

**DEVELOPMENT OF A SMELL AGENT OPTIMIZATION ALGORITHM FOR
COMBINATORIAL OPTIMIZATION PROBLEMS**

BY

Ahmed Tijani SALAWUDEEN

**DEPARTMENT OF COMPUTER ENGINEERING
FACULTY OF ENGINEERING
AHMADU BELLO UNIVERSITY
ZARIA, NIGERIA**

MAY, 2018

**DEVELOPMENT OF A SMELL AGENT OPTIMIZATION ALGORITHM FOR
COMBINATORIAL OPTIMIZATION PROBLEMS**

BY

**Ahmed Tijani SALAWUDEEN, B.Eng 2011, MSc (ABU) 2015
P15EGCP9006
tasalawudeen@abu.edu.ng**

**A DISSERTATION SUBMITTED TO THE SCHOOL OF POSTGRADUATE
STUDIES, AHMADU BELLO UNIVERSITY, ZARIA
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD
OF THE DOCTOR OF PHILOSOPHY (Ph.D) DEGREE IN CONTROL
ENGINEERING**

**DEPARTMENT OF COMPUTER ENGINEERING
FACULTY OF ENGINEERING
AHMADU BELLO UNIVERSITY,
ZARIA, NIGERIA**

MAY, 2018

DECLARATION

I, Ahmed Tijani SALAWUDEEN hereby declare that the work in this thesis entitled **Development of a Smell Agent Optimization Algorithm for Combinatorial Optimization Problems**” has been carried out by me in the Department of Computer Engineering. The information derived from literature has been duly acknowledged in the text and a list of references provided. No part of this dissertation was previously presented for another degree or diploma at this or any other institution.

Ahmed Tijani SALAWUDDEN

Signature

Date

CERTIFICATION

This Thesis entitled Development of a Smell Agent Optimization (SAO) Algorithm For Combinatorial Optimization Problems by Ahmed Tijani SALAWUDEEN meets the regulations governing the award of the degree of Doctor of Philosophy (Ph.D) in Control Engineering of the Ahmadu Bello University and is approved for its contribution to knowledge and literary presentation.

Professor M. B. Mu'azu
(Chairman, Supervisory Committee)

Signature

Date

Dr. Y. A Sha'aban
(Member, Supervisory Committee)

Signature

Date

Dr. E. A. Adedokun
(Member, Supervisory Committee)

Signature

Date

Professor M. B. Mu'azu
(Head of Department)

Signature

Date

Professor S. Z. Abubakar
(Dean, School of Postgraduate Studies)

Signature

Date

DEDICATION

This research is dedicated to kind hearted people, promoters of peace and those who in one way or the other make a significant difference in an ordinary person life.

ACKNOWLEDGEMENT

In the name of Allah (SWT), the most beneficent, the most merciful. All praises are due to Allah (SWT), the creator of heaven and earth, the Lord of the universe. Peace and blessings are upon His prophet, Muhammad (SAW) and his companions as well as those who follow the right path until the day of judgement. Foremost, I would like to express my sincere gratitude and thanks to my role model supervisor Prof. M. B. Mu'azu for his patience, motivation, enthusiasm, and immense knowledge. His immeasurable guidance helped me shape the research problem, and constantly provide me insight towards actualizing the research and writing of the thesis. His strong belief in academics and research activities will surely be rewarded by God. I wouldn't have imagined having a better supervisor and a mentor for my PhD study Sir. My earnest gratitude goes to my co-supervisors Dr. Y. A. Sha'aban for his touch of expertise, advice and wonderful corrections without which this work would not have made it to this point and Dr. E. A. Adedokun for bringing the best out of me and making me realize what I can achieve within a short time. I feel so lucky and honoured having such a wonderful combination of the supervisory committee. To my team mates in computer and control research group, your contributions are truly appreciated. I will also like to thank the Nigerian Liquefied Natural Gas (NLNG) and Prof. Yaro, for providing me access to the mechatronics axes of the multiuser Lab where I carried out my simulations.

The contributions of Prof. B. G. Bajoga, Prof. B. Jimoh, Dr Y. Jibril, Dr A. D. Usman, Dr T. H. Sikiru, Dr I. J. Umoh, Dr I. A. Bello, Dr M. B. Abdulrazak and Dr A. M. S. Tekanyi towards ensuring the success of this research work will never be forgotten. I'm highly grateful sirs. To the entire staff of Computer, Electrical and Communication

Engineering whose names could not be mentioned, I am highly grateful for all your contributions to my academic career.

I am highly grateful to my parents Mr and Mrs Salawudeen for their moral, financial and spiritual support throughout my life thus far. I also wish to thank my brothers, Mall. Abdulrahman and Dr. Hameed, for their unending love and financial support throughout my education. My sisters, Hauwa, Fatima, Khadija and Amina, the care and love showered on me will forever be appreciated.

To my friends and colleagues whom I called brothers, Kashman, M.J Atta, Bashoo, Ansari, Cyrus, Obari, Shehu, Basira, Yusuf, Zainab, Adams, Smog, Kunya, Ya'u, Musaib, Atuman, Ajayi, Suraj, Zion, Valentine and those whose names could not be mentioned I am grateful for all we shared.

Finally, I will like to thank my wonderful Queen Hussenat Salawudeen for her understanding, love and care. To my son Farees and my daughter Baiza (Mama) who always puts a smile on my face and keep me young and Zeenat who makes the home lively thank you all for your understanding and love. Daddy loves you all.

A. T. Salawudeen
May, 2018

ABSTRACT

This thesis presents the development of smell agent optimization (SAO) algorithm. The developed algorithm consists of three modes (sniffing, trailing and random modes). The evaporation of smell molecules from the smell source is modelled into sniffing mode using the concept of the hydrostatic pressure of gas and positions of molecules. The fitness of the sniffing mode is evaluated and the molecule with the most favourable fitness is taken as the agent. The olfaction capacity of the agent is then evaluated and the training mode is developed using the current position of the agent and the position of the molecules with the current worst fitness. In practical scenarios, it is usually difficult for the agent to account for all the evaporating smell molecules due to the Brownian nature of the smell molecules. This is largely responsible for the agent to getting trapped in a “state of confusion” and consequently leading to the loss of smell trail. To account for this situation in the SAO, a random mode which allows the agent to take a random step in the search space is modelled. The agent evaluates the fitness of the random mode and decides whether to continue its trailing process or to start the entire process of the SAO all over again. This process continues until the object (optimum result) generating the smell is identified. The performance of the developed SAO was evaluated using a total of thirty-nine (39) optimization benchmark functions. Simulations were performed using MATLAB R2017a and results were compared with the results obtained using the fruit fly optimization algorithm (FFOA) and gaseous Brownian motion optimization (GBMO). Results showed that the SAO obtained the best results in twenty-two (22) functions (56.41%) while the FFOA and GBMO obtained the best results in four (4) and seven (7) (10.26% and 17.95%) functions respectively. However, there were similar results in six (6) of the functions (15.38%). The convergence rate of the algorithms was also compared and results showed that the FFOA converged faster than the SAO in all the functions except in one, while the GBMO converged faster than the SAO in 24 of the functions. These convergence results are expected because the computation time in FFOA and GBMO is similar to the computation time required to evaluate one and two modes in SAO respectively. The developed SAO was applied to path planning problem and three scenarios of minimum spanning tree (MST) problem and results were compared with particle swarm optimization (PSO) and smell detection agent (SDA). Though all the algorithms obtained an optimized obstacle free path, results showed that SAO performed better than PSO and SDA in terms of cost by 11.41% and 83.29% respectively. On the MST model, the SAO and PSO obtained the same cost in the first scenario and 3.03% improvement over SDA. In the second scenario, the SAO obtained a better cost with 15.97% and 20.67% improvement over PSO and SDA respectively. In the third scenario, the SAO obtained a better cost with 8.94% and 14.14% improvement over PSO and SDA respectively. These results showed that the developed SAO is highly efficient and can compete significantly well with other algorithms reported in the literature.

TABLE OF CONTENTS
CHAPTER ONE: INTRODUCTION

1.1	Background	1
1.2	Motivation	7
1.3	Significance of Research	8
1.4	Statement of Problem	11
1.5	Aim and Objectives	12
1.6	Methodology	14
1.7	Thesis Organisation	17

CHAPTER TWO: LITERATURE REVIEW

2.1	Introduction	18
2.2	Review of Fundamental Concepts	18
2.2.1	Biology of Sense of Smell	18
2.2.1.1	Olfactory system	19
2.2.2	Chemistry of Sense of Smell	27
2.2.2.1	Smell agent algorithm deduction from chemistry perspective	27
2.2.3	Physics of Sense of Smell	30
2.2.3.1	Characteristics of gas	30
2.2.3.2	Mathematical properties of gas	32
2.2.4	Gaseous Brownian motion optimization	33
2.2.5	Fruit Fly Optimization Algorithm	36
2.2.6	Particle Swarm Optimization (PSO)	39
2.2.7	Smell Detection Agent (SDA)	41
2.2.8	Metrics for measuring the Complexity of Optimization Problem	43
2.2.8.1	Qualifying the complexity of function optimization	44
2.2.9	Robot pathfinding	60
2.2.9.1	Visibility graph algorithm	61

2.2.9.2	Artificial potential field	65
2.2.10	Minimum spanning tree problem	67
2.2.10.2	Prim's algorithm	70
2.3	Review of similar works	72

CHAPTER THREE: MATERIALS AND METHOD

3.1	Introduction	85
3.2	Materials	85
3.2.1	Computer System	85
3.2.2	MATLAB	86
3.3	Methods	86
3.3.1	Smell Agent Optimization (SAO) algorithm	86
3.3.2	Sniffing Mode	88
3.3.2.1	Population	88
3.3.2.2	Updating smell velocity and position	92
3.3.3	Trailing mode	94
3.3.4	Random Mode	96
3.4	Flow of SAO Algorithm	98
3.5	Important assumptions	99
3.6	SAO Parameter Selection	102
3.7	Application of SAO on the Benchmark function	103
3.8	Application of SAO in Path Planning	109
3.9	Application of SAO to Minimum Spanning Tree (MST)	115
3.10	SAO GUI Simulator	119
3.11	Performance Comparison	119

CHAPTER FOUR: RESULTS AND DISCUSSION

4.1	Introduction	122
4.2	Performance of the Algorithms on Uni-modal Test Function	122

4.3	Performance of the Algorithms on Multi-modal Test Function	126
4.4	Application to Part Planning	134
4.5	Application to Minimum Spanning Tree	139
4.6	Simulation with SAO GUI	145
CHAPTER FIVE: CONCLUSION AND RECOMMENDATION		
5.1	Summary	147
5.2	Conclusion	148
5.3	Limitations	150
5.4	Contributions to Knowledge	149
5.5	Recommendation for Future Work	150
REFERENCES		153
APPENDIX A		161
MATLAB CODE FOR SAO		161
APPENDIX B		165
MATLAB CODE FOR BENCHMARK FUNCTIONS		165
APPENDIX C		172
Path Planning Cost Function		172
APPENDIX D		174
MST Cost Function		174

LIST OF FIGURES

Figure 2.1: Olfactory System of Man (Buck, 2005)	21
Figure 2.2: Olfactory System of Dog (Correa, 2005)	22
Figure 2.3: Olfactory System of Cat (Bradshaw)	24
Figure 2.4 Overview of Fish Brain (Hamdani & Døving, 2007)	25
Figure 2.5: Process of Smell Perception in Agents	26
Figure 2.6: Molecular Structure of Methyl Salicylate (Wintergreen)(Black, 2014)	28
Figure 2.7: Molecular Structure of Vanillin (Vanilla)(Black, 2014)	28
Figure 2.8: Brownian Motion of Smell Molecules (Chapman & Cowling, 1970; Wheeler, 1990)	31
Figure 2.9: Smell Molecule Path Illustration	31
Figure 2.10: Flowchart of Gaseous Brownian Motion Optimization (Abdechiri et al., 2013; Elyas et al., 2014; Rathore & Roy, 2014).	31
Figure 2.11: Flowchart Implementation of Fruit Fly Optimization Algorithm (Pan et al., 2014; Pan, 2011)	38
Figure 2.12: Flowchart of particle swarm optimization	30
Figure 2.13: Polygon generated Robot Obstacles(Taizhi & Maoyan, 2017)	64
Figure 2.14: Visibility Graph of Polygon Generated Robot Obstacles (Taizhi & Maoyan, 2017)	64
Figure 2.15: An Undirected Weighted MST graph	68
Figure 2.16: Trees Extracted from Figure 2.15	69
Figure 2.17: Minimum Spanning Tree of Figure 2.15	69
Figure 3.1 The MATLAB interface of SAO simulation	87
Figure 3.2: Coordinate Position Representation of the Smell Particles in the Search Space	89

Figure 3.3: Cartesian Coordinate Representation of the Smell Molecules in the Search Space	92
Figure 3.4: Conceptual Framework of the SAO	97
Figure 3.5: Standard Flow Chart for SAO	101
Figure 3.6: Objective Function Flowchart	108
Figure 3.7: Path Planning Environments	112
Figure 3.8: Flow Chart of Path Planning Cost Function	113
Figure 3.9: MST Graph Topology	116
Figure 3.10: Flow Chart of MST Cost Evaluation	118
Figure 3.11: Developed SAO GUI Software	120
Figure 4.1: Minimization of unimodal functions	125
Figure 4.2: Minimization of Multimodal Functions	131
Figure 4.3: Performance Comparison Chart	132
Figure 4.4: Online Path Planned by SAO	135
Figure 4.5: Path Planned by PSO	136
Figure 4.6: Path Planned by SDA	137
Figure 4.7: Path Planning Cost Function	138
Figure 4.8: Minimum Spanning Tree Solution (1st Case)	140
Figure 4.9: Minimum Spanning Tree Solution (2nd Case)	141
Figure 4.10: Minimum Spanning Tree Solution (3rd Case)	142
Figure 4.11: MST Cost Minimization Plots	145

LIST OF TABLES

Table 3.1: Computer Specification	86
Table 3.2: SAO Control Parameters	103
Table 3.3: Classifying the Characteristics of the Selected Subset of Benchmark Functions	106
Table 3.4: Coordinate positions of obstacles	114
Table 3.5: SAO Simulation on Path Planning	114
Table 3.6: MST Vertex Coordinate	117
Table 4.1a: Uni-modal functions evaluations	123
Table 4.1b: Uni-modal functions evaluations	123
Table 4.2a: Performance Evaluation on Multi-modal functions	126
Table 4.2b: Performance Evaluation on Multi-modal functions	127
Table 4.2c: Performance Evaluation on Multi-modal functions	127
Table 4.2d: Performance Evaluation on Multi-modal functions	128
Table 4.2e: Performance Evaluation on Multi-modal functions	128
Table 4.2f: Performance Evaluation on Multi-modal functions	129
Table 4.3 Average Time of Convergence (sec)	133
Table 4.4: Performance comparison on Path Planning	137
Table 4.5: SAO performance evaluation on MST	143

LIST OF ABBREVIATIONS

Acronyms	Definition
ABC	Artificial Bee Colony
ACBFO	Adaptive Chemotactic Bacterial Foraging Optimization
ACO	Ant Colony Optimization
AFSA	Artificial Fish Swarm Algorithm
BFO	Bacterial Foraging Optimization
CA	Cultural Algorithm
CI	Computational Intelligence
CPU	Central Processing Unit
CSA	Clonal Selection Algorithm
Da	Dalton
DG	Distributed Generation
FFOA	Fruit Fly Optimization Algorithm
GA	Genetic Algorithm
GBMO	Gaseous Brownian Motion Optimization
GUI	Graphical User Interface
HFFOA	Hybrid Fruit Fly Optimization Algorithm
IEEE	Institute of Electrical and Electrical Engineering
IJCI	International Journal of Computational Intelligence

MATLAB	MATrix LABoratory
NFL	No Free Lunch
NLNG	National Liquefied Natural Gas
OFA	Optimal Foraging Algorithm
OS	Operating System
olc	olfaction Capacity
PSO	Particle Swarm Optimization
RAM	Random Access Memory
SA	Smell Agent
SAO	Smell Agent Optimization
SSO	Shark Smell Optimization
SDA	Smell Detection Agent
wAFSA	weighted Artificial Fish Swarm Algorithm

CHAPTER ONE

INTRODUCTION

1.1 Background

Efforts to adopt an acceptable definition of intelligence still elicit debates among various disciplines. Dictionaries (Crystal, 2004; English, 2007) have defined intelligence as the power of understanding, comprehending and profiting from experience, the power to interpret and having the capability for thought and reason especially to a high degree. The mechanisms of “intelligence”, which are exhibited by all living systems, share similarities in terms of complexity, organisation and adaptability as a whole. Over the years, experts have understandably sought for means of codifying intelligence systems into algorithms dedicated to solving some complex problems in engineering and related disciplines. This triggered the development of a new field of study called computational intelligence (CI) which was popularized by James C. Bezdek about 24 years ago (Bezdek, 1994). Perhaps, the first appearance of CI was way back in 1983 where the International Journal of Computational Intelligence (IJCI) was reported to be the title of the Canadian journal by its editors and founders Gordon McCalla and Nick Cercone (Bezdek, 2013). Mu'azu stated that “computational intelligence consists of any science supported technologies and approaches for analysing, creating, and developing intelligent systems (Mu'azu, 2006, 2016). ‘Intelligent’ in this case refers to the utilization of engineering techniques that have, to one extent or another, been borne out of human reasoning, adaptation or learning, biological cognitive structures or principles of evolution, natural physical or chemical processes.

Many experts have also referred to the term computational intelligence (CI) as the ability of a computer system to learn precisely a certain task, from a given set of data and/or empirical observation (Siddiqui & Hojjat, 2013). The IEEE World Congress on Computational Intelligence and the IEEE Neural Network Council formalized this term in the summer of 1994 in Orlando, Florida (Xing & Gao, 2014). Perhaps, till date, researchers have not come to a conclusive agreement as to a precise definition of computational intelligence (Zimmermann, 1999). This is because it is difficult to start with anything precise; the precision has to be achieved through a certain process. But, strictly speaking, computational intelligence is a set of nature-inspired computational models and approaches capable of addressing real-world problems to which traditional or mathematical model may be limited due to any or all of the following reasons:

- 1) The problem or process may be too complex for mathematical reasoning.
- 2) The problem or process may be dynamic and stochastic in nature.
- 3) The solution space of the problem may be too large for mathematical computation.
- 4) The problem or process may contain uncertainties.

All these characteristics are exhibited by most real life (non-linear) science, economic, social and engineering problems. These non-linear type problems require several assumptions in order to transform them to their near-linear equivalents for easy computation. However, the outputs of such linear computations usually do not depict entirely the real-life situations, except in a scenario where the exact solution to real life situations is not critical. Thus, agent-based computational intelligent techniques are capable of providing superb and promising alternatives in such scenario.

The obvious high-level intelligent agents are human beings. An intelligent agent is a rational entity that performs actions, in a relatively autonomous (independent) way, in its environment on behalf of its user. However, there are classes of intelligent agents (for example, natural phenomena, such as water drop, flower pollination, river formation, etc. or animals, such as fish, dogs, worms, insects, etc. or even bacterial, amoeboid, etc.) which could be more intelligent than humans in terms of coordination and organizations (Poole & Mackworth, 2010). An example of this intelligent organization is the Ant colonies. A single ant might not be very intelligent, but the entire colony will act more intelligently and organized than any individual ant. The ant colony can discover the location of food and search this location in order to exploit the food source effectively as well as adapt to changes in the environment using some specialized skills (Poole & Mackworth, 2010; Xing & Gao, 2014).

In the past decades, several researchers have developed various agent-based biological and nature-inspired CI algorithms for solving various optimization problems. The foraging behaviours of ant systems were codified into an algorithm in (Dorigo *et al.*, 1996). Algorithms based on the intelligent behaviour of fish (Lei. *et al.*, 2002), bacterial foraging (Passino, 2002), firefly (Yang, 2010), swarm particles (Eberhart & Kennedy, 1995), bee (Karaboga & Basturk, 2007) etc. have also been developed. The performance of all these algorithms on suitable problems such as in (Malarvizhi & Kumar, 2015; Pradhan *et al.*, 2016; Sundaram *et al.*, 2016; Tijani & Mua'zu, 2015; Turabieh & Abdullah, 2011) has demonstrated their effectiveness in solving real word problems. It is, however, important to note that, there is no known single nature-inspired optimization method capable of solving all optimization problems. This is known as the “no free lunch” theorem (Wolpert & Macready, 1997). This is because, the solution to

every real word problem can directly or indirectly be formulated using a particular algorithm that mimics a nature or bio-inspired phenomena. For example, the pheromone trail and pattern movement of ant system can be said to mimic the lane and edge detection problem in transportation and image processing respectively. Similarly, the path planning and trajectory problem of a robot can be linked to an agent seeking to identify the molecular movement of an object (smell) which evaporates and travels into the olfactory organ (nose), where they activate special cells that send a message to the brain for adequate judgment. Thus, it is the focus of this thesis to develop a mathematical model and codify the intuitive movement of an agent towards smell (odorant) molecule as the agent seeks to identify the source of that smell or odorant.

The sense of smell (olfaction) is one of the five senses through which the world is perceived. Through the sense of smell, humans and other animals can perceive a variety of chemicals (Linda, 2005). In fact, with a good sense of smell, humans and other agents (especially dogs) can perceive the molecular concentration of smell and intuitively trace or follow this concentration in order to identify its source. This is possible because the nose has a mechanism that can recognize sensory information within its surroundings and transmit this information to the brain, where it is processed to simulate internal interpretation of the external world (Axel, 2005). These interpretations help the smell agent to determine the optimized path which constitutes the search or solution space. In this case, the object which radiates the odour molecule is the solution that resides in the search space for which the proposed algorithm is based. Thus, it is hoped that a metaheuristic optimization algorithm using the path trailing ability of smell agents be developed for solving various degrees of combinatorial optimization problems.

In virtually all optimization problems, the complexity and structure of the problems increase as the dimension of the problem increases. In such situations, an exhaustive search is not feasible. Thus, a technique capable of operating in a domain of such optimization problems in which a set of possible solution is discrete or can be reduced to discrete with a common goal of finding the best solution is termed combinatorial optimization. Combinatorial optimization is an indication of optimization problems with an extremely large (combinatorial) increase in the number of possible solutions as the problem size increases.

Initially, the performance of the develop smell agent algorithm is to be evaluated using a subset of multidimensional and multi-peak applied mathematical optimization benchmark functions. This is to ascertain the optimality, precision and the convergence of the developed smell agent algorithm. Thereafter, the algorithm would be used to develop an efficient robot path planning technique in a dynamic environment with a static positioning of obstacles and minimum spanning tree problem.

The framework of the proposed smell agent optimization follows the general fundamental framework of nature-inspired algorithm. The properties and structure of every nature-inspired computational algorithm hinge on the following (Abdullah *et al.*, 2010; Poole & Mackworth, 2010):

- 1) Population: Every nature-inspired algorithm is initialized with a randomly generated set of initial population. This initial population are usually depicted as the population of the natural phenomenon on which the algorithm is based. Thus, the population of the proposed SAO is initialized by a set of the randomly generated initial population of smell molecules. These smell molecules were

deployed in the optimization hyperspace and the fitness of each molecule is evaluated.

- 2) Dimension: This is a function of the optimization problem which the computational algorithm seeks to explore. Perhaps, many real-life optimization problems are multi-dimensional in nature. Therefore, to efficiently evaluate the performance of any newly developed optimization algorithm, it is often customary to do so using a selection of multi-dimensional problems. If this is successful, then the general belief is that solving problem with simple dimension will be an efficient process.
- 3) Control Parameters: These parameters are formulated to guide the movement of the algorithm, as it evolves towards the optimum solution in the search space. The modelling choices of these parameters are usually dependent on the natural phenomenon on which the algorithms are based. If these parameters are not properly selected, the optimum performance of the algorithms will be greatly affected. In the proposed SAO, the control parameters include the mass and temperature of the gas molecules. Also, since the trailing capabilities of agent depend on their size of olfactory lobe, an olfaction capacity of the agent will also be assigned.
- 4) Stopping Criteria: These are additional parameters which are used to terminate the evolution process of the optimization algorithms. The basic stopping criteria for all computational intelligent algorithm is the iteration. The iteration is used by the algorithms to repeat the same process (in discrete form) over a specified number of time until a converged solution is obtained.
- 5) Fitness function: The fitness function which is also called the objective or cost function is the optimization problem which the computational intelligence

algorithm hopes to solve efficiently. The complexity of the fitness function depends on the dimensionality of the optimization problem, the nature of the constraints and the number of decision variables it contained.

- 6) Obey the “No free lunch theorem”: There is no single computational intelligent algorithm that is general for solving all optimization problems. Since these algorithms are mostly developed by the observation of the natural phenomenon, it is generally believed that the performance of an algorithm is best and convincingly evaluated using some optimization problems which mimic the behaviour of the algorithm.
- 7) Obey the “The no see the forest for the tree rule”: Every nature-inspired algorithm is expected to determine the best possible solution from a set of available possible alternative solutions. To do this, the algorithm has to be able to evaluate all the possible solutions in the search space from which the best (optimum) solution is to be identified. Thus, the inability of the algorithm to evaluate all the solution in the entire search space (No see the forest) in order to determine effectively the best possible solution (the tree) is termed the “No see the forest for the tree”.

1.2 Motivation

The sense of smell which is also called olfaction is termed as “the ability to detect or discover something by the faculty of smell” (Morrison, 2014). This ability is exhibited by nearly all living organisms. Animals employ the sense of smell for detecting and avoiding danger, mating, searching for food, searching for the conducive environment and changes in climatic conditions (Doty *et al.*, 1985). Due to the usefulness of olfaction, advancement in security and surveillance have employed animals with a

highly-developed sense of smell, like wolfs, dogs etc, to search hard to find targets which release a detectable plume (Furton & Myers, 2001). When these detectable plumes evaporate from a region of high density (source), it is conveyed by the wind toward the region of high concentration and consequently generating an odour plume. This odour plume is typically a conic (elliptical) shape evaporating laterally in the direction of downwind (Marques *et al.*, 2006). The distance covered by the plume is mostly determined using advection phenomena and spreading is usually due to turbulent rotation. In the aftermath of these behaviours, the temporal average and spatial odour concentrations decrease as the distance to the odour source increases. Since the odour plume located near the odour source releases profile of odour with maximum average concentration close to the source, the problem of searching different sources of multiple such odour in a confined environment can therefore be formulated or translated into a stochastic or metaheuristic optimization problem where the goal is to focus on determining the local optimum path leading to the odour source. This algorithm should be able to identify this optimal path quickly since the smell source identification depends on the distance to the smell source, the concentration of the smell and the density of the transmission medium (air).

1.3 Significance of Research

The integral part of modern engineering practice and industrial design revolves around modelling, simulation and optimization. Significant efforts and tremendous achievements have been recorded in all these components over the last few years. Perhaps, one of the challenging issues which still remains a bottleneck to researchers is the unavailability of an optimization resource that is of general application. Due to the quest to address these challenges, most of the present research trend in optimization

techniques has drifted towards using nature-inspired computational methods. Thus, the significance of this research, which aims to develop a smell agent optimization algorithm, hinges on the following:

- 1) **The “no free lunch” theorem:** All computational intelligent algorithms are inspired by observation of various natural phenomena. Mostly, the behaviours of these natural phenomena can be traced and transformed into equivalent real-life optimization problems. Since all real-life problems do not exhibit a common inherent structure and model, it has been stated that no single optimization algorithm or computational intelligent model has the capability of solving all of them (Wolpert & Macready, 1997). In nearly all optimization problems, the characteristics of the cost function for the problem at hand are almost always ignored. However, these optimization algorithms are always applied to these problems with little or no modification(s). When detailed characteristics of the optimization problem’s cost function are considered, it then becomes obvious that every optimization algorithm is unique to a certain kind of optimization problem a priori. In 1995, David Walpert demonstrated that all computational algorithms that seek for the extremum of cost functions act exactly in the same manner according to any performance measure when averaged across all cost functions (Wolpert & Macready, 1995, 1997). This led to the formulation of the “no free lunch” (NLF) theorem which states that, “if algorithm A outperform an algorithm B on a set of cost functions, then, there must exist exactly as many other functions where algorithm B outperforms algorithm A” (Lakkaraju *et al.*, 2017; Wolpert & Macready, 1997). Thus, the performance of every optimization algorithm can be effectively maximized when a suitable cost function is carefully selected. Therefore, in developing a new algorithm, the ideal thing is to

first study the natural phenomenon behind the algorithm and thereafter, a suitable optimization problem can then be identified. It is on the basis of this theorem that, this research hopes to develop efficient algorithm using the path trailing abilities of smell agent for combinatorial optimization problems. The developed algorithm is applied to path planning and minimum spanning tree problem.

- 2) **Real word optimization problem:** Many practical problems are large-scale and complex and could be challenging to fully capture and understand using mathematical models. However, most optimization tools are tested on some very small-scale problems. This is because the non-linearities in the large-scale problems are more than what optimization methods can handle. Thus, assumptions and approximations become a necessity. Since the real-world problems are mostly complex, research tends to use over-simplified models for approximating the real systems. This introduces several unknown factors (uncertainties) that reduce the accuracy of the results and validation of the models (Marques *et al.*, 2006; Pugh & Martinoli, 2007). Robot path planning problem is a typical example of an oversimplified real-world problem. Given a starting and final position of a robot, the path planning problem is to seek for a set of motions which will enable the robot to move between the two positions without colliding or been obstructed or stopped by an obstacle. The robot has to identify the optimal path and be able to locate the final position (target) in the shortest amount of time. This is usually a difficult task to achieve since the robot does not have an intelligent means of identifying the nature of the environment. Researchers have over the years employed various CI algorithms (S. Wang *et al.*, 2016; X. Wang *et al.*, 2016; Zhang *et al.*, 2016) for optimal path planning.

However, selecting a CI technique which will provide the optimal path planning when compared with all other algorithms has always been a critical issue. Since the algorithm proposed in this research is inspired by the path trailing behaviour of a smell agent, validating the algorithm with an engineering problem (path planning and minimum spanning tree) having the same or similar characteristics will be the ideal thing to do.

1.4 Statement of Problem

Combinatorial optimization involves the mathematical study of arraignment, ordering, selection or grouping of discrete objects. The complexity of combinatorial problem usually increases as the problem dimension becomes very large. Solving these large dimensional problems usually requires an enormous amount of time and computational resources before a combination of discrete solution (which may not even be optimum) is obtained. Over the years, researchers have sought for an efficient means of solving these problems with a common focus on obtaining the best possible combination of results within a short possible interval of time. This quest plays a significant role in the development of already well-established and widely accepted fields of computational intelligence algorithms. Interestingly, these algorithms have appeared to be more efficient (in terms of computation, precision and complexity) in solving various degrees of optimization problems when compared with the traditional methods. Generally speaking, all computational intelligence algorithms were developed through careful observation of various natural phenomena. The majority were focussed only on the foraging behaviours of most natural behaviours of some biological agents as the agents seek to identify the location of food source within its environment. In several situations, the nature and concentration of the food do influence the behaviour of the agent. For example, the movement of fish in water is attracted towards the region of high

concentration of food. The implication of this is that, the movement of fish is restricted to only the region of high concentration of food. However, considering only the foraging movement of the agents towards the food source does not truly represent the real-life scenario of a biological system. This leads to the problem of imbalance between exploration and exploitation, increasing the chances of poor convergence and subsequently leading the algorithm into local minima. Considering these limitations and bearing in mind the “no free lunch theorem” this research developed a new optimization algorithm using the natural phenomenon of smell. In the proposed research, the Brownian nature of the smell molecules as they evaporate from the smell source towards the agent is mathematically modelled. Thereafter, the trailing behaviour of the agent towards the smell source after sniffing the smell molecules is also developed. These mathematical models were codified and the resulting bio-inspired optimization algorithm is termed smell agent optimization (SAO). The performances of SAO will be evaluated through comparison with gaseous Brownian motion optimization (GBMO) and fruit fly optimization (FFO) on a total of thirty-nine (39) applied mathematical optimization benchmark functions. After the performance evaluation, the proposed algorithm will be used to develop a model of robot path planning and minimum spanning tree and results were compared with that of particle swarm optimization (PSO) and smell detection agent (SDA) algorithm.

1.5 Aim and Objectives

The aim of this research is to develop a smell agent optimization algorithm (SAO) based on the random concentration of smell particles and detection capability of an agent. This will then be applied to the robot path planning problem in a static environment with dynamic obstacle positions and the minimum spanning tree problem.

In order to achieve the stated aim, the following objectives are formulated:

- 1) To develop the mathematical model of the smell agent optimization algorithm and present the algorithmic procedure necessary for its realization. This will involve examining the smell detection capability of various agents and determining the detection strength and nature of detection organs in order to establish modelling parameters and also determine the influencing factors of most agents towards efficient smell detection.
- 2) To codify the developed mathematical model in 1 using MATLAB R2017a simulation environment and hence develop a generalized user-friendly graphical user interface (GUI) based simulator.
- 3) To evaluate the performance of the developed SAO using a collection of thirty-nine (39) subset of multidimensional and multi-peak applied mathematical optimization benchmark functions. This will contain both multimodal (such as Ackley, Dejong Cosine Mixture etc.) and unimodal (such as Beale, Booth, Easom, ellipsoid etc.) functions and both are used to evaluate the general performance of the developed algorithm in comparison with others.
- 4) To apply the gaseous Brownian motion optimization (GBMO) algorithm and the fruit fly optimization algorithm (FFOA) on the benchmark problems in 3) for the purpose of comparison and validation. The GBMO and FFOA were selected because the former is inspired by the turbulent rotational properties of gas and the later by its abilities to detect food using its strong sense of olfaction.
- 5) To apply the developed SAO in solving two typical combinatorial optimization problems and comparing its performance with those of particle swarm optimization (PSO) and smell detection agent (SDA) based approaches using optimality precision and convergence as performance metrics:

a) To determine the optimal path in a robot path planning problem in a static search environment in the presence of obstacles with dynamically changing positions.

b) To develop an optimized solution for the minimum spanning tree problem.

The PSO and SDA were selected because PSO has been widely used in solving combinatorial optimization problems (Mac *et al.*, 2017; Norouzi *et al.*, 2016) while the SDA was developed purposely for path planning problems (Chandra, 2016).

1.6 Methodology

The step by step procedures adopted to achieve the proposed smell agent optimization algorithm are highlighted as follows:

- 1) Development of smell agent optimization (SAO) algorithm. To efficiently carry out this method, the following procedure is adopted:
 - a) All the necessary parameters (such as velocity, molecular mass, population, temperature, position, iteration, agent, olfaction etc.) required to develop the mathematical model of the proposed algorithm are established.
 - b) The mathematical model of the proposed SAO algorithm is developed using the parameters established in item a).
 - c) The model of SAO algorithm to be developed is codified using MATLAB R2017a simulation environment.
 - d) A generalized user-friendly graphical user interface (GUI) is developed for the codified SAO developed in c)

- 2) The optimality and performance of the proposed algorithm in 1) is evaluated using thirty-nine (39) carefully selected benchmark mathematical optimization test functions.
 - a) Mathematical function are formulated as a metric which is used to determine the complexity of each of the selected benchmark functions.
 - b) The modality, plateau, valley, decomposability and dimensionality of each test function is determined using their respective hyperspace structure and the formulated metric function in 2a)
 - c) The algorithm to be developed in 1) is applied to the selected test functions and precision and convergence time is used as performance metrics.
- 3) Replication of the GBMO and the FFOA.
 - a) The performance of the replicated GBMO is evaluated using the selected test functions in item 2).
 - b) The performance of the replicated FFOA is also evaluated using the same test function.
 - c) Items 3a) and 3b) are compared with the SAO to be developed in 1) as a means of validation.
- 4) The developed SAO is used to develop an efficient path planning model for a mobile robot.
 - a) A suitable model of visibility graph based mobile robot path planning is developed using Euclidean distance as an objective function.
 - b) A static environmental scenario is generated in a square field area with dynamic positions of robot and goal.

