybernet.—Part B: Cybernet. 36, 5, pp. 1068–1080, 2006.
30. G. Ferri, M. V. Jakuba, E. Caselli, V. Mattoli, B. Mazzolai, D.R. Yoerger, and P. Dario, "Localizing multiple gas/odor sources in an indoor environment using Bayesian occupancy grid mapping," In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 566–571, 2007.
31. G. Sandini, G. Lucarini, and M. Varoli, "Gradient driven self-organized systems," In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 429–432, 1993.
32. R. Bachmayer, and N.E. Leonard, "Vehicle networks for gradient descent in a sampled environment," In Proceedings of the 41st IEEE Conference on Decision and Control, pp. 1–6, 2002.
33. T. Hogg, "Coordinating microscopic robots in viscous fluids," Auton. Agent Multi-Agent Syst., 14, pp. 217–305, 2007.
34. D. Borah, and A. Balagopal, "Localization and tracking of multiple near-field sources using randomly distributed sensors," In Proceedings of the 38th Asilomar Conference on Signals, Systems, and Computers, pp. 1323–1327, 2004.
35. A. Lilienthal, D. Reimann, and A. Zell, "Gas source tracing with a mobile robot using an adapted moth strategy," In Proceedings of the IEEE International Conference on Robotics and Automation, pp. 150–160, 2003.
36. A.T. Hayes, A. Martinoli, and R.M. Goodman, "Distributed odor source Localization," IEEE Sens. J. , 2(3), pp. 260–271, 2002.
37. D. Zarzhitsky, D. Spears, and W. Spears, "Swarms for chemical plume tracing," In Proceedings of the Swarm Intelligence Symposium, pp. 249–256, 2005.
38. K.N. Krishnanand, and D. Ghose, "Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions," Swarm Intell, 3(2), pp. 87–124, 2009b.

39. R. Parpinelli, H. Lopes, "New inspirations in swarm intelligence: a survey," *International Journal of Bio-Inspired Computation*, 3(1), pp. 1–16, 2011.
40. J. Pugh and A. Martinoli, "Multi-Robot Learning with Particle Swarm Optimization", In *Proc. of the 5th Int. Joint Conf. on Autonomous Agents and Multi-agent Systems*, pp. 441–448, 2006.
41. F. H. Branin Jr., "Widely Convergent Method of Finding Multiple Solutions of Simultaneous Nonlinear Equations," *IBM Journal of Research and Development*, 16(5), pp. 504-522, 1972.
42. P. Scerri, T. Von Gonten, G. Fudge, S. Owens, and K. Sycara, "Transitioning multi-agent technology to UAV applications," In *Proceedings of 7th International Conference on Autonomous Agents and Multi-agent Systems*, pp. 89–96. 2008.
43. X.X. Chen, J. Huang, "Odor source localization algorithms on mobile robots: A review and future outlook," *Robot. Auton. Syst.*, 112, pp.123-136, 2019.
44. M. Vergassola, E. Villermaux, B.I. Shraiman, "Infotaxis' as a strategy for searching without gradients," *Nature*, 445(7126), pp. 406-409, 2007.
45. E.M. Moraud, D. Martinez, "Effectiveness and robustness of robot infotaxis for searching in dilute conditions," *Frontiers in neurorobotics*, 4, pp. 1, 2010.
46. J. L. Blanco, J. G. Monroy, A. Lilienthal, J. Gonzalez-Jimenez, "A kalman filter based approach to probabilistic gas distribution mapping, in: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ACM, pp. 217–222, 2013.
47. D. Grünbaum, M.A. Willis, "Spatial memory-based behaviors for locating sources of odor plumes," *Movement ecology*, 3, pp.11, 2015.
48. H. Hajieghrary, M.A. Hsieh, I.B. Schwartz, "Multi-agent search for source localization in a turbulent medium," *Physics Letters A380*, pp.1698-1705, 2016.
49. J.A. Farrell, S. Pang, W. Li, "Plume mapping via hidden Markov methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 33(6), pp. 850- 863, 2003.
50. P. Civicioglu and E. Besdok, "A conceptual comparison of the cuckoo search, particle swarm optimization, differential evolution and artificial bee colony algorithms," *Artificial intelligence review*, pp.1–32, 2013.
51. A. Prügel-Bennett, "Benefits of a population: five mechanisms that advantage population-based algorithms," *IEEE Trans. Evol. Comput.*, 14(4), pp. 500-517, 2010.
52. J. Suarez and R. Murphy, "A Survey of Animal Foraging for Directed, Persistent Search by Rescue Robotics," In *Proc. of the 2011 IEEE International Symposium on Safety, Security and Rescue Robotics*, Kyoto, Japan, pp. 314-320, 2011.



53. M. Mitchell, "An introduction to Genetic algorithms," MIT press, fifth edition, 1999.
54. M. Jamil and X.S. Yang, "A literature survey of benchmark functions for global optimization problems," *Int. Journal of Mathematical Modelling and Numerical Optimisation*, 4(2), pp. 150–194, 2013.
55. D. H. Ackley, "A Connectionist Machine for Genetic Hill-Climbing," Kluwer Dordrecht, Academic Publishers, 1987.
56. I. O. Bohachevsky, M. E. Johnson, M. L. Stein, "General Simulated Annealing for Function Optimization," *Technometrics*, 28(3), pp. 209-217, 1986.
57. C. Munteanu and V. Lazarescu, "Improving mutation capabilities in a real-coded genetic algorithm," In *Proceedings of the First European Workshops*, pages 138–149, Goteburg, Sweden, pp. 26.-27, May 1999.
58. F. H. Branin Jr., "Widely Convergent Method of Finding Multiple Solutions of Simultaneous Nonlinear Equations," *IBM Journal of Research and Development*, 16(5), pp. 504-522, 1972.
59. A. Lavi, T. P. Vogel (eds), "Recent Advances in Optimization Techniques," JohnWiley & Sons, 1966.
60. M. M. Ali, C. Khompataporn, Z. B. Zabinsky, "A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems," *Journal of Global Optimization*, vol. 31, pp. 635-672, 2005.
61. C. J. Chung, R. G. Reynolds, "CAEP: An Evolution-Based Tool for Real-Valued Function Optimization Using Cultural Algorithms," *International Journal on Artificial Intelligence Tool*, vol. 7, no. 3, pp. 239-291, 1998.
62. S. S. Rao, "Engineering Optimization: Theory and Practice," John Wiley & Sons, 2009.
63. R. L. Haupt, S. E. Haupt, *Practical genetic Algorithms*, John Wiley and sons Inc, 2004.
64. A. A. Goldstein, J. F. Price, "On Descent from Local Minima," *Mathematics and Computation*, vol. 25, no. 115, pp. 569-574, 1971.
65. H. H. Rosenbrock, "An Automatic Method for Finding the Greatest or least Value of a Function," *Computer Journal*, vol. 3, no. 3, pp. 175-184, 1960.
66. S.K. Mishra, "Some New Test Functions for Global Optimization and Performance of Repulsive Particle Swarm Method", *Social Science Research Network (SSRN) Working Papers Series*, <http://ssrn.com/abstract=927134>, 2006

67. S.K. Mishra, "Global Optimization by Differential Evolution and Particle Swarm Methods: Evaluation on Some Benchmark Functions". SSRN: <http://ssrn.com/abstract=933827>, 2006
68. E. P. Adorio, "MVF – multivariate test functions library in C for unconstrained global optimization," 2005. available from <http://geocities.com/eadorio/mvf.pdf>
69. C. S. Adjiman, S. Sallwig, C. A. Flouda, A. Neumaier, "A Global Optimization Method, aBB for General Twice-Differentiable NLPs-1, Theoretical Advances," *Computers Chemical Engineering*, 22(9), pp. 1137-1158, 1998.
70. J.R. Koza, "Genetic programming: on the programming of computers by means of natural selection," MIT Press, Cambridge, 1992.
71. L. Davis, "Genetic Algorithms and Simulated Annealing," Pitman, London, 1987.
72. G. Syswerda and J. Palmucci, "The Application of Genetic Algorithms to Resource Scheduling," In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, California, pp. 502–508, 1991.
73. X.S. Yang and S. Deb, "Cuckoo search via Lévy flights," In *Nature & Biologically Inspired Computing*, 2009. NaBIC 2009 World Congress on, pp. 210–214, 2009.
74. N. Damavandi, S. Safavi-Naeini, "A Hybrid Evolutionary Programming Method for Circuit Optimization," *IEEE Transaction on Circuit and Systems I*, 52(5), pp. 902-910, 2005.
75. M. Negnevitsky, "Artificial intelligence: a guide to intelligent systems second edition," Pearson Education Limited, Edinburgh Gate, Harlow, 2005
76. R. Rozas, J. Morales, and D. Vega, "Artificial smell detection for robotic navigation," In *Proceedings of the 5th International Conference on Advanced Robotics*, pp. 1730–1733, 1991.
77. Z. Li, Z. Shang, B. Y. Qu, and J. J. Liang, "Feature selection based on manifold-learning with dynamic constraint handling differential evolution," in *Proc. IEEE Congr. Evol. Comput.*, Beijing, China, pp. 332–337, 2014.
78. J. H. Holland, K. J. Holyoak, R. E. Nisbett and P. R. Thagard, "INDUCTION: Processes of Inference, Learning, and Discovery," MIT Press, 1986.
79. J. H. Holland, "Adaptation in Natural and Artificial Systems," MIT Press, 1992.
80. J. Hertz, A. Krogh, and R. G. Palmer, "Introduction to the Theory of Neural Computation," Santa Fe Institute Studies in the Sciences of Complexity lecture notes. AddisonWesley Longman Publ. Co., Inc., Reading, MA, 1991.
81. J. H. Holland, "COMPLEX ADAPTIVE SYSTEMS," 121(1), pp. 17-30, winter 1992.

82. Montana, D. J and L. D. Davis “Training feed forward networks using genetic algorithms,” In Proceedings of 11th Intl. Joint Conf. in Artificial Intelligence (IJCAI), Detroit, MI, Morgan Kaufmann, San Mateo, CA, pp. 762-767, 1989.
83. P.W. Tsai, T.T. Nguyen, T.K. Dao “Robot path planning optimization based on multi-objective grey wolf optimizer,” In: International conference on genetic and evolutionary computing, Springer, 536, pp. 166–173, 2017.
84. D. C. Dang, T. Friedrich, T. Kotzing, M. Krejca, P. K. Lehre, P. Oliveto, D. Sudholt and A. Sutton, ‘Emergence of diversity and its benefits for crossover in genetic algorithms’, in Proc. of PPSN XIV, pp. 890–900, 2016.
85. H. Ishibuchi, H. Masuda, Y. Tanigaki, and Y. Nojima, “Modified Distance Calculation in Generational Distance and Inverted Generational Distance,” in Evolutionary Multi-Criterion Optimization. Guimaraes, Portugal: Springer International Publishing, pp. 110–125, 2015.
86. E. Shelhamer, P. Mahmoudieh, M. Argus, and T. Darrell, “Loss is its own reward: Self-supervision for reinforcement learning,” arXiv preprint arXiv:1612.07307, 2016.
87. D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” In International Conference on Machine Learning, pp. 387–395, 2014.
88. M. Otsuka, J. Yoshimoto, and K. Doya, “Free-energy-based reinforcement learning in a partially observable environment,” ESANN 2010 proceedings, European Symposium on Artificial Neural Networks – Computational Intelligence and Machine Learning, 2010.
89. D. Silver and J. Veness, “Monte-Carlo Planning in Large POMDPs,” in Proc. Neur. Inform. Process. Sys. Vancouver, Canada, pp. 1–9, 2010.
90. H. Kurniawati and V. Yadav, “An online POMDP solver for uncertainty planning in dynamic environment,” In Robotics Research, Springer, pp. 611–629, 2016.
91. S. Omidshafiei, A.A. Agha-Mohammadi, C. Amato, S.Y. Liu, J.P. How, and J. Vian, “Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions,” The International Journal of Robotics Research, 36(2), pp.231–258, 2017.
92. T. Li, H. Fan, S. Sun, “Particle filtering: Theory, approach, and application for multitarget tracking,” Acta Autom. Sin., 41, pp. 1981–2002, 2015.
93. L. Martino, V. Elvira, G. Camps-Valls, “Group importance sampling for particle filtering and mcmc.” arXiv, arXiv:1704.0277, 2017
94. S. Thrun, W. Burgard, and D. Fox, “Probabilistic Robotics,” Cambridge, MA: MIT Press, 2005.