- c) The developed SAO is employed to determine the shortest path to move from a specified initial position to a specified final position while avoiding collision with the obstacles.
- 5) The developed SAO is employed to develop an efficient model of minimum spanning tree
 - a) Defined the vertex of the minimum spanning tree problem containing tree different scenario (the first scenario contains 15 vertices; the second scenario contains 20 vertices and the third scenario contains 30 vertices).
 - b) Initialize the edges of each vertex connecting each order in each of the scenarios of (a.).
 - c) Starting from the edge with the most minimum weight, compute the weight at each vertex (node)
 - d) Perform item 5 c) until the most minimum weight is obtained.
 - 6) Items 4) and 5) are repeated using the PSO and SDA based approaches and the performances compared with that of the SOA based approach using optimality precision and convergence as performance metrics.

1.7 Scope of Study

The scope of this research which developed a novel optimization algorithm (smell agent optimization) inspired by the phenomenon of smell are highlighted as follows:

- 1) The research will not, at this point, develop some empirical formulae for selecting/determining the control parameters (i.e. temperature and mass) of the developed SAO.
- 2) The research will not, at this point, model an empirical formula for the psychological and environmental conditions which have direct effect on the

smell perception of the agent. It will be assumed that these and other physical factors as well as the olfactory lobes contribute to the olfaction capacity of the agent.

1.8 Thesis Organisation

The introduction to the thesis has been presented in Chapter One. The rest of the thesis is structured as follows: Detailed review of related literature and relevant fundamental concept on smell perception, benchmark functions and the selected optimization problems are presented in Chapter Two. In-depth approach and relevant mathematical models describing the smell agent optimization, path planning and minimum spanning tree were presented in Chapter Three. The analysis, performance and discussion of results were analysed in Chapter Four. Conclusions and recommendations make up Chapter Five. Quoted references and Appendices are also provided at the end of this thesis.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter is divided into two subsections. In the first subsection, the review of fundamental concepts which are relevant for the mathematical modelling, parameter establishment and development of the proposed smell agent algorithm are discussed. In the second subchapter, some of the previous research works, which are related to the proposed research area are critically reviewed.

2.2 Review of Fundamental Concepts

The section is focused on smell and the viewpoint of various fields of study on the smell. This is basically to understand the constituent nature of smell, its evaporation from source and mathematical deductions necessary for the formulation of the proposed smell agent optimization (SAO). There are also reviews on some optimization algorithms relevant to the research, benchmark test functions and the performance metrics.

2.2.1 Biology of Sense of Smell

Smell is the ability to perceive (sense) the odour or scent of a chemical substance through the nose by means of the olfactory nerves (Smell-(n.d), 2017). This is usually, made of a chemical compound of different molecules evaporating from a source with a molecular weight of less than 300 Da (Amoore & Hautala, 1983). Detection and discrimination of these chemicals in the environment are critical for the survival of most organisms. Virtually, all chemosensory organisms starting from the unicellular form to the most advanced multicellular creature has the capability to detect chemicals substances within their surroundings through a process called chemosensation

(Sookoian *et al.*, 2011). Chemosensation relies on the olfactory organ which contains a large number of chemoreceptors used by organisms to locate food, mates and avoiding danger. These chemoreceptors are generally classified into odorant receptors and pheromone receptors which mostly detect general odours and pheromones respectively (Ranaldi, 2011; Wysocki & Beauchamp, 1984). The pheromone receptors of an individual create a chemical compound which influences the behaviours of another individual of the same species. The pheromone is an exquisite and highly sensitive receptor capable of detecting low-level pheromone chemicals such that, random odorants molecules in the environment are not mistaken as pheromone cues (Morrison, 2014; Ranaldi, 2011).

2.2.1.1 Olfactory system

As stated earlier, olfaction is the process of detecting the presence of certain chemical compounds in the air through chemosensation. The chemical compounds which are odour molecules travel into the nasal cavity, where they are detected by the olfactory receptors. At this point, the olfactory receptor neuron fires and transmits an impulse into the olfactory bulb at the top of the nasal passage. This connects to the olfactory centre in the cerebral cortex for odour perception and recognition and to the limbic system which controls the expression of emotion and instinctive behaviours (Amoore & Hautala, 1983; Stevenson, 2009). This process is similar to biological organisms which have the ability of olfaction. The olfaction process in man, dog, rat and shark which are potential agents in the developed SAO are therefore discussed as follows.

1) The olfactory system of man

An important element in distal odorous object detection is the capacity to follow the scent trail or odour plume. This element is fundamental to the existence of several

animal species including human beings (Stevenson, 2009). Through odorant chemical identification, human beings can identify harmful objects such as carbon monoxide, button batteries, iron pills, cleaning products, pesticides, hydrocarbons, wild mushrooms etc. Humans and many other animals also use odorant molecules (olfaction) to identify food substances such as ripe fruits etc. Although in industrialized nations, humans rarely use olfaction for the detection of food substance, they still possess a strong capacity to follow odorant trails (Porter *et al.*, 2007; Stevenson, 2009) using specialized olfactory receptors.

Researchers have estimated about five to six million olfactory receptors in man, each having a specific sensation for a particular ligand (ligand, in this case, refers to a specific sensation for a particular smell molecule). This enables man to be able to detect over one trillion different odours (Morrison, 2014). It is however believed that each odorant molecules triggers several olfactory receptors which create a unique combination of neural impulses that are interpreted as a single odour scent. The biological structure of the olfactory system of man is given in Figure 2.1

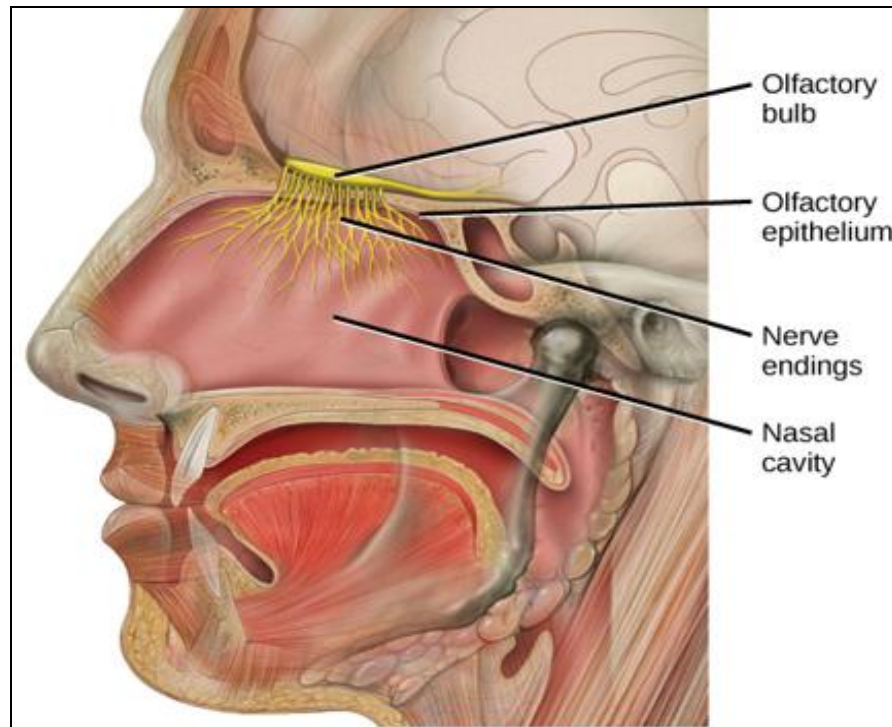


Figure 2.1: Olfactory System of Man (Lynch, 2006)

Figure 2.1 shows the biological peripheral structure of the olfactory system in human beings. This structure basically consists of the nostril, nasal cavity, ethmoid bone and olfactory epithelium. The olfactory epithelium consists of layers of thin tissues which are covered in mucus that lining the nasal cavity. The basic component of the olfactory epithelium tissue is the olfactory gland, olfactory neuron, mucous membrane, olfactory nerves and the nerve fibres. Odorant molecules evaporate into the peripheral pathway during olfaction or through throat as the tongue pushes smell molecules into the nasal cavity during retro-nasal olfaction (Touhara, 2014). The mucus lining inside the wall of the nasal cavity dissolves the odour molecules and the olfactory receptors generate some electrical responses representing the information about the dissolve molecules (Doty, 2017). This response spread through the receptor cells to the brain in the process called sensory transduction. The brain interprets the received sensory information and creates an exact interpretation of the external world.

2) The olfactory system of dog

An interesting thing to note is that despite the olfaction capability of man, agents such as dogs, sharks, cat etc. have a stronger olfaction capacity. For example, dogs have been said to have over 220 million olfactory receptors compared to a man with just about 5 to 6 million. This accounts for the dazzling sense of smell in dogs (Breer, 2008). As an illustration, Figures 2.2 shows the olfactory system of a dog

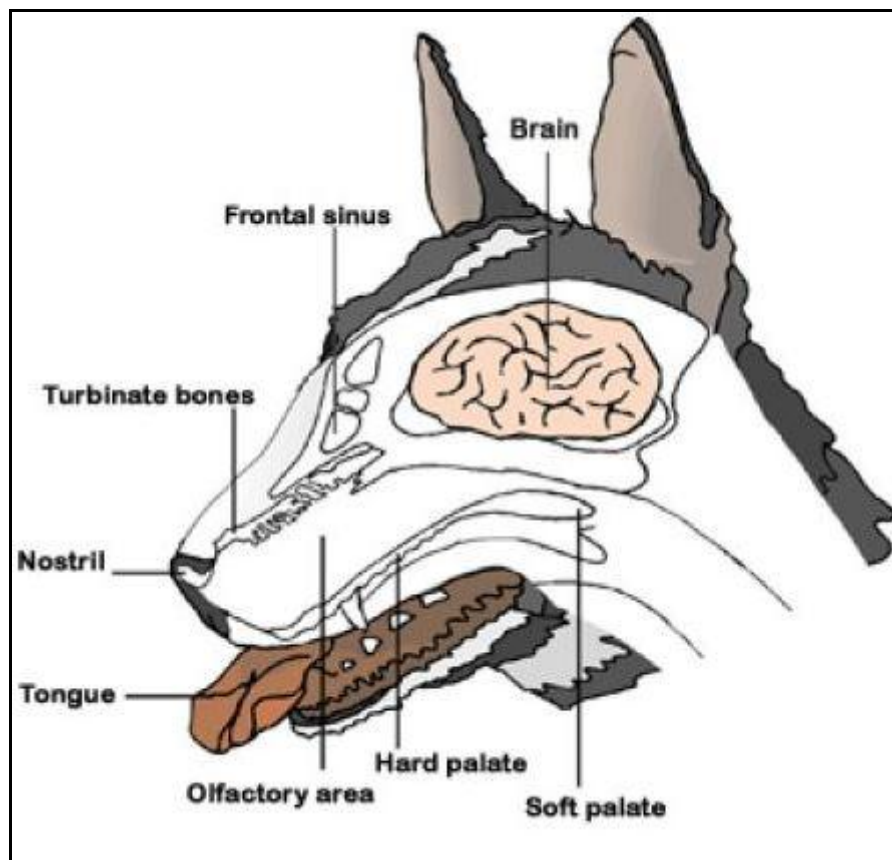


Figure 2.2: Olfactory System of Dog (Correa, 2005)

Just like humans, a dog has one pair of nares (nostrils) with which they inhale chemical substances and a nasal cavity as shown in Figure 2.2. The olfactory system of dog also contains olfactory receptor cells which extend through the olfactory epithelium located in the ethmo-turbinate bones of the nasal cavity. The olfactory system which is located in the mucous membrane of the nasal cavity contains a specialized supply of olfactory nerves connecting the olfactory lobe of the dog's

brain (Correa, 2005). Unlike in human beings, dogs possess another olfactory chamber called vomeronasal organ that contains olfactory epithelium. The vomeronasal organ, which is also known as Jacobson's organ contains a pair of elongated fluid that opens into the nose or the mouth. The olfactory receptors in the nasal cavity are anatomically distinct from that in the vomeronasal organ (Correa, 2005). Each nerves cell (receptor neuron) in the olfactory epithelium of the nasal cavity contain a dendrite which ends in the knob with a number of brain cilia covered by the mucous membrane. However, the receptor cells in the vomeronasal organ typically lack cilia but contain microvilli in the cell surface. The dog's nostril is usually cool and moist which is secreted by a mucous gland in the nasal cavity. This gland captures and dissolves molecules in the air and brings them into contact with the specialized olfactory epithelium (Correa, 2005; Touhara, 2014).

3) The olfactory system of the cat

Cats have a strong sense of adaptation which makes them highly effective in predatory. One of the important sense through which they predate is their strong sense of smell. It has been stated that cat sense of smell is about fourteen times stronger than that of a man (Syufy, 2017). Cats depend heavily on their strong olfaction for their immediate survival. In fact, through sniffing and trailing, cats use olfaction to identify the location of enemies, mates, foods and to seek the previously marked territory (Smith, 2016; Syufy, 2017). The biological structure of the cat olfactory system is given in Figure 2.3

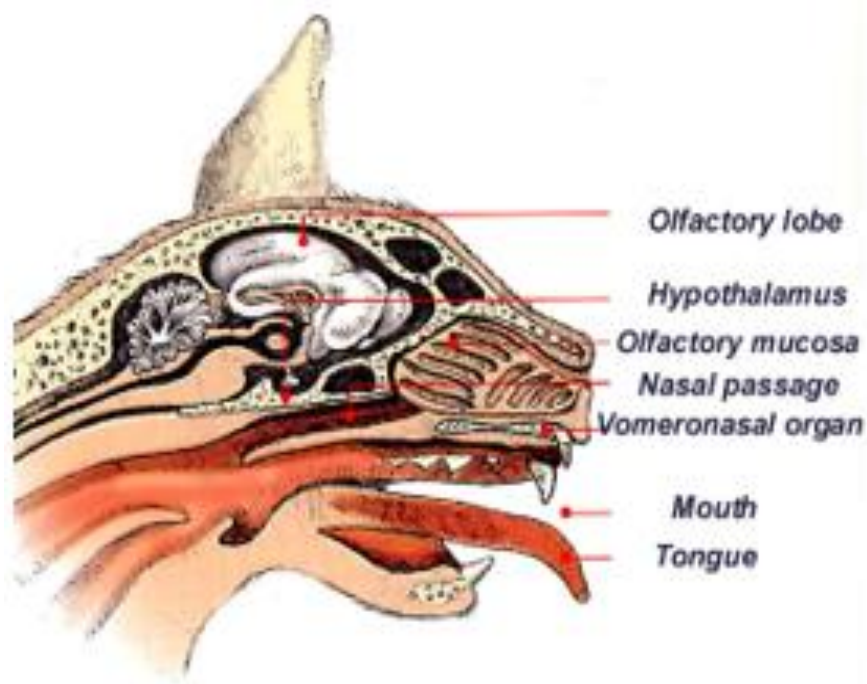


Figure 2.3: Olfactory System of Cat (Bradshaw)

The chemoreceptors in cat olfactory epithelium shown as Figure 2.3 has been said to be twice that of a human being. Similar to the dog, cats have vomeronasal (Jacobson's organ) smell organ which is located on the roof of their mouth (Fox *et al.*, 1944; Syufy, 2017). A cat opens a passage to the vomeronasal organ by lowering its chin, hanging its tongue and wrinkling its muzzle. Smell molecules evaporating through air pass through the vomeronasal into the central olfactory system which eventually connect the amygdala area of the brain where interpretation is performed (Fox *et al.*, 1944).

4) The olfactory system of Fish

The olfactory system of fish plays a significant role in their existence which lies fundamentally on feeding, avoidance of danger and reproduction. Figure 2.4 depicts the layout of the fish brain which exposes the olfactory epithelium (Hamdani & Døving, 2007). Similar to the previous discussion on man, dog and cat, the

chemosensory neurons in fish are located in sensory epithelium called the olfactory rosette. The sensory neuron contains some very thin axons which terminate in the olfactory bulb in specific synaptic glomeruli. The relay neuron axons project via the olfactory tract to a higher centre in the brain (Hamdani & Døving, 2007).

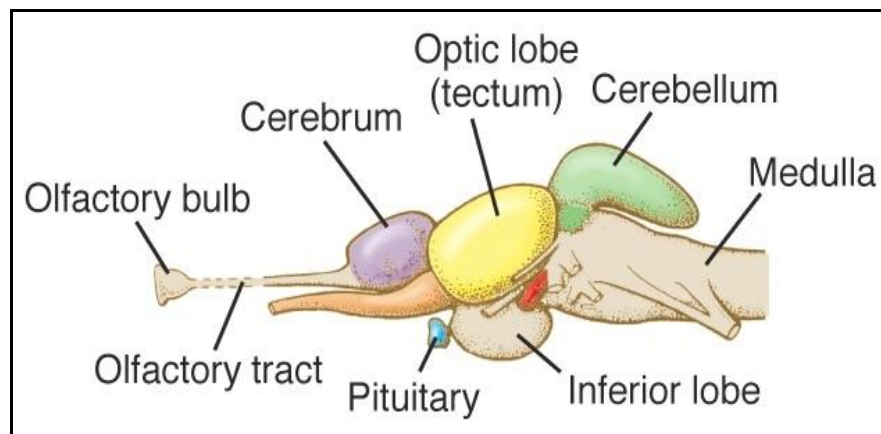


Figure 2.4 Overview of Fish Brain (Hamdani & Døving, 2007)

The chemosensory odorant receptors in fish consist of three types with different shapes. The first is the ciliated sensory neurones which have long dendrites and a few cilia. The second is the microvillous sensory neurones with shorter dendrite compared to the ciliated cell and microvillae extending from apical surface (Martin, 2017). In the olfactory bulb, odorant molecules are represented by a distinct glomerular which is a centre for integration of the olfactory information captured by a specific odorant chemoreceptor (Hamdani & Døving, 2007).

From the discussions so far in this subsection, it can be said that every agent perceives the presence of a chemical substance within their environment through olfaction. Basically, all biological agents use olfaction for the same purpose of identification of food location, survival and reproduction. In all the agents discussed and in fact, every agent which has the faculty of olfaction, chemoreceptors convey sensory chemical

compounds into the central nervous system as smell molecules through the steps shown in Figure 2.5



Figure 2.5: Process of Smell Perception in Agents

At the evaporation stage of Figure 2.5, the molecular vibration of an object which has smell capabilities evaporates chemical compound which travels into an agent nasal cavity at a molecular weight of less than 300Da. At the stimulation stage, the chemical compound interferes with the nerve endings of the olfactory epithelium which generates a specific stimulating energy for the chemical compound. An electrochemical (chemosensation) nerve impulse is therefore produced by the olfactory receptors through transduction. The electrochemical impulse is then transmitted through action potential along different pathways into the brain. The brain then interprets and creates perception from the electrochemical event produced through stimulation. This action of the brain is what an agent perceives as the smell.

In this research, every agent is assumed to have the ability of smell perception. However, sociologically, it has been observed that the olfaction capabilities of agents of the same species do vary. This variation is generally attributed to the variation in the size of the olfactory bulb and the physical and psychological state of the agent. Averagely, the olfactory bulb in humans is about 2.5cm^2 and contains millions of receptor cells having 8 to 20 cilia down in the layer of mucus of about 60 microns thick (Elsaesser & Paysan, 2007). Variation in this specification and the physical condition is what accounts for different olfaction capabilities between people. The implication of

this is that agents with larger olfactory bulbs will have higher chances of detecting the presence of chemical compounds in their environments more than those with smaller olfaction. Thus, in the proposed algorithm, olfaction capacity of the agents is an important parameter that was introduced to guide the movement of the agent towards the global solution. The larger the value of the parameter (olfactory size) the better the exploration capability of the developed smell agent optimization algorithm, while the smaller the value, the better the exploitation capability. Thus, maintaining a balance between these two situations is carefully considered when selecting the olfaction capacity parameter.

2.2.2 Chemistry of Sense of Smell

From the perspective of chemistry, smell (odour or fragrance) is generated by one or more chemical compounds generally at a very low concentration that humans or agents can perceive through olfaction (Breer, 2008). Smell and taste are the chemical sense indicators of every agent as they allow an agent to be aware of the presence of chemical substances in their environments. The ability to detect these substances is predicated on the chemical nature of their molecules (Black, 2014; Wysocki & Beauchamp, 1984). For the purpose of this study, the focus is only on exploring the chemical properties of smell which will be useful in establishing relevant modelling parameters for the proposed smell agent optimization.

2.2.2.1 Smell agent optimization deduction from chemistry perspective

As stated earlier, the sense of smell of an agent is influenced by the vibrations of tiny molecules. These molecules evaporate into an agent's nostril with a specific shape and mass which fit into a particular receptor allowing the agent to detect the specific object having the smell (Ranaldi, 2011). Chemists have over the years identified that

molecular atoms vibrate at a specific frequency depending on their molecular structure. In fact, a single change of atom in the molecular structure can make the smell molecules to vibrate at a much higher frequency resulting in variation of smell perception (Ranaldi, 2011). This accounts for why smell molecules of a very close structure and mass differs in terms of perception. For example, the fragrance (smell) of wintergreen which is used predominantly in mints, candies, mouthwashes and toothpaste has the same molecular structure with vanilla fragrance (smell) which is used to make ice cream and chocolate. The molecular formula of the wintergreen is $C_8H_8O_3$ and the molecular structure is as shown in Figure 2.6 while the molecular formula of vanilla is also $C_8H_8O_3$ and its molecular structure is as shown in Figure 2.7 (Black, 2014)

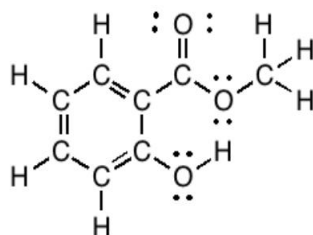


Figure 2.6: Molecular Structure of Methyl Salicylate (Wintergreen)(Black, 2014)

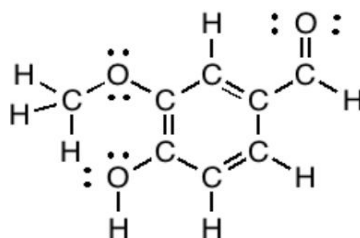


Figure 2.7: Molecular Structure of Vanillin (Vanilla)(Black, 2014)

From, Figures 2.6 and 2.7, it can be observed that, though both wintergreen and vanilla have the same molecular formula and similar structure, the variation in their fragrance is attributed to the positioning and bonding of the functional group (Black, 2014). For a smell molecule to be perceptible by an agent, the chemical composition of the smell has to be lipophilic, small (with a molecular weight of less than 300 Da) and volatile (Black, 2014). Of interest in this research is the molecular mass (weight) of the smell particles

and because the particles are individually very small (in terms of weight and size), their initial situations and states of motion are not known. Even if this is known, there is an unequal power to the task of following the subsequent motion of all the individual smell molecules (Miller, 1924). Thus, the faith of individual smell molecules will not be the focus of the proposed smell agent optimization (SAO) algorithm but, on the collective distribution of properties and motion (diffusion) of these smell molecules. According to the kinetic theory of heat, the amount of translatory kinetic energy possessed by the molecules over a distance is given by (Chapman *et al.*, 1970)

$$K_e = ndx \frac{1}{2} mc^2 \quad (2.1)$$

Where;

n is a number of smell molecules.

c is molecular velocity

m is the mass of the molecules.

Assuming a single molecule of smell at rest, the average kinetic energy is given by (Chapman *et al.*, 1970)

$$K_e = \overline{\frac{1}{2} mv^2} = \frac{3}{2} kT \quad (2.2)$$

Where;

k is the Boltzmann's constant given as $1.38 \times 10^{-23} JK^{-1}$

T is the temperature of the smell molecules in the environment

From, equation (2.2), it can be observed that an increase in the molecular temperature increases the kinetic energy and subsequently affect the movement velocity of the smell molecules. Thus, in the proposed smell agent algorithm, selecting the appropriate

molecular temperature is an important control parameter. Equations (2.1) and (2.2) also showed that the velocity has an inverse relationship with the mass of the smell particles (Chapman *et al.*, 1970). The implication of this is that smell particles with larger mass will move with a lower velocity which will enhance the local search ability of the developed algorithm. Similarly, smell particles with smaller mass will move with a higher velocity which enhances the global searching ability of the algorithm developed in this thesis. Thus, maintaining the appropriate mass of the smell molecules throughout the generation of the proposed algorithm is also an important control parameter that needs to be selected appropriately.

2.2.3 Physics of Sense of Smell

It has already been established that smell sensation results from molecules of chemical compounds which evaporate from a source in form of gas. Thus, the focus of this subsection is to explore the characteristics of gas which is useful in modelling the proposed SAO. It should be noted that, from a physics perspective of the discourse in this subsection, the words smell and gas are sometimes used interchangeably. This is because both smell molecules and general gas molecules are air-like substances which expand freely to fill any available space, irrespective of their density and quantity.

2.2.3.1 Characteristics of gas

Gas particles are usually in constant, rapid and random motion colliding with each other and with the wall of the container making the particles to change its direction constantly. As a demonstration, this movement is as shown in Figure 2.8

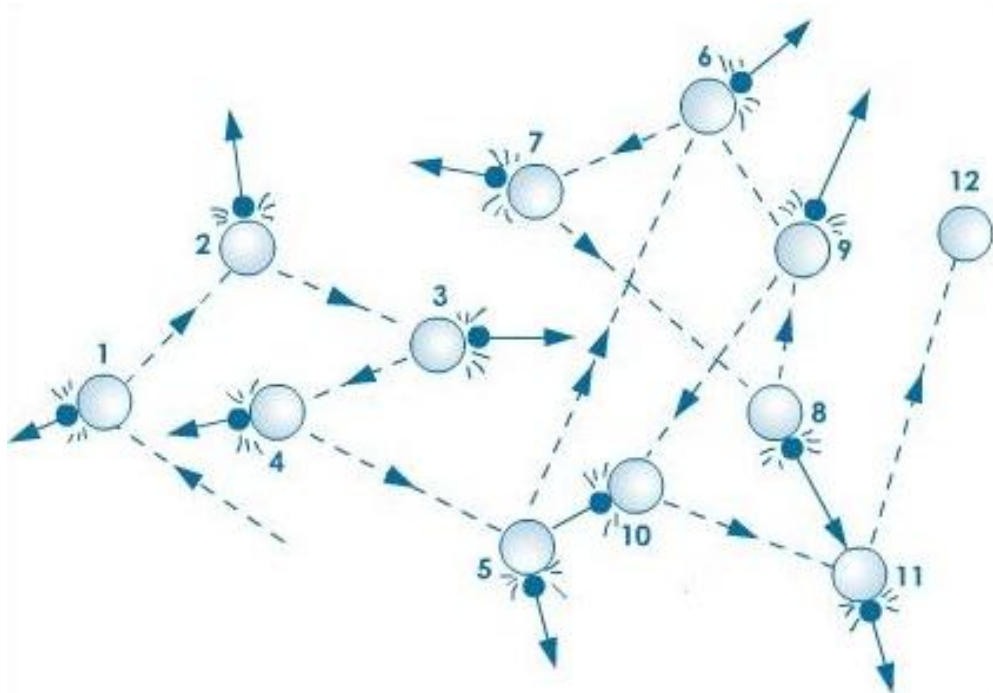


Figure 2.8: Brownian Motion of Smell Molecules(Chapman *et al.*, 1970; Wheeler, 1990)

It can be observed from Figure 2.8 that, point 1 shows the initial position of the gas molecule. The molecule moves through all the other point until the destination point in a Brownian form. The path followed by the gas molecule is as shown in Figure 2.9

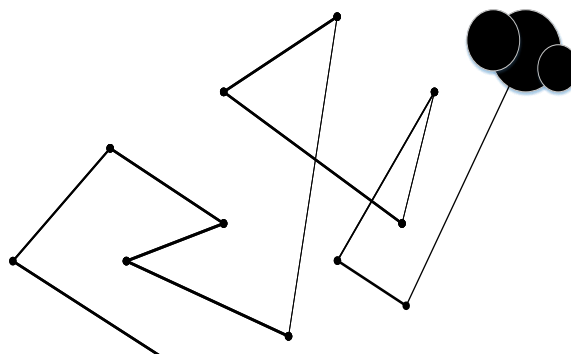


Figure 2.9: Smell Molecule Path Illustration

Trailing the path of the smell molecule shown in Figure 2.9 is usually the main focus of all smell agents. By this, virtually all organisms are aware of the presence of chemical substances in their environments. The physical state of any gas can be described by the pressure (P) exerted on the gas molecules, the temperature (T) of the gas molecules, the volume of the gas (V) and the number of moles of substances present in the gas. With

the proper knowledge of any three of these variables, the properties of the fourth variables can always be determined (Chapman *et al.*, 1970; Rathore & Roy, 2014). The mathematical relationships are discussed in the following subsection

2.2.3.2 Mathematical properties of the gas

In order to set up a mathematical relationship between the molecules of a gas, the following assumption is usually adopted (Richtmyer & Burdorf, 1981).

- 1) The collision between molecules of gas is perfectly elastic
- 2) The volume of an individual molecule of gas is negligible compared to the total volume occupied by the gas.
- 3) All the molecule of a particular gas is identical
- 4) The collision time is negligible
- 5) There are no attraction forces between the gas molecules.

Based on these assumptions, the average kinetic energy of gaseous molecules is given by (Miller, 1924; Vogt, 2017)

$$\frac{1}{2} m \bar{V}^2 = \frac{3R}{2N_A} T \quad (2.3)$$

Both R and N_A are universal constant and their ratio R/N_A is called the Boltzmann's constant denoted by k . Now, substituting k into equation (2.3)

$$\frac{1}{2} m \bar{V}^2 = \frac{3}{2} kT \quad (2.4)$$

Hence, equation (2.4) showed that the average kinetic energy of a gas molecule is proportional to its absolute temperature. This implies that increase in the temperature of the gas molecule will increase its average kinetic energy. Therefore, the temperature of the evaporating smell molecule is considered as an important parameter for the

developed algorithm. Equation (2.4) shows the kinetic energy of the generalized idea gas. Using this equation, the velocity, mass and temperature of the gas molecules can then be obtained. Since the proposed SAO algorithm is inspired by the intuition capability of agents toward smell perception and also since the behaviour of the smell particles is assumed to be the same as that of gas molecules, this research will employ two of the most recently developed gas and smell based computational intelligent algorithms to compare the performance of the developed SAO algorithm. The fundamentals of these algorithms are therefore discussed as in the following subsections

2.2.4 Gaseous Brownian motion optimization

Gaseous Brownian motion optimization (GBMO) is an optimization algorithm which was developed in 2012 for solving various degrees of non-linear optimization problems and is developed using the Brownian nature and turbulent rotational motion of the ideal natural gas (Abdechiri *et al.*, 2013). In this algorithm, the gas molecules are referred as agents and the solution to the optimization problem represents a potential position of the molecules. The steps involved in the implementation of the GBMO are highlighted as follows (Abdechiri *et al.*, 2013)

- 1) Randomized population. The process of GBMO starts with a randomly generated initial population of gas particles in a random position
- 2) A radius of turbulence is assigned to each randomly generated gas particles in the range of [0,1]
- 3) An appropriate temperature is determined and assigned to each gas particles.
- 4) The position and velocity of the gas molecules in GBMO are determined and updated as $x_i^d(t+1)$ and $V_i^d(t+1)$ respectively.

- 5) Fitness evaluation. At this stage, the fitness function for the gaseous molecules is evaluated and updated.
- 6) A chaotic sequence generator called circle map is used to modelled the turbulent rotational motion and is represented by:

$$x_i^d(t+1) = x_i^d(t) + b - \left(\frac{a}{2\pi} \right) \sin(2\pi x_i^d(t)) \text{mod}(1) \quad (2.5)$$

Where $a = 0.5$, $b = 0.2$, and a chaotic sequence generated in an interval of $(0, 1)$ and $x_i^d(t)$ is the current position of the gas molecules in turbulent rotational motion

- 7) Evaluate and compare the fitness of the objective function of the new positions of molecules.
- 8) Updating temperature and mass. The molecular mass is updated by:

$$m_i(t) = \frac{fit_i(t) - worst(t)}{best(t) - worst(t)} \quad (2.6)$$

Where $fit_i(t)$ is the fitness of the molecule i at time t , and $best(t)$ and $worst(t)$ are defined by:

$$\begin{cases} best(t) \\ worst(t) \end{cases} = \begin{cases} \min_{i \in \{1, \dots, N\}} fit_i(t) \\ \max_{i \in \{1, \dots, N\}} fit_i(t) \end{cases} \quad (2.7)$$

The temperature is updated by:

$$T = T - \left(\frac{1}{mean\ fit_i(t)} \right) \quad (2.8)$$

Where $fit_i(t)$ represent the fitness value of the molecule i at the time t ,

- 9) At the end of the algorithm, steps 3 to 9 are repeated until the stop criterion is reached.

The basic flowchart for implementing the GBMO is as shown in Figure 2.10 (Abdechiri *et al.*, 2013; Elyas *et al.*, 2014; Rathore & Roy, 2014).

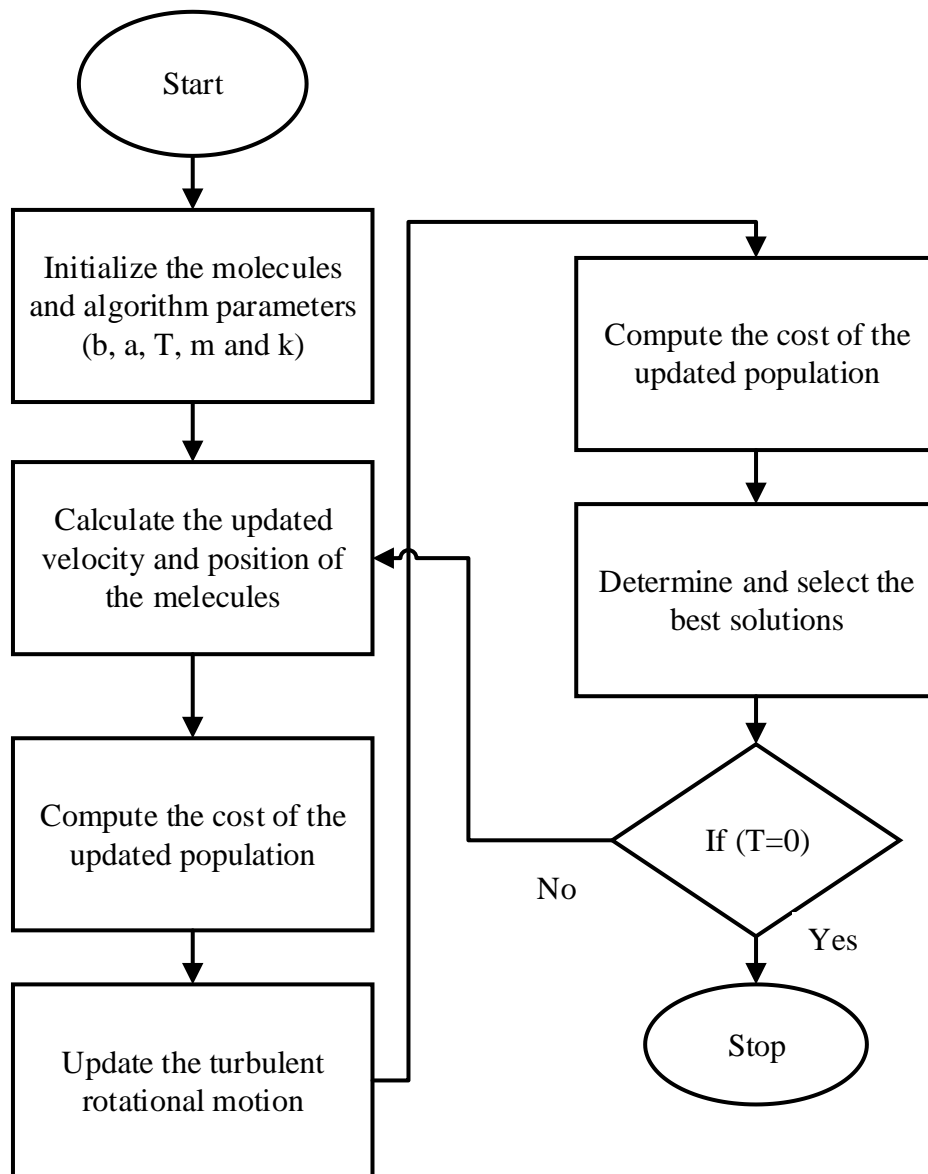


Figure 2.10: Flowchart of Gaseous Brownian Motion Optimization (Abdechiri *et al.*, 2013; Elyas *et al.*, 2014; Rathore & Roy, 2014).

The procedure employed for the replication of GBMO is given in the flowchart in Figure 2.10. Initially, the number of gas molecules and other parameters required for the realization of GBMO were initialized. Then, the position of the molecules was randomly initialized and their update velocity and position are determined. The fitness of each molecule is calculated and the position of molecules with the best fitness is

memorized. The turbulent rotational motion of the molecules is updated and the fitness of the updated motion is calculated. This fitness is compared with the previous fitness and the best fitness is retained while the worst fitness is discarded through greedy selection. The whole process is repeated all over again until the stopping criteria are met. The replicated GBMO is applied to the selected 39 benchmark functions. The performance of the SAO developed in this work will be validated using the GBMO and all relevant information about the GBMO can be found in (Abdechiri *et al.*, 2013; Elyas *et al.*, 2014; Rathore & Roy, 2014).

2.2.5 Fruit Fly Optimization Algorithm

The superior sensing and perception (ospheresis and vision) capability of fruit flies towards food was modelled into an algorithm by Tsao Pan in 2011 (Pan, 2011). The fruit fly can smell fruit from a location of about 40km away and intelligently move towards this location using their ospheresis organ and vision. The basic step by step procedure of implementing the fruit fly optimization algorithm (FFOA) are highlighted as follows (Pan, 2011):

Step 1: The initial position of the fruit fly is randomly generated in X-axis, Y-axis coordinate.

Step 2: Each fruit fly is assigned a random value used in looking food and location (X,Y).

$$X = X\text{-axis} + \text{Random Value} \quad (2.9)$$

$$Y = Y\text{-axis} + \text{Random Value} \quad (2.10)$$

Step 3: Determine the distance of each fruit fly from the zero point and determine the density value which is equal to the inverse of the distance.

$$Dist = \sqrt{X^2 + Y^2}; S = 1/dist \quad (2.11)$$

Step 4: Evaluate the fitness of the smell density function.

$$\text{Smell}=\text{function}(S) \quad (2.12)$$

Step 5: Repeating steps two to four and calculate the smell density of all the fruit flies and determined the flies with lowest and largest density value

Step 6: Retain the position of fruit fly with largest smell density and discard the ones with lowest smell density.

Step 7: Step two to six is repeated until termination criteria are met.

The flowchart implementation of the Fruit fly optimization algorithm shown in Figure 2.11 (Pan *et al.*, 2014; Pan, 2011)

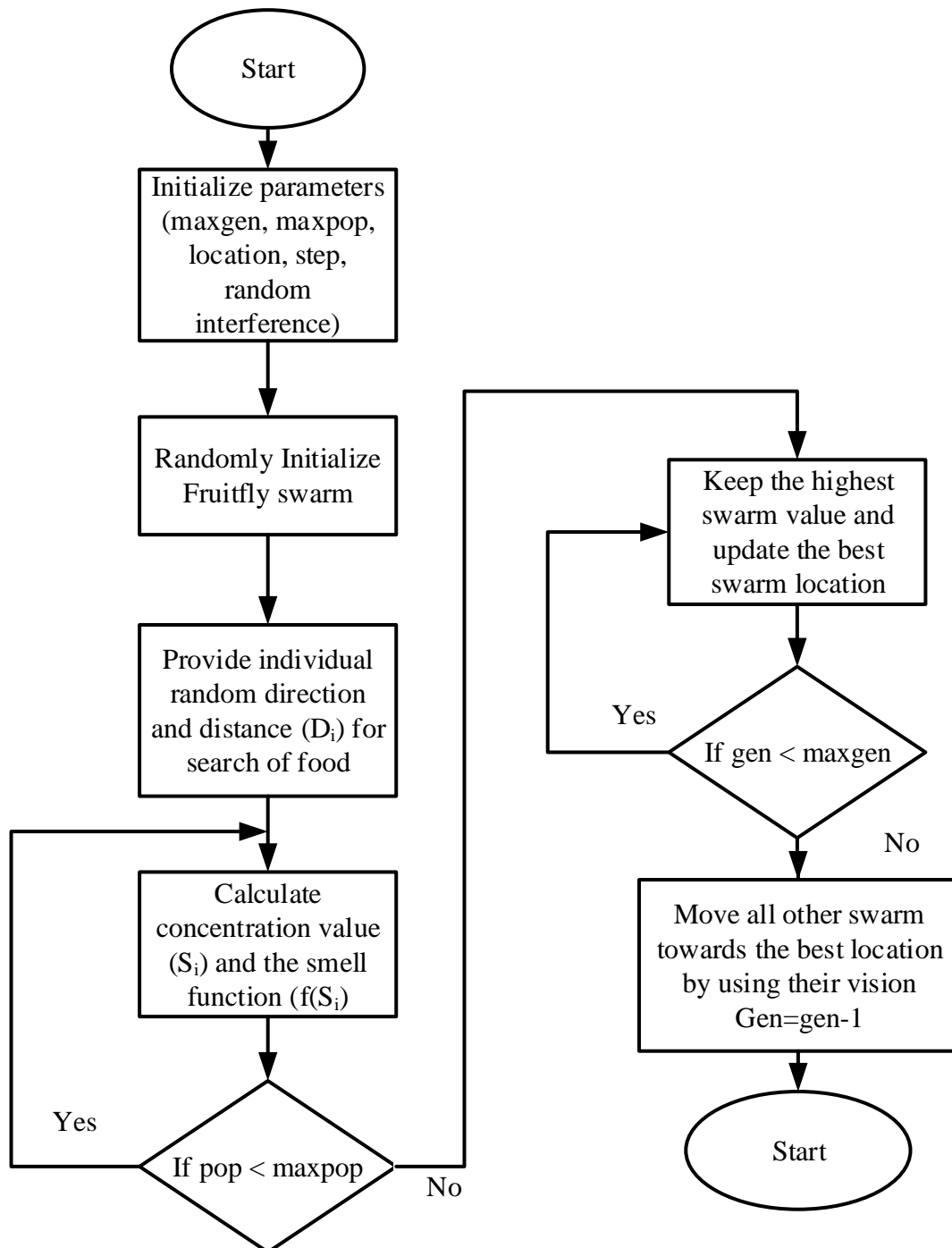


Figure 2.11: Flowchart Implementation of Fruit Fly Optimization Algorithm (Pan et al., 2014; Pan, 2011)

In the flowchart of FFOA algorithm given in Figure 2.11, all the parameters of the algorithm were first initialized and the positions of the fruit flies were randomly initialized. Each fly was assigned a random direction and a distance for food searching are assigned. The concentration value of smell is calculated and the smell function

(fitness function) is evaluated. The position of a fruit fly with the best smell function is kept and this information is used to update the previous position of the fruit fly. All other flies is moved towards the best position until the food source is identified. The performance of the SAO developed in this work is also validated using the FFOA and all relevant information about the FFOA can be found in the works of (Pan *et al.*, 2014; Pan, 2011; Wang *et al.*, 2015; Xing & Gao, 2014).

2.2.6 Particle Swarm Optimization (PSO)

PSO is one of the pioneer swarm optimization algorithms which was developed using the principle of flocks of birds and school of fish. The process of PSO is initialized by a set of randomly generated initial population of solutions (Eberhart & Kennedy, 1995). Each potential solution is assigned a random velocity with which they are flown into the optimization hyperspace. Each particle keeps a record of its own position coordinates in the hyperspace which is associated with the optimum solution obtained so far. This is called the personal best (pbest). The entire particles in the swarm also keep track of the overall best and its location in the hyperspace. This overall best value is called the global best (gbest) (Delice *et al.*, 2017; Eberhart & Kennedy, 1995). Assuming an n -dimensional search space, the position and velocity of a particle i is given as (Park *et al.*, 2005).

$$X_i = [x_{i1}, \dots, x_{in}] \quad (2.13)$$

$$V_i = [v_{i1}, \dots, v_{in}] \quad (2.14)$$

The fitness of each particle in equation (2.13) is evaluated and the pbest and gbest value are determined. This information is used to update the velocity as (Delice *et al.*, 2017; Park *et al.*, 2005):

$$V_i^{k+1} = V_i^k + c_1 r_1 \times (pbest_i^k - X_i^k) + c_2 r_2 \times (gbest_i^k - X_i^k) \quad (2.15)$$

Where;

c_1 and c_2 are the velocity control parameter

r_1 and r_2 are random numbers generated differently

Therefore, the position of the particles is updated as (Park *et al.*, 2005):

$$X_i^{k+1} = X_i^k + V_i^{k+1} \quad (2.16)$$

Equation (2.16) is the position update of the particles. The flow chart for implement the standard particles swarm optimization is given in Figure 2.12.

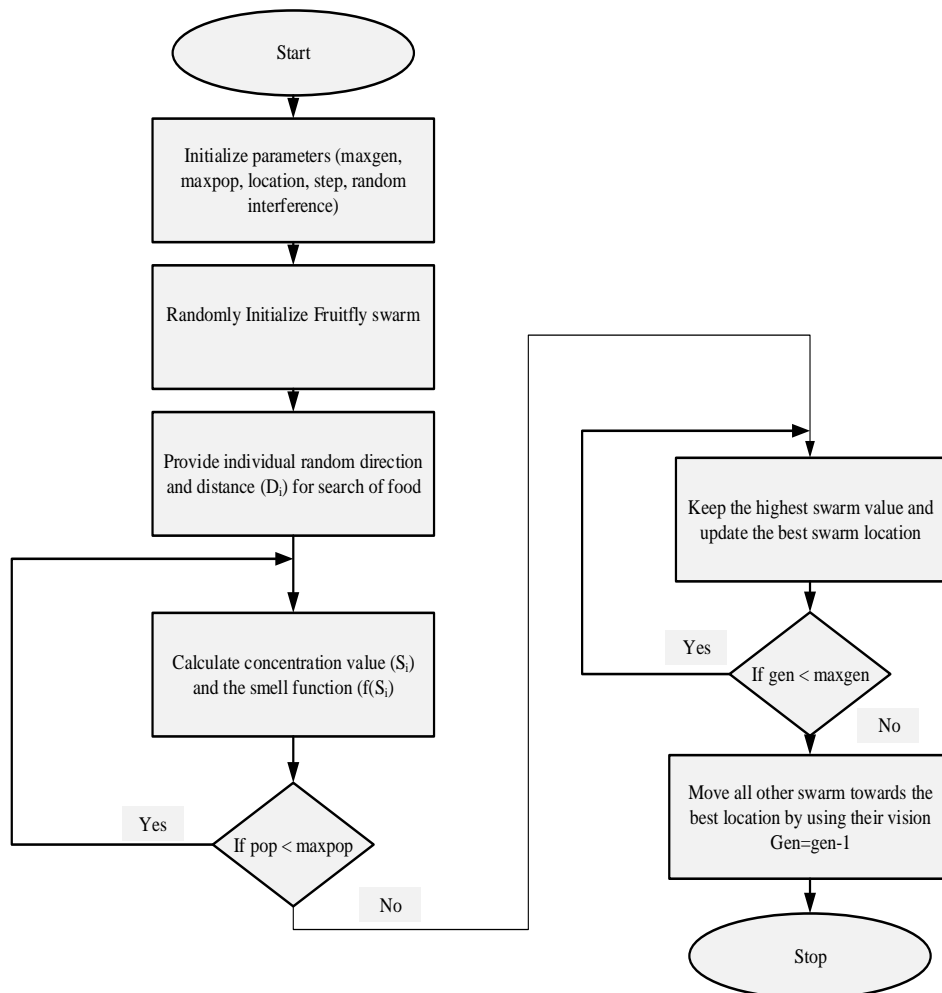


Figure 2.12: Flowchart of particle swarm optimization

From the flowchart in Figure 2.12, all the parameters of the PSO were first initialized.

The initial positions were randomly generated and the fitness of each particle in these

positions was evaluated and the gbest position was determined. The velocity of the particle was updated and the positions of the particles with the updated velocity was determined. The fitness of the updated position of the particles is evaluated and the updated fitness is determined. This fitness is compared with the previous fitness and the most favourable fitness is kept. The pbest and gbest positions were updated and the entire process of PSO repeat itself again until the termination criteria are reached.

2.2.7 Smell Detection Agent (SDA)

The SDA is an optimization algorithm which was developed using the territorial behaviours of dog and their smelling capabilities. In this algorithm, the concept of dog trialling a path is used to develop a suitable path for the SDA. The search environment is modelled into Cartesian coordinate with a specified boundary defining the search area. The points in the Cartesian coordinate selected randomly which can be visited by the SDA. The points are called the smell spots which help the dog in restricting its movement only within the point. The smell spots are stored by two parameters which are the smell trail and the signature value. This parameter is assigned a signature value which the SDA used to mark a smell spot and a radius value which indicate the olfaction capability. The data structure of the SDA is given as

$$D_{SDA} = (s, F, A) \quad (2.17)$$

Where s is the data type used, either integer or real values. F is a set of function such as search, split an SDA etc. A is a set of flags or constants such as start/stop, local maxima.

The data structure for smell spot is:

$$D_{SDA} = (v, F, A) \quad (2.18)$$

Where v is a data type such as real, integer; F is a set of functions such as nearest spots, distance measure and A is a set of axioms such as visited, not visited, radius

identification, smell distribution. In the initialization process of SDA, two parameter values are initialized. These parameters are the number of agents and the number of smell spots that are varied according to the computational resources available to solve some optimization problems. The algorithmic flow of the SDA is highlighted as

I. SDA Algorithm

Let N_1 : Number of SDAs

N_2 : Number of Smell spots

N_3 : Number of SDAs that can reach the destination

- a. Initialize the agents with natural numbers as signature indices and radii value in the inverse order, so that the SDA is considered in the iteration that has the highest radius.

- b. Randomly select N_2 points within the extremities of the Cartesian plot spots.

Assign the smell values' to each smell spot

$$s = 1/(a \pm b \times d)$$

Where d is the cartesian distance between the smell spot, the destination and a & b are proportionality constraints

- c. Initialize every agent with a smell source
- d. For each agent from 1 to N_1
 - i. choose the unmarked point (within the radius) from a set of smell spots that has the highest smell value
 - ii. Move the SDA to the point by marking the value of SDA's signature in the smell spot's index
- e. Repeat step d. until the destination is reached
- f. Count the SDAs that reached the destination and assign to N_3

2.2.8 Metrics for measuring the Complexity of Optimization Problem

For any new or modified computational intelligent (CI) optimization algorithm, it is often customary to validate and compare the various characteristics of such algorithm with other algorithms using a subset of benchmark functions (Jamil & Yang, 2013). These benchmark functions are mathematical representations of landscape structures having one or more contours. The goal of CI algorithms is to find the best possible minimum or maximum (optimum) of these contours while satisfying various equality and/or inequality constraints. These goals are called objective functions which can be expressed mathematically as (Chung, 1997):

$$(x^*) = \text{Find} \left\{ \begin{array}{c} x_2 \\ x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{array} \right\} \text{ which minimizes } f(x) \quad (2.19)$$

Subject to the constraints

$$\begin{aligned} g_i(x) &\geq 0 \quad i = 1, 2, \dots, m \\ h_j(x) &= 0 \quad j = 1, 2, \dots, n \end{aligned} \quad (2.20)$$

where,

x is the decision variable whose optimum value is to be determined by the optimization algorithm.

The optimum value of x^* may be a set of a minimum point $x^* \in D$ rather than a single optimum value. In such a case, the objective function is called a multimodal function. Thus, the algorithm proposed in this research is applied to various optimization test functions taken from different classes. The optimization test functions selected is a representation of a broad spectrum of practical optimization problems and different degrees of complexities is considered. These functions are carefully selected to

efficiently measure the performance of SAO developed in this research. The performance measures of interest are listed as follows (Chung, 1997)

- 1) Execution speed: which is a measure of how fast the algorithm can obtain the optimum solution
- 2) The precision of the solution: this is a measure of how well or how close the obtained solution is to the exact or global solution.
- 3) The rate of convergence: This is used to determine at what point the algorithm obtains its solution.
- 4) The probability of finding global optimum solutions.

2.2.8.1 Qualifying the complexity of function optimization

Usually, the objective function of an optimization problem can be characterized as linear, non-linear, continuous, discontinuous, unimodal, multimodal, convex, non-convex, separable and non-separable. In qualifying the complexity, it is important to provide answers to the following questions (Chung & Reynolds, 1998):

- 1) At what part of the function landscape did the optimization problem become difficult?
- 2) What is the most effective type of knowledge for searching specific function landscapes a priori?

In order to provide an acceptable answer to these questions, five basic features which contribute to reducing the effective true knowledge of the optimization problem have been described (Chung & Reynolds, 1998; Winston, 1992):

- 1) Modality: The modality of an optimization problem is determined by the number of unknown peaks in the landscape of the optimization problem. Population-based search methods have performed significantly well in these kinds of

problems compared to other methods. However, false local minima still attract individuals into early convergence like a magnet. Once this individual is trapped into the local minima, it is very difficult to escape from it, especially at the end of the generation, since the step size usually becomes very small.

- 2) Plateau: This characterizes a function with basin shape surface. It is simpler to find the flat area of the basin surface at the beginning of the search. An optimization algorithm can be easily attracted to this region. Once individuals reach this flat region, the individuals may experience no progress at all, since not only the surface is flat but also the step size is small (Chung & Reynolds, 1998).
- 3) Valley: This occurs when a little change in the narrow area of the optimization problem is surrounded by a region of a steep descent. Just like the plateau, the individuals in the optimization algorithm are initially attracted to this region. Thus, the probability of finding the global solution is slow, since it will take time for individuals to go through this valley.
- 4) Decomposability: This is a measure of the complexity of different optimization problem. If each parameter in an optimization problem is independent of all other parameters, the function is regarded easy to optimized since optimization can be performed in a sequence of n independent optimization process. The general condition of decomposability is given as follows (Momin & Yang, 2013):

$$\frac{\partial f(\vec{x})}{\partial(x_i)} = h(\vec{x})g(x_i) \quad (2.21)$$

Where f is any function of only x_i and $h(\vec{x})$ is any function of any? If this condition is satisfied, the function is as easy to optimize as decomposable

functions. This is because the solution for x_i can be obtained independently of all other parameters.

- 5) **Dimensionality:** Generally, the complexity of an optimization problem depends on its dimension. As the number of parameter dimensions increases, the optimization search space increases correspondingly making exploration process difficult. For a highly non-linear optimization problem, this dimensionality may prevent the optimization algorithm from having effective true knowledge of the optimization problem.

Using these basic features, this research employed a total of thirty-nine test functions is employed for evaluating the performance of the developed algorithm. The mathematical models of these functions are described as follows:

1) Ackley

This function is one of the commonly used test function which has a lot of local minima around the global minima. This test function is expressed as (Salawudeen, 2015)

$$f_{Ackley}(x) = -20 \exp\left[-\frac{1}{5} \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2}\right] - \exp\left[\frac{1}{n} \sum_{i=1}^n \cos(2\pi x_i)\right] + 20 + e \quad (2.22)$$

Where $n = 1, 2, \dots$, and $-32.768 \leq x_i \leq 32.768$ for $i = 1, 2, \dots, n$.

The global minimum of this function is $f_* = 0$; at $x_* = (0, 0, \dots, 0)$

2) Adjiman

The Adjiman function is a non-separable, continuous, differentiable, non-scalable and multimodal optimization function given mathematically as

$$f(x) = \cos(x_1) \sin(x_2) - \frac{x_1}{(x_2^2 + 1)} \quad (2.23)$$

The minimum of this function is located at $f(x^*) = -2.02181$ which occurs at $x^* = (2, 0.10578)$ subject to the constraint $-1 \leq x_1 \leq 2$ and $-1 \leq x_2 \leq 1$

3) **Alpine F1**

The Alpine function is a multimodal non-scalable, separable, non-differentiable continuous applied mathematical benchmark function given mathematically as:

$$f(x) = \sum_{i=1}^D |x_i \sin(x_i) + 0.1x_i| \quad (2.24)$$

The global minimum of this function is located at the origin $x^* = (0, \dots, 0)$, with an optimum value of $f(x^*) = 0$ subject to the constraint $-10 \leq x_i \leq 10$

4) **Beale**

The Beale function is a continuous, non-separable, non-scalable, differentiable and unimodal optimization benchmark function used for validating the performance of optimization algorithms.

$$f(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2 \quad (2.25)$$

This function is constrained by $-4.5 \leq x_i \leq 4.5$. The global optimum $f(x^*) = 0$ is located at $x^* = (3, 0.5)$.

5) **Bird**

The Bird function is differentiable, continuous, non-scalable, non-separable, and multimodal benchmark function used for evaluating the performance of optimization algorithms.

$$f(x) = \sin(x_1)e^{(1-\cos(x_2))^2} + (x_1 - x_2)^2 \quad (2.26)$$

This function is constrained at $-2\pi \leq x_i \leq 2\pi$ with a global minimum of

$$f(x^*) = -106.764537 \text{ which occur at } (x_1^*, x_2^*) = (-1.58214, -3.13024)$$

6) Bohachevsky

The Bohachevsky function is a multi-modal benchmark minimization function used for evaluating the performance of optimization algorithm. The function is represented mathematically as

$$f(x) = \sum_{i=1}^{n-1} [x_i^2 + 2x_{i+1}^2 - 0.3 \cos(3\pi x_i) - 0.4 \cos(4\pi x_{i+1}) + 0.7] \quad (2.27)$$

where, n is the number of dimensions. The global minimum and other information about this function are described in chapter 3.

7) Booth

The function is also a non-separable, non-scalable, continuous, differentiable and unimodal given mathematically as follows

$$f(x) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 \quad (2.28)$$

The global optimum of this function is $f(x^*) = 0$ which occurs at

$(x_1^*, x_2^*) = (1, 3)$ with a constraint of $-10 \leq x_i \leq 10$.

8) Box

The box function is global optimization function which is usually used for testing the minimization capability of optimization algorithms. This function is multi-modal with the following mathematical expression

$$f(x) = \sum_{i=1}^k \left(e^{-0.1(i+1)x_1} - e^{-0.1(i+1)x_2} - \left[\left(e^{-0.1(i+1)} \right) - e^{-(i+1)x_3} \right]^2 \right) \quad (2.29)$$

Detailed information about the test will be presented in chapter 3.

9) Bukin F6

The Bukin N.6 function is a multi-modal test function with several local minima around the global minima. It is also a continuous, differentiable non-separable and non-scalable function. The mathematical expression describing this function is as follows (Hansen, 2006):

$$f(x, y) = 100\sqrt{|y - 0.01x^2|} + 0.01|x + 10| \quad (2.30)$$

Bukin function has a global optimal of $f(-10, 1) = 0$ with the constraint of $-15 \leq x \leq -5$ and $-3 \leq y \leq 3$ (Hansen, 2006).

10) Colville

The Colville is a continuous, multi-dimensional test function which is used for evaluating the performance of optimization algorithm. The mathematical representation of the Colville function is given as

$$f(x) = 100(x_1 - x_2)^2 + (x_1 - 1)^2 + (x_3 - 1)^2 + 90(x_3^2 - x_4)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1) \quad (2.31)$$

The global minimum of this function is $f^* = 0$ which occurs at $x^* = (1, 1, 1, 1)$ subject to the constraint of $-10 \leq x_i \leq 10$ for $i = 1, \dots, 4$

11) Cosine Mixture

The Cosine Mixture (CM) function is a multimodal optimization test function described using the following (Salawudeen, 2015; Wu *et al.*, 2011):

$$f(x) = (-0.1) \times \sum_{i=1}^n \cos(5\pi x_i) + \sum_{i=1}^n x_i^2 \quad (2.32)$$

Where $n = 1, 2, \dots$, and $i = 1, 2, \dots, n$ with a global minimum of -3 for $f_* = -3$ at $x_* = (0, 0, \dots, 0)$.

12) Cross-in-tray

the cross-in-tray is continuous, non-convex, non-differentiable and multimodal optimization benchmark function used for evaluating the performance of optimization algorithm. The mathematical representation of this function is given as

$$f(x, y) = -0.0001 \left(\left| \sin(x) \sin(y) \exp \left(\left| 100 - \frac{\sqrt{x^2 + y^2}}{\pi} \right| \right) + 1 \right| \right)^{0.1} \quad (2.33)$$

This function has four global minimal $f(x^*) = -2.06261218$ which occurs at $x^* = (\pm 1.349406685353340 \pm 1.349406685353340)$

13) Dejong F4

Dejong 4th-order function is represented as follows (Salawudeen, 2015; Wu *et al.*, 2011):

$$f(x) = \sum_{i=1}^n i \times x_i^4 \quad (2.34)$$

$$-5.12 \leq x_i \leq 5.12$$

The function has the global minimal $f_* = 0$ at $x_* = (0, 0, \dots, 0)$ (Ben-Tal & Nemirovski, 2001).

14) Easom

The Easom benchmark function is a unimodal function, whose global optimum lies in a small region relative to the entire search space. The function was developed for testing the capability of optimization algorithm on minimization problem. the function is given as:

$$f(x_1, x_2) = -\cos(x_1) \cos(x_2) \exp \left(-(x_1 - \pi)^2 - (x_2 - \pi)^2 \right) \quad (2.35)$$

The Easom's function is constrained to a square area of $-100 \leq x_1 \leq 100$ and $-100 \leq x_2 \leq 100$. The global minimum is $f(x^*) = -1$ which is obtained for $(x_1^*, x_2^*) = (\pi, \pi)$.

15) Egg Crate

Egg Crate: (Continuous, Separable, Non-Scalable)

$$f(x) = x_1^2 + x_2^2 + 25(\sin^2(x_1) + \sin^2(x_2)) \quad (2.36)$$

Subject to $-5 \leq x_i \leq 5$ and the global solution of the decision variable is located at $x^* = f(0,0)$ with an optimum of $f(x^*) = 0$

16) Ellipsoid

The axis parallel ellipsoid is a benchmark function also called the weighted sphere function. The ellipsoid is continuous, convex and uni-modal benchmark function with the following equation (Wu *et al.*, 2011):

$$f(x) = \sum_{i=1}^n i^2 x_i^2 \quad (2.37)$$

This function has global minimal $f_* = 0$ at $x_* = (0,0,\dots,0)$ (Li *et al.*, 2013)

Where $i = 1,2,\dots,n$ and $i = 1,2,\dots,n$ for $-5.12 \leq x_i \leq 5.12$.

17) Goldstein price

This function is continuous, non-separable, differentiable, multimodal and non-scalable global optimization test function. The function is defined by two decision variables as follows

$$f(x_1, x_2) = \left[1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2) \right] \times \left[30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 - 36x_1x_2 + 27x_2^2) \right] \quad (2.38)$$

The constraint of this function is defined as $-2 \leq x_1 \leq 2$ and $-2 \leq x_2 \leq 2$ the global minimum of this function is equal $f(x) = 3$ and is obtained for

$$(x_1, x_2) = (0, -1)$$

18) Griewank

Griewank's has several widespread local minima (Hansen, 2006). However, the position of the minima is generally distributed

$$f(x) = \frac{1}{4000} \sum_{i=1}^n x_i^2 - \prod_{i=1}^n \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad (2.39)$$

Where $n = 1, 2, \dots$, and $i = 1, 2, \dots, n$ for $-600 \leq x_i \leq 600$.

This function has a global minimum of $f_* = 0$ at $x_* = (0, 0, \dots, 0)$ (Tang *et al.*, 2007)

19) Holder Table 1

There are about three Holder Table benchmark function, however, the Holder Table 1 function is continuous, differentiable, separable, non-scalable and multi-modal benchmark function given as follows

$$f(x) = - \left| \cos(x_1) \cos(x_2) e^{|1 - (x_1 + x_2)^{0.5/\pi}|} \right| \quad (2.40)$$

This function has four global minima which are located at $x^* = (\pm 9.646168, \pm 9.646168)$ and the global minima is $f(x^*) = -26.920336$ subject to the constraint of $-10 \leq x_i \leq 10$

20) Kowalik

The Kowalik is a multi-dimension global optimization benchmark function used for evaluating the minimization capability of optimization algorithms. This

function is multi-modal continuous, non-scalable and non-separable written mathematically as follows

$$f(x) = \sum_{i=1}^{10} \left[a_i - \frac{x_i (b_i^2 + b_i x^2)}{b_i^2 + b_i x_3 + x_4} \right]^2 \quad (2.41)$$

where

$$a = [4, 2, 1, 1/2, 1/41/8, 1/10, 1/12, 1/14, 1/16]$$

$$b = [0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342, 0.0323, 0.0235, 0.0246]$$

The global optimum of this function is $f(x^*) = 0.00030748610$ which occurs the optimal values of the decision variable as $x^* = [0.192833, 0.190836, 0.123117, 0.135766]$

21) Levi & Montalvo F1

Levy and Montalvo (LM F1) is a multimodal optimization test function which has been widely used to verify the performance of optimization algorithm. This function is represented as follows (Momin & Yang, 2013):

$$f(x) = \left(\frac{\pi}{n} \left(10 \sin^2(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 \left[1 + \sin^2(\pi y_{i+1}) \right] + (y_n - 1)^2 \right) \right) \quad (2.42)$$

This function has the same global minimum which is $f_* = 0$ at $x_* = (0, 0, \dots, 0)$ (Momin & Yang, 2013)

22) Michalewicz

This is a class of multi-modal global optimization function which is defined by the following mathematical expression

$$f(x) = -\sum_{i=1}^2 \sin(x_i) \sin^{2m} \left(\frac{i x_i^2}{\pi} \right) \quad (2.43)$$

The optimal value of this function is $f(x^*) = -1.8013$ for $x^* = 0$ and the value of $m = 10$.

23) Matyas

The Matyas function is a unimodal, non-separable differentiable and continuous benchmark function given mathematically as

$$f(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2 \quad (2.44)$$

The global minimum of this function is $f(x^*) = 0$ which occurs subject to the following constraint $-10 \leq x_i \leq 10$.

24) Mishra

This function is a global optimization benchmark function whose global minimum is $f(x^*) = 2$. The function is continuous, non-separable, differentiable and multi-modal function given as:

$$f(x) = \left(1 + D - \sum_{i=1}^{N-1} x_i\right)^{N - \sum_{i=1}^{N-1} x_i} \quad (2.45)$$

The global minimum of this function occurs within the constraint of $0 \leq x_i \leq 1$.

25) McCormick

The McCormick function is a continuous, differentiable, non-separable multimodal benchmark function used to validate the performance of optimization algorithm. Though this function is not as common as others, but, its complexity and modality makes it suitable for the algorithm proposed in this work.

$$f(x) = \sin(x_1 + x_2) + (x_1 + x_2)^2 - \left(\frac{3}{2}\right)x_1 + \left(\frac{5}{2}\right)x_2 + 1 \quad (2.46)$$

The constraints of this function is given as $-1.5 \leq x_1 \leq 4$ and $-3 \leq x_2 \leq 3$. The global optimum of this function is $f(x^*) = -1.9133$ which is located at $x^* = f(-0.547, -1.547)$.

26) Neumaier F3

The Neumaier3 function is expressed as (Wu *et al.*, 2011):

$$f(x) = \sum_{i=1}^n (x_i - 1)^2 - \sum_{i=2}^n x_i x_{i-1} \quad (2.47)$$

This function has a lot of local minima around the global minima with a global solution of $f_* = -4930$ at $x_* = (0, 0, \dots, 0)$ (Wu *et al.*, 2011).

27) Quadratic

The Quadratic function is a multi-modal optimization test function defined mathematically as follows

$$f(x) = -3803.84 - 138.08x_1 - 232.92x_2 + 128.08x_1^2 + 203.64x_2^2 + 182.25x_1x_2 \quad (2.48)$$

This function is continuous, non-separable and scalable with an optimal solution of $f(x^*) = -3877.2418$ which occur when the values of $x^* = [0.19388, 0.485113]$.

28) Rastrigin

Rastrigin's function is based on exponential function with the addition of cosine modulation to produce many local minima. The mathematical expression for this function is as follows (Wu *et al.*, 2011):

$$f(x) = 10n + \sum_{i=1}^n [x_i^2 - 10 \cos(2\pi x_i)] \quad (2.49)$$

This function is highly multi-modal and has global minimum $f_* = 0$ at

$x_* = (0,0,\dots,0)$ with a constraint of $-5.12 \leq x_i \leq 5.12$ (Momin & Yang, 2013)

29) Rosenbrock

The Rosenbrock's valley also called Banana function is a classical optimization problem (Tang *et al.*, 2007). The global optimum of this function is inside a narrow, parabolic shaped flat and long valley. Making it trivial and hard to find. Also, convergence to the optimum solution is difficult and thus, this function has been widely used in assessing the performance of several optimization algorithms given by (Wu *et al.*, 2011).

$$f(x) = \sum_{i=1}^{29} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2) \quad (2.50)$$

This function has global minimum $f_* = 0$ occurs at $x_* = (1,1,\dots,1)$ in the domain of $-2.048 \leq x_i \leq 2.048$ where $i = 1,2,\dots,n-1$. In the 2D case, it is often written as (Momin & Yang, 2013)

$$f(x, y) = (x - 1)^2 + 100(y - x^2)^2 \quad (2.51)$$

30) Sphere

This is the simplest form of DeJong function. This function is unimodal and convex with an obvious local minimum of $f_* = 0$ at $x_* = (0,0,\dots,0)$ in a domain of $-5.12 \leq x_i \leq 5.12$ (Wu *et al.*, 2011).

$$f(x) = \sum_{i=1}^{30} x_i^2 \quad (2.52)$$

All other information about this function is detailed in chapter 3.

31) Styblinski's Tang

This function is a multi-modal, separable, continuous and non-scalable benchmark function written mathematically as follows

$$f(x) = \sum_{i=1}^n (x_i^4 - 16x_i^2 + 5x_i) \quad (2.53)$$

The global minimum of this function is

$$f(x^*) = -39.16616570377142n \quad \text{which occurs at}$$

$$x^* = -2.9035340181590 \quad \text{under a constraint of } -5 \leq x \leq 5.$$

32) Step

The step function is a multi-modal benchmark function which is defined mathematically as follows

$$f(x) = \sum_{i=1}^n (\lfloor x_i \rfloor + 0.5)^2 \quad (2.54)$$

The optimal solution to this function is given by $f(x^*) = 0$ for value of $x^* = 0.5$

which is constrained as $-100 \leq x \leq 100$.

33) Sal

Sal function is another multi-modal optimization test function considered in this research. The function is described as follows (Wu *et al.*, 2011):

$$f(x) = 1 - \cos\left(2\pi \sqrt{\sum_{i=1}^{30} x_i^2}\right) + 0.1 \sqrt{\sum_{i=1}^{30} x_i^2} \quad (2.55)$$

The Sal function has the global minimum $f_* = 0$ at $x_* = (0, 0, \dots, 0)$ in the domain $-2\pi \leq x_i \leq 2\pi$ (Li *et al.*, 2013).

34) Schaffer

Schaffer is a multimodal non-convex optimization test function expressed in the following equation (Wu *et al.*, 2011):

$$f(x) = \sum_{i=1}^{30} (x_i^2 + x_{i+1}^2)^{0.25} \{ [\sin 50(x_i^2 + x_{i+1}^2)^{0.1}]^2 + 1 \} \quad (2.56)$$

The Schaffer function has the global minimum $f_* = 0$ at $x_* = (0,0,\dots,0)$ in the domain $-100 \leq x_i \leq 100$ (Li *et al.*, 2013).

35) Schwefel

This function is deceptive because the global optimum is geometrically distant, over the optimization hyperspace from the next best local minima. Thus, the optimization algorithms are exposed to convergence towards the wrong direction (Tang *et al.*, 2007).

$$f(x) = 418.9829 \times n + \sum_{i=1}^n (-x_i \times \sin(\sqrt{|x_i|})) \quad (2.57)$$

This function has global minimum $f_* \approx -418.9829n$ occurs at $x_i = 420.9687$ in the domain of $-500 \leq x_i \leq 500$. Where $n = 1,2,\dots$, and $i = 1,2,\dots,n$ (Tang *et al.*, 2007):

36) Subert f1

There are about 4 subert benchmark functions, however, this research employed the first subert function which is given mathematically as follows

$$f(x) = \left(\sum_{i=1}^n i \cos[(i+1)x_1 + i] \right) \left(\sum_{i=1}^n i \cos[(i+1)x_2 + i] \right) \quad (2.58)$$

The optimal solution to this function is $f(x^*) = -186.7309$ which occur when the value of $x = [-7.0835, 48580]$.