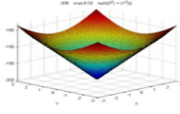
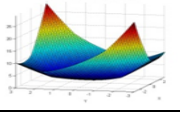
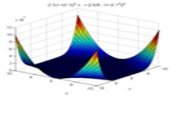
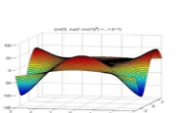
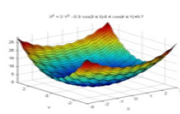
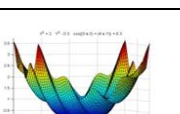
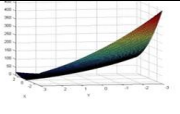
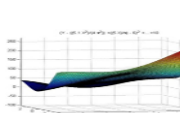
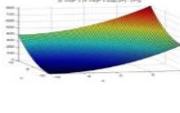
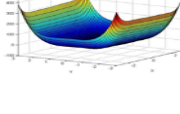
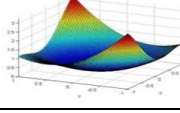
95. G. Best, J. Faigl and R. Fitch, “Online planning for multi-robot active perception with self-organizing maps,” *Autonomous Robots* doi: 10.1007/s10514-017-9691-4, 2018.
96. J. Asmuth and M. Littman, “Approaching Bayes-optimality using Monte-Carlo tree search,” In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, pages 19–26, 2011.
97. M. Toussaint, “Robot trajectory optimization using approximate inference,” In *Int. Conf. on Machine Learning*, ACM, pp. 1049–1056, 2009.
98. K. A. De Jong, “An analysis of a class of genetic adaptive systems,” Ph.D. thesis, University of Michigan, 1975.
99. D. E. Goldberg, and J. Richardson, “Genetic algorithm with sharing for multimodal function optimization,” In *Proceedings of the second international conference on genetic algorithms and their applications*, Cambridge, Massachusetts, USA, pp. 41-49, 1987.
100. I. J. Eshelman, and J. D. Schaffer, “Preventing premature convergence in genetic algorithms by preventing incest,” In: R. Belew, L.B. Booker, (eds) *Proc. of the Fourth Int. Conf. on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, pp. 115-122, 1991.
101. J. H. Holland, “Adaptation in neural and artificial systems,” University of Michigan press second edition, 1975.
102. Y. Liu, and K.M. Passino, “Biomimicry of social foraging bacteria for distributed optimization: Models, principles, and emergent behaviors,” In *J. Opt. Theor. Appl.*, 115, 3, pp.603–628, 2002.
103. N. Barton and T. Paixão, “Can quantitative and population genetics help us understand evolutionary computation?,” in *Proc. of GECCO*, 13, pp. 1573–1580, 2013.
104. B. Doerr, C. Doerr and F. Ebel, “From black-box complexity to designing new genetic algorithms,” *Theor. Comput. Sci.*, 567, pp. 87–104, 2015.
105. R.R. Murphy, “An Introduction to AI Robotics (Intelligent Robotics and Autonomous Agents),” Chapters 3-6, MIT Press, 2000.
106. C. Bishop, M. Svensen, & C. Williams, “GTM: the generative topographic mapping,” *Neural Computation*, 10 (1), 215-234. 1998.
107. J. M. Moore, A. J. Clark, and P. K. McKinley. Evolution of station keeping as a response to flows in an aquatic robot. In *Proceedings of the 15th Genetic and Evolutionary Computation Conference*, ACM Press, New York, NY, pp. 239–246, 2013.

108. R. Olfati-Saber, "Distributed tracking for mobile sensor networks with information-driven mobility," In Proceedings of the American Control Conference, pp. 4606–4612, 2007.
109. V. Chandrasekhar, W.K.G. Seah, Y.S. Choo, H.V. Ee, "Localization in Underwater Sensor Networks - Surveys and Challenges," In Proc. of the WUWNet, pp.33–40, 2006.
110. M. Hahn and J. Rice, "Undersea Navigation via a Distributed Acoustic Communication Network", Proceedings of the Turkish International Conference on Acoustics, pp.4-8, 2005.
111. N.Y. Ko, T.G. Kim, "Comparison of kalman filter and particle filter used for localization of an underwater vehicle," In: 2012 9th international conference on ubiquitous robots and ambient intelligence (URAI), pp.350–352, 2012.
112. A. Savvides, C.-C. Han, and M. Srivastava, "Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors," Proc. 7th ACM MobiCom, pp. 166–79, 2001.
113. A. Boukerche, H. Oliveira, E. Nakamura, and A. Loureiro, "Localization systems for wireless sensor networks," IEEE Wireless Commun. Mag., 14(6), pp. 6–12, 2007.
114. J.N. Ash and R.L. Moses, "Acoustic time delay estimation and sensor network self-localization: Experimental results," The journal of acoustical society of America, 118, pp.841, 2006.
115. Y. T. Chan and K. C. Ho, "A simple and efficient estimator for hyperbolic location," IEEE Trans. Signal Process., 42(8), pp. 1905–1915, 1994.
116. X.U. Yaosong, W. Dandan, F. Hua, "Underwater acoustic source localization method based on TDOA with particle filtering," In Proceedings of the 26th Chinese Control and Decision Conference (2014 CCDC), Changsha, China, pp.4634–4637, 2014.
117. K.V. MacKenzie, "Nice-term equation of sound speed in the oceans," Journal of the acoustic society of America, 70, pp. 807, 1981.
118. A. Martins, A. Dias, J. Almeida, H. Ferreira, C. Almeida, G. Amaral, D. Machado, J. Sousa, P. Pereira, A. Matos, V. Lobo, E. Silva, "Field experiments for marine casualty detection with autonomous surface vehicles," In Proceedings of the 2013 OCEANS, San Diego, CA, USA, pp. 1-5, 2013.
119. R. Murphy, E. Steimle, M. Lindemuth, D. Trejo, M. Hall, D. Slocum, S. Hurlebaus, Z. Medina-Cetina, "Use of unmanned marine vehicles for hurricane damage inspection," In Proceedings of the OCEANS 2009, Biloxi, MS, USA, 2, pp. 783–790, 2009.
120. J. A. Bagnell, & J. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), , Seoul, South Korea. IEEE Press, pp. 1615-1620, 2001.

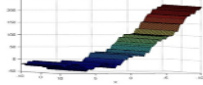
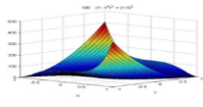
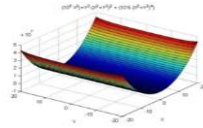
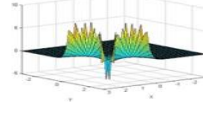
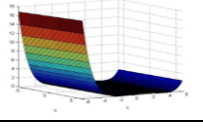
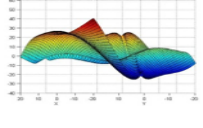
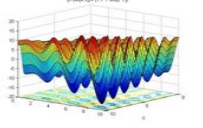
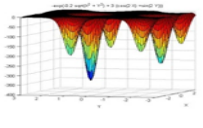
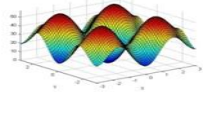
121. C. Boutilier, & D. Poole, "Computing optimal policies for partially observable Markov decision processes using compact representations," In Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), pp. 1168-1175, 1996.
122. E. Hansen, and Z. Feng, "Dynamic programming for POMDPs using a factored state representation," In Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling, pp.130–139, 2000.
123. Z.N. Sunberg and M. J. Kochenderfer, "Online algorithms for POMDPs with continuous state, action, and observation spaces," in Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS), pp. 259–263, 2018.
124. M. Corah, and N. Michael, "Efficient online multi-robot exploration via distributed sequential greedy assignment," In: Proceedings of Robotics: Science and Systems, 2017.
125. R.S. Sutton, and A.G. Barto, "Reinforcement learning: An introduction," volume 1. MIT press Cambridge, 1998.
126. P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," arXiv:1709.06560, 2017.
127. J. Schulman, P. Abbeel, and X. Chen, "Equivalence between policy gradients and soft Q-learning," arXiv preprint arXiv:1704.06440, 2017.
128. J.S. Gutmann and D. Fox, "An experimental comparison of localization methods continued," In Proceedings of International Conference on Intelligent Robots and Systems, pp.454-459, 2002.
129. T. Patten, W. Martens, & R. Fitch, "Monte Carlo planning for active object classification," Autonomous Robots. 2017. <https://doi.org/10.1007/s10514-017-9626-0>. Retrieved 2021
130. N. Cao, K.H. Low, & J.M. Dolan, "Multi-robot informative path planning for active sensing of environmental phenomena: A tale of two algorithms," In Proceedings of AAMAS, pp. 7–14, 2013.
131. G. Best, M. Forrai, R. R. Mettu, and R. Fitch, "Planning-aware communication for decentralised multi-robot coordination," in 2018 IEEE International Conference on Robotics and Automation, Brisbane, Australia, pp. 1050–1057, 2018.
132. H. Li, H. Gao, T. Lv, and Y. Lu, "Deep q-learning based dynamic resource allocation for self-powered ultra-dense networks," in IEEE ICC (ICC Workshops), pp. 1–6, 2018.
133. N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," CoRR, vol. abs/1810.07862, 2018.

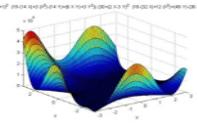
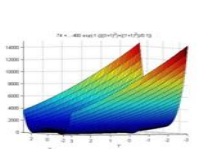
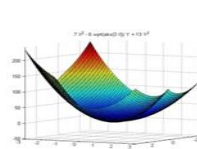
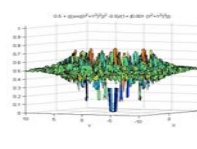
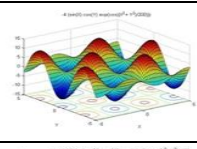
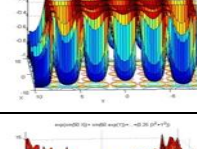
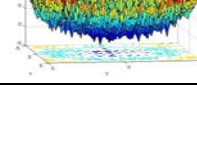
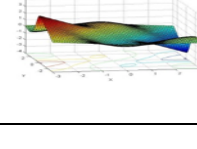
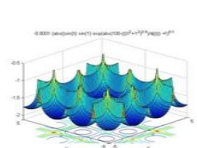
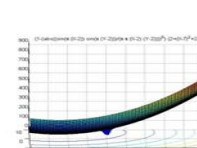
134. Y. Lin, X. Dai, L. Li, and F.-Y. Wang, "An efficient deep reinforcement learning model for urban traffic control," arXiv preprint arXiv:1808.01876, 2018
135. N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous networks," in IEEE GLOBECOM, Abu Dhabi, UAE, Dec., pp. 1–6, 2018
136. M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in The Thirty-Second AAAI Conference on Artificial Intelligence, 2018.
137. P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in Proceedings of the 2000 Congress on Evolutionary Computation, 1, pp. 256–263, 2000.
138. W. Wang, J. Hao, Y. Wang, and M. Taylor, "Towards cooperation in sequential prisoner's dilemmas: a deep multiagent reinforcement learning approach," arXiv preprint arXiv:1803.00162, 2018.
139. Li, T.; Corchado, J.M.; Sun, S.; Fan, H. Multi-EAP, "Extended EAP for multi-estimate extraction for SMC-PHD filter," Chin. J. Aeronaut, 30, pp. 368–379, 2017.
140. D. Sun, F. Geißer, and B. Nebel, "Towards effective localization in dynamic environments," in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, pp. 4517–4523, 2016.
141. S. Sun, T. Li, and T. P. Sattar, "Adapting sample size in particle filters through KLD-resampling," Electronics Letters, 49(12), pp. 740–742, 2013.
142. G. Peng, W. Zheng, Z. Lu, J. Liao, L. Hu, G. Zhang, and D. He, "An improved AMCL algorithm based on laser scanning match in a complex and unstructured environment," Complexity 2018, vol. 11, Article ID 2327637, 2018. <https://doi.org/10.1155/2018/2327637>. Retrieved 2021
143. Y. Li, C. hi, "Localization and Navigation for Indoor Mobile Robot Based on ROS," Chinese Automation Congress (CAC), Xi'an, China, IEEE, 2018, doi: 10.1109/CAC.2018.8623225, pp. 1135–1139, 2018.