37) Watson

The Watson function is a uni-modal global optimization benchmark function which is given mathematically as follows:

$$f(x) = \sum_{i=1}^{29} \left\{ \sum_{j=0}^4 \left((j-1)a_i^j x_j + 1 \right) - \left[\sum_{j=0}^5 a_i^j x_j + 1 \right]^2 - 1 \right\}^2 + x_1^2 \quad (2.59)$$

The optimum value of this function is obtained as $f(x^*) = 0.002288$ for the value of $x = [-0.0158, 1.012, -0.2329, 1.260, -1.513, 0.9928]$ which is constrained as $-5 \leq x_i \leq 5$.

38) Yang F3

The Yang F3 is a generic stochastic and non-smooth continuous benchmark function for evaluating the performance of optimization algorithms. About four yang test functions exist in literature, but for the purposed of this research, only the third function is used due to its complexity.

$$f(x) = \left[\frac{e^{-\sum_{i=1}^n (x_i)}}{\beta^{2m}} - 2e^{-\sum_{i=1}^n (x_i)^2} \cdot \prod_{i=1}^n \cos^2(x_i) \right] \quad (2.60)$$

The global values of $m=5$ and $\beta=15$ subject to the constraint of $-20 \leq x_i \leq 20$ and a global objective function value of $f(x^*) = 0$ for $x^* = (0, \dots, 0)$.

39) Zirilli

This is a uni-modal global optimization benchmark function whose global minimum is $f(x^*) = -0.3523$. The mathematical expression for this function is given by

$$f(x^*) = \frac{1}{4}x_1^4 + \frac{1}{2}x_1^2 + \frac{1}{10}x_1 + 0.5x_2^2 \quad (2.61)$$

The optimal solution to this function occurs at $x^* = (-1.0465, 0)$ with constraint of $-1 \leq x_i \leq 5$. This function is continuous, non-separable and scalable with detailed information given in row 39 of Table 2.1

The proposed SAO algorithm, the gaseous Brownian motion optimization and fruit fly optimization is applied to the thirty-nine (39) test function presented in this subsection. Thereafter, the proposed algorithm is applied to robot path planning in a static environment with a dynamic obstacle and the minimum spanning tree problem.

2.2.9 Robot pathfinding

Robot motion or path planning which is also called the “Piano mover’s problem” is a process of breaking down the desired movement into some discrete motions that satisfy certain movement constraints and optimize some performance criterion (Chung *et al.*, 2013). Developing practical control algorithms for these kinds of problems have been very challenging due to the complexities involved. Mobile robots are primarily employed for repetitive tasks, program execution and some distance waypoint navigation (Ni *et al.*, 2016). When an autonomous robot navigates from an initial point to a destination point in an environment, it is necessary to plan a feasible path avoiding obstacles in its way while satisfying some criterion of autonomy requirements such as thermal, energy, time and safety (Rashid *et al.*, 2016). Due to the complexities involved, the piano mover problem search environment has been classified into the static and dynamic environment (Abdelkader *et al.*, 2014; Achtelik *et al.*, 2014).

1) Static environment

In this environment, the positions of the obstacles are fixed and the robot already has the prior knowledge of these obstacles as it trails the pre-planned path

hoping to avoid a collision. Since the environmental information is available to the robot, the robot can plan the optimal path a priori.

2) **Dynamic environment**

In the dynamic environment, the nature and conditions of the obstacles are not known. In most cases, some of the obstacles are in a fixed position while some are moving in a random direction. This usually makes a collision-free movement of the robot in this environment very difficult.

In either of the environments, an efficient pathfinding algorithm is required to ensure an optimized path while minimizing time requirement and distance travelled. Two of the popular methods of robot path planning are the visibility graph algorithm and the artificial potential field (Ma *et al.*, 2015; Mandal *et al.*, 2013; Masehian & Amin- Naseri, 2004; Ninomiya *et al.*, 2015).

2.2.9.1 *Visibility graph algorithm*

The concept of visibility graph involves the extraction of important lines in the free space which connects some properties of an object to another object. The graph is usually constructed by joining the edges which are visible between two vertices. The vertices (V) and the edges (E) form a visibility graph as (Ma *et al.*, 2015; Mandal *et al.*, 2013; Masehian & Amin- Naseri, 2004; Ninomiya *et al.*, 2015):

$$VG = \{V, E\} \tag{2.62}$$

In the original form of the visibility graph, the features are vertices of polygon obstacles and the visibility graph contain $O(n^2)$ edges which can be constructed in $O(n^2)$ time in a 2-dimensional (2D) space, where n is the number of features (Becerra *et al.*, 2015; Masehian & Amin- Naseri, 2004).

Usually, the shortest path of a robot to the goal can be determined in $O(n^2 \log n)$ using the A* algorithm with the Euclidean distance to the goal as the objective function as follows:

$$f(x) = \min(\|X_j - X_i\|) \quad (2.63)$$

where X_j and X_i are two different positions of the robot in the search area.

The A* (A-star) pathfinding algorithm was proposed as a result of the quest to find an improved method for path planning. The A* algorithm is the most popular choice for pathfinding due to its flexibility and ease of implementation. The A* has similar characteristics of Dijkstra's algorithm and can be used in solving the shortest path problem. The A* can also be used as a Greedy Best-First Search algorithm, thus, can serve as a heuristic for the self-guiding problem-solving process. The formal syntax of A* Algorithm is presented in the following pseudocode (Seet *et al.*, 2004).

```

Put nodeStart in OPEN list with  $f(\text{nodeStart}) = h(\text{nodeStart})$  (initialization)
while OPEN list is not empty {
  pick from the open list the node nodeCurrent with lowest
   $f(\text{nodeCurrent}) = g(\text{nodeCurrent}) + h(\text{nodeCurrent})$ ;
  if
    nodeCurrent is nodeGoal the solution is found;
    break
  Generate each state nodeSuccessor which come after nodeCurrent
  for each nodeSuccessor of nodeCurrent {
    Set successorCurrentCost =  $g(\text{nodeCurrent}) + w(\text{nodeCurrent}, \text{nodeSuccessor})$ 
    if nodeSuccessor in OPEN list {
      if  $g(\text{nodeSuccessor}) \leq \text{successorCurrentCost}$  continue
    }
    elseif (nodeSuccessor is CLOSED list) {
      if  $g(\text{nodeSuccessor}) \leq (\text{successorCurrent\_cost})$  continue
      Move nodeSuccessor from the CLOSED list to OPEN list}
    else{
      Add nodeSuccessor to the OPEN list
      set  $h(\text{nodeSuccessor})$  to be the heuristic distance to
      node_gaol
    }
    set  $g(\text{nodeSuccessor}) = \text{seccessorCurrentCost}$ 
    set the parent of nodeSuccessor to node
  }
  Add nodeCurrent to the CLOSE list
}
if (nodeCurrent  $\neq$  nodeGoal) exist with error (the OPEN list
empty)

```

Detail information on A* algorithm implementation can be found in (Flinn *et al.*, 2016;

Seet *et al.*, 2004; Sudhakara & Ganapathy, 2016; Yang *et al.*, 2016; Yao *et al.*, 2010)

As an illustration, Figure 2.13 shows a search environment of the robot with generated detectable polygon obstacles (Taizhi & Maoyan, 2017).

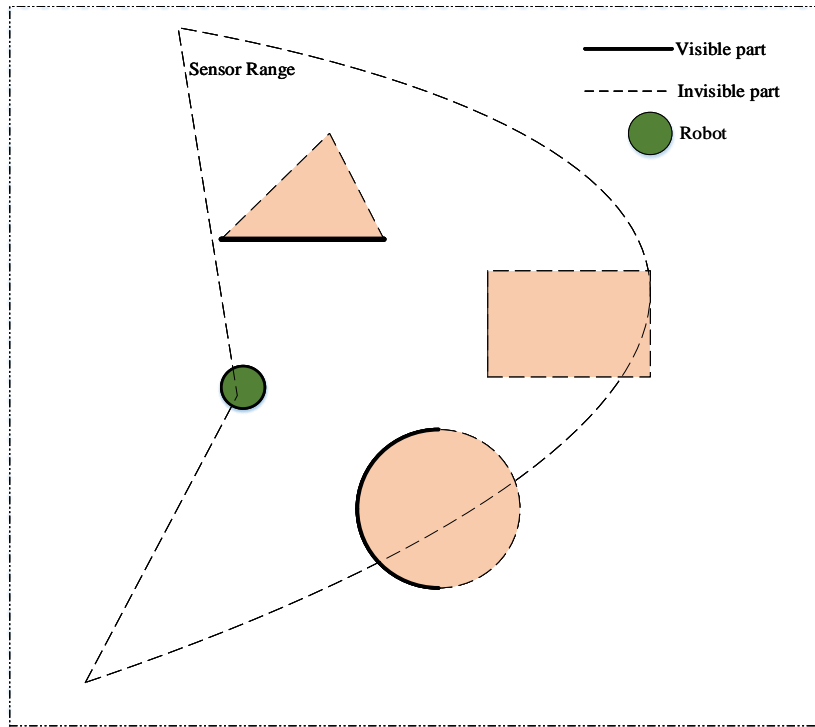


Figure 2.13: Polygon generated Robot Obstacles (Taizhi & Maoyan, 2017)

The complete visibility graph of the polygon generated obstacles given in Figure 2.13 is shown in Figure 2.14 (Taizhi & Maoyan, 2017)

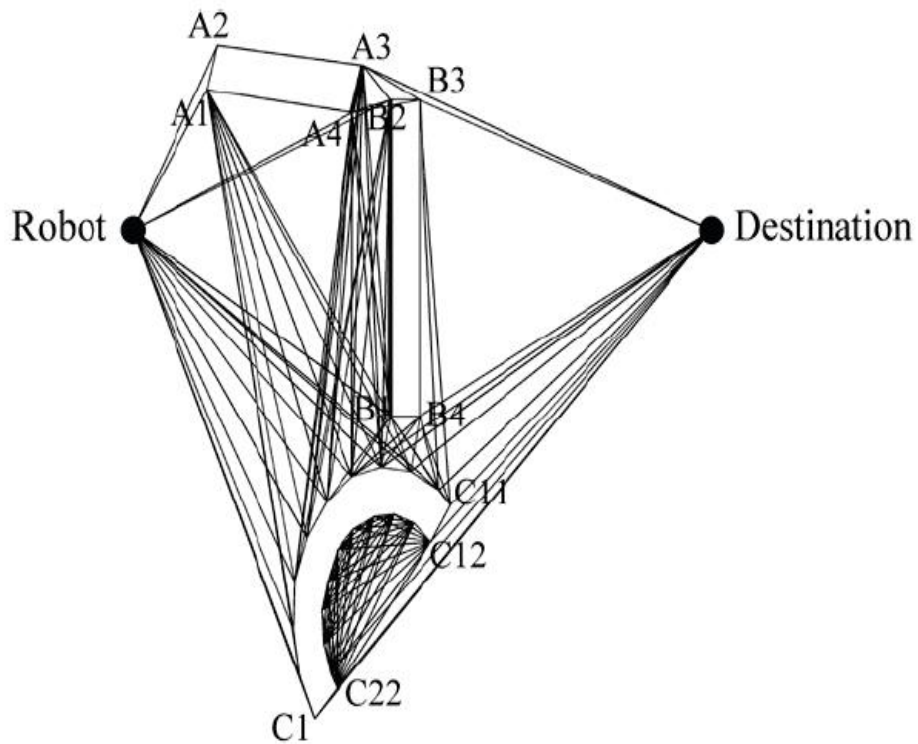


Figure 2.14: Visibility Graph of Polygon Generated Robot Obstacles (Taizhi & Maoyan, 2017)

Figure 2.14 shows the visibility graph of the polygon generated robot obstacles. This shows a total of one hundred and sixty-six vertices which the robot needs to compute and compare before choosing an optimal path from the source to the destination.

2.2.9.2 Artificial potential field

In artificial potential fields method, the robot is taken as a point represented in a configuration space and as a particle under the influence of an artificial potential field U whose local variations reflect the structure of the free space (Chen *et al.*, 2013). To make the robot attraction towards the goal more effectively, the potential field is constructed as the sum of an attractive potential pulling the robotic vehicle towards the goal and a repulsive potential pulling the robotic vehicle way from the obstacle (Masehian & Amin- Naseri, 2004), thus, the path planning is performed iteratively. During each iteration step, the artificial force evoked by the potential function at current configuration is recorded as the direction of motion and the path planning proceeds along this direction subsequently. The general form of the potential function is given as (Bentes & Saotome, 2012; Cheng *et al.*, 2015):

$$U = U_{att} + U_{rep} \quad (2.64)$$

Where

U_{att} is the attractive potential which pulls the robot towards the target position and U_{rep} is the repulsive potential which pulls the robot away from the obstacle.

The artificial vector field force $F(q)$ of the potential field is given by the gradient of the potential function as (Bentes & Saotome, 2012; Cheng *et al.*, 2015):

$$F(q) = -\nabla U_{att} + \nabla U_{rep} \quad (2.65)$$

In this manner, $F(q)$ can be defined as the sum of the vector force of the attractive potential and vector force of the repulsive potential as (Bentes & Saotome, 2012; Cheng *et al.*, 2015):

$$F(q) = F(q)_{att} + F(q)_{rep} \quad (2.66)$$

The attractive potential field of the robot is given by (Bentes & Saotome, 2012; Cheng *et al.*, 2015):

$$U_{att} = \frac{1}{2} \xi d^2 \quad (2.67)$$

where $d = |q - q_a|$; q is the current position of the robot, q_a is the position of an attractive point and ξ is the adjustable constant.

The corresponding attractive force function is given by (Zhu *et al.*, 2013; Ziaei *et al.*, 2014):

$$F_{att}(q) = -\nabla U_{att} = -\xi(q - q_a) \quad (2.68)$$

Also, the repulsive potential field of the robot is given by (Zhu *et al.*, 2013; Ziaei *et al.*, 2014):

$$U_{rep} = \begin{cases} \frac{1}{2} \eta \left(\frac{1}{d} - \frac{1}{d_0} \right)^2 & d \leq d_0 \\ 0 & d > d_0 \end{cases} \quad (2.69)$$

The corresponding repulsive force function is given by (Zhu *et al.*, 2013; Ziaei *et al.*, 2014):

$$F_{rep}(q) = -\nabla U_{rep} = \begin{cases} \frac{1}{2} \eta \left(\frac{1}{d} - \frac{1}{d_0} \right) \frac{(q - q_0)}{d^3} & d \leq d_0 \\ 0 & d > d_0 \end{cases} \quad (2.70)$$

where $d = |q - q_0|$ for the robot position q and the obstacle position q_0 . d_0 is the influencing distance of the force and η is an adjustment operator.

Over the years, both the visibility graph and artificial potential methods have been extensively used in robot path planning due to their simplicity, mathematical elegance and ease of implementation. However, both methods suffer from oscillations which subsequently lead the robot into local minima under some obstacle configurations. In order to address this challenge, researchers have employed CI techniques for robot or trajectory path planning under different environments and promising results have been achieved. For example, an efficient mobile robot path planning scheme using ant colony optimization (ACO) was presented in (Habib *et al.*, 2016; Rashid *et al.*, 2016; Wang *et al.*, 2017). A hierarchical mobile robot path planning using a multi-objective PSO was presented in (Mac *et al.*, 2017). An improved genetic algorithm (GA) based robot path planning was presented in (Ni *et al.*, 2016) etc. All these and other CI-based methods have provided promising alternatives in robot path planning techniques. Thus, the SAO proposed in this research is employed for robot planning in a static environment with dynamic obstacle position for improved path planning.

2.2.10 Minimum spanning tree problem

The minimum spanning tree (MST) is one of the typical and well known fundamental problems in combinatorial optimization. This problem finds application in virtually all fields of engineering such as communication engineering, power system engineering, process engineering etc. The minimum spanning tree problem is generally stated as Given a weighted undirected graph which represents cities, the minimum spanning tree problem is to select for construction a set of communication links that would connect all

the cities and have minimum total cost or minimum total length (Graham & Hell, 1985). The weighted graph in this case is a graph whose node represent cities, the edges represent communication links and the edge weights represent the cost of construction or lengths of the links (Graham & Hell, 1985; Sheng *et al.*, 2017). In order to clearly understand the minimum spanning tree problem, it is important to be familiarized with some of the important terminologies, such as, a tree, a spanning tree and minimum spanning tree. Consider undirected weighted graph haven five nodes (A, B, C, D and E) and eight edges with their corresponding weight given in Figure 2.15, a tree is defined as a connected subgraph, which is obtained from a given connected graph (Figure 2.15) and must not necessarily contain all the nodes and vertices of that original graph.

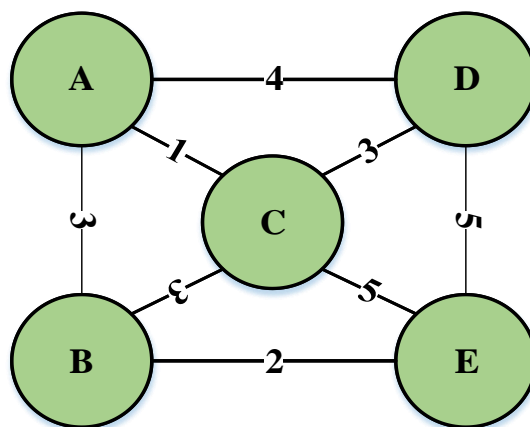


Figure 2.15: An Undirected Weighted MST graph

As an illustration, Figure 2.15 shows three possible trees which can be deduced from the given connected graph of Figure 2.15

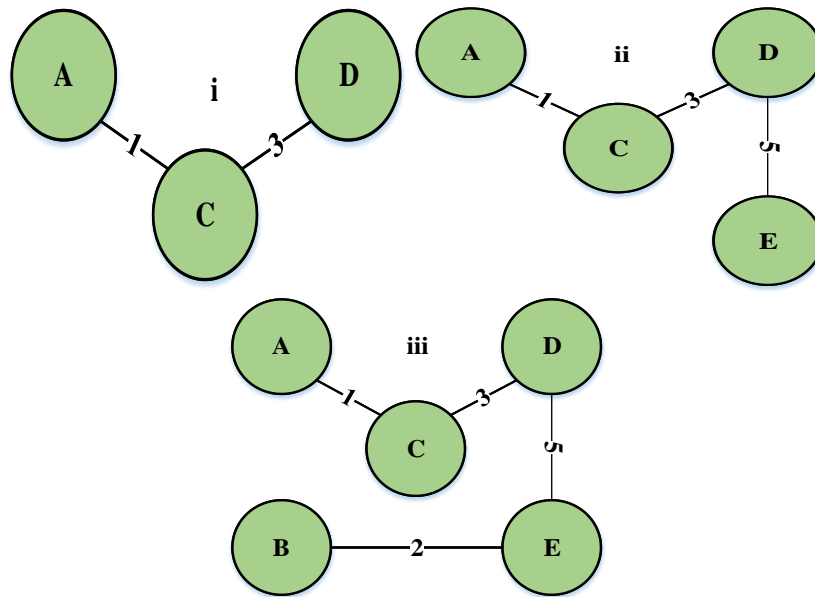


Figure 2.16: Trees Extracted from Figure 2.15

The spanning tree is a connected subgraph which is obtained from a given connected graph and contained all the vertices (node) of that original graph. Therefore, a spanning tree has n vertices, $n-1$ nodes and is connected. Thus, the spanning tree which has the minimum amount of weight is called the minimum spanning tree. Therefore, two minimum spanning trees can be deduced from the given connected graph of Figure of 2.15. Both minima spanning tree has a weight $W=11$ as shown in the third tree of Figure 2.16 and the given tree in Figure 2.17

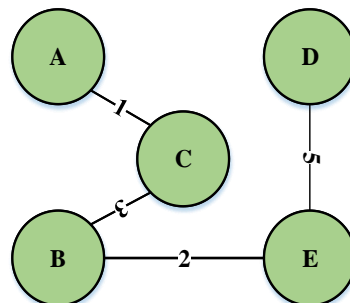


Figure 2.17: Minimum spanning tree of Figure 2.15

Two of the most widely used generic algorithms for solving the MST problem are the Kruskal's and Prim's algorithms. The basic concept of each of these algorithms is discussed in the following subsection.

2.2.10.1 Kruskal's algorithm

The Kruskal's algorithm is an algorithm in graph theory which employed the greedy approach to finding the minimum spanning tree of a connected weighted graph. Given a connected undirected graph, the Kruskal's algorithm is a subset of the edges that forms a tree which includes every vertex, where the total weight of all the edges in the tree is minimized. If the given graph is not connected, then the Kruskal's algorithm may find a minimum spanning forest (Nešetřil *et al.*, 2001; Sheng *et al.*, 2017). The basic framework of the Kruskal's algorithm are highlighted as follows (Held & Karp, 1970; Nešetřil *et al.*, 2001; Sheng *et al.*, 2017):

- 1) Create a forest T (a set of trees), where each vertex in the graph is a separate tree
- 2) Create a set S containing all the edges in the graph
- 3) While S is nonempty and T is not yet spanning
 - a) Remove an edge with minimum weight from S
 - b) If that edge connects two different trees, then add it to the forest combining two trees into a single tree otherwise discard that edge.

2.2.10.2 Prim's algorithm

Just like the Kruskal's, the Prim's algorithm also employed the greedy algorithm to find the minimum spanning tree of a weighted undirected graph. It finds a subset of the edges that forms a tree which includes every vertex, where the total weight of all the edges in the tree is minimized (Delaite & Pesant, 2017; Mashreghi & King, 2017;

Nešetřil *et al.*, 2001). The algorithm operates by building this tree taking one vertex at a time, from an arbitrary starting vertex, at each step adding the cheapest possible connection from the tree to another vertex. The basic framework of the Prim's algorithm is highlighted as follows:

- 1) Identify the Arcs (edges) with the minimum weight
- 2) Then separate the vertices into two set

$$S_1 : \{Set\ of\ nodes\ with\ min.\ weight\}$$

$$S_2 : \{Set\ of\ all\ other\ nodes\}$$

- 3) Determine the connectivity between set S_1 and S_2 and add the connectivity with the minimum weight into the spanning tree
- 4) Update the set S_1 until the hole element of set S_2 are eliminated
- 5) Stop the algorithm when all the vertices are completed in the spanning tree or when set S_2 becomes a null set.

Researches have shown that, both Kruskal's and Prim's algorithms guarantee a minimum spanning tree. However, the efficiency of both approaches decrease significantly when the number of vertices and edges in the graph becomes very large and capacitated. Since both algorithms employed the greedy approach in determining the MST, the reliability of the algorithm in practical problem is not guaranteed. Thus, researches have over the years, employed various nature inspired algorithm such as GA (Salgueiro *et al.*, 2017; Zhou & Gen, 1999), artificial bee colony (Bindima & Elias, 2017; Singh, 2009) PSO (Awad *et al.*, 2017; Tsai, 2017) in solving the MST problem. Therefore, the SAO is used to develop an efficient MST which will guarantee the minimum weight.

2.3 Review of similar works

The review is based on works focussed on development of CI-based algorithms highlighting the underlying mechanisms (how the agent identifies the “food source” and the influence of the “food source” on the behaviour of the agent), strengths and weaknesses that necessitated the development of modified versions or subsequent algorithms.

Eberhart and Kennedy (1995) developed a novel optimization algorithm called the PSO by careful observation of the biological phenomena based on the behaviour of school of fish and flock of birds. The PSO is initialized by a set of randomly generated initial population of particles. Each particles in the swarm is also assigned a random velocity and then the potential solutions are flown through the hyperspace. Each particle in the swarm keeps track of its coordinate in the search space which is affiliated with the best fitness (solution) it has achieved so far and this value is termed the personal best “*pbest*”. The particles also collectively keep track of the overall best fitness (solution) and its location in the swarm. This overall best solution is termed as the global best “*gbest*”. At each time step in the PSO, a particle changes its velocity (or acceleration) as it moves towards the “*pbest*” and the “*gbest*” values. The performance of the PSO algorithm over several engineering problems has demonstrated its effectiveness in solving non-linear and complex optimization problems. However, the original PSO had significant chances of getting trapped into local minima due the imbalance between intensification (exploitation) and diversification (exploitation). This is usually attributed to the lack of guiding (control) parameter as the algorithm swarm through the optimization hyperspace. Over the years, several researchers have proposed

modifications to the PSO with a common goal of improving its precision and rate of convergence. For example, Basu (2015) developed an improved PSO for solving non-convex economic load dispatch problem. In this work, a Gaussian random variable is introduced in the generation of particles velocity. This is with the hope of improving the searching efficiency and ensuring a high probability of obtaining the global optimum without hindering the acceleration convergence and increasing the complexity of the structure of PSO. Also, Norouzi *et al.* (2016) presented a modified PSO for time dependent vehicle routing problem (TDVRP) with the aim of minimizing travel time and reducing carbon emission. Crossover and mutation operators were introduced in order to increase the diversity of the original PSO by replacing half of the worst individual with a newly generated particle. In the work of Delice *et al.* (2017) a modified PSO algorithm with negative knowledge is proposed for solving the mixed-model two sided assembly line balancing problem. A combine selection and decoding procedure was employed at the generation stage of the PSO. This enhanced the exploration capability of the original PSO and also allowed the PSO to search at different points in the hyperspace. However, in both the standard and modified PSO, attention is focused on modelling the movement of the particles towards the food source. This usually does not depict the real-life situation, because, in most cases, the behaviour of the particles in the swarm is usually influenced by the nature, location and concentration of foods.

lei Li (2001) developed a novel metaheuristic algorithm using the intelligent swarming behaviour of swarm of fish called the artificial fish swarm algorithm (AFSA). In the developed AFSA, the swarming behaviour of fish is classified into three distinct behaviours (preying swarming and chasing). In each of these behaviours, three

communication rules (synergetic rule, reconnaissance rule and stochastic rule) were employed to decide the movement of each fish in the algorithm. An appropriate control parameter such as *visual* which is a function how far the food source from the location of the fish, *step* which penalize the amount of movement the fish will take to get to the food source and crowd factor, which determine the concentration of other fishes around the food source. At the initial stage of AFSA, the population of the artificial fish were generated randomly and are placed in the hyperspace for foraging. Then, the preying behaviour is implemented. If the solution obtained during preying is not better than that obtained during foraging then, the swarming behaviour is implemented. Also, the solution obtained during swarming is not better than that obtained during preying, the chasing behaviour is implemented. This process continued until the best food (solution) source is obtained. The performance of the algorithm over several optimization problems demonstrated the efficiency and capability of the AFSA. Just like the PSO, the major challenge of the AFSA is the ease at which the algorithm fell into local minima. This is attributed to the constant effect of the control parameters when the algorithm evolved from generation to generation. To address these challenges, several researchers have proposed modifications to the standard AFSA algorithm. In Salawudeen (2015) an improved cultural AFSA (with four variants) called the weighted AFSA (wAFSA) was developed using the normative and situational knowledge inherent in cultural algorithm (CA) to increase the diversification of the algorithm. An inertial weight was introduced such that the control parameters could be selected adaptively and a crossover operator was introduced such that, a generated offspring could inherit some of the characteristics of its parents. Simulation results on standard benchmark functions showed a significant improvement over the original AFSA. The work presented in Luan *et al.* (2016) introduced a three parameter Lorentzian function into the AFSA with the hope of

improving its visual distance through adaptive selection. At the initial stage of the AFSA, a large visual distance is selected to enhance both the searching capability and convergence of the algorithm while at the later stage of AFSA, a small visual distance is adopted to improve the local searching ability and the accuracy of the optimal solution. Also, to keep a balance between the iteration speed and precision, a variable step adaptive operator is introduced. Xian *et al.* (2017) presented a modified AFSA for fuzzy time series forecasting in which, the chemotaxis behaviour of bacterial foraging was employed in the foraging phase of the AFSA. Thereafter, the Levy flight was introduced as a mutation operator for a mutation strategy. Simulation results demonstrated the efficiency of this approach. However, modelling only the foraging behaviour of fish towards the food source did not truly depict the real-life relationship between the fish and the food. For example, in the preying, swarming and chasing behaviours of fish, only the regions of high concentration of food is considered in modelling these behaviours while the nature and environmental conditions of the food source were not considered.

Passino (2002) developed a new optimization algorithm called the bacteria foraging optimization (BFO) based on the foraging behaviour of *E. coli* bacteria. At the initial stage of the BFO, a set of artificial bacterial are randomly generated. The randomly generated bacteria forage and communicate between one another by sending signals using four basic stages (chemotaxis, swarming, reproduction, elimination and dispersal). The chemotaxis stage describes the movement of the bacteria through swimming and tumbling. In the swarming stage, the bacterium congregates and moves in groups towards the location of high nutriment. In the reproduction stage, fitter bacterium survives and those bacteria that are least healthy die. In the elimination and dispersal,

bacteria with very low probabilities are eliminated and new bacteria are randomly generated for replacement. This procedure continues until the termination criteria are reached. The performance of BFO on various optimization problems has demonstrated the efficiency of the algorithm in comparison with candidate nature inspired optimization algorithms. However, the major challenge of the standard BFO is the constant effect of its step size parameter resulting in an unnecessary chemotactic step as the BFO moves close to the optimal solution. In order to avoid this, researchers have proposed various modifications on the standard BFO. Chen *et al.* (2017) developed two modified BFO algorithms for feature selection called adaptive chemotaxis BFO (ACBFO) and improved swarming and elimination-dispersal BFO (ISEDDBFO) respectively. In the ACBFO, the bacterium is redefined to establish mapping relationship between the bacterium and the feature selection subset. In the ISEDDBFO, a hyperbolic tangent function was introduced to enhance accurate relation between cell to cell attraction-repulsion. In the elimination-dispersal phase, roulette wheel technique was introduced in order to retain the primary features of the eliminated bacteria. The performance of the algorithm was compared with similar optimization methods and results demonstrated the efficiency of the proposed approach. Tang *et al.* (2017) developed a multilevel thresholding approach based modified BFO for enhancing the applicability and practicality of optimal thresholding techniques. A diversity of solution is adopted in the reproduction stage to ensure that, weak bacterium randomly selects a strong bacterium from the healthiest bacteria as its moves towards the global solution. The idea of PSO is introduced at the chemotactic step to strengthen the global searching capability and quicken the convergence rate of the BFO. Finally, the modified BFO was applied in maximizing the Tsallis thresholding function in order to obtain the optimal threshold. Simulation results were compared with similar algorithms and results showed

that the modified BFO required less computation time. However, the inherent constant effect of the BFO step size was not effectively addressed. In addition, the direct relationship between the nature of the nutrient and the behaviours of the bacteria based on this relationship were not considered in the modifications of the BFO.

Dorigo and Stützle (2003) studied the ant system in 1992 which eventually led to the development of an optimization algorithm called the ACO. The idea of ACO was inspired by the pheromone trail laying-and-following behaviour of real ants which use pheromones as a medium of communication. The pheromone trail in the ACO served as the distributed numerical information which the ant used to stochastically provide solution to several problems. The ACO adapted this technique during its execution to update its searching experience in order to collectively search for solution in an optimal form. Just like others, the major challenge of the ACO is the ease at which the algorithm fell into local minima. This is a result of the constant values of its control parameters which are the pheromone influencer, heuristic matrix influencer, the evaporation rate and the pheromone decay coefficient. If proper values of these parameters were not properly selected with respect to a particular problem, the performance of the ACO is greatly influenced. To address this challenge the work presented in Ma *et al.* (2017) developed a modified ACO framework for delimiting optimal urban growth boundaries. A dynamic process and planning intervention was introduced into the standard ACO in order to overcome or reduced the constant effect of the control parameters. Simulation results showed that the approach proposed was more efficient in comparison with the standard ACO. However, the quality and nature of the deposited pheromone by the leading ant were not considered in the modifications of the ACO.

Pan (2012) presented a novel optimization algorithm based on the foraging behaviour of fruit flies towards food. Due to sense (osphresis) and vision capability of the fruit fly, it can sense the location of food up to about 40km away. At the initial stage of the fruit fly optimization algorithm (FFOA), a set of initial fruit flies is generated randomly. A random direction and distance for the search of food is then assigned using osphresis by an individual fly. The food location distance to the estimated and the smell concentration judgement value is calculated. The fitness of the smell concentration judgement value is then calculated and the smell concentration of the individual fruit fly location is determined. By this, the fruit fly location with maximum smell concentration can be determined and all other fruit flies will fly toward this location. This procedure is repeated until the overall best fruit fly is determined and termination condition is reached. The FFOA has a simple structure and uses few parameters which make its convergence toward optimum solution efficient. Due to its simple structure, it may fall into local convergence with slow searching speed, poor stability and low efficiency when applied to complex high dimensional optimization problems. To address these shortcomings, researchers have proposed various modification of the FFOA. In the work of Jiang *et al.* (2017) a dynamic variable step size is introduced in order to increase the searching capability of the FFOA and its performance was evaluated using a total of five standard optimization test functions before applying the algorithm in identifying the parameters of Jiles-Atherton (J-A) flux gate sensor model. Simulation results demonstrated the effectiveness of the modified FFOA over the standard FFOA. Also, Xiang *et al.* (2017) presented a hybrid location information exchange mechanism for improving fruit fly swarm diversity in a more efficient way called the hybrid fruit fly optimization (HFFOA) in order to improve the global searching and local searching abilities of the algorithm. Also, a mutation strategy called cataclysm policy was

designed to help the flies jump out of the local extreme points and move towards the global extreme points. Simulation on eighteen (18) benchmark functions demonstrated the efficiency of the developed approach. However, the smell concentration and the density of the medium (air) through which the fruit smell evaporate from its source were not considered in developing the FFOA or its variations.

Abdechiri *et al.* (2013) developed a new metaheuristic optimization algorithm which was based on the Brownian motion and turbulent rotational motion of gas called the gases Brownian motion optimization (GBMO). In the work, each position of the gas molecule represents a potential solution and the searching process is guided by properly adjusting the gases Brownian motion and turbulent rotational motion of gas. The algorithm starts with a set of randomly generated initial position and velocity of gas molecules and then each molecule is assigned a random radius of turbulence and a temperature is assigned and the turbulent rotational motion is then modelled using chaotic sequence. The algorithm is iterated until the optimum solution is obtained. Just like other algorithms, the GBMO suffered from the problem of imbalance between exploration and exploitation. To address this challenge, researchers have proposed modified GBMO using important features from other optimization algorithms. **Elyas *et al.* (2014)** developed a hybrid algorithm using the positive features of clonal selection algorithm (CSA), GBMO and PSO for local search and improving the quality of initial population respectively. The proposed algorithm was employed for solving the non-linear constrained economic load dispatch problem in power systems. Simulation results demonstrated an improved performance over the original GBMO, PSO and CSA respectively. Since the total particle in the gas molecules could not be accounted for at the end of the evaporation process, the GBMO technique did not consider a proper

guiding parameter capable of accommodating the loss of gaseous molecules during the Brownian process.

Abedinia et al. (2016) developed a new metaheuristic algorithm which was inspired by the superior hunting ability of shark in nature. This algorithm was modelled based on the smell sense of shark and its movement towards the odour source. Various behaviours of the shark in the sea were considered in the mathematical model of the proposed algorithm and the resulting algorithm is termed as shark smell optimization (SSO). In the SSO, it is assumed that, the food is injured and injects blood into the sea regularly. Therefore, the velocity of the food is negligible compared with that of the shark. At the initial stage of the SSO, a population of blood particles (which represent possible solutions to the optimization problem) is randomly generated within the feasible region and an initial velocity is assigned to the shark which enables its movement towards the stronger odour particle. The movement of the shark was modelled using the gradient of the objective function. Simulation results demonstrated the effectiveness of the SSO. Since the SSO employed the gradient of the objective function to guide its movement, it may lead to trapping in local minimum. To address this short coming, Ahmadigorji and Amjady (2016) proposed a chaotic shark smell optimization algorithm for solving a multi-year distributed generation (DG)-incorporated framework for expansion planning of distribution networks. In this work, a chaotic operator which generated four other candidate solutions aside from the standard SSO was introduced to convert the SSO into chaotic SSO. Simulation results demonstrated the efficiency of the developed approach. However, both the standard and the variant of the SSO did not consider the intensity of the flow of water through which the shark perceived the smell of blood.