## ADDENDUM A

Label	Function name	Function Plot	F(x)*
<b>F1</b>	<b>Ackley2 Function</b>		<b>-200</b>
<b>F2</b>	<b>Bartels Conn Function</b>		<b>1</b>
<b>F3</b>	<b>Beale Function</b>		<b>0</b>
<b>F4</b>	<b>Bird Function</b>		<b>-106.76</b>
<b>F5</b>	<b>Bohachevsky 1 Function</b>		<b>0</b>
<b>F6</b>	<b>Bohachevsky 3 Function</b>		<b>0</b>
<b>F7</b>	<b>Booth Function</b>		<b>0</b>
<b>F8</b>	<b>Branin RCOS-2 Function</b>		<b>-39.195</b>
<b>F9</b>	<b>Brent Function</b>		<b>0</b>
<b>F10</b>	<b>Camel Function – Six Hump</b>		<b>-1.0316</b>
<b>F11</b>	<b>Camel Function – Three Hump</b>		<b>0</b>



<b>F12</b>	<b>Chichinadze Function</b>		<b>-42.9444</b>
<b>F13</b>	<b>Cube Function</b>		<b>0</b>
<b>F14</b>	<b>Deckkers-Aarts Function</b>		<b>-24776.5</b>
<b>F15</b>	<b>Easom Function</b>		<b>-1</b>
<b>F16</b>	<b>Freudenstein Roth Function</b>		<b>0</b>
<b>F17</b>	<b>Haupt Function 16</b>		<b>-25.2305</b>
<b>F18</b>	<b>Haupt Function 07</b>		<b>-18.5547</b>
<b>F19</b>	<b>Haupt Function 15</b>		<b>-345.36</b>
<b>F20</b>	<b>Egg Crate Function</b>		<b>0</b>

<b>F21</b>	<b>Goldstein Price Function</b>		<b>3</b>
<b>F22</b>	<b>Rosenbrock Modified Function</b>		<b>34.0412</b>
<b>F23</b>	<b>Rotated Ellipse Function</b>		<b>0</b>
<b>F24</b>	<b>Scahffer-1 Function</b>		<b>0</b>
<b>F25</b>	<b>Test-tube Holder Function</b>		<b>-10.872</b>
<b>F26</b>	<b>Pen-Holder Function</b>		<b>-0.9635</b>
<b>F27</b>	<b>Trefethen Function</b>		<b>-3.388</b>
<b>F28</b>	<b>Adjiman Function</b>		<b>-2.02181</b>
<b>F29</b>	<b>Cross-in-Tray Function</b>		<b>-2.06261</b>
<b>F30</b>	<b>Damavandi Function</b>		<b>0</b>

## Ackley 2 Objective function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	-3988.9	-199.445	0.118271263	0.343905893	
SA	20	-3967.465	-198.37325	0.913960724	0.956012931	
GA	20	-3998.991	-199.94955	0.019270892	0.138819639	
PSO	20	-3850.843	-192.54215	6.943544766	2.635060676	
POLY	20	-3998.679	-199.93395	0.011629103	0.107838317	
HYB	20	-3999.997	-199.99985	1.34211E-07	0.000366348	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	854.3341	5	170.866824	128.0432518	4.42439E-45	2.293911156
Within Groups	152.1269	114	1.334446147			
Total	1006.461	119				
SCHEFFE'S TEST						
	1.237154					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	4318	215.9	56649.25263		
PSO	20	10000	500	0		
POLY	20	8360	418	30275.89474		
HYB	20	3592	179.6	15848.25263		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	2230919	5	446183.7133	26.04859117	1.82898E-17	2.293911156
Within Groups	1952695	114	17128.9			
Total	4183613	119				
SCHEFFE'S TEST						
	140.1645					

### Adjiman Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-40.43548	-2.021774	2.29895E-09		
SA	20	-40.13538	-2.006769	0.001939014		
GA	20	-35.57537	-1.7787685	0.006532401		
PSO	20	-28.83175	-1.4415875	0.045489191		
POLY	20	-35.64871	-1.7824355	0.007121065		
HYB	20	-40.4362	-2.02181	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	5.193505093	5	1.038701019	102.030705	2.02948E-40	2.293911156
Within Groups	1.160551778	114	0.010180279			
Total	6.35405687	119				
SCHEFFE'S TEST						
	0.108057057					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	8520	426	19991.05263		
SA	20	9590	479.5	8405		
GA	20	10000	500	0		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	1040	52	414.3157895		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	3150984.167	5	630196.8333	131.2437573	1.34422E-45	2.293911156
Within Groups	547397	114	4801.72807			
Total	3698381.167	119				
SCHEFFE'S TEST						
	74.21164864					

### Bartels Conn Objective Tunction

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	248.21488	12.410744	148.7868805	12.19782278	
SA	20	318.40678	15.920339	196.9263221	14.03304393	
GA	20	33.97332	1.698666	2.624100498	1.619907559	
PSO	20	23069.9544	1153.49772	838371.3859	915.6262261	
POLY	20	58.3618	2.91809	6.619946637	2.572925696	
HYB	20	20.00025	1.0000125	2.63026E-09	5.12861E-05	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	21919448	5	4383889.508	31.36104793	5.73609E-20	2.293911156
Within Groups	15935801	114	139787.7239			
Total	37855248	119				
SCHEFFE'S TEST						
	400.4127					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	10000	500	0	0	
SA	20	10000	500	0	0	
GA	20	6317	315.85	53646.13421	231.6164	
PSO	20	10000	500	0	0	
POLY	20	8628	431.4	22895.51579	151.3126	
HYB	20	5544	277.2	9837.642105	99.18489	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1011314.442	5	202262.8883	14.0494012	1.07E-10	2.293911156
Within Groups	1641206.55	114	14396.54868			
Total	2652520.992	119				
SCHEFFE'S TEST						
	128.4998125					

## The BEALE Objective Function

Anova: Single Factor						
SUMMARY						
<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>	<i>STD</i>	
CK	20	0.08706108	0.004353054	2.64591E-05	0.005143836	
SA	20	0.1150291	0.005751455	3.55581E-05	0.005963062	
GA	20	0.777546	0.0388773	0.002022793	0.044975476	
PSO	20	31.0703028	1.55351514	2.484417358	1.576203463	
POLY	20	2.754056	0.1377028	0.092070831	0.303431757	
HYB	20	0.000089	0.00000445	9.29974E-11	9.64351E-06	
ANOVA						
<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	38.58432764	5	7.716865527	17.95613045	4.2247E-13	2.293911156
Within Groups	48.992887	114	0.429762167			
Total	87.57721463	119				
SCHEFFE'S TEST						
	0.702081273					

Anova: Single Factor						
SUMMARY						
<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>	<i>STD</i>	
CK	20	10000	500	0	0	
SA	20	10000	500	0	0	
GA	20	10000	500	0	0	
PSO	20	10000	500	0	0	
POLY	20	10000	500	0	0	
HYB	20	3256	162.8	19249.01053	138.7408	
ANOVA						
<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	1895064	5	379012.8	118.1399323	2.08E-43	2.293911156
Within Groups	365731.2	114	3208.168421			
Total	2260795	119				
SCHEFFE'S TEST						
	60.65993					

## The BIRD Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	-2129.187	-106.45935	0.088339187	0.297219089	
SA	20	-2127.148	-106.3574	0.140063305	0.374250324	
GA	20	-2134.5	-106.725	0.008304316	0.091128019	
PSO	20	-1573.3125	-78.665625	533.1795981	23.09068206	
POLY	20	-2134.917	-106.74585	0.002407503	0.049066309	
HYB	20	-2135.276	-106.7638	1.01053E-06	0.001005249	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	13017.9	5	2603.579823	29.28558474	5.09685E-19	2.293911156
Within Groups	10134.96	114	88.90311891			
Total	23152.85	119				
SCHEFFE'S TEST						
	10.09792					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	10000	500	0	0	
SA	20	10000	500	0	0	
GA	20	3463	173.15	41401.08	203.4726	
PSO	20	10000	500	0	0	
POLY	20	4725	236.25	49225.14	221.8674	
HYB	20	456	22.8	28.8	5.366563	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	4281640	5	856328.1	56.67605	2.5E-29	2.293911156
Within Groups	1722446	114	15109.17			
Total	6004086	119				
SCHEFFE'S TEST						
	131.6417					

### Bohachevsky 1 Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	22.449307	1.12246535	0.51041125	0.714430717	
SA	20	33.050106	1.6525053	1.198825436	1.094908871	
GA	20	1.42714022	0.071357011	0.024381045	0.156144307	
PSO	20	841.14948	42.057474	967.9726211	31.11225837	
POLY	20	1.95057286	0.097528643	0.039640905	0.199100238	
HYB	20	0.000000002	1E-10	2E-19	4.47214E-10	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	28706.33	5	5741.265881	35.52229095	9.08241E-22	2.293911156
Within Groups	18425.17	114	161.6243133			
Total	47131.5	119				
SCHEFFE'S TEST						
	13.61528					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	10000	500	0	0	
SA	20	10000	500	0	0	
GA	20	6082	304.1	41333.98947	203.3076228	
PSO	20	10000	500	0	0	
POLY	20	10000	500	0	0	
HYB	20	5976	298.8	11434.27368	106.9311633	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1051537	5	210307.3933	23.91294093	2.20353E-16	2.293911156
Within Groups	1002597	114	8794.710526			
Total	2054134	119				
SCHEFFE'S TEST						
	100.4348					



### Bohachevsky 3 Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	13.1775651	0.658878255	0.241792246	0.49172375	
SA	20	32.380988	1.6190494	3.280491338	1.811212671	
GA	20	1.854823335	0.092741167	0.05750863	0.23980957	
PSO	20	876.8378	43.84189	438.2693257	20.93488299	
POLY	20	1.355626115	0.067781306	0.005949545	0.077133291	
HYB	20	0.000074329	3.71645E-06	3.6991E-11	6.08202E-06	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	31364.03	5	6272.806338	85.17914765	7.49381E-37	2.293911156
Within Groups	8395.246	114	73.64251124			
Total	39759.28	119				
SCHEFFE'S TEST						
	9.190467					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	10000	500	0	0	
SA	20	10000	500	0	0	
GA	20	9559	477.95	9724.05	98.6106	
PSO	20	10000	500	0	0	
POLY	20	10000	500	0	0	
HYB	20	7960	398	80	8.944272	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	166509.4	5	33301.875	20.38048052	1.73E-14	2.293911156
Within Groups	186277	114	1634.008333			
Total	352786.3	119				
SCHEFFE'S TEST						
	43.29128					

## The BOOTH Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	0.2791125	0.013955625	0.000246978		
SA	20	0.6944204	0.03472102	0.00172919		
GA	20	0.758061158	0.037903058	0.020742254		
PSO	20	46.024655	2.30123275	4.777250376		
POLY	20	0.449638197	0.02248191	0.001713		
HYB	20	0.00003941	1.9705E-06	4.82269E-12		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	86.6152	5	17.32303928	21.64621483	3.49218E-15	2.293911156
Within Groups	91.23195	114	0.800280299			
Total	177.8472	119				
SCHEFFE'S TEST						
	0.958064					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	9152	457.6	17092.04211		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	8000	400	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	168362.6667	5	33672.53333	11.82042489	3.21E-09	2.293911
Within Groups	324748.8	114	2848.673684			
Total	493111.4667	119				
SCHEFFE'S TEST						
	57.16031991					

## The Branin RCOS 2 Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	-752.1784	-37.60892	2.595382701	1.61101915	
SA	20	-656.3342	-32.81671	1.385618784	1.1771231	
GA	20	-663.3616	-33.16808	12.64434158	3.5558883	
PSO	20	-508.7088	-25.43544	17.57675793	4.19246442	
POLY	20	-591.3664	-29.56832	9.700322721	3.11453411	
HYB	20	-783.7805	-39.189025	0.000223472	0.014949	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	2571.988367	5	514.3976734	70.30068203	3.2879E-33	2.293911156
Within Groups	834.1502966	114	7.317107865			
Total	3406.138664	119				
SCHEFFE'S TEST						
	2.896963528					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	10000	500	0		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	7336	366.8	6152.589474		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	295704	5	59140.8	57.67405765	1.23E-29	2.293911156
Within Groups	116899.2	114	1025.431579			
Total	412603.2	119				
SCHEFFE'S TEST						
	34.29467		HYB & ALL			

## The BRENT Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	0	0	0		
SA	20	0	0	0		
GA	20	121.33776	6.066888	5.187505916		
PSO	20	183.3034	9.16517	81.56954893		
POLY	20	99.57983	4.9789915	2.969227128		
HYB	20	0	0	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1550.33481	5	310.066962	20.73418993	1.10236E-14	2.293911156
Within Groups	1704.799357	114	14.95438033			
Total	3255.134168	119				
SCHEFFE'S TEST						
	4.141498513					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	938	46.9	1178.2		
SA	20	321	16.05	27.83947368		
GA	20	10000	500	0		
PSO	20	9675	483.75	4917.671053		
POLY	20	10000	500	0		
HYB	20	456	22.8	15.32631579		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	6528721.467	5	1305744.293	1276.17181	2.75464E-98	2.293911156
Within Groups	116641.7	114	1023.172807			
Total	6645363.167	119				
SCHEFFE'S TEST						
	34.25687899					

### The Camel 3 Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	0.02507991	0.001253996	1.32163E-06		
SA	20	0.0762615	0.003813075	1.28594E-05		
GA	20	0.001284207	6.42104E-05	1.54594E-08		
PSO	20	5.5450521	0.277252605	0.067162084		
POLY	20	0.004391179	0.000219559	1.10038E-07		
HYB	20	0.000001004	5.02E-08	3.74912E-15		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1.271487642	5	0.254297528	22.71311625	9.35908E-16	2.293911156
Within Groups	1.276351423	114	0.011196065			
Total	2.547839065	119				
SCHEFFE'S TEST						
	0.113319854					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	7005	350.25	44032.82895		
PSO	20	10000	500	0		
POLY	20	9550	477.5	10125		
HYB	20	7456	372.8	7682.694737		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	483322.0417	5	96664.40833	9.378744154	1.6558E-07	2.293911156
Within Groups	1174969.95	114	10306.75395			
Total	1658291.992	119				
SCHEFFE'S TEST						
	108.7262109					