Chandra (2016) presented a novel nature inspired optimization algorithm based on the training behaviour of dogs in detecting smell for solving optimization problems called the smell detection algorithm (SDA). The principle of the SDA involved the creation of an artificial surface with smell trails and the use of the dog in resolving a path around this surface. The SDA was focused on the ability of a dog to lock onto a scent using their highly developed olfactory cells and their territorial behaviours through urination. The dogs in the SDA hold two parameters, namely the signature value which is used by the dogs to mark smell spots and the radius value, which indicates the smelling ability of the dogs. In the SDA, the search environment is initialized with a randomly selected smell spot and smell values are assigned to be inversely related to the Cartesian distance between the source and the destination. At this point, the agents are initialized to each spot. The smell spot that has the highest value is chosen as the unmarked point during iteration and the SDA is moved to that spot. This procedure is repeated until the best solution is obtained. In the work, the smell is not considered as a molecule of gas which did not appropriately depict the nature and properties of smell molecules. Also, since the smell did not evaporate, the agent practically needed to have prior knowledge of every smell location in the search space which is also not a true representation of real-life situation. The major differences between the developed SAO and the SDA are that;

- 1) The agent in SDA is the dog which marks its territory by urinating and identifying an established smell spot whereas the agent in the developed SAO is any biological system that has the ability of olfaction.
- 2) In SDA, the smell is considered as a static spot distributed randomly in a rectangular cartesian coordinate whereas in SAO, the smell is considered as

molecules of gas randomly evaporated from a source with a specific velocity and position.

- 3) The SAO is developed using three distinctive behaviours: the sniffing mode which is based on the kinetic theory of gas, the trailing mode which is based on the olfaction capability of the agent and its ability to sniff the location of best and worst molecules of smell and a random behaviour which is the strategy the agent employs to escape from being trapped in a local minimum. The SDA is focused only on the territorial behaviour of dogs in identifying a smell spot.
- 4) The agent in SDA is passive and has no active participation in the computational process, whereas the agent in SAO is active and plays a significant role in the intelligent decision making during the optimization process.

Zhu and Zhang (2017) developed a new optimization algorithm which was based on the animal behavioural ecology called the optimal foraging algorithm (OFA). The OFA position is expressed as a two-dimensional variable of longitude and latitude and the foragers in the OFA can move along the two directions respectively. When a forager searches the location with the abundance of food, these foragers pass information to others through recruitment. Individual members of the OFA are generated randomly and are regarded as foragers whose positions denoted a candidate solution to the objective function. The fitness of the initial position is determined and the updated positions of each individual are determined. The intensity of this new position is compared with the previous position for judgement. If the new position is better, the position is preserved for the next foraging and its fitness is determined otherwise the previous position is retained. This process continues repeatedly over a large number of iteration until the optimum solution is obtained. The performance of the algorithm was compared with six

other algorithms including PSO and simulation results demonstrated the efficiency of the OFA. However, the movement rule and the type of movement for all the foragers were not considered in the work. Also, the influence of the food source on the behaviours of the foragers was not also considered in modelling the algorithm.

From the literatures reviewed, it is obvious that several computational intelligent algorithms which are inspired by various natural phenomena have been codified into an algorithm for solving various degree of complex optimization problems. In most cases, an algorithm is conceived through careful study of the behaviour and nature of the natural system on which the algorithm is based. This is why nearly all nature inspired computational models are named after a certain natural phenomenon (such as ant colony optimization named after the natural behaviours of ant system). The frame work of all the nature inspired computational models is usually focused on the collective foraging (swarm) behaviours of a group of agents and the formal communication between individual foragers as they move toward the location of more advantage. Perhaps, in practical situations, there is always an information communication between the location (i.e. food) source and the foragers. For example, the region of higher concentration of food tends to attract the attention of the foragers than any other region with lesser concentration of food. For an algorithm to be efficient, the natural conditions of the food source have to be considered in developing its mathematical model. Thus, in this research, the diffusion of the smell molecules from the smell source in the direction of the agent is modelled. Thereafter, the trailing behaviour of an agent in the direction of these molecules is then modelled and the resulting algorithm called the smell agent optimization (SAO). The performance of the developed algorithm is then evaluated using a subset of thirty-nine (39) applied mathematical optimization test functions. The

algorithm is used to develop an efficient path planning model for robot in static environment with a dynamic position of obstacles and solve the minimum spanning tree problem as typical examples of combinatorial optimization problems.

CHAPTER THREE

MATERIALS AND METHOD

3.1 Introduction

This chapter presents the development of the mathematical model of the proposed smell agent optimization SAO algorithm. The relevant assumptions necessary for the development of SAO are also presented. The procedures adopted for the implementation of gaseous Brownian motion optimization (GBMO) and fruit fly optimization algorithm (FFOA), which were used to benchmark the performance of SAO, are presented. All the mathematical equations governing the selected standard optimization benchmark test functions for performance evaluation are also presented. The models of the path planning cost function and the minimum spanning tree problems are also presented.

3.2 Materials

In this section, the materials used for the implementation of the research presented in this report are discussed. These materials involve the specification of the computer system used and the software used for the implementation.

3.2.1 Computer System

All the simulations performed in this research were carried out using HP Pro desktop computer situated at the Mechatronics Laboratory of the NLNG Multi-user laboratory, Ahmadu Bello University Zaria. The specifications of this computer are given in Table 3.1.

Table 3.1: Computer Specification

SN	Items	Specification
1	Operating System (OS)	Windows 7 Professional
2	RAM	4.00GB
3	System Type	32-bit Operating System
4	Processor	Intel® Pentium® CPU G2030 @ 3.00GHz
5	Rating	4.7
6	Dedicated Video Graphics	64MB

3.2.2 MATLAB

The MATrix LABoratory (MATLAB) is the software which is used to implement all the models developed in this research. Several versions of MATLAB have been developed by MathWorks. For the purpose of this research, MATLAB R2017a version was used.

3.3 Methods

In this section, the step by step procedure employed in the development of the SAO algorithm are discussed. The methods used to evaluate the test functions, the path planning and minimum spanning tree are also presented in this section.

3.3.1 Smell Agent Optimization (SAO) algorithm

The SAO was programmed into a file called the SAO_run.m using the programming platform (editor window) of the MATLAB software. All the thirty nine test functions were programmed into a function file called the Cost_function.m and each function were assigned a serial number which correspond to the function the algorithm is evaluating. The screen shot showing the MATLAB interface for the selection choice for the test functions is shown in Figure 3.1

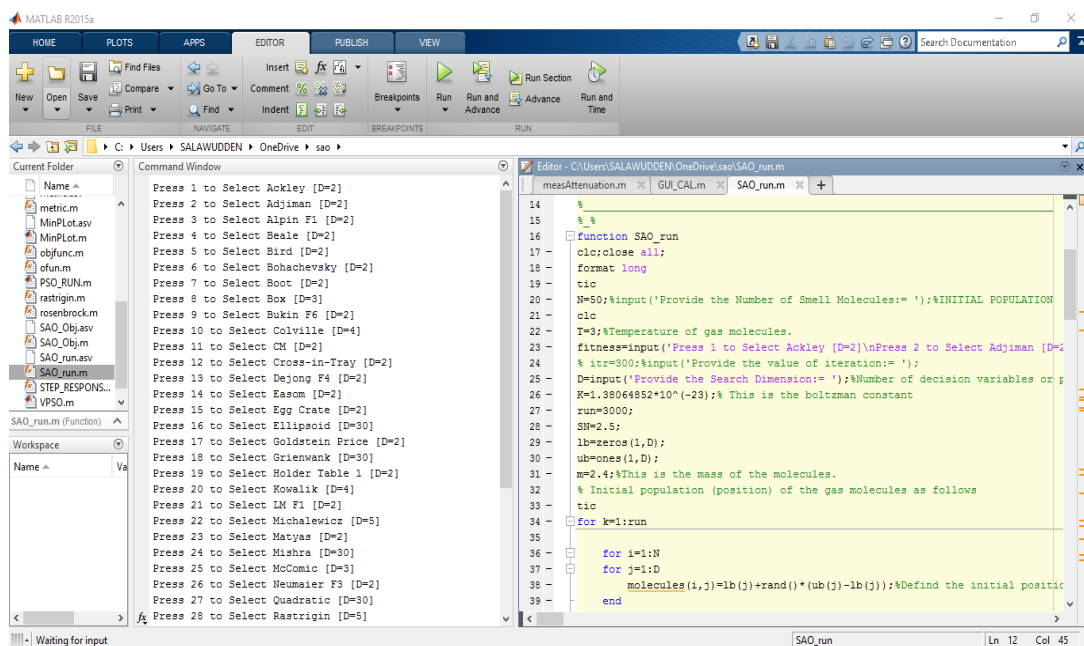


Figure 3.1 The MATLAB Interface of SAO Simulation

Figure 3.1 shows the simulation of SAO in MATLAB R2017a environment. It can be seen that when the SAO is run, the simulation will request the user to select which of the function to be evaluated. MATLAB is a powerful simulation software which has such a user-friendly environment, thus inspired its choice as the simulation software for this research.

As previously discussed in chapter one and chapter two, every biologically inspired optimization algorithm is based on a number of critical modelling parameters. These parameters are decided through careful observation and understanding of the biological system on which the algorithm is inspired. The SAO developed in this research has three sub modes:

- 1) The gaseous molecules of smell evaporate in the direction of the smell agent (SA). This is termed the “*sniffing mode*”
- 2) The SA trailing the part of the gaseous molecules of smell and eventually identify it source. This is termed the “*trailing mode*”

- 3) In case the agent loses its trail during the search, a position is selected randomly and the agent move towards this position hoping to sniff the smell molecule again. This is termed the “*random mode*”

3.3.2 Sniffing Mode

The smell agent algorithm is modelled using population (which represents the total number of molecules evaporating from the smell source), dimension (which is a representation of the search space within the region of the gaseous molecules) and velocity (which determine the movement pattern of the molecules). These determine the position and movement pattern of the smell agent, as the agent seek to identify the smell source. The detailed mathematical models of this mode are described in the subsequent subsection.

3.3.2.1 Population

The population of an optimization algorithm are usually generated randomly from a set of initial positions. In this research, the population are assumed to be a set of randomly generated molecules or particles of smell which represents the candidate solution set. The size of the population depends on the total number of molecules of smell evaporating from the smell source (decision variables). Assuming the molecular population of the smell molecule is denoted as N . If the evaporation space (search space) of the smell molecules are denoted as D , then, the initial population of the smell particles (molecules) in the search space is generated using:

$$X_i = \sigma(N, D) \tag{3.1}$$

The σ operator in equation (3.1) is a random number generator with a mean of zero and standard deviation of 1.

Individual members of this population are assigned a vector position using:

$$X_i^t = [x_{N,1}^t, x_{N,2}^t, \dots, x_{N,D}^t] \quad (3.2)$$

The position vector in equation (3.2) enables the agent to determine the region with highest concentration in the search space. For example, consider the coordinate positions given in Figure 3.2. Each cycle in the cell represents a molecule of smell. The number of columns in the figure represent the dimension (D) or solution search space to be explore while the number of rows represent the population of the smell molecule exploiting the search space.

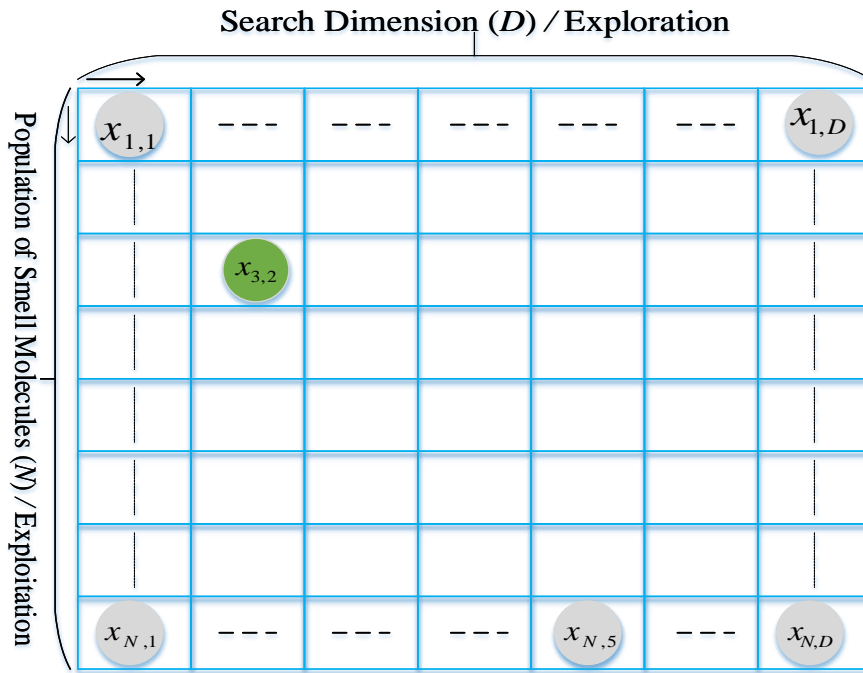


Figure 3.2: Coordinate Position Representation of the Smell Particles in the Search Space

If N is 3 and the index of the molecule with the highest concentration is $x_{3,2}^t$, this indicate that, the smell molecule with the highest concentration in this case is at the coordinate (3,2) in the entire search space indicated as cycle with green colour in Figure 3.2.

Since the smell molecules evaporate and travel through the air in the direction of the agent, each molecule can be said to maintain a uniform velocity in the direction of the agent, provided the intensity of the air medium is constant. This velocity of the smell molecule is denoted vectorially as V . The velocity-vector V is regarded as the diffusion vector (which is a displacement of the smell molecule from the smell source/origin). The velocity of the smell molecules is assigned using:

$$V_i^t = [v_{N,1}^t, v_{N,2}^t, \dots, v_{N,D}^t] \quad (3.3)$$

The velocity vector in equation (3.3), provide a suitable velocity for each molecule in the proposed SAO.

In the geometric number space, individual members in the smell population represent a possible solution that moves within the problem search space. The position of each candidate solution (smell molecule) is determined by the position vector $X_i^t \in R^N$ given in equation (3.2) and the molecules velocity $V_i^t \in R^N$ given in equation (3.3).

From, the fundamental theory of physics, velocity is the rate of change of displacement over change in time. Since the movement of smell molecules is non-uniform in a six-dimensional coordinate, the velocity is obtained as follows

$$\frac{\Delta X_{(x,y,z)}}{\Delta t} = V_{(u,v,\omega)} \quad (3.4)$$

Where;

$\Delta X_{(x,y,z)}$ is change in displacement and x, y, z are displacement coordinates.

Δt is interval of time.

$V_{(u,v,\omega)}$ is the velocity in the velocity coordinates u, v, ω .

From the definition of equation (3.4)

$$\Delta X_{(x,y,z)} = V_{(u,v,\omega)} \Delta t \quad (3.5)$$

The following is therefore obtained from equation (3.5):

$$X_{(x,y,z)}^{(t+1)} = V_{(u,v,\omega)}^{(t)} \times \Delta t + X_{(x,y,z)}^{(t)} \quad (3.6)$$

For a gas molecule in a fixed position, the change in time Δt is 1. For example, the algorithms increase its iteration progressively by 1 until all the iteration is exhausted. Putting $\Delta t = 1$ in equation 3.6 gives the generalized population of the proposed algorithm is formulated as follows

$$X_i^{(t+1)} = X_i^{(t)} + V_i^{(t+1)} \quad (3.7)$$

The component of this initial position of the smell molecule relative to Cartesian coordinate can be represented in equation (3.8)

$$X_j^{(t)} = X_{(x,y,z)}^{(t+1)} + V_{(u,v,\omega)}^{(t)} \quad (3.8)$$

Where the coordinates (x, y, z) and (u, v, ω) are the position and velocity coordinates of the smell molecule in the search space respectively. This is demonstrated pictorially as shown in Figure 3.3

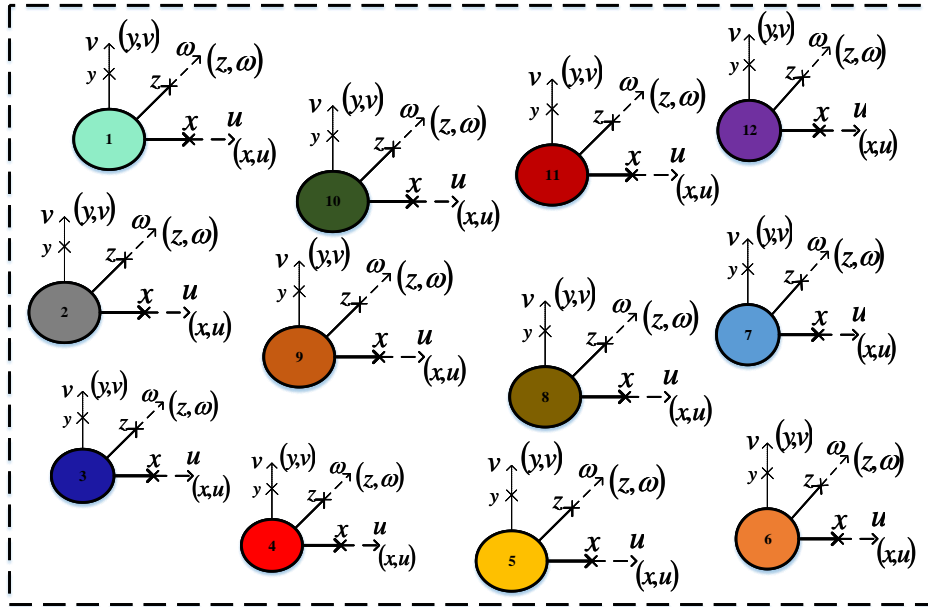


Figure 3.3: Cartesian Coordinate Representation of the Smell Molecules in the Search Space

From equation (3.5) and Figure 3.3, it can be said that, the smell molecules can move in six different coordinates a priori: three coordinates (x, y, z) representing the position and three coordinates (u, v, ω) representing the velocity. At this point, the population of the smell agent optimization algorithm have been initialized. The fitness of this population is evaluated and the agent (X_{agent}) sniffs the molecule with the best fitness and assumes this position immediately.

3.3.2.2 Updating smell velocity and position

Every smell molecules have their corresponding velocity for which they move and update their position in the search environment. As stated in chapter two, Maxwell developed a preliminary theory of gas molecules movement and provided the heuristic derivation of the velocity distribution function. Since gas molecules moves in a non-uniform manner, the velocity of the gas can be obtained from the hydrostatic pressure of the gas as

$$p = \frac{1}{2}nmv^2 = \frac{3}{2}knT \quad (3.9)$$

Where n is the amount of substance in the gas. Therefore, the velocity of the non-uniform gaseous molecules moving in the search space is obtained as

$$v = \sqrt{\frac{3kT}{m}} \quad (3.10)$$

The velocity of the smell molecule is then updated from equation (3.3) to:

$$V_i^{(t+1)} = V_j^{(t)} + r_0 \times \sqrt{\frac{3kT}{m}} \quad (3.11)$$

where; $V_i^{(t+1)}$ is the updated velocity, $V_i^{(t)}$ and $V_j^{(t)}$ are the previous and current velocity respectively. k is the Boltzmann's constant given as $k = 1.38 \times 10^{-23} JK^{-1}$, T is the temperature of the smell molecules in the environment and m is the mass of the molecules and r_0 is a random number.

The choice of these parameters, most especially temperature (T) and mass (m) are critical to performance of the developed SAO. The effect of these parameters and selection choice is discussed later in this chapter.

Thus, the current position of the agent which is the updated position of the smell molecule is given by:

$$X_i^{(t+1)} = X_j^{(t)} + \left(V_j^{(t)} + r_0 \times \sqrt{\frac{3kT}{m}} \right) \quad (3.12)$$

Where; $X_i^{(t+1)}$ is the current position of the molecule, $X_j^{(t)}$ is the previous position of molecule r_0 is a random number which stochastically guide the velocity update

Both the position and velocity in equation (3.12) are generated randomly with the same vector size. The fitness of the current position of the smell molecule given in equation (3.12) is evaluated. At this point, the agent sniffs the molecule with the best fitness again and updates its position with the most favourable fitness. The path trailing capability of the agent is therefore modelled in the following subsection.

3.3.3 Trailing mode

The molecule $X_j^{(t)}$ in equation (3.12) are representation of a potential agent. The agent (X_{Agent}) in this case is the molecule with the best fitness and its position at the initial stage is fixed i.e. equation (3.8). While exploring the search space, the concentration of smell molecules may become higher than the current position of the agent, in this case, the agent moves towards this position. This way, the agent continues to trail the position of all molecules with higher concentrations until the molecule with the overall best fitness (smell source) is identified.

In some practical situations, the agent should be able to sniff the smell particles and intuitively follow these particles with the hope of identifying their source. Due to variation in the temperature and molecular mass of the smell particles, path trailing by an agent becomes a challenging task. Also, every agent has a specific capacity of olfaction, depending on the size of their olfactory lobe, psychological and physical conditions. For example, larger size of olfactory lobe indicates an efficient olfaction which favours exploitation while smaller size of olfactory lobe indicates poor olfaction which is an indication of poor exploration. Since the proposed SAO is generalized for all agents, the SAO precision and convergence will obviously be influenced by the olfaction capacity of the agent. Thus, a control parameter called the olfaction capacity (olc) of the agent is introduced.

As discussed in chapter two, the olfaction is a chemo-sensation process. Therefore, to determine whether an agent will be able to perceive and trail some smell molecules, the smell has to first evaporate in the direction of the agent. The sniffing process is performed and then, the olfaction capacity of the agent can then be determined. Thus, the olfaction capacity in this research is determined as a function of the fitness of the agent and the entire sniffing process as shown in equation 3.13:

$$olf = \frac{f(x_{Agent})}{\sum_{i=1}^N f(x_i) / N} \quad (3.13)$$

Where;

olf is the olfaction capacity, $f(x_{Agent})$ is the fitness of the agent, $f(x_i)$ is the fitness of the individual smell molecules and N is number of smell molecules.

The olfaction capacity given in equation (3.13) will enable the agent to have prior knowledge of the nature of object originating the smell as the agent trails the part of the molecule. For the agent to efficiently trail the path of the smell, the agent should be able to remember the position of the molecule with the best smell concentration (which is the present position of the agent) and the position with the worst concentration (position of worst fitness) of smell. The agent will use this information during trailing and avoid moving towards the region with unfavourable fitness. The trailing behaviour of the agent is modelled in equation 3.14:

$$X_i^{(t+1)} = X_i^{(t)} + r_1 \times olf \times (X_{agent}^{(t)} - X_{(i)}^t) - r_2 \times olf \times (X_{worst}^{(t)} - X_{(i)}^t) \quad (3.14)$$

Where; r_1 and r_2 are random numbers generated at different intervals.

During trailing by the agent, the random number r_1 stochastically penalize the influence of the olfaction capacity parameter on the position of the agent while the random number r_2 stochastically penalize the influence of the olfaction capacity on the worst position of the smell molecule. Both r_1 and r_2 ensure that *olf* has no significant influence on the behaviour of SAO

3.3.4 Random Mode

The smell molecules are discrete in nature and if these molecules are separated by large distances in comparison with the molecular search dimension, the intensity/concentration of the smell molecules varies over time from one point to another. This makes smell perception a challenge for the agent which may subsequently lead to the loss of the smell trail. At this point, the agent maybe trapped into local minima leading to its inability to continue trailing. In such case the agent goes into another mode called the random mode which is described as:

$$X_i^{(t+1)} = X_i^{(t)} + r_3 \times SM \quad (3.15)$$

Where;

SM is a constant indicating the step movement and r_3 is random number which stochastically penalize the value of the step movement.

In a situation where the trailing mode fails to obtain the best fitness or identify the smell source or the agent losses its trail, the agent takes a step movement randomly in the search space using the equation described in equation (3.15). The conceptual framework of the proposed algorithm is given in Figure 3.4

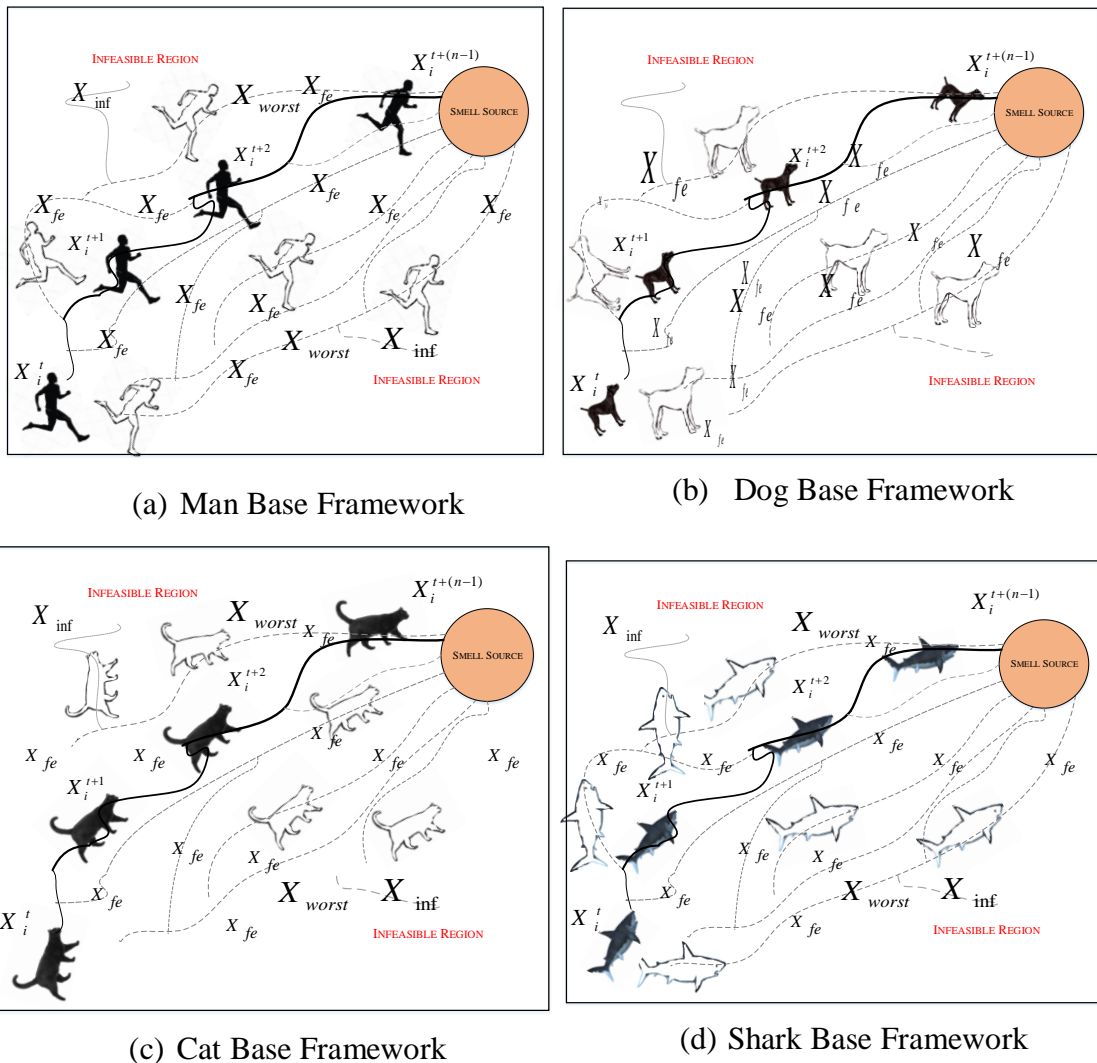


Figure 3.4: Conceptual Framework of the SAO

The, conceptual frame work of the intuitive behaviour of some agents whose olfaction capabilities have been discussed in subsection 2.2.1.1 are represented Figure 3.4. In the figure, the agent is represented as the man, dog, cat or fish and the object evaporating the smell molecule is depicted in the grey circle. In the SAO, the smell object evaporates the smell molecules in the direction of the agent which is indicated by the “dotted” lines labelled X_{fe} and the “thick” line which represent the path with high concentration of smell (optimum path). The paths labelled X_{fe} are all feasible paths which if followed could lead the agent to the smell source. However, due to the

optimality factor, the agent is constrained to follow only the optimum path all through the searching process. At every stage in the searching process, the agent take note of the X_{worst} position (worst feasible paths in generation) and make use of this information to restrict it movement only to the optimum path. This ideology is modelled as negative path of equation (3.14). The path labelled X_{inf} is an infeasible path which leads the agent into an infeasible region. Movement towards this region is avoided by appropriately selecting a suitable value for the SAO control parameters such as temperature (T), mass (m), olfaction capacity (olc) and step movement (SM). During the search process, the agents in SAO identify the region with high concentration of smell by performing the sniffing mode which is represented as the first position (X_i') of the man, dog, cat or fish in Figure 3.3. The agents update this position by performing the entire process of SAO and update (iterate) its positions until the optimum solution is obtained.

3.4 Flow of SAO Algorithm

The step involved in implementing the smell agent optimization algorithm are highlighted as follows:

Step 1: Initialize all the parameters required to implement the algorithm. These parameters are: the population (positions) of the smell molecules, the initial velocity, search dimension, temperature, Boltzmann constant, random mode step movement (SM) and the number of iteration.

Step 2: Randomly generate initial position of the smell molecules in the search space and assign the velocity of each molecule. This is done using equations (3.7) or (3.8).

Step 3: Evaluate the fitness of the generated molecules position and the velocity in Step 2 and determine the smell agent and its olfaction capacity using equation (3.13). The agent (X_{agent}) in this case is the molecules position with the best fitness.

Step 4: Update velocity using equation (3.11) and perform the sniffing mode as described in equation (3.12).

Step 5: Evaluate the fitness of the sniffing mode and replace the position of the agent in step 2 with the position of molecule having the best sniffing fitness.

Step 6: Determine the position of the molecule with the worst sniffing fitness (which is the unfavourable position to be avoided by the agent) and perform the trailing mode behaviour as described in equation (3.14).

Step 7: Evaluate the fitness of the trailing mode described in step 6.

Step 8: Compare the fitness of the trailing mode with the fitness obtained during the sniffing mode.

Step 9: If the trailing mode fitness is better than the sniffing mode fitness, repeat step 6 to step 8 until the smell source is determined. If the sniffing mode fitness is better than the trailing mode fitness, then, move to step 10.

Step 10: Perform the random mode behaviour as described in equation (3.15).

Step 11: Evaluate the fitness of the random mode.

Step 12: If the fitness of the random mode is better than the fitness of the trailing mode, then determine the new random position of the agent and the worst random position of the molecule and perform the trailing mode again, otherwise repeat step 4 to step 9.

Step 13: Terminate if the stopping condition is satisfied else repeat step 1 to step 12 until the stopping criteria is met.

3.5 Important assumptions

Some of the important assumptions adopted for the successful development of the proposed SAO are listed as follows:

- 1) The smell molecules evaporate from the smell source constantly in the direction of the agent until the smell object is found.
- 2) The velocity of the object evaporating the smell molecule is negligible compared to the velocity of the smell agent. In other words, the object originating the smell is in a fixed position and cannot move.
- 3) The smell source could be more than one and each source evaporates the same number of smell molecules with varying concentration.

The flow chart for implementing the smell agent optimization is given in Figure 3.5.

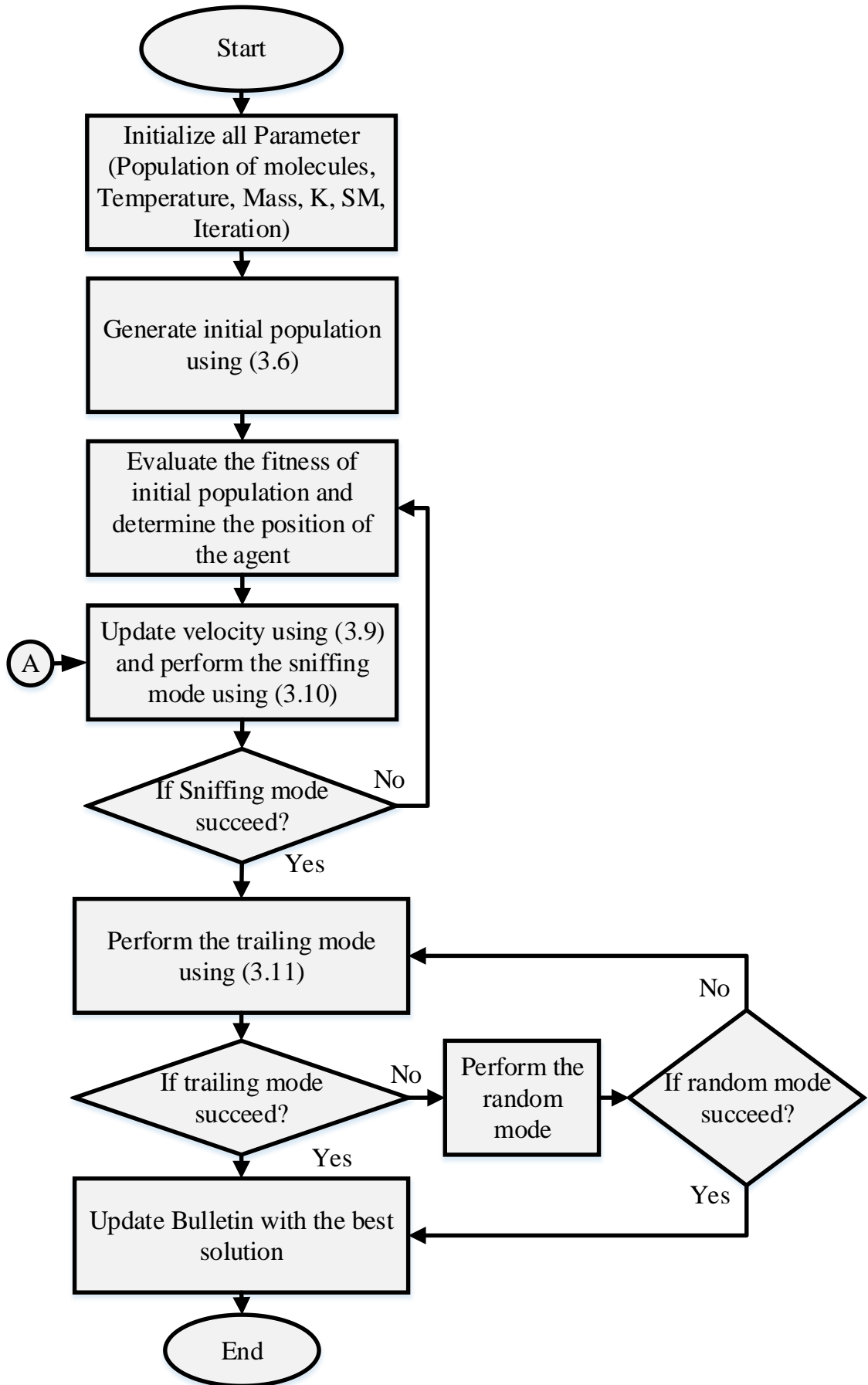


Figure 3.5: Standard Flow Chart for the Developed SAO

The standard flowchart for implementing the algorithm proposed in this research is given in Figure 3.5. From the figure, all the parameters required for successful implementation of the algorithm are initialized. The initial position of the smell molecules was then randomly assigned (generated) using equation (3.7). The fitness of the molecules was determined and the molecule which has the best fitness is selected as the agent whose olfaction capacity is determined using equation (3.13). The velocity of the molecule is updated and the sniffing mode is performed using equations (3.11) and (3.12) respectively. The position of the agent is updated and the molecule with the worst fitness is determined. Then the trailing mode behaviour of the agent is performed using equation (3.14). The fitness of the trailing mode is evaluated and the position of the agent is updated until the best position is obtained. In a situation where the trailing mode fails to obtain the best solution, the agent goes into the random mode hoping to obtain better solution. If a better solution is obtained, the agent starts trailing from this position again. If not, the agent goes back to the sniffing (default) mode and start the SAO process all over again. The MATLAB script for implementing the SAO can be found in Appendix A.

3.6 SAO Parameter Selection

In the developed SAO and in fact all other bio-inspired algorithms, the appropriate value of control parameters plays a significant role in their optimality and convergence. In some algorithms such as SAO as in Table 3.2, the sensitivity of the algorithms to initial value of this control parameters are negligibly small making the algorithms efficient in terms of exploration and exploitation.

Table 3.2: SAO Control Parameters

SN	Parameter	Symbol	Value	Unit
1	Temperature	T	3	K
2	Mass	m	2.4	Kg
3	Olfaction Capacity	olf	Equation 3.13	---
4	Boltzmann's constant	k	1.38×10^{-23}	JK^{-1}
5	Population (Molecules)	N	50	---
6	Dimension	D	Function dependent	---
7	Step Movement	SM	2.5	m
8	Iteration	Itr	3000, 500	---

The values of the parameters given in Table 3.2 were selected after several trials by the researcher. However, looking at equation (3.13), the choice of temperature and mass should be selected such that the following constraints must be respectively satisfied.

$$T \geq 0 \quad (3.16)$$

$$m > 0 \quad (3.17)$$

The constraint given in equation (3.16) is to ensure that whatever value selected for temperature should not become negative (non-negativity restriction). This restriction when violated will result in SAO given a complex output. Also, if the constraint given in equation (3.17) is violated, the entire process of SAO becomes undefined. Keeping in mind these restrictions, the values given Table 3.2 were carefully selected for this research.

3.7 Application of SAO on the Benchmark function

As discussed in section 2.2.8, the effectiveness of an optimization algorithm cannot be measured against other algorithms by the problems that it solves. This is more so, if the set of problems are too specialized and without diverse properties. To effectively evaluate the performance of the algorithm developed in this research, a subset of thirty-

nine benchmark function were employed based on the features described in subsection 2.2.8.1. The complexity of each of these functions are modelled into a metric function given as:

$$f_{metrics}(N, M, P, V, D) = \psi + \rho \cdot N \cdot \beta^{(M+P+V+D)} \quad (3.18)$$

Where

ψ is a constant which can take any value depending on the optimization problem, ρ is a coefficient of modality which depend on the optimization problem, N is the problem dimensionality or the number of parameters.

M is the modality

- 0 – Uni-modal
- 1 – Medium-modal
- 2 – Multi-modal
- 3 – Highly multi-modal

P is the Plateau

- 0 – No Plateau
- 1 – Medium Plateau
- 2 – Large Plateau

V is the Valley

- 0 – No Valley
- 1 – Narrow Valley
- 2 – Wide Valley

D is the decomposability

- 0 – Easy-as-decomposable
- 1 – Not easy-as-decomposable

This metric function is used to classify the thirty-nine-benchmark functions used for validating the performance of the smell agent optimization.

Equation (3.18) is used to describe the complexity of the selected benchmark functions.

The larger the value obtained for $f_{metrics}(N, M, P, V, D)$ the more complex the optimization function. Since the value of ψ , ρ and β can be constant, the values 200 for ψ , 15 for ρ and 2 for β respectively were used in this research. The values of the other parameters (N , M , P , V and D) are determined from the landscape structure and characteristics of the benchmark functions. Table 3.3 show the characteristics and metric values obtained for each test function.

Table 3.3: Classifying the Characteristics of the Selected Subset of Benchmark

Functions									
SN	Function Name	SD	$f(x^*)$	N	M	P	V	D	$f_{metrics}$
1	Ackley	$[-32.0, 32.0]^n$	0	2	1	0	0	1	320
2	Adjiman	$[-1.0, 2.0]^n$	-2.02181	2	1	0	1	1	440
3	<i>Alpine 1</i>	$[-10, 10]^n$	0	2	2	0	0	1	440
4	Beale	$[-5.0, 5.0]^n$	0	2	0	1	0	1	320
5	Bird	$[-2\pi, 2\pi]^n$	-106.76453	2	1	0	1	1	440
6	Bohachevsky	$[-6.0, 6.0]^n$	0	2	1	0	0	1	320
7	Booth	$[-10, 10]^n$	0	2	1	0	0	0	260
8	Box	$[-10.0, 10.0]^n$	0	3	0	2	0	1	560
9	Bukin F6	$[-15,5; -3,3]^n$	0	2	1	0	0	0	260
10	Colville	$[-10.0, 10.0]^n$	0	4	1	0	1	1	680
11	Cosine Mixture	$[-1.0, 1.0]^n$	-3	2	1	0	0	1	320
12	Cross-in-tray	$[-10.0, 10.0]^n$	-2.06261218	2	1	1	0	1	440
13	Dejong F4	$[-5.12, 5.12]^n$	0	2	0	1	0	0	260
14	Easom	$[-100.0, 100.0]^n$	-1	2	1	0	0	1	320
15	Egg Crate	$[-5, 5]^n$	0	2	1	0	0	0	260
16	Ellipsoid	$[-5.12, 5.12]^n$	0	30	1	0	0	0	1100
17	Goldstein price	$[-2.0, 2.0]^n$	3.0000	2	0	1	0	1	320
18	Griewank	$[-600.0, 600.0]^n$	0	30	1	0	0	1	2000
19	Holder Table	$[-10, 10]^n$	-19.2085	2	2	0	0	1	440
20	Kowalik	$[-1.0, 1.0]^n$	4.0750e-04	4	0	2	2	1	2120
21	Levi & Montalvo F1	$[-10.0, 10.0]^n$	0	2	1	0	0	1	2000
22	Michalewicz	$[0.0, \pi]^n$	-4.687	5	2	0	0	1	800
23	Matyas	$[-10.0, 10.0]^n$	0	2	0	0	1	1	320
24	Mishra	$[-10, 10]^n$	0	30	1	0	0	1	2000
25	McCormick	$[-1.5, 4; -3, 3]^n$	-1.9133	3	2	0	0	1	560
26	Neumaier F3	$[-N^2; N^2]^n$	-2	2	1	0	0	1	320
27	Quadratic	$[-5.12, 5.12]^n$	-3877.24	30	0	0	0	1	1100
28	Rastrigin	$[-5.12, 5.12]^n$	0	5	2	0	0	1	800
29	Rosenbrock	$[-6.0, 6.0]^n$	0	2	0	0	2	1	440
30	Sphere	$[-1, 1]^n$	0	30	1	0	1	1	3800
31	Styblinkis Tang	$[-5, 5]^n$	-78.33198	2	1	1	1	1	7400
32	Step	$[-5.12, 5.12]^n$	-3.000e+01	5	0	0	0	0	275
33	Sal	$[-2\pi, 2\pi]^n$	0	30	1	0	0	1	2000
34	Schaffer	$[-100, 100]^n$	0	2	1	0	1	1	440
35	Schwefel	$[-10.0, 10.0]^n$	0	2	2	0	1	0	440
36	Subert	$[-10.0, 10.0]^n$	-186.73	5	3	1	0	1	1160
37	Watson	$[-2.0, 2.0]^n$	2.288e-03	6	0	2	2	1	3080
38	Yang	$[-2.0, 2.0]^n$	-3.791e-03	2	0	1	0	1	320
39	Zirilli	$[-10, 10]^n$	-0.3523	2	0	0	1	1	320

In Table 3.3, the first column represents the identification number (serial number) of the functions used for validating the performance of the proposed algorithm and the name of the functions is given in the second column. The search domain of each function is presented in column three. The $f(x^*)$ column provides the optimal solution to each of the functions. The N column gives the number of dimensionality or decision variables used. Column M , P , V and D describe the modality, plateau, valley and decomposability of the functions respectively. The $f_{metrics}$ column describe the maximum number of generations allowed for each function as calculated using equation (3.18). Detail mathematical expression for each of the test functions have been discussed in chapter two.

In order to solve for each of the test functions using SAO, a serial number n which correspond to the test functions serial numbers given in Table 3.3 are assigned. The functions are then programmed such that, when the SAO is simulated, the program prompts for a message requiring the serial number which correspond to the function the user wishes to run. By this, the complexity of the program is then reduced to only one function program, instead of 39 different function programs representing each test functions. The flowchart of the test functions program is given in Figure 3.6

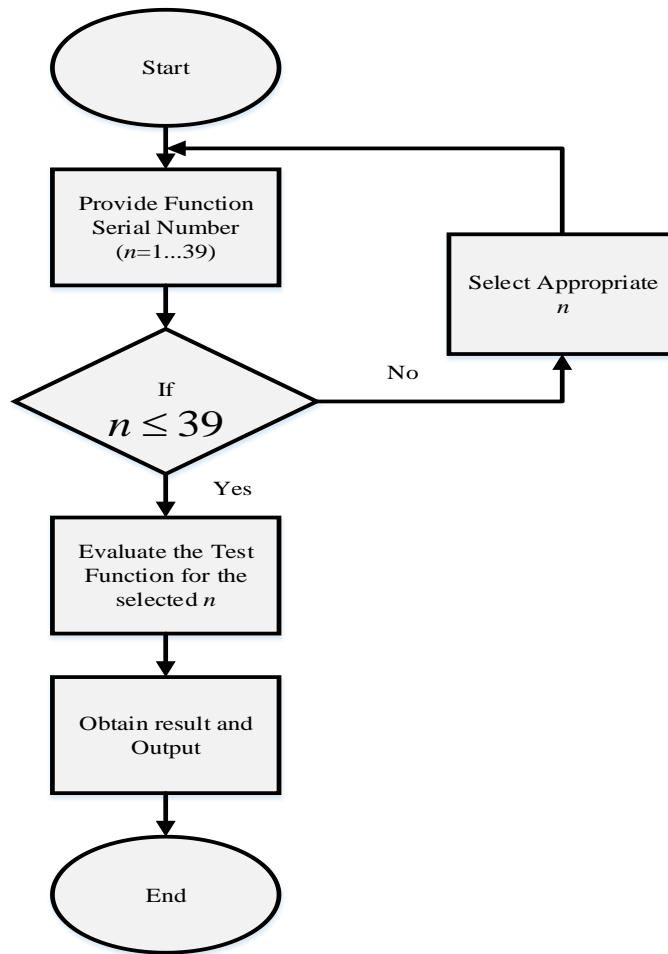


Figure 3.6: Objective Function Flowchart

Figure 3.6 shows the flow chart for implementing the test function. At the initial stage, a serial number n which corresponds to the test function the user wants to evaluate is initialized. Thereafter, a decision is made to verify if the value of n provided by the user is within the available range. If this is so, the test function whose serial number correspond to the value enter by the user is selected otherwise a new serial number would have to be provided. This selected function is passed to the SAO for evaluation and the result is displayed for the user to make decision. The MATLAB code for implementing the benchmark functions is given in Appendix B.

3.8 Application of SAO in Path Planning

In applying the developed SAO to solve the path planning problem, the model of the path planning was formulated. In the developed model, the navigation field of the robot were assumed to be a two-dimensional field mapped from one vector.

$$v = [x, y] \quad (3.18)$$

To generate the navigation field, the gradient vector is formulated from the gradient of x and y as

$$\Delta = [\Delta x, \Delta y] \quad (3.19)$$

In order to formulate the navigational field, Δx and Δy were define in the following step

Step one:

Let the target (goal) position coordinate be represented as (x_{goal}, y_{goal}) and the spread of the navigational field be represented as s . Let r represent the distance from the centre to the edge of the goal and let $[x, y]$ represent the coordinate position of the robot (x_{robot}, y_{robot}) .

Step two:

Evaluate the distance from the robot position to the goal position as follows:

$$D(x_{robot}, y_{robot}) = \|(x_{goal}, x_{robot}), (y_{goal}, y_{robot})\| \quad (3.20)$$

Equation (3.20) is a direct distance between the robot position and the goal position, represented as:

$$D(x, y) = \sqrt{(x_{goal} - x_{robot})^2 + (y_{robot} - y_{goal})^2} \quad (3.21)$$

Step three

The angle between the goal and the robot is evaluated as

$$\phi = \tan^{-1} \left(\frac{y_{goal} - y_{robot}}{x_{goal} - x_{robot}} \right) \quad (3.22)$$

Step four

To set the gradient vector, Δx and Δy are set considering the following constraint.

$$1) \text{ if } D(x, y) < r$$

$$\Delta x = \Delta y = 0 \quad (3.23)$$

$$2) \text{ if } r \leq D(x, y) \leq s + r$$

$$\begin{aligned} \Delta x &= (d - r) \cos(\phi) \\ \Delta y &= (d - r) \sin(\phi) \end{aligned} \quad (3.24)$$

$$3) \text{ if } D(x, y) > s + r$$

$$\begin{aligned} \Delta x &= s \cos(\phi) \\ \Delta y &= s \sin(\phi) \end{aligned} \quad (3.25)$$

So far, the attraction of the robot to the goal have been modeled. Now the robot need to avoid any obstacle along it path to the goal. The obstacle avoidance model is highlighted in the following steps:

Step one

Let the position of the obstacles in the two-dimensional field be denoted as $(x_{obstacle}, y_{obstacle})$ and let the radius of the obstacle be denoted as r . The vector position of the robot can be denoted as equation (3.18).

Step two

The distance between the robot position and the obstacle is determined as

$$D(x, y) = \left\| (x_{obstacle}, x_{robot}), (y_{obstacle}, y_{robot}) \right\| \quad (3.28)$$

Equation (3.29) is a direct distance between the robot position and the obstacle vertex, represented as:

$$D(x, y) = \sqrt{(x_{obstacle} - x_{robot})^2 + (y_{robot} - y_{obstacle})^2} \quad (3.29)$$

Step three

Evaluate the angle between the robot and the obstacle as:

$$\phi = \tan^{-1} \left(\frac{y_{obstacle} - y_{robot}}{x_{obstacle} - x_{robot}} \right) \quad (3.30)$$

Step four

Then, Δx and Δy are set considering the following:

1) If $D(x, y) < r$

$$\begin{aligned} \Delta x &= -\text{sign}(\cos(\phi)) \\ \Delta y &= -\text{sign}(\sin(\phi)) \end{aligned} \quad (3.31)$$

2) If $r \leq D(x, y) \leq s + r$

$$\begin{aligned} \Delta x &= (s + r - d) \cos(\phi) \\ \Delta y &= (s + r - d) \sin(\phi) \end{aligned} \quad (3.32)$$

3) If $D(x, y) > s + r$

$$\Delta x = \Delta y = 0$$

The introduction of the negative signs in equation (3.31) is an indication that the robot points away from the obstacle. When $d = r$, it corresponds to the robot being on the edge of the obstacle.

Now, the total gradients generated by the goal and obstacle function is then summed as follows

$$\begin{aligned} \Delta x &= \Delta x_{goal} + \Delta x_{obstacle} \\ \Delta y &= \Delta y_{goal} + \Delta y_{obstacle} \end{aligned} \quad (3.33)$$

During navigation, the robot evaluates Δx and Δy using equation (3.33) and find the angle:

$$\varphi = \tan^{-1} \left(\frac{\Delta y}{\Delta x} \right) \quad (3.34)$$

The path followed by the navigating robot is then formulated as

$$D(\Delta x, \Delta y) = \|\Delta x, \Delta y\| \quad (3.35)$$

The objective function which the smell agent optimization hopes to optimize is given in equation (3.36)

$$f(x) = \min \sum_{i=1}^N D(\Delta x, \Delta y) \quad (3.36)$$

Where $D(\Delta x, \Delta y)$ is Euclidean distance given between given in equation (3.35).

Equation (3.36) is the cost functions which serves as evaluation criteria for the optimality of the solution obtained from the SAO. The modelled environment showing the randomized positions of the robot, goal and obstacles is given in Figure 3.7

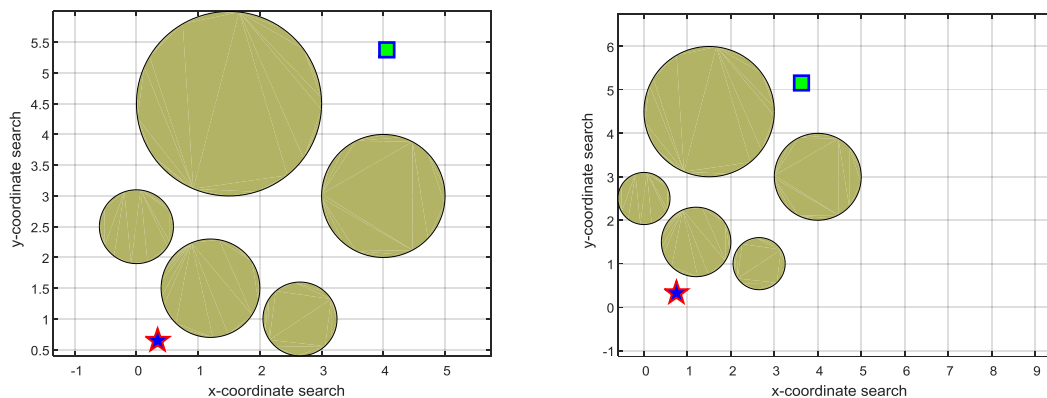


Figure 3.7: Path Planning Environments

In order to demonstrate the dynamic positioning of the robot, obstacles and the goal, Figure 3.7 was generated. The figures show two environments simulated at different intervals. In the figure, the robot is represented as the blue star shape with red outline. The environment also contains five circled obstacles made of different diameters and the goal is represented as the green square with blue outline. In the environment the position of the robot, the goal and the obstacles vary every time the SAO is run. In each case, the position of the robot is such that, there is no direct line of sight between the

robot and the goal, and the robot is always positioned below the obstacles while the goal is always positioned above the obstacles. This way, the complexity of the search environment during robot exploration is made as complex and realistic as possible. The flowchart for implementing the path planning cost functions is given in Figure 3.8.

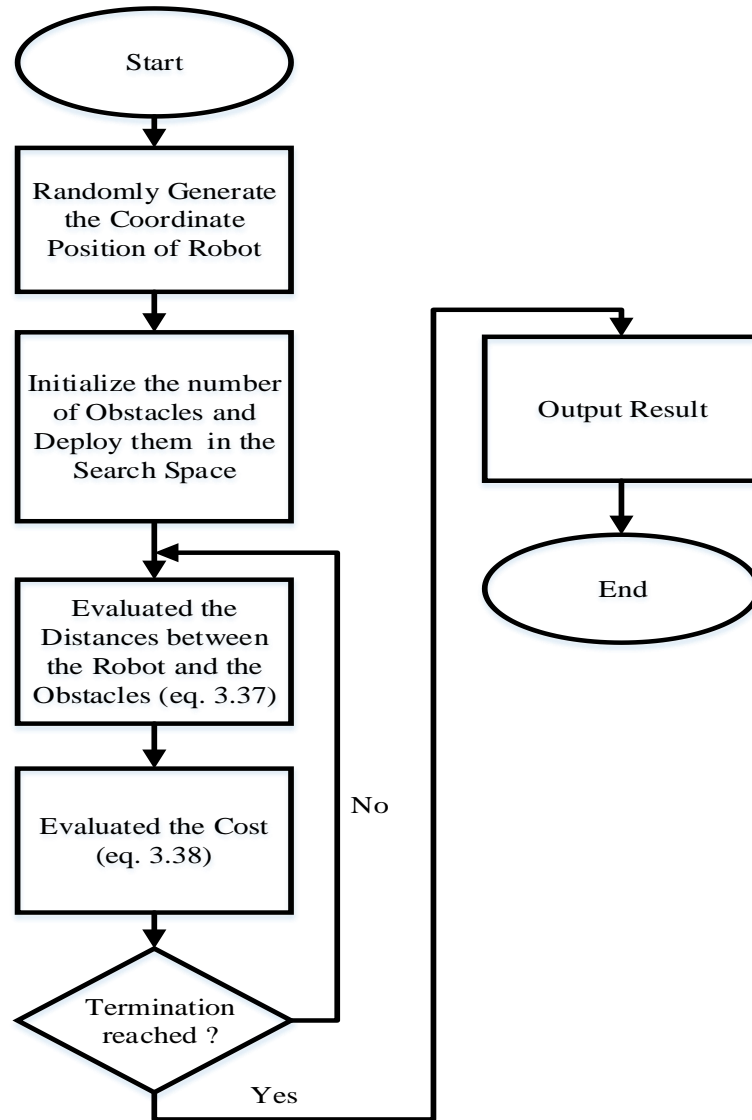


Figure 3.8: Flow Chart of Path Planning Cost Function

In Figure 3.8, the coordinate positions of the robot (x_{robot}, y_{robot}) were randomly generated and the position is restricted by the constrain given in equation (3.37)

$$\begin{aligned}
 0 &\leq x_{robot} \leq 1 \\
 0 &\leq y_{robot} \leq 1
 \end{aligned}
 \tag{3.37}$$

Just like the robot, the position of the goal was also generated randomly and is restricted by the constrain given in equations (3.38) and (3.39).

$$3 \leq x_{goal} \leq 5 \quad (3.38)$$

$$4 \leq y_{goal} \leq 6 \quad (3.39)$$

The number of obstacles were then initialized and each obstacle coordinate is given in the Table 3.4.

Table 3.4: Coordinate positions of obstacles

Coordinate	Obstacle 1	Obstacle 2	Obstacle 3	Obstacle 4	Obstacle 5
$x_{obstacle}$	1.5	4.0	1.2	2.65	0
$y_{obstacle}$	4.5	3.0	1.5	1.0	2.5

Since the obstacles considered in this thesis are circle, the radius of each obstacle was selected as 1.5m, 1.0m, 0.8m, 0.6 and 0.6m for Obstacle1, Obstacle2, Obstacle3, Obstacle4 and Obstacle5 respectively. The planning cost function is then optimized until the termination criteria is met.

The SAO parameters given in Table 3.2 were adopted for the path planning simulation. However, the number of search dimension or decision variables were selected to be 3. This is purely the choice of the researcher as the decision variable influence the number of points the robot should evaluate at every iteration. Nevertheless, more decision variables may improve the path planning cost, but have negative influence on the time the robot takes to reach the goal. The parameters used for simulating the path planning model along with the SAO parameters is given in Table 3.5:

Table 3.5 SAO Simulation on Path Planning

SN	Parameters	Value	Unit
1	Population	250	--
2	Dimension	3	--
3	Iteration	500	--

Table 3.5 shows some of the parameters used during SAO simulation on the developed model of the robot path planning. These parameters were selected carefully by the researcher according to the problems under consideration. Since there is no empirical parameter established in literature for solving path planning problems, these parameters can be varied to improve the optimality of the planned path. All other parameters of SAO are as presented in Table 3.2. The MATLAB program for implementing the path planning cost function is given in appendix C

3.9 Application of SAO to Minimum Spanning Tree (MST)

As discussed in subsection 2.2.10, for a given undirected and completely connected graph $G(V, E)$, the MST is a close-free subgraph $T(V, E_T)$, $E_T \in E$, so much that all the vertices in V are connected. The spanning tree consists of $n - 1$ edges and a total of V^{n-2} can be obtained from the given graph. In this research, some numerical costs $w_{i,j} \geq 0$ are randomly assigned to each edge $(i, j) \in E, (i, j) \in V$. To solve the minimum spanning tree problem using the developed SAO, the MST was conceptualized as a simple network topology. In the MST network, each smell molecules were considered as a vertex in the graph. Each pair of molecules were connected by a weighted edge. Assuming a pair of molecules x_i and x_j in the MST search space, the weighted edge connecting the two molecules is computed using equation (3.40)

$$w_{(i,j)} = \|x_i - x_j\| \quad (3.40)$$

Since the developed algorithm is iterative based, the distance between two vertices is computed with reference to the best vertex available in memory. This is because the vertex in memory is more stable than the present vertex. Since the minimum spanning tree in MST is connecting vertex through different edge of minimum weight, a number

of minimum weighted edges in the spanning tree are cut to avoid drift. Hence the resulting MST of the entire network is computed using the cost function given in equation (3.41)

$$C = \min \sum_{(i,j) \in E_T} w_{(i,j)} \quad (3.41)$$

Where C , is the cumulative sum of successive distances between the vertices whose weight constitute the minimum weights. Figure 3.9 shows the undirected minimum spanning tree graph topology used for evaluating the performance of the developed SAO.

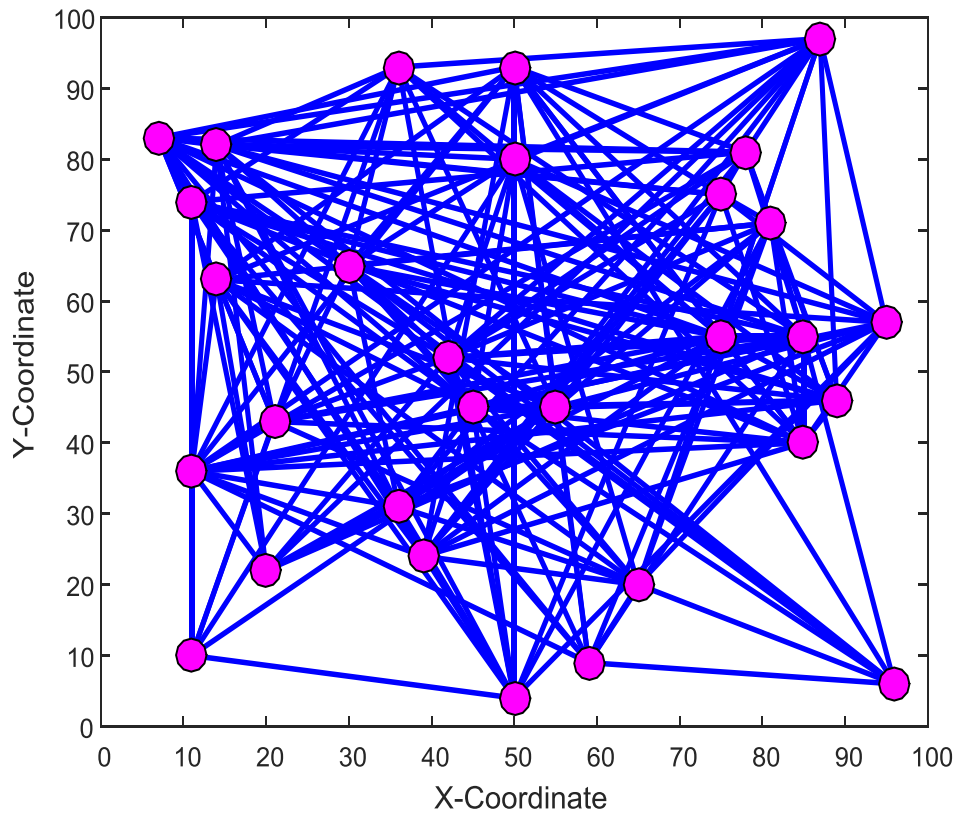


Figure 3.9: MST Graph Topology

The modelled MST graph topology given in Figure 3.9, contain 30 vertices, statically distributed in a hyperspace of 100×100 meters. Each vertex is placed in a specified position based on the coordinates given in Table 3.6

Table 3.6: MST Vertex Coordinate

SN	Vertex (V)	$X_{coordinate}$	$Y_{coordinate}$
1	V ₁	14	63
2	V ₂	42	52
3	V ₃	50	93
4	V ₄	7	83
5	V ₅	36	31
6	V ₆	87	97
7	V ₇	96	6
8	V ₈	50	4
9	V ₉	11	10
10	V ₁₀	21	43
11	V ₁₁	36	93
12	V ₁₂	14	82
13	V ₁₃	81	71
14	V ₁₄	39	24
15	V ₁₅	11	74
16	V ₁₆	59	9
17	V ₁₇	78	81
18	V ₁₈	89	46
19	V ₁₉	11	36
20	V ₂₀	55	45
21	V ₂₁	65	20
22	V ₂₂	75	55
23	V ₂₃	30	65
24	V ₂₄	75	75
25	V ₂₅	20	22
26	V ₂₆	50	80
27	V ₂₇	85	55
28	V ₂₈	95	57
29	V ₂₉	45	45
30	V ₃₀	85	40

The vertices coordinate given in Table 3.6 forms the MST topology which are connected by a total of 435 decision variables (edges). The weight of the edges is determined by the distances between a pair of directly connected vertices. These distances are what the SAO optimizes in order to compute the vertices which makes up the spanning tree with the minimum weight. In this research, three scenarios of MST were considered. The first scenario consists of the first fifteen vertices of Table 3.6, the second vertex consists of the first twenty vertices and the third scenario consist of the

entire 30 vertices given in Table 3.6. The flow chart for the minimum spanning tree cost function is given in Figure 3.10.

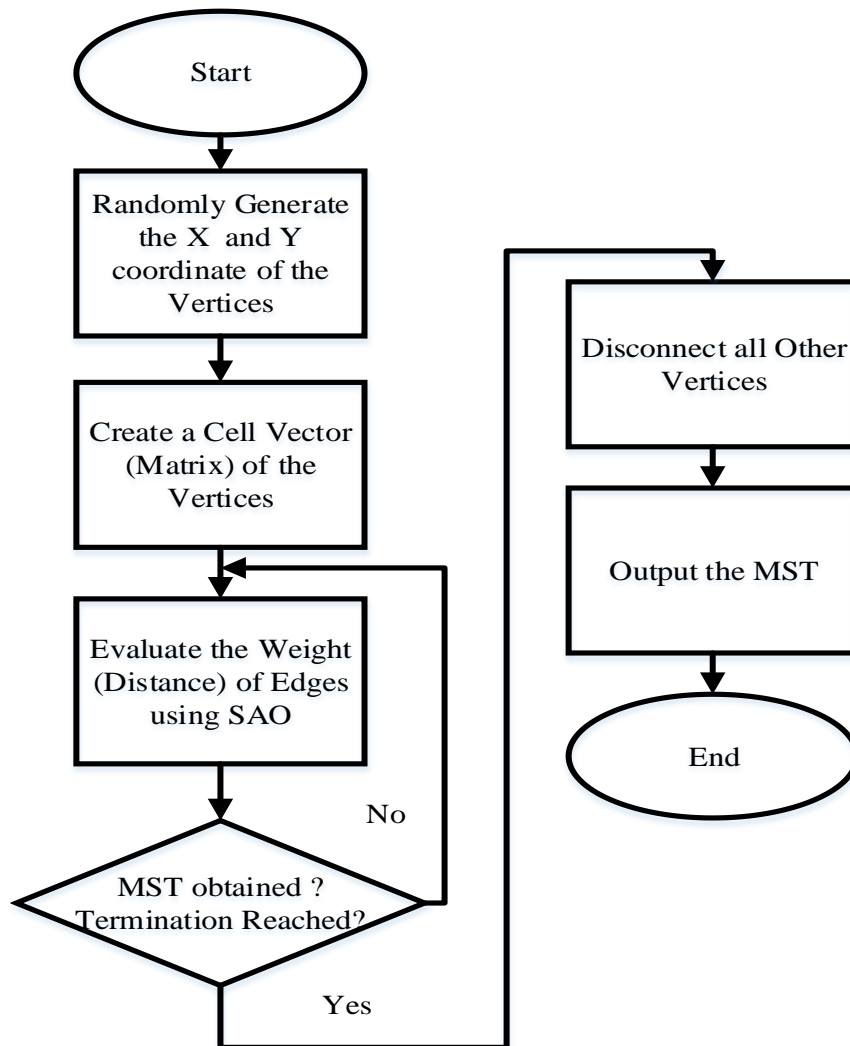


Figure 3.10: Flow Chart of MST Cost Evaluation

Figure 3.10 shows the procedures used for implementing the MST cost function. At the initial stage, the coordinate positions of each of the vertices given in Table 3.6 are assigned. The vertices are then positioned in the network using their respective coordinate. The distance (weight) between a pair of vertex are then evaluated and a combination of all $n - 1$ edges which constitute all the vertices and have the minimum weight are then computed as the MST. The parameters used for simulating the MST problem is given in Table 3.7:

Table 3.7 SAO Simulation on MST

SN	Parameters	Value	Unit
1	Population	50	--
2	Dimension	105, 190,435	--
3	Iteration	500	--

From Table 3.7, the dimensions which are the edges connecting the vertices are computed by the algorithms using the Euclidean distance model described in equation 3.41. The number of edges in the first, second and third scenario were 105, 190 and 435 respectively. The MATLAB program for implementing the MST cost function is given in appendix D.

3.10 SAO GUI Simulator

In order to ensure efficient use of the developed SAO, a user-friendly GUI simulator was developed as a standalone software for user who have little or no knowledge of the programming platform used in the research. The simulator is simple and it runs the SAO program and the objective function program in the background. The interface of the developed simulator is given in Figure 3.10

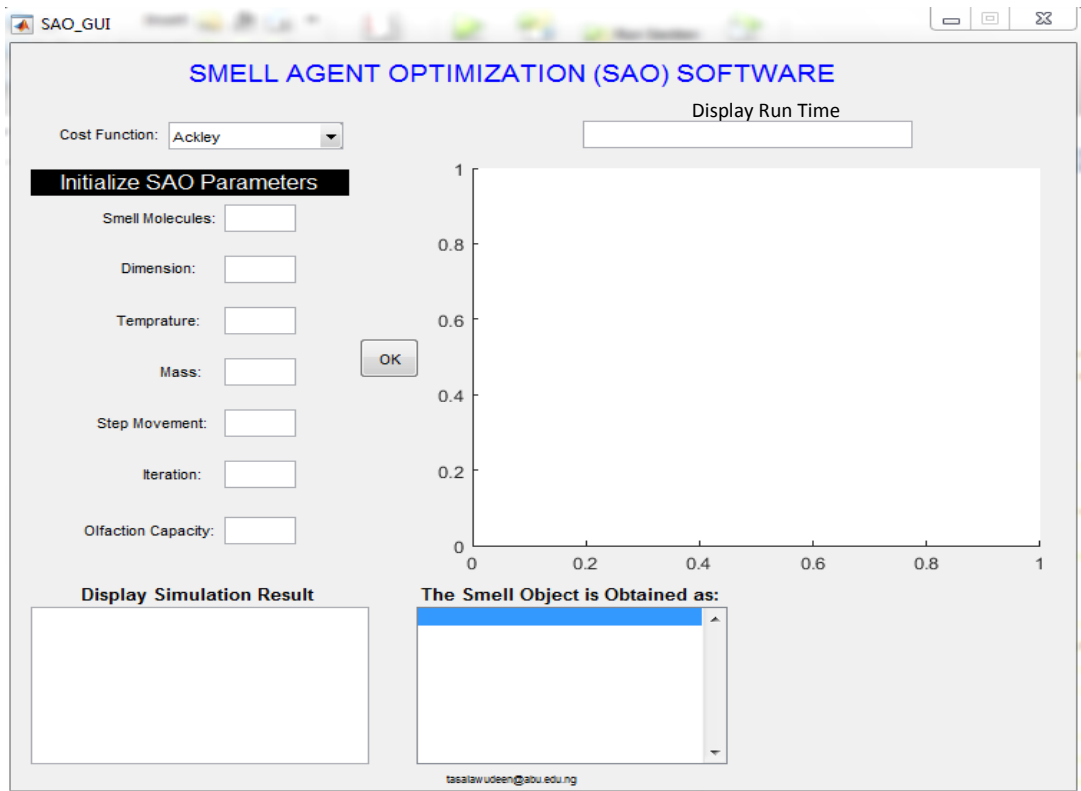


Figure 3.8: Developed SAO GUI Software

The SAO simulator given in Figure 3.8 contain basically six parts. In the first part, all the benchmark functions used for evaluating the performance of the SAO has been programmed into a drop-down menu item. A user has to simply click on the drop-down menu and select a function the user wishes to evaluate. Also, the drop-down menu contains a “*defined cost*” which allows the user to define a cost function for a different optimization problem. In the second part of the simulator, all the parameters of the SAO are provided as input into their respective space. When this is done, the user simply clicks on the “*OK*” button to perform the simulation. Upon clicking the OK button, the simulator displays the progress of the simulation in the third part titled “*Display Simulation Result*”. In the fourth part of the simulator, the time it takes to evaluate the cost function is displayed in the “*Display Run Time*”. The convergence process of the SAO is plotted and displayed in the fifth part of the simulator while the optimized result obtained at the end of the simulation is displayed in the sixth part of the simulator as the

“*smell object*”. The MATLAB script for the developed simulator is given in Appendix E.