### The Camel 6 Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-20.575	-1.02875	1.02691E-05		
SA	20	-20.5202	-1.02601	3.38176E-05		
GA	20	-20.62509	-1.0312545	2.36904E-06		
PSO	20	-16.38875	-0.8194375	0.023624432		
POLY	20	-20.62802	-1.031401	3.63736E-07		
HYB	20	-20.6326	-1.03163	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	0.738073379	5	0.147614676	37.41618988	1.51277E-22	2.293911156
Within Groups	0.449753785	114	0.003945209			
Total	1.187827164	119				
SCHEFFE'S TEST						
	0.067267965					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	3435	171.75	48308.51316		
PSO	20	10000	500	0		
POLY	20	6066	303.3	51609.48421		
HYB	20	568	28.4	286.1473684		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	4066537.242	5	813307.4483	48.69903039	9.69512E-27	2.293911156
Within Groups	1903878.75	114	16700.69079			
Total	5970415.992	119				
SCHEFFE'S TEST						
	138.4014106					

### The Chichinadze Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-854.3203	-42.716015	0.034757686		
SA	20	-852.7634	-42.63817	0.022081309		
GA	20	-857.4154	-42.87077	0.046660257		
PSO	20	-772.034	-38.6017	17.44265872		
POLY	20	-858.76	-42.938	0.000217371		
HYB	20	-858.888	-42.9444	2.12577E-28		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	298.2912781	5	59.65825563	20.40019815	1.68892E-14	2.293911156
Within Groups	333.3811315	114	2.92439589			
Total	631.6724096	119				
SCHEFFE'S TEST						
	1.831434459					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	4264	213.2	57724.48421		
PSO	20	10000	500	0		
POLY	20	8370	418.5	20984.47368		
HYB	20	1240	62	1434.105263		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	3447744.167	5	689548.8333	51.62384412	1.0131E-27	2.293911156
Within Groups	1522718.2	114	13357.17719			
Total	4970462.367	119				
SCHEFFE'S TEST						
	123.7743466					

### The Cross-in-Tray Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-41.23184	-2.061592	4.61438E-07		
SA	20	-41.21868	-2.060934	3.68666E-06		
GA	20	-41.2522	-2.06261	0		
PSO	20	-38.76471	-1.9382355	0.005218201		
POLY	20	-41.24562	-2.062281	1.47576E-06		
HYB	20	-41.2522	-2.06261	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	0.255358986	5	0.051071797	58.6602294	6.21067E-30	2.293911156
Within Groups	0.099252678	114	0.000870638			
Total	0.354611664	119				
SCHEFFE'S TEST						
	0.031600357					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	9650	482.5	6125		
GA	20	2038	101.9	6797.568421		
PSO	20	10000	500	0		
POLY	20	6770	338.5	33924.68421		
HYB	20	1256	62.8	3417.431579		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	4090970.7	5	818194.14	97.66628234	1.52471E-39	2.293911156
Within Groups	955029	114	8377.447368			
Total	5045999.7	119				
SCHEFFE'S TEST						
	98.023262					



### The Cube Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	0.708874	0.0354437	0.00115818		
SA	20	2.176152	0.1088076	0.012527446		
GA	20	4.6078116	0.23039058	0.043157828		
PSO	20	66.87996	3.343998	6.78616384		
POLY	20	4.417183	0.22085915	0.042561687		
HYB	20	0.00055645	2.78225E-05	1.15149E-09		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	174.2126251	5	34.84252502	30.36134714	1.62595E-19	2.293911156
Within Groups	130.8258106	114	1.14759483			
Total	305.0384357	119				
SCHEFFE'S TEST						
	1.14727516					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	10000	500	0		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	8000	400	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	166666.6667	5	33333.33333	65535	#NUM!	2.293911156
Within Groups	0	114	0			
Total	166666.6667	119				
SCHEFFE'S TEST						
	0					

## The Damavandi Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	30.360738	1.5180369	0.540883502		
SA	20	54.72077	2.7360385	1.052938267		
GA	20	40.00009	2.0000045	1.52368E-10		
PSO	20	70.17623	3.5088115	1.259215246		
POLY	20	40.00602	2.000301	9.29409E-07		
HYB	20	4.37330122	0.218665061	0.041391927		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	124.4730529	5	24.89461057	51.60521072	1.0275E-27	2.293911156
Within Groups	54.99416755	114	0.482404979			
Total	179.4672204	119				
SCHEFFE'S TEST						
	0.743839419					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	10000	500	0		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	8000	400	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	166666.6667	5	33333.33333	65535	#NUM!	2.293911156
Within Groups	0	114	0			
Total	166666.6667	119				
SCHEFFE'S TEST						
	0					

## The Deckkers Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-491312.6	-24565.63	37559.62958		
SA	20	-487629.7	-24381.485	116521.0319		
GA	20	-494872.7	-24743.635	5047.242395		
PSO	20	-359688.2	-17984.41	19211429.33		
POLY	20	-494615.3	-24730.765	8765.293974		
HYB	20	-495530	-24776.5	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	740392648.4	5	148078529.7	45.8463487	9.59387E-26	2.293911156
Within Groups	368207128	114	3229887.088			
Total	1108599776	119				
SCHEFFE'S TEST	1924.717385					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	5888	294.4	43008.98947		
PSO	20	10000	500	0		
POLY	20	7542	377.1	35502.93684		
HYB	20	1568	78.4	3473.515789		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	2826946.567	5	565389.3133	41.37729569	4.22754E-24	2.293911156
Within Groups	1557723.4	114	13664.24035			
Total	4384669.967	119				
SCHEFFE'S TEST	125.1889639					

## The Easom Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-14.996741	-0.74983705	0.032922883		
SA	20	-8.883957	-0.44419785	0.095688493		
GA	20	-16.158748	-0.8079374	0.128700008		
PSO	20	-0.00036	-0.000018	4.58526E-09		
POLY	20	-17.1723474	-0.85861737	0.073812373		
HYB	20	-20	-1	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	13.31010731	5	2.662021461	48.23612988	1.39766E-26	2.293911156
Within Groups	6.291351469	114	0.055187294			
Total	19.60145877	119				
SCHEFFE'S TEST						
	0.251589694					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	6851	342.55	43176.57632		
PSO	20	10000	500	0		
POLY	20	8083	404.15	38684.55526		
HYB	20	3288	164.4	6268.463158		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1776091	5	355218.2	24.18380802	1.6E-16	2.293911156
Within Groups	1674462.3	114	14688.26579			
Total	3450553.3	119				
SCHEFFE'S TEST						
	129.79518					

### The EggCrate Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	0.4044233	0.020221165	0.000649855	0.025492251	
SA	20	1.414664	0.0707332	0.00188674	0.043436616	
GA	20	0.000869089	4.34545E-05	7.5331E-09	8.67935E-05	
PSO	20	36.41302	1.820651	4.712023609	2.170719606	
POLY	20	0.013013349	0.000650667	2.13567E-06	0.001461395	
HYB	20	0.000006416	3.208E-07	1.80635E-13	4.25012E-07	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	54.21401622	5	10.84280324	13.79912168	1.55505E-10	2.293911156
Within Groups	89.57668458	114	0.785760391			
Total	143.7907008	119				
SCHEFFE'S TEST						
	0.949332536					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	6100	305	49121.05263		
PSO	20	10000	500	0		
POLY	20	8740	437	23811.57895		
HYB	20	8000	400	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	612666.6667	5	122533.3333	10.08053575	5.20026E-08	2.293911156
Within Groups	1385720	114	12155.4386			
Total	1998386.667	119				
SCHEFFE'S TEST						
	118.0751799					

### The F-stein Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	8.9822934	0.44911467	0.248755201	0.498753648	
SA	20	10.566518	0.5283259	0.257907203	0.507845649	
GA	20	21.8856253	1.094281265	4.563204661	2.136165879	
PSO	20	266.04813	13.3024065	98.75035892	9.937321516	
POLY	20	34.62513049	1.731256525	12.97881797	3.602612659	
HYB	20	0.00481006	0.000240503	9.46513E-08	0.000307654	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	2657.249636	5	531.4499272	27.3007334	4.45976E-18	2.293911156
Within Groups	2219.181837	114	19.46650734			
Total	4876.431473	119				
SCHEFFE'S TEST						
	4.725168693					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	10000	500	0		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	8000	400	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	166666.6667	5	33333.33333	65535	#NUM!	2.293911156
Within Groups	0	114	0			
Total	166666.6667	119				
SCHEFFE'S TEST						
	0					

## The Goldstien Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	61.30657	3.0653285	0.00438984	0.06625587	
SA	20	61.65316	3.082658	0.006136956	0.078338723	
GA	20	60.377837	3.01889185	0.000989981	0.031463965	
PSO	20	93.5223	4.676115	3.536544004	1.880570127	
POLY	20	61.14961	3.0574805	0.008780943	0.093706686	
HYB	20	60.00027	3.0000135	2.34474E-10	1.53125E-05	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	44.442944	5	8.8885888	14.99406972	2.68003E-11	2.293911156
Within Groups	67.57999275	114	0.592806954			
Total	112.0229368	119				
SCHEFFE'S TEST						
	0.824574583					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	8571	428.55	30451.73421		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	5920	296	22635.78947		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	681513.0417	5	136302.6083	15.40504422	1.48091E-11	2.293911156
Within Groups	1008662.95	114	8847.920614			
Total	1690175.992	119				
SCHEFFE'S TEST						
	100.7381353					

## The Haupt07 Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-368.6178	-18.43089	0.013100481		
SA	20	-297.8059	-14.890295	2.934066536		
GA	20	-363.5067	-18.175335	0.218930458		
PSO	20	-253.3386	-12.66693	4.151312353		
POLY	20	-365.7245	-18.286225	0.110389805		
HYB	20	-371.094	-18.5547	5.31443E-29		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	611.2292937	5	122.2458587	98.74729915	9.19098E-40	2.293911156
Within Groups	141.128193	114	1.237966606			
Total	752.3574867	119				
SCHEFFE'S TEST						
	1.191592507					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	8665	433.25	25233.35526		
PSO	20	10000	500	0		
POLY	20	8547	427.35	31484.66053		
HYB	20	1672	83.6	4436.042105		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	2632738.767	5	526547.7533	51.66111013	9.84919E-28	2.293911156
Within Groups	1161927.1	114	10192.34298			
Total	3794665.867	119				
SCHEFFE'S TEST						
	108.1210647					



## The Haupt 15 Objective Function

Anova: Single Factor						
SUMMARY						
<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>	<i>STD</i>	
CK	20	-6868.465	-343.42325	2.877373987	1.696282402	
SA	20	-6758.875	-337.94375	17.91420336	4.232517378	
GA	20	-6907.137	-345.35685	4.73974E-05	0.006884575	
PSO	20	-5133.362	-256.6681	1731.617687	41.61271064	
POLY	20	-6906.258	-345.3129	0.005590095	0.074766936	
HYB	20	-6907.197	-345.35985	1.34211E-07	0.000366348	
ANOVA						
<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	126424.5161	5	25284.90322	86.57163277	3.62808E-37	2.293911156
Within Groups	33295.88313	114	292.0691502			
Total	159720.3992	119				
SCHEFFE'S TEST						
	18.30274135					

Anova: Single Factor						
SUMMARY						
<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	3659	182.95	49156.78684		
PSO	20	10000	500	0		
POLY	20	5727	286.35	53198.55526		
HYB	20	1736	86.8	18568.58947		
ANOVA						
<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	3368201.267	5	673640.2533	33.42466183	7.07297E-21	2.293911156
Within Groups	2297554.7	114	20153.9886			
Total	5665755.967	119				
SCHEFFE'S TEST						
	152.0385794					

## The Haupt 16 Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-503.9332	-25.19666	0.000758701		
SA	20	-504.1301	-25.206505	0.000239229		
GA	20	-503.1299	-25.156495	0.015789127		
PSO	20	-459.7374	-22.98687	3.032267906		
POLY	20	-500.8998	-25.04499	0.07418491		
HYB	20	-504.61	-25.2305	1.32861E-29		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	79.64768179	5	15.92953636	30.60194605	1.26316E-19	2.293911156
Within Groups	59.34155761	114	0.520539979			
Total	138.9892394	119				
SCHEFFE'S TEST						
	0.772681197					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	9574	478.7	9073.8		
PSO	20	10000	500	0		
POLY	20	9145	457.25	17661.77632		
HYB	20	744	37.2	628.3789474		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	3404065.442	5	680813.0883	149.2795355	2.50691E-48	2.293911156
Within Groups	519915.15	114	4560.659211			
Total	3923980.592	119				
SCHEFFE'S TEST						
	72.32477805					

## The Penholder Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-19.265064	-0.9632532	1.45567E-07		
SA	20	-19.268495	-0.96342475	1.62005E-08		
GA	20	-19.102375	-0.95511875	0.000110112		
PSO	20	-17.381878	-0.8690939	0.002027256		
POLY	20	-19.206067	-0.96030335	5.08039E-05		
HYB	20	-19.2707	-0.963535	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	0.142215749	5	0.02844315	77.98575912	3.7071E-35	2.293911156
Within Groups	0.041578348	114	0.000364722			
Total	0.183794097	119				
SCHEFFE'S TEST						
	0.020452881					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	9330	466.5	10760.78947		
GA	20	10000	500	0		
PSO	20	10000	500	0		
POLY	20	8563	428.15	30801.08158		
HYB	20	1976	98.8	2582.063158		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	2489612.575	5	497922.515	67.67713715	1.65387E-32	2.293911156
Within Groups	838734.75	114	7357.322368			
Total	3328347.325	119				
SCHEFFE'S TEST						
	91.86142786					

### The Rosenbrock Mod Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	695.2752	34.76376	0.424771456		
SA	20	725.4301	36.271505	3.734558768		
GA	20	1365.1013	68.255065	194.3300008		
PSO	20	1510.631	75.53155	1.553354168		
POLY	20	1459.14737	72.9573685	21.7784793		
HYB	20	681.7307	34.086535	0.002739317		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	42126.315	5	8425.263	227.8905795	1.30234E-57	2.293911156
Within Groups	4214.654173	114	36.97065064			
Total	46340.96917	119				
SCHEFFE'S TEST						
	6.51181188					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	10000	500	0		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	8000	400	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	166666.6667	5	33333.33333	65535	#NUM!	2.293911156
Within Groups	0	114	0			
Total	166666.6667	119				
SCHEFFE'S TEST						
	0					

### The Rotated\_ellipes Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	1301.53423	65.0767115	3729.749462	61.0716748	
SA	20	1625.65761	81.2828805	6027.517374	77.6370876	
GA	20	56.37013704	2.818506852	17.94979694	4.23672007	
PSO	20	72637.8719	3631.893595	10935960.82	3306.95643	
POLY	20	224.192295	11.20961475	156.7529441	12.5201016	
HYB	20	0.000771032	3.85516E-05	5.25192E-09	7.247E-05	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	216094538.7	5	43218907.74	23.69047928	2.8735E-16	2.293911156
Within Groups	207971963.1	114	1824315.465			
Total	424066501.7	119				
SCHEFFE'S TEST						
	1446.51609					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	10000	500	0		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	8000	400	0		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	166666.6667	5	33333.33333	65535	#NUM!	2.293911156
Within Groups	0	114	0			
Total	166666.6667	119				
SCHEFFE'S TEST						
	0					

## The Scahffer1 Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance	STD	
CK	20	0.21498762	0.010749381	0.000121256	0.011011611	
SA	20	1.30225962	0.065112981	0.004062579	0.063738368	
GA	20	0.013790696	0.000689535	3.13806E-06	0.001771457	
PSO	20	9.040595	0.45202975	0.010977102	0.104771667	
POLY	20	0.091304112	0.004565206	5.93482E-05	0.00770378	
HYB	20	0	0	0	0	
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	3.226663998	5	0.6453328	254.34468	4.33902E-60	2.293911156
Within Groups	0.289245048	114	0.002537237			
Total	3.515909046	119				
SCHEFFE'S TEST						
	0.053945328					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	5472	273.6	47086.98947		
PSO	20	10000	500	0		
POLY	20	9670	483.5	5445		
HYB	20	6136	306.8	9332.378947		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1143164.967	5	228632.9933	22.17428214	1.8132E-15	2.293911156
Within Groups	1175423	114	10310.72807			
Total	2318587.967	119				
SCHEFFE'S TEST						
	108.7471704					

### The T- tube holder Objective Function

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	-216.9775	-10.848875	0.000307542		
SA	20	-216.7395	-10.836975	0.000697397		
GA	20	-217.2057	-10.860285	9.95129E-05		
PSO	20	-207.7473	-10.387365	0.057071407		
POLY	20	-217.2133	-10.860665	0.000142716		
HYB	20	-217.446	-10.8723	3.32152E-30		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	3.671868804	5	0.734373761	75.55470309	1.47518E-34	2.293911156
Within Groups	1.108052912	114	0.009719762			
Total	4.779921716	119				
SCHEFFE'S TEST						
	0.105584732					

Anova: Single Factor						
SUMMARY						
Groups	Count	Sum	Average	Variance		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	7593	379.65	45739.50263		
PSO	20	10000	500	0		
POLY	20	8501	425.05	24635.31316		
HYB	20	1080	54	2363.789474		
ANOVA						
Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	3009466.867	5	601893.3733	49.64846696	4.61192E-27	2.293911156
Within Groups	1382033.5	114	12123.10088			
Total	4391500.367	119				
SCHEFFE'S TEST						
	117.9180146					

## The Trefethen Objective Function

Anova: Single Factor						
SUMMARY						
<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>		
CK	20	-56.44144	-2.822072	0.077400066		
SA	20	-53.97352	-2.698676	0.07998127		
GA	20	-63.48537	-3.1742685	0.025552876		
PSO	20	-33.1763	-1.658815	0.180158923		
POLY	20	-61.72655	-3.0863275	0.061547481		
HYB	20	-67.62152	-3.381076	3.53545E-05		
ANOVA						
<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	37.45382883	5	7.490765765	105.8326763	3.70578E-41	2.293911156
Within Groups	8.068843451	114	0.070779329			
Total	45.52267228	119				
SCHEFFE'S TEST						
	0.28492235					



Anova: Single Factor						
SUMMARY						
<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>		
CK	20	10000	500	0		
SA	20	10000	500	0		
GA	20	10000	500	0		
PSO	20	10000	500	0		
POLY	20	10000	500	0		
HYB	20	8000	400	0		
ANOVA						
<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	166666.6667	5	33333.33333	65535	#NUM!	2.293911156
Within Groups	0	114	0			
Total	166666.6667	119				
SCHEFFE'S TEST						
	0					

## ADDENDUM B

### Source Code for the Hybrid Optimization

#### Model

```
#ifndef CMAP_H
```

```
#define CMAP_H
```

```
////////////////////////////////////  
////////////////////////////////////
```

```
//
```

```
// Author : Obadan Samiel
```

```
//
```

```
// Special Thanks to: Mat Buckland's Data  
//structures syntax in AI for game programming  
// 2005
```

```
//
```

```
////////////////////////////////////  
////////////////////////////////////
```

```
#include "stdlib.h"
```

```
#include <windows.h>
```

```
#include <vector>
```

```
#include "utils.h"
```

```
#include "defines.h"
```

```
using namespace std;
```

```
class SMap
```

```
{
```

```
public:
```

```
SMap()
```

```
{
```

```
    //Multiirate input
```

```
    double TestFunction(double xin, double  
    yin);
```

```
};
```

```
#endif
```

```
#include " SMap.h"
```

```
double SMap::TestFunction(double xin, double  
yin)
```

```
{
```

```
    // double cost = xin * sin(4*xin) + 1.1 * yin  
    * sin(2* yin);
```

```
    //McCormick function
```

```
    //double cost = sin(xin + yin) + ((xin -  
yin)*(xin - yin))- (1.5* xin) + (2.5 *yin) + 1;
```

```
    //Easom function -100 -> 100
```

```
    // double a = (xin - PI);
```

```
    // double b = (yin - PI);
```

```
    //double cost = -cos(xin) * cos(yin)* exp(-  
( a*a) + (b*b));
```

```
    //Ackley 2 Function
```

```
    //double cost = -200 * exp(-0.02 * sqrt(  
    (xin*xin) + (yin*yin)));
```

```
    //Ackley 3 Function
```

```

// double    cost = -200 * exp(-0.02 * sqrt(
(xin*xin) + (yin*yin))) + 5 * exp(cos(3*xin) +
sin(3*yin));

//Bartels Conn Function

/*double a = (xin*xin) + (yin*yin) + (xin *yin);

double b = sin(xin);

double c = cos(yin);

if (a <0)

{ a *= -1;}

if (b <0)

{ b *= -1;}

if (c <0)

{ c *= -1;}

double cost = a + b + c;*/

// Beale Function -4.5 -> 4.5

// double cost = ((1.5 - xin + (xin*yin))*(1.5 -
xin + (xin*yin))) + ((2.25 - xin +
(xin*yin*yin))*(2.25 - xin + (xin*yin*yin)))+
((2.625 - xin + (xin*yin*yin*yin))*(2.625 - xin +
(xin*yin*yin*yin)));

// Birds Function

// double cost = sin(xin)*exp((1-cos(yin))* (1-
cos(yin))) + cos(yin)*exp((1-sin(xin))* (1-
sin(xin))) + ((xin - yin)*(xin - yin));

//Bohachevsky 1 Function

//double cost = (xin *xin) + 2*(yin *yin) - 0.3 *
cos(3*PI*xin)- 0.4 * cos(4*PI*yin) + 0.7;

//Bohachevsky 3 Function

```

```

//double cost = (xin *xin) + 2*(yin *yin) - 0.3 *
cos((3*PI*xin)+(4*PI*yin)) + 0.3;

//Booth Function

// double j = (2 * xin) + yin -5;

// double k = xin + 2*(yin) - 7;

//double cost = (k*k) + (j*j);

//Brent Function

/* double j = (10 + xin) * (10 + xin);

double k = (10 + yin) * (10 + yin);

double a = xin *xin;

double b = yin * yin;

double cost = j + k + exp(-a-b); */

//Bohachevsky 1 Function

//double cost = (xin *xin) + 2*(yin *yin) - 0.3 *
cos(3*PI*xin)- 0.4 * cos(4*PI*yin) + 0.7;

//Bohachevsky 3 Function

// double cost = (xin *xin) + 2*(yin *yin) - 0.3 *
cos((3*PI*xin)+(4*PI*yin)) + 0.3;

//Booth Function -10 -> 10

// double j = (2 * xin) + yin -5;

// double k = xin + 2*(yin) - 7;

//double cost = (k*k) + (j*j);

//Branin RCOS 2 Function

// double a = yin - ((5.1*xin*xin)/(4* PI*PI)) +
((5*xin)/PI) - 6;

```

```

// double cost = pow(a,2) + 10*(1 -
(1/(8*PI))) * cos(xin) * cos(yin) * log(pow(xin,2) +
pow(yin,2) +1) +10;

```

```

//Camel Function – Six Hump

```

```

//double cost = ((4- (2.1 * pow(xin,2))
+(pow(xin,4)/3))*pow(xin,2)) + (xin *yin) +
((4*pow(yin,2))-4)*pow(yin,2);

```

```

//Camel Function – Three Hump

```

```

//double cost = 2 * pow(xin,2)- 1.05*pow(xin,4)
+ (pow(xin,6)/6) +(xin *yin) + pow(yin,2);

```

```

//Chichinadze Function

```

```

//double a = yin -0.5;

```

```

//double cost = pow(xin,2) - 12*xin +11
+10*cos((PI*xin)/2) +8*sin((5*PI*xin)/2)-
pow(0.2,0.5)*exp(-0.5*pow(a,2));

```

```

///cube Function -10 ->10

```

```

//double a = yin-pow(xin,3);

```

```

//double b = 1-xin;

```

```

//double cost = 100* pow(a,2) + pow(b,2);

```

```

//.....
.....

```

```

//Deckkers-Aarts Function

```

```

double cost = pow(10,5)*pow(xin,2)
+pow(yin,2)-
pow((pow(xin,2)+pow(yin,2)),2)+pow(10,-
5)*pow((pow(xin,2)+pow(yin,2)),4);

```

```

//.....
.....

```

```

//Cross-in-Tray Function -10 -> 10

```

```

//double a = sin(xin)*sin(yin)*exp(100-
(pow((pow(xin,2)+pow(yin,2)),0.5)/PI));

```

```

//double cost = -0.0001*pow((a+1),0.1);

```

```

// % Chen Bird Function -500 ->500

```

```

// double cost = -
(0.001/(pow(0.001,2)+pow((xin-(0.4*yin)-
0.1),2))) - (0.001/(pow(0.001,2)+pow(((2*xin)-
yin-1.5),2)));

```

```

//Branin RCOS 2 Function -5 -> 15

```

```

// double a = yin - ((5.1*xin*xin)/(4* PI*PI)) +
((5*xin)/PI) - 6;

```

```

// double cost = pow(a,2) + 10*(1 -
(1/(8*PI))) * cos(xin) * cos(yin) * log(pow(xin,2) +
pow(yin,2) +1) +10;

```

```

//El-Attar-Vidyasagar-Dutta Function -500-> 500

```

```

//double cost = pow((pow(xin,2) + yin-10),2) +
pow((xin +pow(yin,2)-7),2) + pow((pow(xin,2) +
pow(yin,3)-1),2);

```

```

//Goldstein Price Function -2 -> 2

```

```

//double cost = (1 + pow((xin+yin+1),2))*(19-
(14*xin)+(3*pow(xin,2))-
(14*yin)+(6*xin*yin)+3*pow(yin,2))) * (30 +
pow((2*xin-3*yin),2)*(18-
(32*xin)+(12*pow(xin,2))+48*yin)-
(36*xin*yin)+27*pow(yin,2)));

```

```

//Rosenbrock Modified Function -2 to 2

```

```

//double cost = 74 + (100* pow((yin-
pow(xin,2)),2)) + pow((1-xin),2) - 400*exp(-
1*((pow((xin+1),2) + pow((yin+1),2))/0.1));

```

```

//Camel Function – Six Hump -5 -> 5

```

```

// double cost = ((4- (2.1 * pow(xin,2))
+(pow(xin,4)/3))*pow(xin,2)) + (xin *yin) +
((4*pow(yin,2))-4)*pow(yin,2);