3.11 Performance Comparison

The results obtained by the smell agent optimization developed in this research will first be compared with that of fruit fly optimization and gaseous Brownian motion optimization algorithm on the benchmark functions. Result from each algorithm will be compared with the global minimum of each functions and the algorithm that obtain the best result on any of the benchmark functions will be indicated. At the end of the day, the overall performance of the algorithms on the total test functions and over each other will be presented using percentage given in equation (3.42).

$$Pect(\%) = \left(\frac{Alg_{ref} - Alg_{alt}}{Alg_{ref}} \right) \times 100 \quad (3.42)$$

Where;

$Pect(\%)$ is the percentage difference in performance, Alg_{ref} is the reference algorithm (i.e. Algorithm with better performance) and Alg_{alt} is the alternative algorithm (i.e., algorithm which did not perform better than Alg_{ref}).

The percentage equation given (3.42) will also be used to compared the performance of the algorithm using for the path planning and the minimum spanning tree problem.

CHAPTER FOUR

RESULTS AND DISCUSSIONS

4.1 Introduction

In this chapter, the results obtained when the developed smell agent optimization was applied on the thirty-nine-standard benchmark functions in comparison with the results obtained using fruit fly optimization algorithm and shark smell optimization algorithm are presented. The results were presented based on the modality (unimodal and multimodal) of the test functions. The performance of the algorithm on robot path planning and minimum spanning tree when compared with particle swarm optimization and smell detection agent optimization are also presented.

4.2 Performance of the Algorithms on Uni-modal Test Function

The thirty-nine test functions selected were classified into unimodal and multimodal function. Ten of these functions were unimodal and twenty-nine were multimodal. Table 4.1 shows the results obtained when the developed SAO, the GBMO and the FFOA were used to evaluate the optimum of the unimodal functions. The algorithms were run ten times and each run was performed for three thousand iterations. Out of the ten (10) runs, the best, worst, mean, median and standard deviations were recorded and only the best run was used as a metric for comparing the performance of the developed SAO with FFOA and GBMO. Due to the number of data contained in Table 4.1, the table is divided into sub tables a and b. However, the test functions in both sub tables were assigned a continuous function number ranging from F_1 to F_{10} . In Table 4.1a, the first five unimodal functions were presented while in Table 4.1b, the last five unimodal functions were presented.

Table 4.1a: Uni-modal Functions Evaluations

Algorithms	Metrics	Booth	Dejong F4	Easom	Egg Crate	Ellipsoid
		F ₁	F ₂	F ₃	F ₄	F ₅
	global	0.0000	0.0000	-1.0000	0.0000	0.0000
SAO	best	40.6642	3.8273e-23	-0.9999	1.1450e-05	2.3575e-11
	worst	43.4335	1.5382e-11	-0.9965	6.3271e-04	2.9097e-06
	mean	42.4351	1.5382e-12	-0.9984	2.6336e-04	3.9293e-07
	median	42.7198	3.6552e-20	-0.9989	1.2610e-04	4.6759e-10
	std	0.9168	4.8642e-12	0.0013	2.3060e-04	9.3995e-07
FFOA	best	72.0000	5.0716e-13	-0.9999	3.7384e-05	7.0985e-07
	worst	72.0000	5.4123e-13	-0.9999	3.8188e-05	7.3411e-07
	mean	72.0000	5.2667e-13	-0.9999	3.7853e-05	7.2628e-07
	median	72.0000	5.2629e-13	-0.9999	3.7816e-05	7.2715e-07
	std	1.6389e-10	1.1176e-14	4.0453e-07	2.5925e-07	6.7575e-09
GBMO	best	64.0001	0.3109	-0.9834	1.4775e-03	0.2144
	worst	22.3804	5.5563	-0.0192	2.4060e-03	2.5902
	mean	6.1956	1.7868	-0.3009	1.8813e-03	0.9397
	median	64.0000	1.0468	-0.2617	1.0870e-03	0.7022
	std	6.2408	2.1914	0.2937	4.5363e-03	0.7220

Table 4.1b: Uni-modal Functions Evaluations

Algorithms	Metrics	Matyas	Step	Schaffer	Watson	Zirilli
	global	F ₆	F ₇	F ₈	F ₉	F ₁₀
		0.0000	-3.000	0.0000	2.288e-03	-0.3523
SAO	best	2.8770e-08	0.0000	0.0000	1.3497e-04	-0.2440
	worst	2.2591e-06	0.0000	1.4917e-10	1.5165e-05	-0.1526
	mean	4.8443e-07	0.0000	3.5755e-11	5.3346e-04	-0.1759
	median	1.9435e-07	0.0000	2.2865e-12	7.0329e-05	-0.1526
	std	7.4539e-07	0.0000	5.9690e-11	1.5477e-04	0.0385
FFOA	best	2.8620e-08	0.0000	0.0000	9.6703e-03	8.4827e-05
	worst	2.9310e-08	0.0000	0.0000	9.6703e-03	8.6197e-05
	mean	2.8996e-08	0.0000	0.0000	9.6703e-03	8.5408e-05
	median	2.8995e-08	0.0000	0.0000	9.6703e-03	8.5369e-05
	std	2.3852e-10	0.0000	0.0000	7.7607e-12	4.0235e-07
GBMO	best	0.0000	0.0000	1.4500e-10	0.0855	-0.1451
	worst	0.0000	0.0000	9.5960e-10	0.7688	1.0707
	mean	0.0000	0.0000	2.1256e-09	0.2756	0.2215
	median	0.0000	0.0000	6.7245e-10	1.8905	0.0000
	std	0.0000	0.0000	3.2699e-09	0.0728	0.5071

Table 4.1a and Table 4.2a shows the comparison of the developed SAO with the FFOA and GBMO on unimodal benchmark functions. The results presented in the table are obtained after each function was run ten times. Though, the best, worst, mean, median and standard deviation of the ten were determined, for the purpose of this research, only the best of the ten runs were used for discussion. It should be observed from the table that, values in bold indicate the algorithm which produces the best result in the test functions. Those in bold-italics indicate a situation where the algorithms obtained the same result for a test function concerned.

Comparing the performance of SAO with the FFOA in both Table 4.1a and Table 4.1b, it can be observed that, the SAO obtained the best result in five (5) of the ten (10) unimodal functions which constituted 50%, while the FFOA produced the best result in two (2) of the ten (10) which constituted (20%) of the test functions. However, both algorithms produced the same results in three (3) of the ten (10) (30% of the test functions). This is an indication of the efficiency of the developed SAO on solving optimization problems of unimodal form. The improved performance of SAO over FFOA is predictable since the SAO employed three (sniffing, trailing and random) mode when searching for the optimum solution of the function unlike the FFOA which has only one mode.

Comparing the performance of SAO with the GBMO also in Table 4.1a and Table 4.1b, it can be observed that, the SAO obtained the best results in eight (8) functions which constituted 80% of the ten unimodal test functions while the GBMO obtained the best result in one (F_6) which is 10%. However, both the SAO and the GBMO obtained the same result in one (F_7) of the test functions (10% of the unimodal functions). This is also an indication that the developed SAO can compete favourably in unimodal optimization problems with other bio-inspired computational models. In order to

demonstrate the searching process of the algorithms, the optimization plots of each algorithm on Eggcrate, Matyas, Watson and Booth functions were superimposed as shown in Figure 4.1.

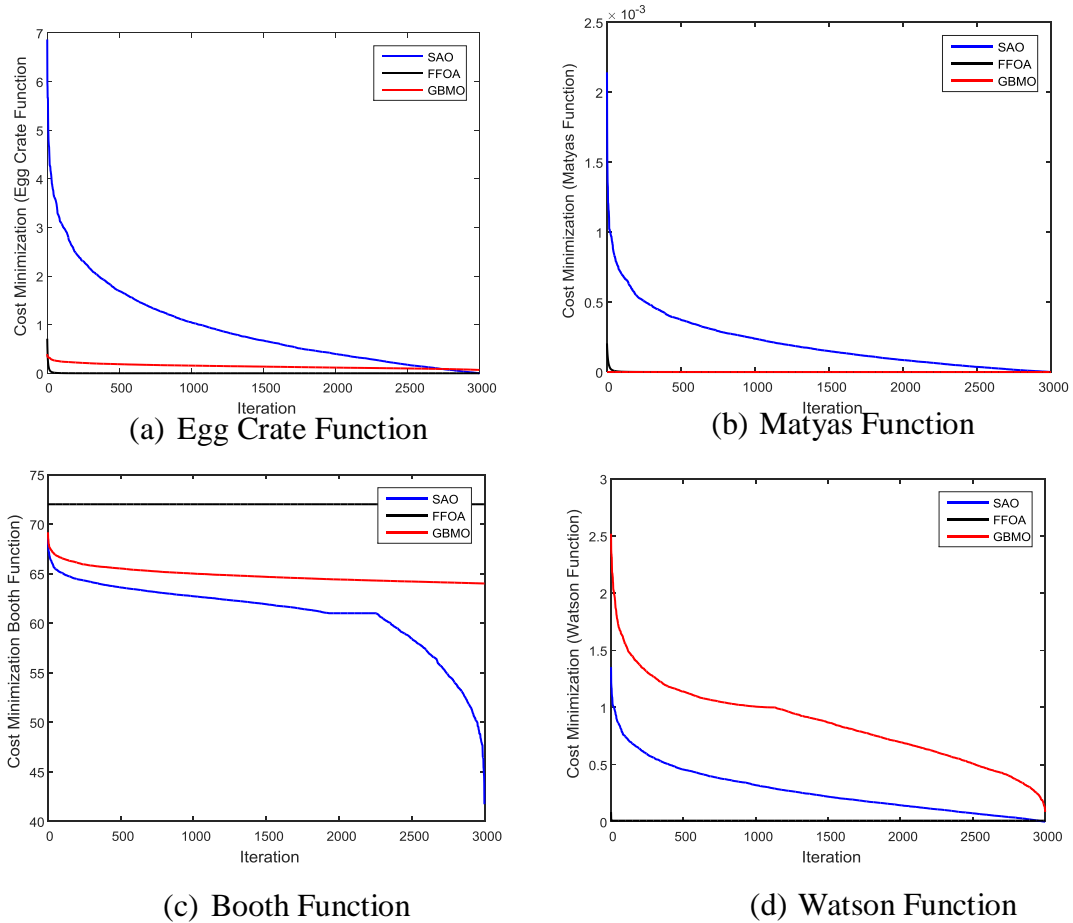


Figure 4.1: Minimization of unimodal functions

It can be observed from Figure 4.1 that, both the FFOA and the GBMO converged faster than the SAO in all the functions, leading the algorithm to converge into local minima. This premature convergence is attributed to the lack of alternative behaviour (mode) and control parameters inherent in the algorithms. However, the SAO explored the search space more efficiently and was able to avoid getting trapped into local minima due to the availability of alternative behaviours (sniffing, trailing and random). In the case of the Booth function which is shown in Figure 4.1 (c), it can be observed

that the SAO did not converge within the number of specified iteration. This implies that the optimum of this function exists outside of the specified 3000 iterations.

4.3 Performance of the Algorithms on Multi-modal Test Function

The developed algorithm was also evaluated on multi-modal test functions. The performance on the twenty-nine-multimodal test function is given in Table 4.2. Due to the large number of the test functions, the table was divided into five sub tables (Table 4.2a, Table 4.2b, Table 4.2c, Table 4.2d, Table 4.2e and Table 4.2f). In each of the sub table, results of five test functions were represented and a functions number was assigned. However, in the last sub-table (Table 4.2f), only the results of the last four test functions were presented.

Table 4.2a: Performance Evaluation on Multi-modal functions

Algorithm	Metric	Ackley	Adjiman	Alpine 1	Beale	Bird
		F_1	F_2	F_3	F_4	F_5
	global	0	-2.02181	0	0	-106.7645
SAO	best	8.8818e-16	-6.4033	1.3867e-05	5.4738E-05	-53.936
	worst	8.8818e-16	-1.2305	5.1515e-04	0.00052887	-43.6227
	mean	8.8818e-16	-2.4225	3.0256e-04	0.00024334	-47.0169
	median	8.8818e-16	-1.9557	09/01/2018	0.00024715	-46.3884
	std	0	1.5278	1.7017e-04	0.00016554	2.7293
FFOA	best	3.4248e-03	2.0202e-10	1.2624E-05	9.7112	4.0981
	worst	3.4642e-03	2.1067e-10	8.6539E-05	9.7112	4.0981
	mean	3.4435e-03	2.0595e-10	8.6036E-05	9.7112	4.0981
	median	3.4434e-03	2.0530e-10	8.6073E-05	9.7112	4.0981
	std	1.3013e-05	2.8775e-12	2.7323E-05	4.0336E-09	3.138E-10
GBMO	best	9.5235e-11	-1	1.0653e-07	1.5078	-85.3384
	worst	1.5999e-09	-9.3999	3.4982E-06	8.5973	-25.755
	mean	1.46e-09	-7.63999	1.0006E-06	4.2474	-56.3287
	median	1.854e-09	-9.25	3.395E-08	3.6212	-53.8452
	std	3.0295e-09	15.5765	1.3554E-06	2.624	22.0376

Table 4.2b: Performance Evaluation on Multi-modal functions

Algorithm	Metric	Bohachevsky	Box	Bukin F6	Colville	Cosine Mixture
		F_6	F_7	F_8	F_9	F_{10}
	global	0	0	0	0	-3.0000
SAO	best	0	6.3543E-07	0.1	0	-3.0439
	worst	0	1.2114E-05	0.1	0	-1.7076
	mean	0	5.0116E-06	0.1	0	-2.6569
	median	0	3.9358E-06	0.1	0	-2.908
	std	0	3.6985E-06	1.4628E-17	0	0.499
FFOA	best	0	1.0653E-13	3.0088	1.9544	-2.7095
	worst	0	1.5329E-08	3.0265	1.9544	-0.0999
	mean	0	1.7708E-09	3.0188	1.9544	-0.5341
	median	0	5.2169E-11	3.0191	1.9544	-0.0999
	std	0	4.7931E-09	0.0062883	2.0477E-14	0.9382
GBMO	best	0.2674	0.0024068	7.6884	0.8096e-03	-57.1589
	worst	4.9331	0.4574	15.3638	0.4312e-02	-33.6361
	mean	2.4657	0.1888	9.2523	0.7125e-02	-46.7087
	median	2.7097	0.0853	7.8523	0.9314e-02	-46.1834
	std	1.4123	0.1884	3.56493	1.5693e-02	8.7847

Table 4.2c: Performance Evaluation on Multi-modal functions

Algorithm	Metric	Cross-in-tray	Goldstein price	Griewank	Holder Table	Kowalik
		F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
	global	-2.062612	3	0	-19.2085	0.0004075
SAO	best	-2.062611	84.0071	7.835E-08	-9.8376	0.027
	worst	-2.062586	84.7101	7.796E-06	-7.9215	0.039
	mean	-2.062604	84.2397	3.69E-06	-9.0532	0.0293
	median	-2.062607	84.1581	4.27E-06	-9.0407	0.0283
	std	7.5081E-06	0.21781	2.582E-06	0.5376	0.035
FFOA	best	-2.62612	601.2938	-1	-1.6487	2.698E-11
	worst	-2.6261	601.3128	-1	-1.6487	1.468E-08
	mean	-2.6261	601.304	-1	-1.6487	4.891E-09
	median	-2.6261	601.3039	-1	-1.6487	3.625E-09
	std	0	0.0053	9.181E-13	7.043E-10	4.824E-09
GBMO	best	-2.062611	18.4951	0.03094	-0.0024	0.004
	worst	-2.0626	19321.82	0.03094	-0.001	0.0781
	mean	-2.0626	3073.826	0.03094	-0.00116	0.0368
	median	-2.0626	979.6697	0.03094	-0.001	0.03855
	std	2.1071E-10	5885.708	0.0000	0.00044	0.026

Table 4.2d: Performance Evaluation on Multi-modal functions

Algorithm	Performance	Levi & Moltelvo	Michalewicz	Mishra	McCormick	Neumaier F3
		F ₁₆	F ₁₇	F ₁₈	F ₁₉	F ₂₀
	global	0	-4.687	0	-1.9133	-2
SAO	Best	1.4998E-32	-4.9981	4.0205E-118	-0.9346	-1.9999
	worst	1.4998E-32	-4.9999	4.8112E-115	-0.9346	-1.9991
	mean	1.4998E-32	-4.9994	7.8931E-116	-0.9346	-1.9996
	median	1.4998E-32	-4.9996	1.5524E-116	-0.9346	-1.9996
	Std	0	59397	1.4842E-115	3.0363E-07	0.00031391
FFOA	Best	0.0841	-0.998	2	0	1.0945E-13
	worst	0.0846	-0.9876	2	0	5.7125E-09
	mean	0.0844	-0.991	2	0	9.9389E-10
	median	0.0844	-0.9895	2	0	2.9283E-10
	Std	0.00017905	0.0037	0	0	1.7549E-09
GBMO	Best	0.0000	-29.9499	5.8843E-07	1.4774	-1.0000
	worst	0.0000	-29.5912	3.3146E-06	1.4775	-1.0000
	mean	0.0000	-29.8139	2.1325E-06	1.4778	-1.0000
	median	0.0000	-29.8638	2.09489E-06	1.4784	-1.0000
	Std	0.0000	0.1413	0.00015649	1.4774	0.0000

Table 4.2e: Performance Evaluation on Multi-modal functions

Algorithm	Metric	Quadratic	Rastrigin	Rosenbrock	Sphere	S. Tang
		F ₂₁	F ₂₂	F ₂₃	F ₂₄	F ₂₅
	global	-3877.24	0	0	0	-78.3319
SAO	best	-3873.72	2.198E-05	0.0000	3.7096E-12	-78.2771
	worst	-3873.73	0.0028	0.0000	2.1461E-10	-63.9429
	mean	-3873.71	0.0013	0.0000	6.2752E-11	-69.3073
	median	-3873.75	0.0013	0.0000	2.9897E-11	-67.4573
	std	0.000254	0.0009758	0.0000	7.3726E-11	5.6842
	FFOA	best	-3870.8	0.0001408	0.0000	7.1799E-07
worst		-3870.8	0.0001463	0.0000	7.3227E-07	0.0021
mean		-3870.8	0.000144	0.0000	7.2543E-07	-5.0042
median		-3870.8	0.000144	0.0000	7.2514E-07	0.0021
std		4.2E-09	1.588E-06	0.0000	4.644E-09	10.5542
GBMO	best	-3873.72	7.9375e-02	0.0000	0.0400	-10.0000
	worst	-3873.71	1.99025	0.0000	0.04010	-10.0000
	mean	-3873.71	1.9904	0.0000	0.0402	-10.0000
	median	-3873.72	1.9908	0.0000	0.04011	-10.0000
	std	0.0009318	0.0081	0.0000	5.3325e-05	0.0000

Table 4.2f: Performance Evaluation on Multi-modal functions

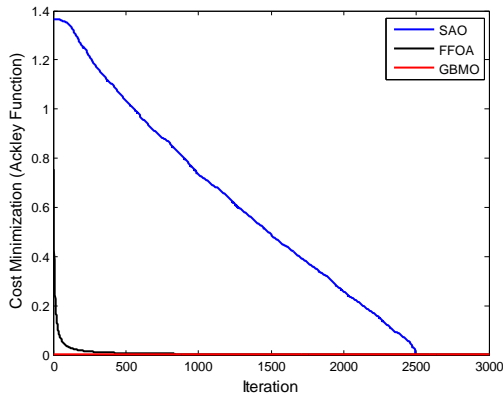
Algorithm	Metric	Sal	Schwefel	Subert	Yang
	global	F_{26} 0	F_{27} 0	F_{28} -186.73	F_{29} 0
SAO	best	4.4111E-07	5.9032E-09	-186.5393	0.0000
	worst	4.2088E-06	7.3149E-08	-185.4934	0.0000
	mean	1.4153E-06	3.7148E-08	-186.1144	0.0000
	median	1.0483E-06	3.715E-08	-186.1111	0.0000
	std	1.1045E-06	2.2601E-08	0.3172	0.0000
FFOA	best	9.8854E-05	0.00002472	7.9782E-13	0.00084502
	worst	0.0001003	0.000025314	1.7455E-10	0.000858
	mean	9.9531E-05	0.000024974	4.6382E-11	0.00085219
	median	9.9548E-05	0.000024984	2.2246E-11	0.00085212
	std	4.7598E-07	2.1575E-07	5.9111E-11	4.6259E-06
GBMO	best	0.095212	0.1742	-14.4274	7.0936e-03
	worst	0.099874	1.7823	-4.8745	0.3299
	mean	0.099407	0.3360	-6.0978	0.1783
	median	0.099873	0.1753	-6.1038	0.1857
	std	0.0014738	0.5081	0.06788	0.1073

The comparisons of SAO with the FFOA and GBMO on multimodal functions are given in Tables 4.2a to 4.2f. The results presented in these tables were obtained after each of the twenty-nine multimodal functions were run ten (10) times. Just like the unimodal test case, the best, worst, mean, median and standard deviations of the ten runs were recorded and only the best of the runs were used for discussion. However, the same discussions using the best results can also be done using any of the other evaluation as a metric.

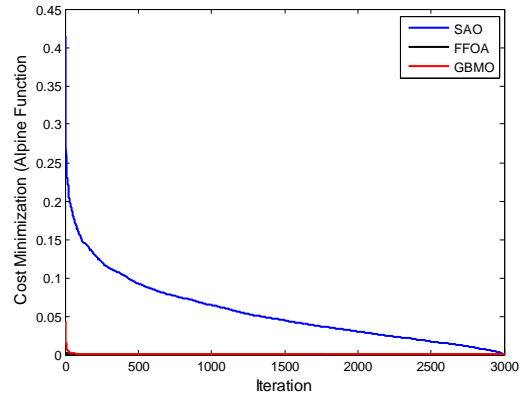
Comparing the performance of SAO with that of the FFOA on the multimodal functions, it can be observed that the SAO obtained the best result in twenty-three (23) out of the twenty-nine (29) multimodal test functions. This constituted 79.31% of the test functions. The FFOA obtained the best result in four (4) out of the twenty-nine (29) which constituted a total of 13.79% of the multimodal test functions. However, both

algorithms obtained the same optimal result for Bohachevsky and Rosenbrock test functions as shown in Tables 4.2b and 4.2e respectively. This constituted 6.80% of the total multimodal test functions. Thus, it can be seen that the developed algorithm in this research is also efficient in solving optimization problems of multimodal form.

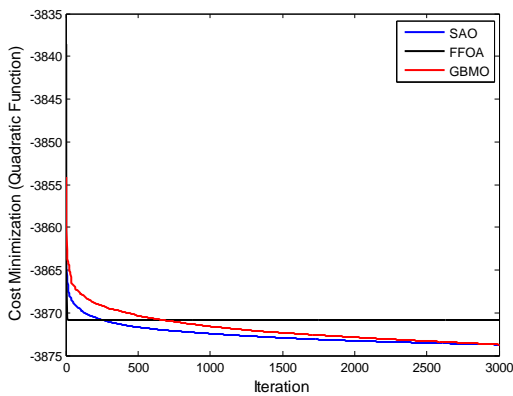
Also, comparing the performance of SAO with that of the GBMO, the SAO obtained the best results in twenty-one (21) out of the twenty-nine (29) test functions. This is an indication of 72.41% of the test functions. The FFOA produced the best results in six (6) constituting a total of 20.69% of the multimodal test functions. However, both algorithms obtained the same result in Cross-in-tray and Rosenbrock test functions also constituting a total of 6.89% of the multimodal test functions as given in Table 4.2c and Table 4.2d respectively. The improved performance of SAO over FFOA can be attributed to the number of mode (sniffing, trailing and random) modelled into the SAO algorithm. To show the searching of the algorithms on the multimodal functions, the optimization process of the algorithms on Ackley, Alpine, Quadratic and Levi & Montelvo function are shown in Figure 4.2



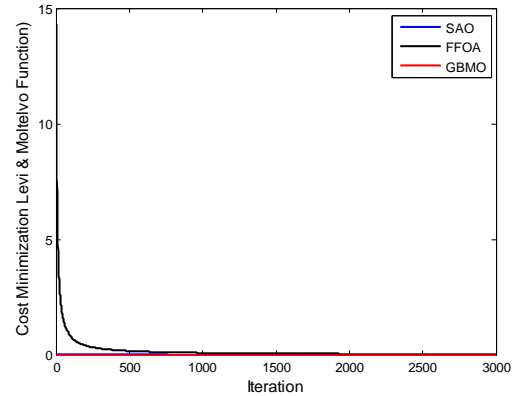
(b) Ackley Function



(a) Alpine F1 Function



(d) Quadratic Function



(c) LM1 Function

Figure 4.2: Minimization of Multimodal Functions

From Figure 4.2, it can be observed that, both the FFOA and GBMO converged faster than the SAO except in quadratic test functions where SAO and GBMO obtained similar optimum result. Just like in the case of unimodal test functions, this early convergence of FFOA and GBMO is attributed to lack of proper guiding parameters and unavailability of alternative mode.

The pie chart given in Figure 4.3 shows the overall percentage performance of the SAO, FFOA and GBMO on the thirty-nine (unimodal and multimodal) benchmark functions used to evaluate the performance of the algorithm. From the result given in Tables 4.1 and 4.2, the total number of functions where SAO obtained the best results is 22 when compared with FFAO and GBMO. The FFOA obtained the best results in 4 of the functions while the GBMO obtained the best results in 7 of the test functions. The

number of times all the algorithms obtained the same results were determined to be 6.

This information is used to generate the pie chart given in Figure 4.3

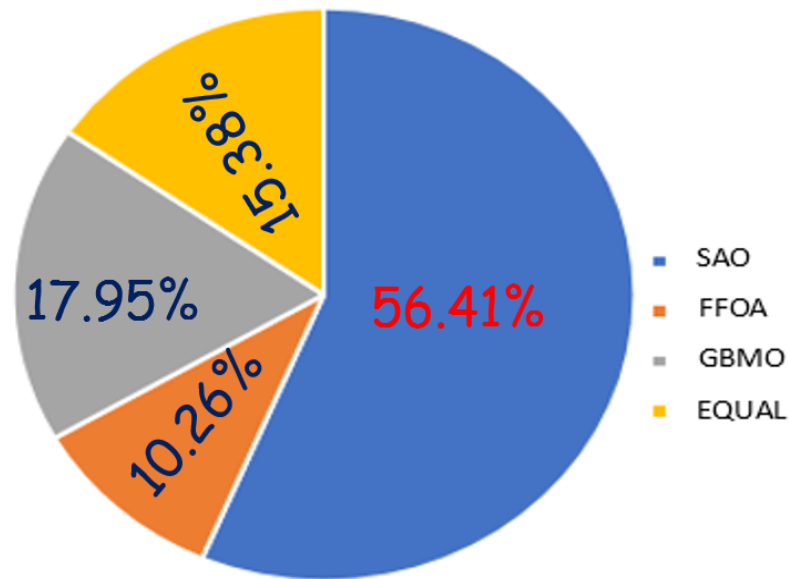


Figure 4.3: Performance Comparison Chart

From Figure 4.3, it can be observed that the SAO obtained the best results in 56.41% of the total thirty-nine benchmark functions. The FFOA obtained the best results in 10.26% of the functions while the GMBO obtained the best result in 17.95% of the total test functions. However, all or any two of the algorithms obtained the same results in 15.38% of the functions.

The performance of the developed algorithm was also evaluated using convergence to optimal results as performance metric. Table 4.3 show the time it took each algorithm to converge towards the best result. From the table, the f_{metric} of each function is given in the $f_{metrics(*)}$ column and the time it takes the algorithms to evaluate the functions are given in columns 4, 5, 6 for SAO, FFOA and GBMO respectively. The essence of including the f_{metric} in this table is to ascertain if the convergence of the algorithms on each function is dependent on the complexity of the function represented by the f_{metric} values.

Table 4.3 Average Time of Convergence (sec)

SN	Function Name	$f_{\text{metrics}}(^*)$	SAO	FFAO	GBMO
1	Ackley	320	6.3296	2.2341	9.7699
2	<i>Adjiman</i>	440	6.4326	2.6706	7.0205
3	<i>Alpine 1</i>	440	6.5437	2.37656	7.3627
4	Beale	320	6.1117	2.32564	8.7575
5	Bird	440	6.2225	2.43996	7.9693
6	Bohachevsky	320	6.2741	2.06609	10.1313
7	Booth	260	5.7771	2.31092	6.7086
8	Box	560	8.9532	2.69029	10.7175
9	Bukin F6	260	6.1250	2.44285	8.6679
10	Colville	680	6.6628	2.53470	6.9503
11	Cosine Mixture	320	8.6947	2.84637	7.6559
12	Cross-in-tray	440	7.6968	2.66483	11.9607
13	Dejong F4	260	7.7881	2.75462	8.8606
14	Easom	320	7.2393	2.61329	8.9666
15	Egg Crate	260	7.4377	2.60536	6.2853
16	Ellipsoid	1100	13.5542	2.80670	6.0591
17	Goldstein price	320	12.7933	3.31124	13.9504
18	Griewank	2000	14.0259	3.16405	20.7109
19	Holder Table	440	12.8667	3.09322	8.3786
20	Kowalik	2120	12.3252	3.33186	13.2540
21	Levi & Montalvo F1	320	9.73788	3.36002	7.4933
22	Michalewicz	800	10.3615	3.16325	11.9271
23	Matyas	320	8.18485	2.91108	9.6448
24	Mishra	2000	19.7427	3.29485	6.8498
25	McCormick	560	14.0176	3.94137	9.9128
26	Neumaier F3	320	10.3724	3.46227	18.9298
27	Quadratic	1100	9.93026	3.08961	14.5705
28	Rastrigin	800	14.4531	3.32078	5.4842
29	Rosenbrock	440	10.2580	2.88137	7.8792
30	Sphere	3800	11.1713	3.48528	10.7685
31	Styblinkis's Tang	7400	10.8942	3.48052	14.5925
32	Step	275	10.2662	3.45589	11.8376
33	Sal	2000	19.2401	3.67368	12.9371
34	Schaffer	440	10.2531	3.10969	7.3860
35	Schwefel	440	10.9808	3.61163	8.6351
36	Subert	1160	10.2781	3.59844	15.6946
37	Watson	3080	20.7386	21.3093	10.5310
38	Yang	320	11.6526	3.83180	8.1109
39	Zirilli	320	10.1939	3.505632	11.5264

From Table 4.3, it can be observed that, the FFOA evaluated the functions much faster than the SAO and the GBMO. Comparing the convergence of SAO with that of the FFOA, it can be seen that the FFOA converged at least three times faster than the SAO except in the case of Watson functions where the SAO had a slightly faster convergence. The fast convergence of FFOA over SAO is attributed to the number of functions evaluated by each algorithm. For example, the time required by FFOA to evaluate a function is more or less the same as the time it takes SAO to evaluate the same function using one of its available three modes. In the case of Watson function, the SAO converged faster than the FFOA by 0.5706sec. It was actually not expected that the SAO will converge faster than the FFOA in any of the test functions due to its number of modes, perhaps this situation may be due to the characteristics of Watson functions described in item 37 of subsection 2.2.6 and in Table 3.3. The practical implication of this is that, even though the FFOA converged faster than the SAO, the nature and characteristics of the function also play a significant role in the convergence rate of algorithms.

Comparing the convergence rate of the SAO and that of the GBMO, it can be observed that the SAO had faster convergence in fifteen (15) of the test functions while the SAO converged faster in twenty-four (24) test functions. Just like in the case of FFOA, the convergence rate of both the GBMO and the SAO are also influenced by the nature of the functions to be evaluated.

4.4 Application to Path Planning

The developed smell agent optimization was applied to the path planning model described in Section 3.8 of chapter three. As described in chapter three, the path planning environment contains a total of five obstacles of different sizes strategically placed, such that, the robot has no direct line of sight with the goal. This makes it trivial

for the smell agent algorithm to plan an optimized robot path easily. The optimum path obtained by the smell agent optimization is given in Figure 4.4.

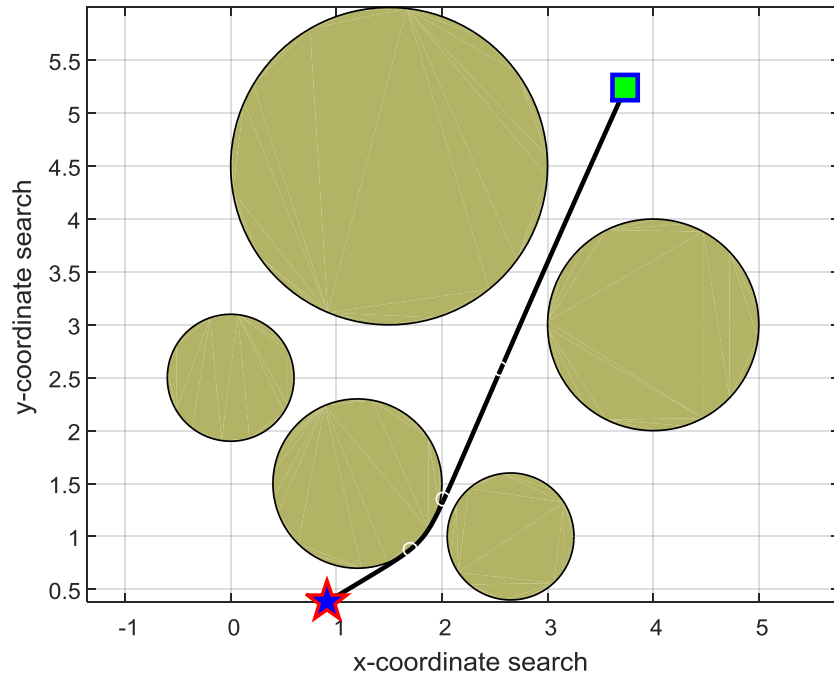


Figure 4.4: Online Path Planned by SAO

Figure 4.4 shows the optimum path found when the developed smell agent optimization algorithm was used. It can be seen that the SAO obtained an obstacle free optimal path and the robot can then follow this path to the goal in an online manner. The SAO obtained the optimized path after a series of an online alternative path has been evaluated.

The optimum path found by the SAO was compared with the path determined using PSO and SDA optimization. The path planning environment model setting is made to have the same condition and parameters as the one modelled when the SAO is used. This is to ensure that the algorithms have the same conditions and basis for comparisons and validation. Figure 4.5 shows the path planned by particle swarm optimization.

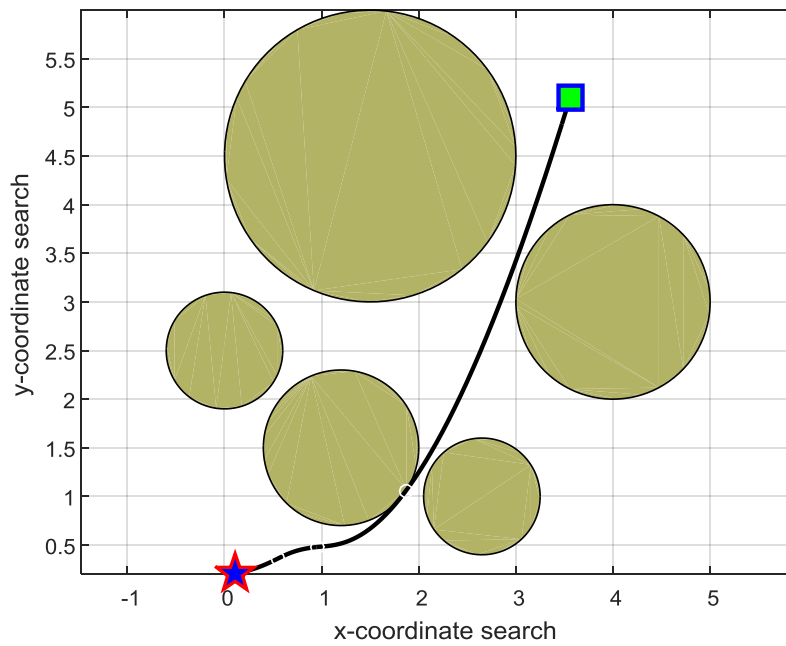


Figure 4.5: Path Planned by PSO

From Figure 4.5, it can be observed that the PSO algorithm also found an obstacle free path which the robot can follow to the goal. Comparing Figure 4.4 with Figure 4.5, it can be seen that, both SAO and PSO obtained the same path between the robot and the goal. However, the smell agent optimization optimized the path much better than the PSO as detailed in Table 4.4. Figure 4.6 shows the path planned when the SDA algorithm was used.