```

```

//Hosaki Function 0 -> 6
//double cost = (1-(8*xin) + 7*pow(xin,2)-
(7/(3*pow(xin,3))) +
(1/(4*pow(xin,4))))*pow(yin,2)*exp(-yin);
//%haupt 15 -5 -> 5
//double cost = -exp(-0.2*sqrt(pow(xin,2) +
pow(yin,2)) + 3*(cos(2*xin) +sin(2*yin)));
//%haupt 16 -20 -> 20
// double a = xin - (yin+9);
// double b = yin + (0.5*xin)+9;
// if (a <0)
// { a *= -1;}
// if (b <0)
// { b *= -1;}
//double cost = -xin* sin(sqrt(a))- (yin +9)*
sin(sqrt(b));
//Ackley 4 orModified -35 -> 35 (summed)
//double cost = exp(-
0.2)*sqrt(pow(xin,2)+pow(yin,2))+ 3*
(cos(2*xin)+sin(2*yin));
// Brown Function -1 -> 4 summed
//double cost = ( pow(xin,2)*(pow(yin,2)+1)) +
((pow(yin,2)+1)*(pow(xin,2)+1));
//Egg Holder Function -512 ->512
/*double a = yin + (xin/2) + 47;
double b= xin - (yin+47);
if (a <0)
{ a *= -1;}
if (b <0)
{ b *= -1;}
double cost = -1*(yin +47)*sin(sqrt(a))-
xin*sin(sqrt(b)); */
//Pathological Function -100 ->100
//double a = sin(sqrt((100*pow(xin,2)) +
pow(yin,2)));
//double b = pow(xin,2)-(2*xin*yin)
+pow(yin,2);
//double cost = 0.5 + ((pow(a,2) -0.5)/
(1+0.001*pow(b,2)));
// test tube holder -10 -> 10
//double cost = -
4*(sin(xin)*cos(yin)*exp(cos((pow(xin,2)+pow(y
in,2))/200)));
return value = 1
return cost;
}
#endif CGA_H
#define CGA_H
#include <vector>
#include <sstream>
#include "defines.h"

```

```

#include " SMap.h"
#include "utils.h"

using namespace std;

//-----
-
// structure
//-----
-
struct SGenome
{
    vector<double> vecBits;
    //....add
    vector<double> vBits;
    double      dFitness;

    double      iRank;

    SGenome():dFitness(0), iRank(0){}

    SGenome(const int
num_bits):dFitness(0), iRank(0)
    {
        //create a random bit string
// the struc craetes a single genome

        for (int i=0; i<num_bits; ++i) //
num_bits will be 2 ( xin,yin)
    }
}

{
    vecBits.push_back(RandomClamped2())
}

//...add
    SGenome(const int n_perm, const int
n_ratio ):dFitness(0), iRank(0) // n_perm = 4
....COUNTER
    {
        //create a random bit string
// the struc craetes a single genome

        vBits.push_back(RandInt(0,5));

        for (int i=1; i<n_perm; ++i) // num bits will
be 4 ( RAND, ratio1, ratio2 ratio3)
        {
            vBits.push_back(RandInt(0,n_ratio)); //
n_ratio = 5 ....for a total of 15 generations max
        }
    }

//overload '<' used for sorting

    friend bool operator<(const SGenome&
lhs, const SGenome& rhs)
    {
        return (lhs.dFitness <
rhs.dFitness);
    }
}

```

```

};
//-----
-
// genetic algorithm
//-----
--
class Cga
{
private:
    //the population of genomes
    vector<SGenome>
    m_vecGenomes;
    vector<SGenome> m_vecGenomesold;

    //.....add
    vector<SGenome>
    m_vecGenomesMajor;

    //size of population
    int m_iPopSize;

    double m_dCrossoverRate;

    double m_dMutationRate;

    //how many bits per chromosome
    int m_iChromoLength;

    //how many bits per gene
    int m_iGeneLength;

    int m_iFittestGenome;

    double m_dBestFitnessScore;
    double m_dbestRank;
    double m_dGlobal_best;
    double m_dDisplaymin;
    double m_dDisplaymin2;
    double m_xinbest;
    double m_yinbest;

    double m_dTotalFitnessScore;

    double m_dAverageFitnessScore;

    int m_iGeneration;
    int m_iGeneration2;

    vector<double> m_vecAvFitness;
    vector<double> m_vecBestFitness;

    int icount0;
    int icount1;

```

```

int icount2;
int icount3;
int icount4;
int icount5;

    SMap    m_Map;

    //lets you know if the current run is in
progress.

    bool        m_bBusy;

int    m_plength;

int    m_ratio;

    void    Mutate(vector<double>
&vecBits);

    //...add

    void    Mutate2(vector<double>
&vBits);

    void    Crossover(const
vector<double> &mum,

        const vector<double> &dad,

        vector<double>    &baby1,

        vector<double>    &baby2);

    SGenome&
    RouletteWheelSelection();

    SGenome    TournamentSelection(int
N);

                                SGenome
TournamentSelection2(int N);

.

void        UpdateFitnessScores();

void
UpdateFitnessScores2();

void
UpdateFitnessScoresMajor();

        bit strings

void
CreateStartPopulation();

string    m_sequence;

double    m_rat1;

double    m_rat2;

double    m_rat3;

public:

        // rem bracket into variable.

        Cga(double cross_rat,

double mut_rat,

int pop_size,

int num_bits,

int p_len,

int
r_ratio):m_dCrossoverRate(cross_rat),

        m_dMutationRate(mut_rat),

```



```

m_iPopSize(pop_size),          {
m_iChromoLength(num_bits),    CreateStartPopulation();
m_plength(p_len),             }
m_ratio(r_ratio),
m_dTotalFitnessScore(0.0),    void          Run(HWND
hwnd);
m_iGeneration(0),             void          Render(int
cxClient, int cyClient, HDC surface);
m_dAverageFitnessScore(0.0),
m_dBestFitnessScore(0.0),
m_dbestRank(0.0),             void          Epoch();
m_dGlobal_best(99999999.0),   void          Epoch2();
m_dDisplaymin(0.0),           void          Epoch3();
m_dDisplaymin2(9999999.0),    void          Epoch4Major();
m_xinbest(0.0),               void          ResetVariables();
m_yinbest(0.0),               void GrabNBest(int    NBest, const int
m_bBusy(false),               NumCopies, vector<SGenome> &vecPop);
m_rat1(0),                     //accessor methods
m_rat2(0),                     int
m_rat3(0),                     Generation(){return m_iGeneration;}
icount0(0),                    int
icount1(0),                    GetFittest(){return m_iFittestGenome;}
icount2(0),                    bool    Started(){return m_bBusy;}
icount3(0),                    void          Stop(){m_bBusy =
icount4(0),                    false;}
icount5(0)                      };
                                #endif
                                #include " Cga.h"

```

```

//-----TournamentSelection-----
//
//-----
SGenome Cga::TournamentSelection(int N)
{
    double BestFitnessSoFar = -999999;
    int ChosenOne = 0;

    for (int i=0; i<N; ++i)
    {
        int ThisTry = RandInt(0, m_iPopSize-1);

        if (m_vecGenomes[ThisTry].iRank >
            BestFitnessSoFar)
        {
            ChosenOne = ThisTry;
            BestFitnessSoFar =
                m_vecGenomes[ThisTry].iRank;
        }
    }

    return m_vecGenomes[ChosenOne];
}

```

```

//.....add.....tournament
selection
//.....
SGenome Cga::TournamentSelection2(int N)
{
    double BestFitnessSoFar = -999999;
    int ChosenOne = 0;

    for (int i=0; i<N; ++i)
    {
        int ThisTry = RandInt(0, m_iPopSize-1);

        if (m_vecGenomesMajor[ThisTry].iRank >
            BestFitnessSoFar)
        {
            ChosenOne = ThisTry;

            BestFitnessSoFar =
                m_vecGenomesMajor[ThisTry].iRank;
        }
    }

    return m_vecGenomesMajor[ChosenOne];
}

//-----RouletteWheelSelection---

```

```
SGenome& Cga::RouletteWheelSelection()
```

```
{  
    double fSlice = RandFloat() *  
m_dTotalFitnessScore;  
  
    double cfTotal = 0.0;  
  
    int SelectedGenome = 0;  
  
    for (int i=0; i<m_iPopSize; ++i)  
    {  
        cfTotal +=  
m_vecGenomes[i].iRank;  
  
        if (cfTotal > fSlice)  
        {  
            SelectedGenome = i;  
            break;  
        }  
    }  
  
    return  
m_vecGenomes[SelectedGenome];  
}  
  
//-----Mutate-----  
  
// iterates through each genome flipping  
the bits according to the  
  
// mutation rate
```

```
void Cga::Mutate(vector<double> &vecBits)
```

```
{  
    for (int curBit=0; curBit<vecBits.size();  
curBit++)  
    {  
        if (RandFloat() <  
m_dMutationRate)  
        {  
            vecBits[curBit] =  
RandomClamped2() * 0.9;  
        }  
    }  
}
```

```
void Cga::Mutate2(vector<double> &vBits)
```

```
{  
    for (int curBit=0; curBit<vBits.size();  
curBit++)  
    {  
        //do we flip this bit?  
        double Rand_result =  
RandFloat();  
        if (Rand_result <  
m_dMutationRate)
```



```

        double R = RandFloat();

        double mold = mum[1];

        double dold = dad[1];

        double  mumNew = mold - R * mold + R *
dold;

        double  dadNew = dold + R * mold - R *
dold;

        baby1.push_back(dad[0]);

        baby2.push_back(mum[0]);

        baby1.push_back(mumNew);

        baby2.push_back(dadNew);

    }
}

//-----Run-----
void Cga::Run(HWND hwnd)
{
    CreateStartPopulation();

    m_bBusy = true;

}

//-----CreateStartPopulation---
void Cga::CreateStartPopulation()
{
        double R = RandFloat();

        //clear existing population
        m_vecGenomes.clear();

        //...add

        m_vecGenomesMajor.clear();

        for (int i=0; i<m_iPopSize; i++)//
m_iPopSize would be from 8-> maybe 20

        {

            m_vecGenomesold.push_back(SGenome
e());

            m_vecGenomes.push_back(SGenome(m_iChro
moLength)); // m_iChromoLength will be 2 ie.
(xin, yin)

        }

        //...add

        for (int i=0; i<POP_SIZE2; i++)

        {

            m_vecGenomesMajor.push_back(SGenome(m_
plength,m_ratio));

        }

        //reset all variables

        m_iGeneration          = 0;

        m_iGeneration2          = 0;

        m_iFittestGenome        = 0;

        m_dBestFitnessScore = 0;

        m_dTotalFitnessScore = 0;

```

```

m_dAverageFitnessScore = 0;
m_dbestRank      =0;
    m_dDisplaymin  =0;
m_dDisplaymin2 = 9999999.0;
icount0 =0;
icount1 =0;
icount2 =0;
icount3 =0;
icount4 =0;
icount5 =0;
}
void Cga::ResetVariables()
{
    m_vecGenomes.clear();

    for (int i=0; i<m_iPopSize; i++)//
m_iPopSize would be from 8-> maybe 20
    {
        m_vecGenomesold.push_back(SGenome());

        m_vecGenomes.push_back(SGenome(m_iChromoLength)); // m_iChromoLength will be 2 ie.
(xin, yin)
    }

        m_dDisplaymin  =0;
    }
//-----Epoch-----
void Cga::Epoch()
{
    UpdateFitnessScores();

    if (!(NUM_COPIES_ELITE * NUM_ELITE
% 2))
    {
        GrabNBest(NUM_ELITE,
NUM_COPIES_ELITE, vecBabyGenomes);
    }

    while (vecBabyGenomes.size() <
m_iPopSize)
    {
        SGenome mum =
TournamentSelection(iTournamentCompetitors
);

        SGenome dad =
TournamentSelection(iTournamentCompetitors
);

```

```

        vector<SGenome> vecBabyGenomes;

        SGenome baby1, baby2;
        Crossover(mum.vecBits,
dad.vecBits, baby1.vecBits, baby2.vecBits);

        //operator - mutate
        Mutate(baby1.vecBits);
        Mutate(baby2.vecBits);

        //add to new population

vecBabyGenomes.push_back(baby1);

vecBabyGenomes.push_back(baby2);
    }

    //copy babies back into starter
population
    m_vecGenomes = vecBabyGenomes;

    //increment the generation counter
    ++m_iGeneration;
}

void Cga::Epoch4Major()
{
    UpdateFitnessScoresMajor();

        vector<SGenome> vecBabyGenomes;

        if (!(NUM_COPIES_ELITE * NUM_ELITE
% 2))
        {
            GrabNBest(NUM_ELITE,
NUM_COPIES_ELITE, vecBabyGenomes);
        }
        while (vecBabyGenomes.size() <
m_iPopSize)
        {
            //select 2 parents

            //SGenome mum =
RouletteWheelSelection();

            //SGenome dad =
RouletteWheelSelection();

            SGenome mum =
TournamentSelection2(iTournamentCompetitor
s);

            SGenome dad =
TournamentSelection2(iTournamentCompetitor
s);

            Mutate2(mum.vBits);

            Mutate2(dad.vBits);

            //add to new population

vecBabyGenomes.push_back(mum);

```

```

        vecBabyGenomes.push_back(dad);
    }

    //copy babies back into starter
population

    m_vecGenomesMajor =
vecBabyGenomes;

    //increment the generation counter

    ++m_iGeneration2;

}

//.....2nd
epoch.....

//.....
..

void Cga::Epoch2()
{

    UpdateFitnessScores();

    vector<SGenome> vecBabyGenomes;

    if (!(NUM_COPIES_ELITE * NUM_ELITE
% 2))
    {

        GrabNBest(NUM_ELITE,
NUM_COPIES_ELITE, vecBabyGenomes);

    }

    while (vecBabyGenomes.size() <
m_iPopSize)
    {

        //select 2 parents

        SGenome mum =
vecBabyGenomes[1];

        //SGenome dad =
RouletteWheelSelection();

        SGenome dad =
TournamentSelection(iTournamentCompetitors
);

        //operator - crossover

        SGenome baby1, baby2;

        Crossover(vecBabyGenomes[1].vecBits,
dad.vecBits, baby1.vecBits, baby2.vecBits);

        //operator - mutate

        Mutate(baby1.vecBits);

        Mutate(baby2.vecBits);

        //add to new population

        vecBabyGenomes.push_back(baby1);

        vecBabyGenomes.push_back(baby2);

    }

    m_vecGenomes = vecBabyGenomes;

    //increment the generation counter

    ++m_iGeneration;

}

void Cga::Epoch3()

```



```

{ UpdateFitnessScores2();

vector<SGenome> vecBabyGenomes;

if (!(NUM_COPIES_ELITE * NUM_ELITE
% 2))
{
    GrabNBest(NUM_ELITE,
NUM_COPIES_ELITE, vecBabyGenomes);
}

int z =2;

while (vecBabyGenomes.size() <
m_iPopSize)
{
    if ( m_vecGenomes[z].dFitness <
m_vecGenomesold[z].dFitness)
    {
vecBabyGenomes.push_back(m_vecGenomes[z
]);
    }

    else if (RandFloat() > 0.50)// if true, then the
nest is not discovered ..perform randomwalk
    {
vecBabyGenomes.push_back(SGenome());

for (int j=0; j<m_iChromoLength; ++j)
    {
double arg =
m_vecGenomesold[z].vecBits[j] +
(RandomClamped2() * 0.9);

vecBabyGenomes[z].vecBits.push_back( Clamp(a
rg,-20,20));// (arg, min, max)
    }
}

else
{
vecBabyGenomes.push_back(SGenome());

for (int j=0;
j<m_iChromoLength; ++j)
    {
vecBabyGenomes[z].vecBits.push_back(Rando
mClamped2());
    }
}

z++;

m_vecGenomesold = m_vecGenomes;

m_vecGenomes = vecBabyGenomes;

//increment the generation counter
++m_iGeneration;
}

void Cga::UpdateFitnessScores2()
{
m_iFittestGenome = 0;
}

```

```

        m_dTotalFitnessScore = 0;

m_dDisplaymin =0;
double Global_best =0;
m_dbestRank = 0;
int counter = 50;

        double total = 0;
        vector<double> vel;
        vel.clear();

        for (int i=0; i<m_iPopSize; ++i)
        {
                //get it's fitness score

                m_vecGenomes[i].dFitness =
m_BobsMap.TestFunction(m_vecGenomes[i].ve
cBits[0], m_vecGenomes[i].vecBits[1]); // for
(xin, yin).. each chromosome

                if (m_vecGenomes[i].dFitness <=
m_dGlobal_best)
                {
                        m_xinbest =
m_vecGenomes[i].vecBits[0];

                        m_yinbest =
m_vecGenomes[i].vecBits[1];

                        m_dBestFitnessScore =
m_vecGenomes[i].dFitness;
                }

                total += m_vecGenomes[i].dFitness;
        }

        sort(m_vecGenomes.begin(),
m_vecGenomes.end());

        //m_dBestFitnessScore =
m_vecGenomes[0].dFitness;

        m_dAverageFitnessScore = total/m_iPopSize;

double static global= 0;

        if (global == 0 )
        {
                global = m_dBestFitnessScore;

                m_dGlobal_best = global;

                m_dDisplaymin = global;
        }

                else if (m_dBestFitnessScore < global)
                {

                        global = m_dBestFitnessScore;

                        m_dGlobal_best = global;

                        m_dDisplaymin = global;
                }

        else
        {

```

```

    m_dGlobal_best = global;
    m_dDisplaymin = global;
}

// make fitnesses gaussian
for (int k=0; k<m_iPopSize; ++k)
    {
    m_vecGenomes[k].dFitness /= total;
    }
/*
    if (m_dAverageFitnessScore< 0)
    {
    m_dAverageFitnessScore *= -1;
    }

    if (m_dBestFitnessScore< 0)
    {
    m_dBestFitnessScore *= -1;
    }

    m_vecAvFitness.push_back(fabs(m_dAverageFitnessScore));

    m_vecBestFitness.push_back(fabs(m_dBestFitnessScore));

    */
}

```

```

void Cga::UpdateFitnessScores()
{
    m_iFittestGenome = 0;
    // m_dBestFitnessScore = 0;
    m_dTotalFitnessScore = 0;
    // m_dAverageFitnessScore = 0;
    m_dDisplaymin = 0;
    m_dbestRank = 0;
    int counter = 50; // set to
    chromosize.....REMEMBER.....
    ....
    double total = 0;

    for (int i=0; i<m_iPopSize; ++i)
    {
        //get it's fitness score
        m_vecGenomes[i].dFitness =
        m_BobsMap.TestFunction(m_vecGenomes[i].vecBits[0], m_vecGenomes[i].vecBits[1]); // for
        (xin, yin).. each chromosome

        if (m_vecGenomes[i].dFitness <=
        m_dGlobal_best)
        {
            m_xinbest =
            m_vecGenomes[i].vecBits[0];

            m_yinbest =
            m_vecGenomes[i].vecBits[1];
        }
    }
}

```

```

    total += m_vecGenomes[i].dFitness;
    //
    ResetVariables();
}

//sort the population (for scaling and elitism)
sort(m_vecGenomes.begin(),
     m_vecGenomes.end());
for (int i=0; i<m_iPopSize; ++i)
    {
        m_vecGenomes[i].iRank = 2*counter--;

        m_dTotalFitnessScore +=
m_vecGenomes[i].iRank;
    }
for (int i=0; i<m_iPopSize; ++i)
    {

if (m_vecGenomes[i].iRank >= m_dbestRank)
        {
            m_dBestFitnessScore =
m_vecGenomes[i].dFitness;

            m_dbestRank =
m_vecGenomes[i].iRank;// useful if elitism was
used.

        }
    }
}

```

```

    if (m_dBestFitnessScore <=
m_dGlobal_best)// extream large used here
        {
            m_dGlobal_best = m_dBestFitnessScore;
        }
m_dBestFitnessScore = m_dGlobal_best;
m_dDisplaymin = m_dGlobal_best;
}
//.....add.....
void Cga::UpdateFitnessScoresMajor()
{
    vector<SGenome> vecTempGenomes =
m_vecGenomesMajor;
    int iterator=0;
    while ( iterator != (POP_SIZE2-1))
        {
            //int duration=
m_vecGenomesMajor[iterator].vBits[1]
+m_vecGenomesMajor[iterator].vBits[2]
+m_vecGenomesMajor[iterator].vBits[3];
            if( vecTempGenomes[iterator].vBits[0] == 0)
                {
                    for (int i = 0; i <
vecTempGenomes[iterator].vBits[1]; ++i)
                        { Epoch();}
                    for (int j = 0; j <
vecTempGenomes[iterator].vBits[2]; ++j)

```

```

    { Epoch2();}
        for (int k = 0; k <
vecTempGenomes[iterator].vBits[3]; ++k)
    { Epoch3();}
ResetVariables();

}

else if (
vecTempGenomes[iterator].vBits[0] == 1)
{
    for (int i = 0; i <
vecTempGenomes[iterator].vBits[1]; ++i)
    { Epoch();}
    for (int j = 0; j <
vecTempGenomes[iterator].vBits[2]; ++j)
    { Epoch3();}
        for (int k = 0; k <
vecTempGenomes[iterator].vBits[3]; ++k)
    { Epoch2();}
ResetVariables();

} // end if for GA, CK, POLY ---1

else if (
vecTempGenomes[iterator].vBits[0] == 2)
{
    for (int i = 0; i <
vecTempGenomes[iterator].vBits[1]; ++i)
    { Epoch2();}

```

```

    for (int j = 0; j <
vecTempGenomes[iterator].vBits[2]; ++j)
    { Epoch();}
        for (int k = 0; k <
vecTempGenomes[iterator].vBits[3]; ++k)
    { Epoch3();}
ResetVariables();

} // end if for POLY, GA, CK, ---2

else if (
vecTempGenomes[iterator].vBits[0] == 3)
{
    for (int i = 0; i <
vecTempGenomes[iterator].vBits[1]; ++i)
    { Epoch2();}
    for (int j = 0; j <
vecTempGenomes[iterator].vBits[2]; ++j)
    { Epoch3();}
        for (int k = 0; k <
vecTempGenomes[iterator].vBits[3]; ++k)
    { Epoch();}
ResetVariables();

} // end if for POLY, CK, GA ---3

else if (
vecTempGenomes[iterator].vBits[0] == 4)
{
    for (int i = 0; i <
vecTempGenomes[iterator].vBits[1]; ++i)

```

```

    { Epoch3();}

    for (int j = 0; j <
vecTempGenomes[iterator].vBits[2]; ++j)

    { Epoch();}

        for (int k = 0; k <
vecTempGenomes[iterator].vBits[3]; ++k)

        { Epoch2();}

        ResetVariables();

    } // end if for CK, GA, POLY ---4

    else

    {

        for (int i = 0; i <
vecTempGenomes[iterator].vBits[1]; ++i)

        { Epoch3();}

        for (int j = 0; j <
vecTempGenomes[iterator].vBits[2]; ++j)

        { Epoch2();}

        for (int k = 0; k <
vecTempGenomes[iterator].vBits[3]; ++k)

        { Epoch();}

        ResetVariables();

    } // end if for CK,POLY, GA ---5

    // vecTempGenomes[iterator].dFitness=
m_dDisplaymin;

```

```

        vecTempGenomes[iterator].dFitness=
m_dBestFitnessScore;

        iterator++;

    } // close while

    m_vecGenomesMajor = vecTempGenomes;

    //sort the population (for scaling and elitism)

    sort(m_vecGenomesMajor.begin(),
m_vecGenomesMajor.end());

    // if ( m_vecGenomesMajor[0].dFitness <
m_dDisplaymin2)

    m_dDisplaymin2 =
m_vecGenomesMajor[0].dFitness;

    if( m_vecGenomesMajor[0].vBits[0] == 0)

    {

        m_sequence = "GA, POLY, CK";

        m_rat1 =
m_vecGenomesMajor[0].vBits[1];

        m_rat2 =
m_vecGenomesMajor[0].vBits[2];

        m_rat3 =
m_vecGenomesMajor[0].vBits[3];

        icount0++;

    }

    else if( m_vecGenomesMajor[0].vBits[0] ==
1)

    {

```

```

        m_sequence = "GA, CK, POLY";
        m_rat1 =
m_vecGenomesMajor[0].vBits[1];

        m_rat2 =
m_vecGenomesMajor[0].vBits[2];

        m_rat3 =
m_vecGenomesMajor[0].vBits[3];

        icount1++;
    }
    else if( m_vecGenomesMajor[0].vBits[0] ==
2)
    {
        m_sequence = "POLY, GA, CK";
        m_rat1 =
m_vecGenomesMajor[0].vBits[1];

        m_rat2 =
m_vecGenomesMajor[0].vBits[2];

        m_rat3 =
m_vecGenomesMajor[0].vBits[3];

        icount2++;
    }
    else if( m_vecGenomesMajor[0].vBits[0] ==
3)
    {
        m_sequence = "POLY, CK, GA";
        m_rat1 =
m_vecGenomesMajor[0].vBits[1];

        m_rat2 =
m_vecGenomesMajor[0].vBits[2];

        m_rat3 =
m_vecGenomesMajor[0].vBits[3];

        icount3++;
    }
    else if( m_vecGenomesMajor[0].vBits[0] ==
4)
    {
        m_sequence = "CK, GA, POLY";
        m_rat1 =
m_vecGenomesMajor[0].vBits[1];

        m_rat2 =
m_vecGenomesMajor[0].vBits[2];

        m_rat3 =
m_vecGenomesMajor[0].vBits[3];

        icount4++;
    }
    else
    {
        m_sequence = "CK,POLY, GA";
        m_rat1 =
m_vecGenomesMajor[0].vBits[1];

        m_rat2 =
m_vecGenomesMajor[0].vBits[2];

        m_rat3 =
m_vecGenomesMajor[0].vBits[3];

        icount5++;
    }
}

//-----elitism

```

```

void Cga::GrabNBest(int NBest, const int
NumCopies, vector<SGenome> &vecPop)
{
    //add the required amount of copies of
the n most fittest

    //to the supplied vector
    while(NBest--)
    {
        for (int i=0; i<NumCopies; ++i)
        {
            vecPop.push_back(m_vecGenomes[(3 -
1) - NBest]);
        }
    }
}

```

```

void Cga::Render(int cxClient, int cyClient, HDC
surface)
{
    string s = "Best Fitness: " +
ftos(m_dBestFitnessScore);

    TextOut(surface, 5, 20, s.c_str(),
s.size()); // ( x, y) arrangement

    s = "Average Fitness: " +
ftos(m_dAverageFitnessScore);

    TextOut(surface, 5, 40, s.c_str(),
s.size());

```

```

s = "Generation: " + itos(m_iGeneration2);

    TextOut(surface, 5, 0, s.c_str(), s.size());

    s = "wining sequence " + m_sequence;

    TextOut(surface, 180, 20, s.c_str(),
s.size());

    s = " ratio 1= " + ftos(m_rat1);

    TextOut(surface, 220, 40, s.c_str(),
s.size());

    s = " ratio 2= " + ftos(m_rat2);

    TextOut(surface, 220, 60, s.c_str(),
s.size());

    s = " ratio 3= " + ftos(m_rat3);

    TextOut(surface, 220, 80, s.c_str(),
s.size());

    s = "Minimun Cost: " +
ftos(m_dDisplaymin2);

    TextOut(surface, 5, 60, s.c_str(),
s.size());

    s = "GA, POLY, CK = " + itos(icontains0);

    TextOut(surface, 5, 120, s.c_str(),
s.size());

```



```

s = "GA, CK, POLY = " + itos(icontains1);
TextOut(surface, 5, 140, s.c_str(),
s.size());

```

```

s = "POLY, GA, CK = " + itos(icontains2);
TextOut(surface, 5, 160, s.c_str(),
s.size());

```

```

s = "POLY, CK, GA = " + itos(icontains3);
TextOut(surface, 5, 180, s.c_str(),
s.size());

```

```

s = "CK, GA, POLY = " + itos(icontains4);
TextOut(surface, 5, 200, s.c_str(),
s.size());

```

```

s = "CK,POLY, GA = " + itos(icontains5);
TextOut(surface, 5, 220, s.c_str(),
s.size());

```

```

// s = "Simulated activity graph below ";
// TextOut(surface, 5, 100, s.c_str(),
s.size());

```

```

s = " x = " + ftos(m_xinbest);
TextOut(surface, 150, 0, s.c_str(),
s.size());

```

```

s = " y = " + ftos(m_yinbest);
TextOut(surface, 290, 0, s.c_str(),
s.size());

```

```

/* //render the graph
float HSlice = (float)400/(m_iGeneration+1);
float VSlice =
(float)400/((m_dBestFitnessScore+1) * 1.5);

```

```
HPEN OldPen;
```

```
HPEN m_RedPen = CreatePen(PS_SOLID, 2,
RGB(255,0,0));
```

```
HPEN m_BluePen = CreatePen(PS_SOLID, 2,
RGB(0,0,255));
```

```
HPEN m_GreenPen = CreatePen(PS_SOLID, 2,
RGB(0,255,0));
```

```
//plot the graph for the best fitness
```

```
float x = 0;
```

```
OldPen = (HPEN)SelectObject(surface,
m_RedPen);
```

```
MoveToEx(surface, (int)0, (int)400, NULL);
```

```
for (int i=0; i<m_vecBestFitness.size(); ++i)
{

```

```

    LineTo(surface, (int)x, (int)(400 -
VSlice*m_vecBestFitness[i]));

    x += HSlice;
}

//plot the graph for the average fitness
x = 0;
SelectObject(surface, m_BluePen);

MoveToEx(surface, (int)0, (int)400, NULL);

for (int i=0; i<m_vecAvFitness.size(); ++i)
{
    LineTo(surface, (int)x, (int)(400 -
VSlice*m_vecAvFitness[i]));

    x += HSlice;
}

//replace the old pen
SelectObject(surface, OldPen);
DeleteObject(m_BluePen);
DeleteObject(m_RedPen);
DeleteObject(m_GreenPen);    */

//.....
.....

    if (!m_bBusy)

```

```

    {
        string Start = "Press Return to
start a new run";

        TextOut(surface, cxClient/2 -
(Start.size() * 3), cyClient - 20, Start.c_str(),
Start.size());
    }
else
{
    string Start = "Space to stop";

    TextOut(surface, cxClient/2 -
(Start.size() * 3), cyClient - 20, Start.c_str(),
Start.size());
}

}

#endif UTILS_H
#define UTILS_H

#include <stdlib.h>
#include <math.h>
#include <sstream>
#include <string>
#include <iostream>

//#include <vector>

```

```

using namespace std;

inline int      RandInt(int x,int y) {return
rand()%(y-x+1)+x;}

//returns a random float between zero and 1

inline double RandFloat()
{return (rand()/(RAND_MAX+1.0));}

//returns a random bool

inline bool  RandBool()
{
    if (RandInt(0,1)) return true;
    else return false;
}

//returns a random float in the range -1 < n < 1

inline double RandomClamped()
{return RandFloat() - RandFloat();}

//inline double RandomClamped1()
{return RandFloat()*4 -
RandFloat()*1.5;}//.....X

inline double RandomClamped2()
{return (RandFloat()*20 - RandFloat()*20);}//
range is(-3 -> 4).....Y

//-----
-----

//

//    some handy little functions

//-----
-----

```

```

//converts an integer to a std::string

string itos(int arg);

//converts an float to a std::string

string ftos (float arg);

//    clamps the first argument between the
second two

double Clamp(double &arg, float min, float
max);

void Clamp(int &arg, int min, int max);

//-----
-----

//

// the structure used to define a vertex

//-----
-----

struct SPoint
{
    double x, y;

    SPoint(double a = 0, double b = 0):x(a),y(b){}
};

#endif

#include <windows.h>

#pragma comment(lib, "winmm.lib") //Tools-
>compiler options, type: -lwinmm .for add
linker command

```

## MAIN CODE SEGMENT

```
#include <windows.h>

#include <stdlib.h>

#include <time.h>

#include "Cga.h"

#include "SMap.h"

#include "defines.h"

#include "resource.h"

/* Declare Windows procedure */
LRESULT CALLBACK WindowProcedure (HWND,
UINT, WPARAM, LPARAM);

//pointer to the GA object
Cga* g_pGA;

/* Make the class name into a global variable
*/
char szClassName[ ] = "WindowsApp";

int WINAPI WinMain (HINSTANCE
hThisInstance,
HINSTANCE hPrevInstance,
LPSTR lpszArgument,
int nFunsterStil)
{
    HWND hwnd; /* This is the handle for
our window */

    MSG messages; /* Here messages to
the application are saved */

    WNDCLASSEX wincl; /* Data structure for
the windowclass */

    //MessageBox(NULL, "hello world", "MsgBox",
MB_OK | MB_ICONASTERISK);

    //return 0;

    /* The Window structure */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure;
    /* This function is called by windows */
    wincl.style = CS_DBLCLKS; /* Catch
double-clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Use default icon and mouse-pointer */
    wincl.hIcon = LoadIcon(hThisInstance,
MAKEINTRESOURCE(IDI_ICON_LRG)); //LoadIcon
(NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon(hThisInstance,
MAKEINTRESOURCE(IDI_ICON_SM));
    wincl.hCursor = LoadCursor (NULL,
IDC_ARROW);
    wincl.lpszMenuName = NULL; /* No
menu */
}
```

```

    wincl.cbClsExtra = 0;          /* No extra
bytes after the window class */

    wincl.cbWndExtra = 0;        /*
structure or the window instance */

    /* Use Windows's default color as the
background of the window */

    wincl.hbrBackground =
(HBRUSH)GetStockObject
(WHITE_BRUSH); //(HBRUSH)
COLOR_BACKGROUND;

    /* Register the window class, and if it fails
quit the program */

    if (!RegisterClassEx (&wincl))

        return 0;

    /* The class is registered, let's create the
program*/

    if (!(hwnd = CreateWindowEx (

        0,          /* Extended possibilites for
variation */

        szClassName, /* Classname */

        "Function-Genetic Algorithm", /* Title
Text */

        WS_OVERLAPPED|WS_VISIBLE|WS_CAPTION|
WS_SYSMENU, /* default window ...ADDDDED BY
ME */

        CW_USEDEFAULT, /* Windows
decides the position.. ininitial x position */

```

```

        CW_USEDEFAULT, /* where the
window ends up on the screen..ininitial y
position */

        400,      /* The programs width
..ininitial x size */

        400,      /* and height in pixels
..ininitial y size*/

        HWND_DESKTOP, /* The window is a
child-window to desktop */

        NULL,      /* No menu ....ADDRESSED
later in chapter.....hMenu*/

        hThisInstance, /* Program Instance
handler */

        NULL      /* No Window Creation
data */

        )))

    return 0;

    /* Make the window visible on the screen */

    ShowWindow (hwnd, nFunsterStil);

    /* Run the message loop. It will run until
GetMessage() returns 0 */

    // the differnce in creating a looping main
program

    // Enter the message loop

    bool bDone = false;

    while(!bDone)

    {

```

```

//this will call
WM_PAINT which will render our scene
InvalidRect(hwnd,
NULL, TRUE);
UpdateWindow(hwnd);
Sleep(200);
}/** your game loop goes here **/
} //end while
//*****
*****
// while (GetMessage (&messages, NULL, 0, 0))
//{
/* Translate virtual-key messages into
character messages */
//TranslateMessage(&messages);
/* Send message to WindowProcedure */
// DispatchMessage(&messages);
//}
/* The program return-value is 0 - The value
that PostQuitMessage() gave */
//*****
*****
return messages.wParam;
}
/* This function is called by the Windows
function DispatchMessage() */
while( PeekMessage(
&messages, NULL, 0, 0, PM_REMOVE ) )
{
if( messages.message
== WM_QUIT )
{
// Stop loop if
it's a quit message
bDone = true;
}
else
{
TranslateMessage(&messages);
DispatchMessage( &messages );
}
}
//if the user has started the run update the GA
and display
//accordingly
if (g_pGA->Started())
{
//update the gun
g_pGA->Epoch4Major();
}

```

```

LRESULT CALLBACK WindowProcedure (HWND
hwnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
//device context for our window
HDC          hdc;
PAINTSTRUCT ps;

//these hold the dimensions of the
client window area
static int cxClient, cyClient;
/*-----
--
//used to create the back buffer
static HDC          hdcBackBuffer;
static HBITMAP      hBitmap;
static HBITMAP      hOldBitmap;
-----
-- */
switch (message) /* handle the
messages */
{
//to get get the size of the client window
first we need to create
//a RECT and then ask Windows to fill in
our RECT structure with
//the client window size. Then we assign to
cxClient and cyClient
//accordingly
case WM_CREATE:
{
//seed the random
number generator
srand((unsigned)
time(NULL));

//create the GA class
g_pGA = new
Cga(CROSSOVER_RATE,
MUTATION_RATE,
POP_SIZE,
CHROMO_LENGTH,
P_LENGTH,
R_RATIO);

//get the size of the
client window
RECT rect;
GetClientRect(hwnd,
&rect);
cxClient = rect.right;
cyClient = rect.bottom;
}
}
}

```

```

}
break;
//...BitBlt(hdcBackBuffer, 0, 0, cxClient,
cyClient, NULL, NULL, NULL, WHITENESS);

case WM_DESTROY:
//render the map and
best route
{
//..SelectObject(hdcBackBuffer,
g_pGA-
hOldBitmap);
>Render(cxClient, cyClient, hdc);

//clean up our
backbuffer objects
//now blit backbuffer to
front

//...DeleteDC(hdcBackBuffer);
//...BitBlt(hdc, 0, 0,
cxClient, cyClient, hdcBackBuffer, 0, 0,
SRCCOPY);

//....DeleteObject(hBitmap);
ReleaseDC(hwnd, hdc);

//delete our GA object
delete g_pGA;
EndPoint(hwnd, &ps);
}

// kill the application,
this sends a WM_QUIT message
break;
PostQuitMessage(0);

//has the user resized the client area?....all
new appart form the 1st 2 lines
}
break;
case WM_SIZE:
{
//if so we need to update our variables so
that any drawing
//we do using cxClient and cyClient is
scaled accordingly
cxClient =
LOWORD(IParam);

//fill our backbuffer
with white

```



```

        cyClient =
HIWORD(IParam);
    }
    break;
//check key press messages
    case WM_KEYUP:
    {
        switch(wParam)
        {
            case
            VK_RETURN:
            {
                g_pGA->Stop();
                break;
            }
            default: /* for messages that
we don't deal with */
            {
                return DefWindowProc (hwnd, message,
wParam, IParam);
            }
        }
    }
    break;
    case
    VK_ESCAPE:
    {
        return 0;
    }

    PostQuitMessage(0);
}

```