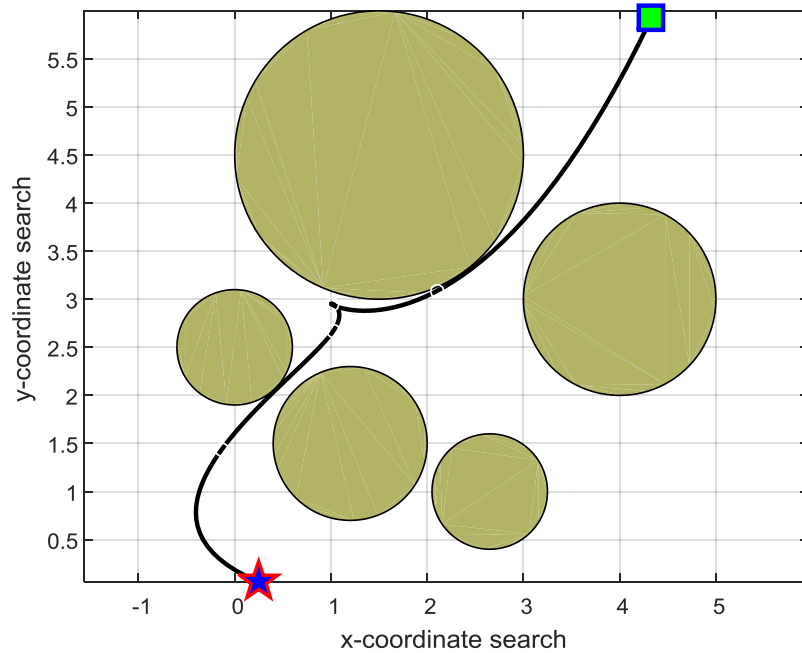


Figure 4.6: Path Planned by SDA

From Figure 4.7, it can be observed that the smell detection agent algorithm also found an obstacle free path. Unlike the SAO and PSO, the SDA obtained a much complicated and longer path for the robot. Though this path is longer, the robot can also move successfully from the starting position to goal position in an online manner without any obstruction by the obstacles. Table 4.4 shows the path planning fitness function obtained by each algorithm and the time it takes the algorithms to plan the optimized paths.

Table 4.4: Performance Comparison on Path Planning

SN	Algorithm	Fitness Value	Time (Sec)
1	SAO	5.7635	81.8764
2	PSO	6.4209	86.7963
3	SDA	10.5642	77.1479

Comparing the performance of the algorithms given in Table 4.4, it can be observed that the fitness value obtained by SAO is better than that obtained by PSO and SDA. With

respect to PSO, the SAO obtained a fitness improvement of 11.41%. When compared with SDA, the SAO recorded a fitness improvement of 83.29%. In terms of simulation time, the SAO obtained the optimized path faster with 5.67% less computational time when compared with the PSO. However, the SDA performed better than the SAO in computation time by obtaining its optimum path in 6.13% less computational time than the SAO.

The plots showing the convergence process of the algorithms on the path planning cost functions is shown in Figure 4.7.

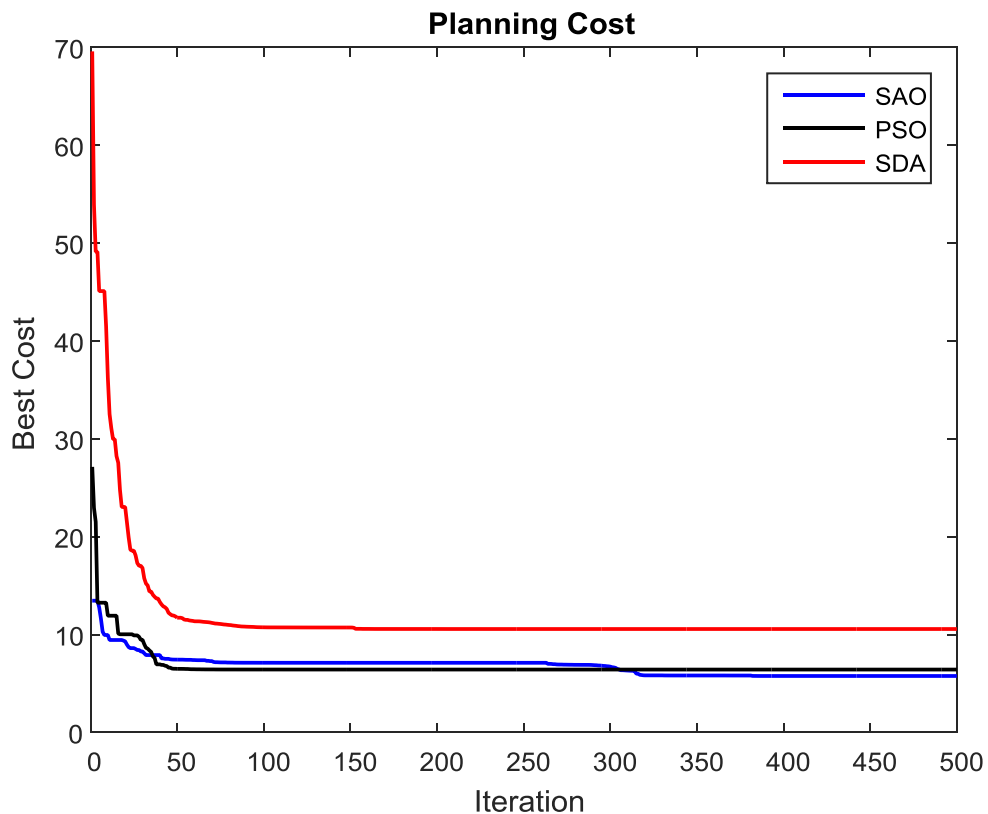


Figure 4.7: Path Planning Cost Function

From Figure 4.7, it can be observed that, both PSO and SDA converged to a solution in less than 200 iterations. However, the SAO was able to improved it results beyond 200 iterations, resulting in the percentage improvement stated in the discussion of Table 4.4.

Unlike in the PSO and SDA, the SAO have a mechanism to restrict it searching within the region of best solutions, accounting for its improvement over the PSO and SDA.

4.5 Application to Minimum Spanning Tree

Just like in the case of path planning model, the developed smell agent optimization was applied to the model of minimum spanning tree problem described in subsection 3.9. In order to justify the efficiency of the developed SAO on the MST problem, three scenarios of the MST model described in subsection 3.9 were used. In the first scenario, only the first fifteen (15) vertices given in Table 3.5 were used. In the second scenario, the number of vertices was increased to twenty (20). While in the third scenario the entire thirty (30) vertices were used. In each scenario, Euclidian distance and all possible spanning edge connection were used. The Euclidian distance forms the edge weights whose minimum combination is to be determined by the algorithms. Thus, the cost function in this case is a function of the distances between the vertices. The result was compared with the result obtained using PSO and SDA algorithms. Figure 4.8 shows the MST obtained when the algorithms were used in the first scenario.

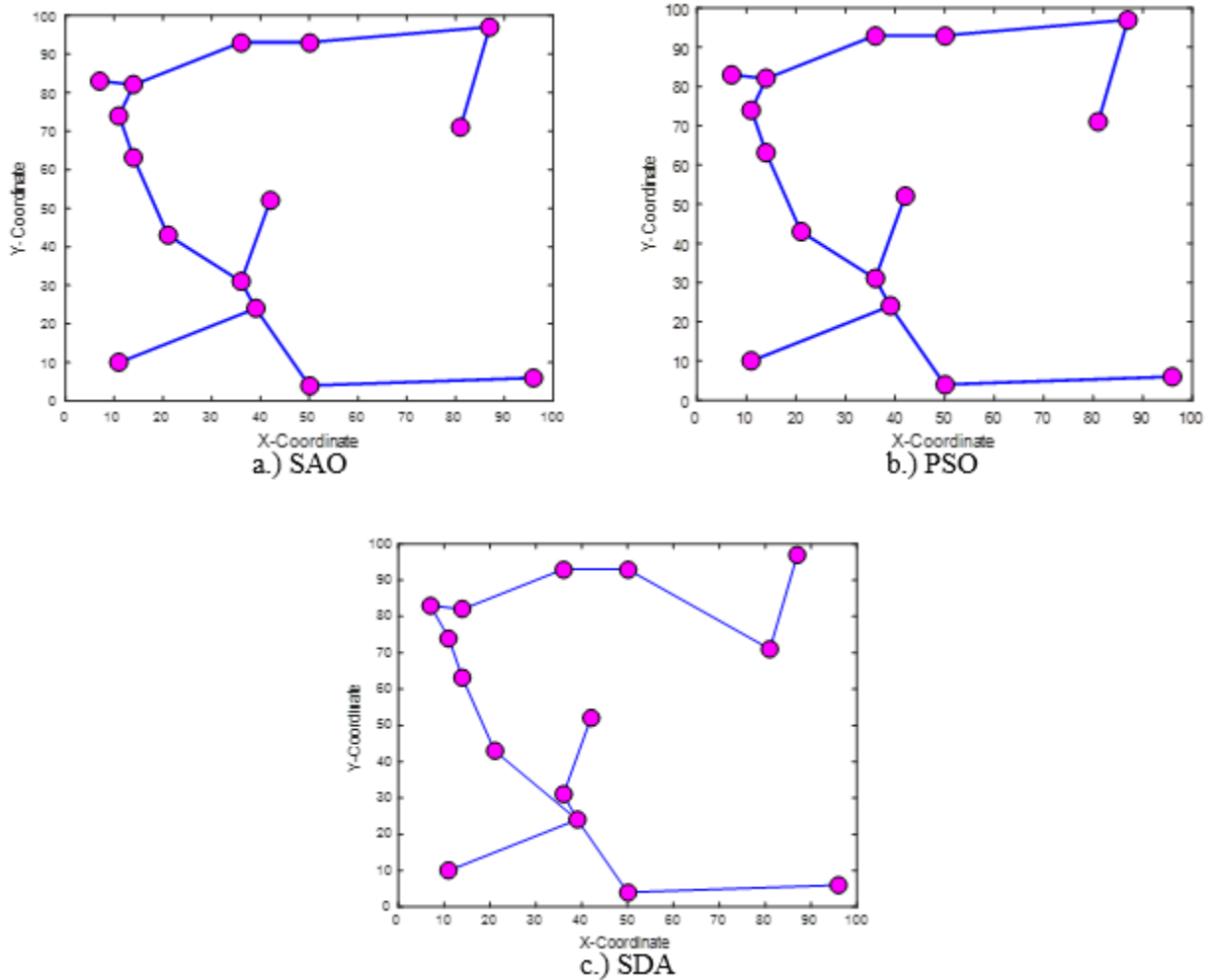


Figure 4.8: Minimum Spanning Tree Solution (First Scenario)

From Figure 4.8, it can be observed that both SAO and PSO obtained the same minimum spanning tree. With respect to table 3.6, the optimum combination of vertices which gives the MST using SAO and PSO is deduced from the Figure 4.8a as (V7, V8), (V8, V14), (V14, V9), (V14, V5), (V5, V2), (V5, V10), (V10, V1), (V1, V15), (V15, V12), (V12, V4), (V12, V11), (V11, V3), (V3, V6) and (V6, V13). In the case of SDA optimization, the optimum combination of the vertices obtained was deduced from Figure 4.8c as (V7, V8), (V8, V14), (V14, V9), (V14, V5), (V5, V2), (V14, V10), (V10, V1), (V1, V15), (V15, V4), (V4, V12), (V12, V11), (V11, V3), (V3, V13) and (V13, V6). This shows that the vertices of the MST obtained by SDA are different from those obtained by SAO and PSO at the connection of (V14, V10), (V15, V4), (V4, V12), (V3,

V13) and (V13, V6). The MST obtained by the SAO, PSO and SDA algorithm on the second scenario is given in Figure 4.9.

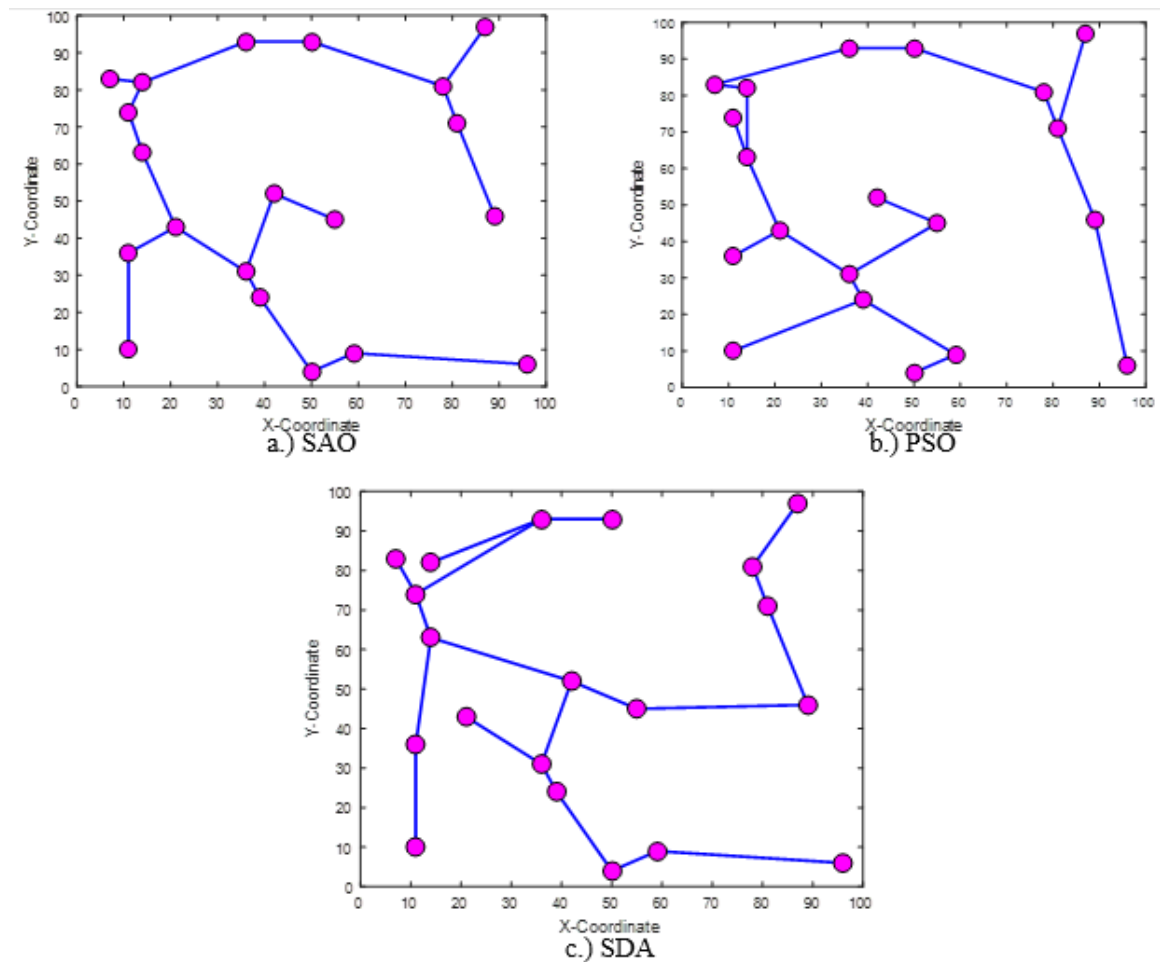


Figure 4.9: Minimum Spanning Tree Solution (Second Scenario)

From Figure 4.9, it can be seen that the three algorithms obtained different optimized MST. The optimum Euclidian distances between vertices which forms the MST obtained by SAO is deduced from the figure as (V7, V16), (V16, V8), (V8, V140), (V14, V5), (V5, V2), (V5, V2), (V2, V20), (V5, V10), (V10,V19), (V19, V9), (V10, V1), (V1, V15), (V15, V12) (V12, V4), (V12, V11), (V11, V3), (V3, V17), (V17, V13), (V13, V18) and (V17, V6). In the case of PSO, the Euclidian distances which form the minimum spanning tree is obtained as (V8, V16), (V16, V14), (V14, V9), (V14, V5), (V5, V20), (V20, V2), (V5, V10), (V10, V19), (V10, V1), (V1, V15), (V1, V12), (V12, V4), (V4, V11), (V11, V3), (V3, V17), (V17, V13), (V13, V6), (V13,

V18), (V18, V7). Likewise, the MST path obtained by the SDA in the second scenario is deduced as (V17, V16), (V16, V8), (V8, V14), (V14, V5), (V5, V10), (V5, V2), (V2, V20), (V20, V18), (V18, V13), (V13, V17), (V17, V6), (V5, V1), (V1, V19), (V19, V9), (V1, V15), (V15, V4), (V15, V11), (V11, V12) and (V11, V3). It can be seen that all the spanning tree obtained by the algorithms obey the spanning tree condition of $(n - 1)$ edges. The MST obtained by the SAO, PSO and SDA algorithm on the third scenario is given in Figure 4.10.

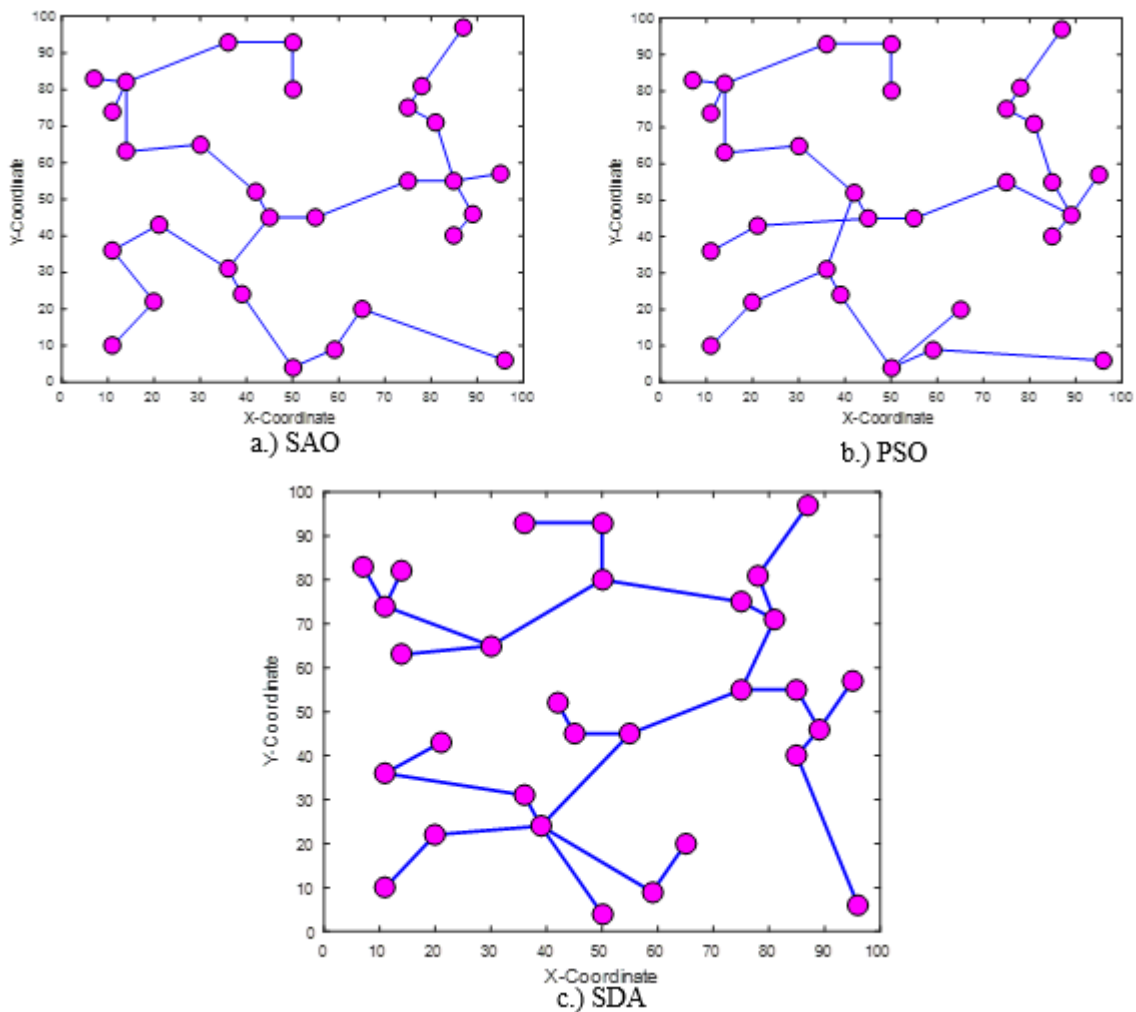


Figure 4.10: Minimum Spanning Tree Solution (Third Scenario)

From Figure 4.10, it can be observed that each algorithm obtained a unique MST when compared to the others. In the case of SAO, the Euclidean distances which form the MST is deduced as (V7, V21), (V21, V16), (V16, V8), (V8, V14), (V14, V5), (V5,

V10), (V10, V19), (V19, V25), (V25, V9), (V5, V29), (V29, V20), (V20, V22), (V22, V27), (V27, V18), (V18, V30), (V27, V28), (V27, V13), (V13, V24), (V24, V17), (V17, V6), (29, V2), (V2, V23), (V23, V1), (V1, V12), (V12, V15), (V12, V4), (V12, V11), (V11, V3) and (V3, V26). When PSO is used, the MST obtained is deduced as (V7, V16), (V16, V8), (V8, V21), (V8, V14), (V14, V5), (V5, V25), (V25, V9), (V5, V2), (V2, V29), (V29, V10), (V10, V19), (V29, V20), (V20, V22), (V22, V18), (V18, V30), (V18, V28), (V18, V27), (V27, V13), (V13, V24), (V24, V17), (V17, V6), (V29, V2), (V2, V23), (V23, V1), (V1, V12), (V12, V4), (V12, V15), (V12, V11), (V11, V3) and (V3, V26). Likewise, the spanning tree obtained by the SDA is deduced as (V9, V25), (V25, V14), (V14, V5), (V5, V19), (V19, V10), (V14, V8), (V14, V16), (V16, V21), (V14, V20), (V20, V29), (V29, V2), (V20, V22), (V22, V27), (V27, V18), (V18, V30), (V18, V28), (V30, V7), (V22, V13), (V13, V24) (V13, V17), (V17, V6), (V24, V26), (V26, V3), (V3, V11), (V26, V23), (V23, V15), (V23, V1), (V15, V4), and (V15, V12). In order to compare the performance of the algorithms on each of the test scenario, the optimum cost obtained for the total edges and the computational time by each algorithm was computed as shown in Table 4.5.

Table 4.5: Performance evaluation on MST

Algorithm	Cost	Time (s)
First Scenario		
SAO	599.0828	53.4974
PSO	599.0828	63.1263
SDA	617.2140	52.1038
Second Scenario		
SAO	658.3846	69.2453
PSO	763.5591	74.8316
SDA	794.4630	67.8509
Third Scenario		
SAO	819.3417	107.6576
PSO	892.5694	121.7420
SDA	935.2266	110.0202

From Table 4.5, it can be observed that, both SAO and PSO obtained the same result for the cost function in the first scenario consisting of fifteen vertices. Comparing with the SDA, both SAO and PSO had 3.03% improvement with respect to the cost function. In terms of simulation time, the SAO obtained the minimum spanning tree faster than the PSO, with 18.00% less computational time. When compared with the SDA, the computational time of SAO was 2.61% more than the SDA.

In the second scenario, the SAO obtained the best result with a percentage of 15.97% and 20.67% over PSO and SDA respectively. The SAO also took 8.07% less computational time to obtain its optimum spanning tree in comparison with the PSO. However, comparing the computational time of SAO with SDA, the SDA appears to be faster with 2.06% less computational time than the SAO.

Similarly, in the third scenario, the SAO also obtained the best results for the cost function. When compared with PSO, the SAO has 8.94% cost improvement while a 14.14% cost improvement over SDA was also recorded by the SAO. With respect to the computational time, the SAO obtained the optimum spanning tree faster than the PSO with 13.08% less computational time. However, the SAO also obtained the optimum spanning tree faster than the SDA in this scenario with a 2.19% less computational time. This is an indication that the simulation time of SDA become higher as the complexity of the minimum spanning tree problem increases. The plots, showing the minimization process of the algorithms on the three scenario is shown as in Figure 4.11.

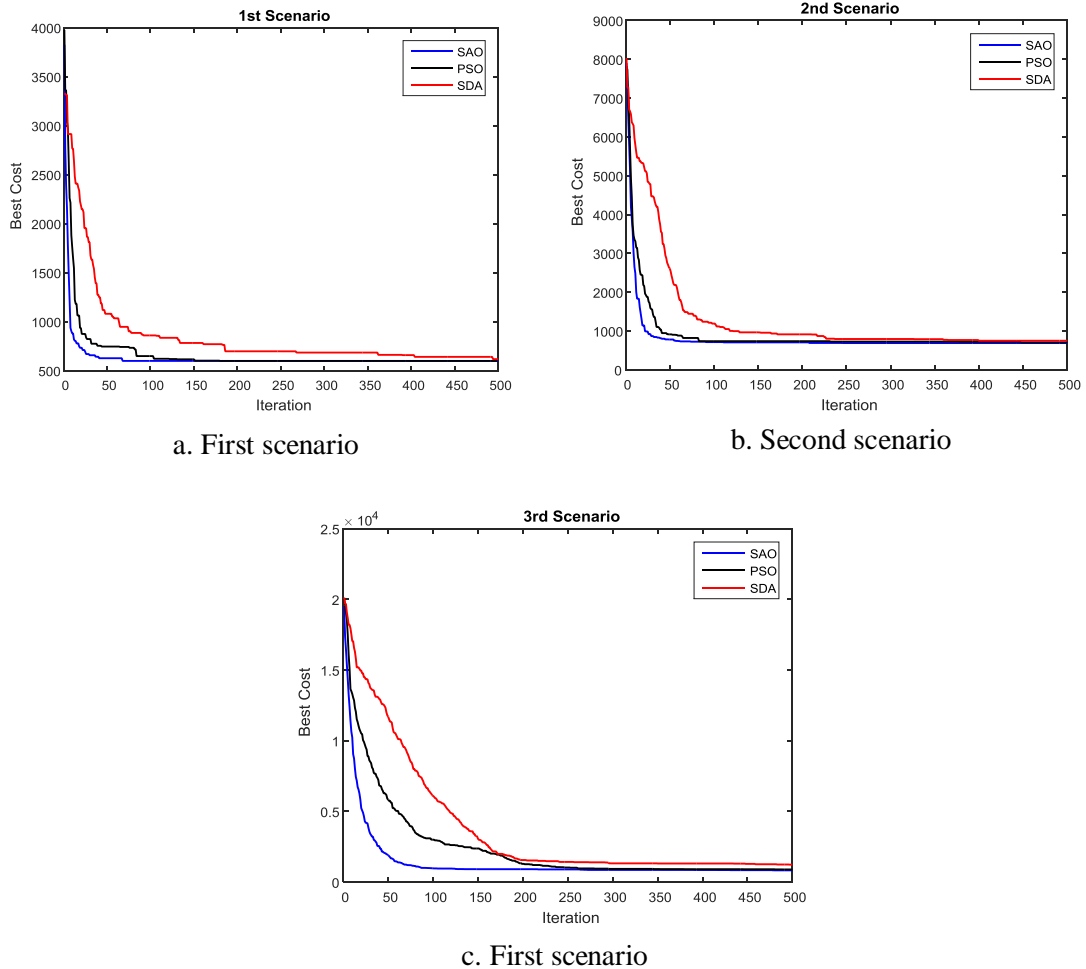


Figure 4.11: MST Cost Minimization Plots

It can be observed from Figure 4.11 that, the SAO and PSO converges faster than the SDA in all the scenario. In the first scenario, both SAO and PSO converges to a solution in less than 200 iterations while the SDA still shows possibility of improvement beyond the 500 iterations. In the second scenario, the SAO and PSO also converges within 200 iterations while the SDA converges after 300 iterations. In the third scenario, SAO still converges before 200 iterations. However, both PSO and SDA took a longer iteration (over 300 iteration) to converge to a solution.

4.6 Simulation with SAO GUI

The developed smell agent optimization graphical user interface was used to simulate all the benchmark functions with the aim of verifying the simulation performance of the

developed GUI. Figure 4.12 shows the simulation of Ackley benchmark function using the SAO GUI.

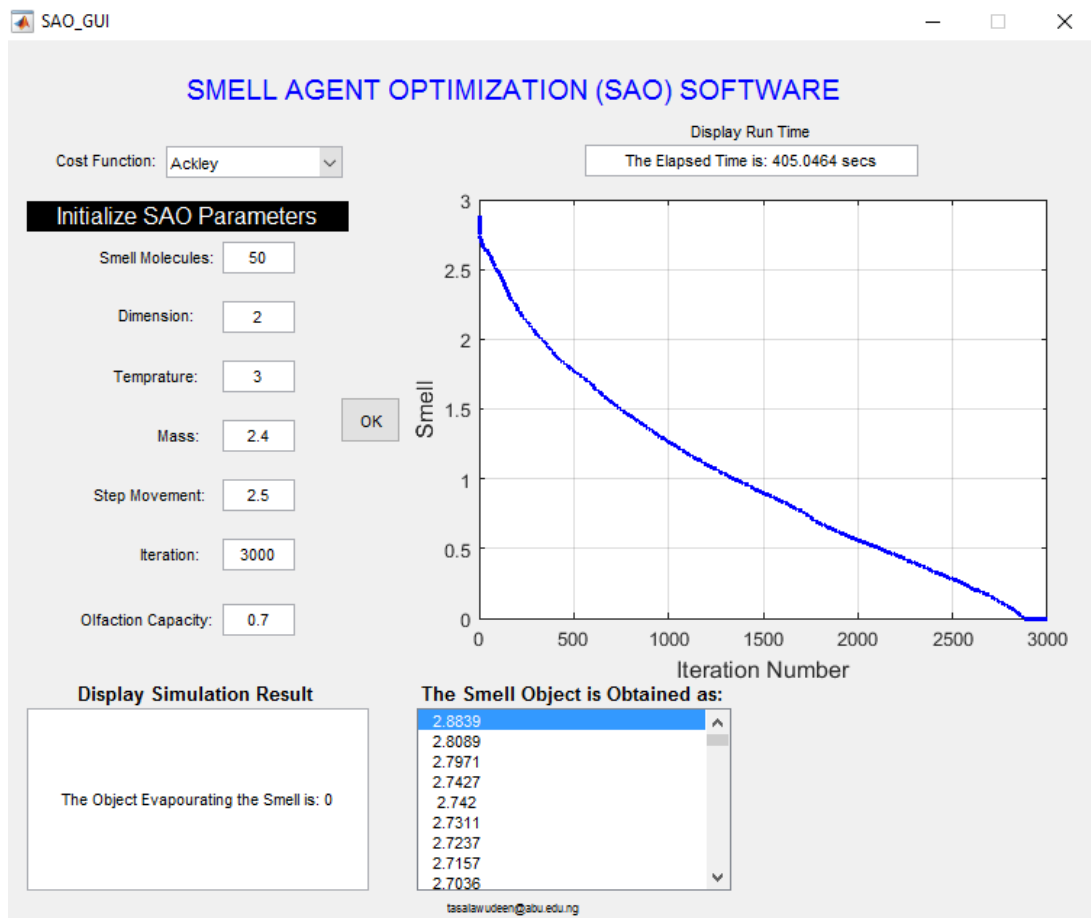


Figure 4.12: Simulation with SAO GUI

From Figure 4.12, it can be observed that, the developed SAO GUI has a feature that displays the response of the optimization process. During simulation, the progress of the algorithm is displaced in the smell object screen while the overall best result obtained by the algorithm is displayed as the object evaporating the smell.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 Summary

In this thesis, the development of a novel computational intelligent algorithm called the smell agent optimization has been presented. The concept of the algorithm was inspired by the smelling ability of various biological organisms and the intuitive behaviours of the agent to follow the smell trails and eventually identify the object generating the smell. The developed SAO consists of three basic modes, named as sniffing mode, trailing mode and random mode. The process of the SAO is initialized by randomly generating initial population of smell molecules. Each molecule is assigned a random velocity and the fitness of their initial position is evaluated. The molecule with the overall best fitness is selected as the agent and the sniffing mode (which is the basic mode of the SAO) was modelled from the concept of hydrostatic pressure of gas. The fitness of the sniffing mode is evaluated and the position of the agent is updated to the position of the sniffed molecule with the best fitness. The trailing mode is then developed using the current position of the agent and the position of the molecules with the worst sniffed fitness. The Brownian motion nature of the smell molecules makes it difficult for the agent to account for all the evaporating smell molecules. This can lead to the agent getting trapped in a “state of confusion” and consequently leading to the loss of smell trail. To account for this situation in the SAO, a random mode which allows the agent to take a random step in the search space was introduced. The performance of the SAO was first evaluated using a collection of 39 applied mathematical optimization benchmark functions. Performance of the SAO on the benchmark functions was compared with that FFOA and GBMO. Thereafter, the SOA was applied to path planning model and MST problems and results were compared with

PSO and SDA based path planning and MST problems. In order to ensure an easier simulation with the SAO, a graphical user interface was developed for users which have little or no knowledge of programming in MATLAB.

5.2 Conclusion

This thesis has presented the development of the SAO algorithm for solving various optimization problems. The SAO was developed using the information deduced from sense of smell of an agent based on chemistry, physics and biology perspective. This information was mathematically modelled into three modes (Sniffing, Trailing and Random) and the modes were codified as SAO using MATLAB 2017a.

The developed SAO was applied to evaluate the optimum value of thirty-nine applied mathematical optimization benchmark functions. The benchmark functions were carefully selected with diverse property and their complexity was determine through a formulated function metric. The performance of SAO on the benchmark functions was compared with FFOA and GBMO algorithm. All the algorithms were simulated for a total of 3000 iterations and results showed that, the SAO is highly efficient in obtaining the optimum of the test functions.

When compared with FFOA and GBMO, the developed SAO algorithm obtained the best result for the functions in 56.41% of the thirty-nine test functions while the FFOA and the GBMO obtained the best results in 10.26% and 17.95% of the total test functions respectively. Nonetheless, all the algorithms obtained the same results in 15.38% of the benchmark functions. The performance of the developed algorithm in terms of convergence rate was also evaluated and results showed that, FFOA is much faster than the SAO in all the test functions except in Watson functions. When the SAO

was compared with GBMO, it was observed that, both algorithms converged to their optimum solution almost at the same time.

The performance of SAO on MST and path planning problem and results shows that the SAO is efficient in solving these problems. Results showed that SAO performed better than PSO and SDA by 11.41% and 83.29% on the path planning respectively. In the case of MST, the SAO had similar performance with PSO but performed better with 3.03% over the SDA in the first scenario and had 15.97% and 20.67%, 8.94% and 14.14% over PSO and SDA on the second and third scenarios respectively in terms of the cost function.

5.3 Contributions to Knowledge

The contributions to knowledge of this thesis, which involves the development of a novel optimization algorithm inspired by the phenomenon of smell are listed as follows:

1. The thesis has developed a novel Smell Agent Optimization (SAO) algorithm for solving various degree of optimization problems was developed
2. The performance evaluation of the developed SAO through comparison with FFOA and GBMO on a total of 39 carefully selected applied mathematical optimization test functions. Results showed that the SAO performed better in 56.41% of the functions, the FFOA performed better in 10.26% of the function while the GBMO performed better in 17.95% of the functions. However, the algorithms obtained similar results in 15.38% of the functions.
3. The developed SAO was applied to solve a model of path planning problem. Simulation results when compared with PSO and SDA algorithm, showed that the SAO obtained the best results in terms of cost function with a percentage improvement of 11.41% and 83.29% over PSO and SDA respectively.

4. The developed SAO was also applied to minimum spanning tree problem and simulation results was also compared with that of PSO and SDA algorithms. In the three scenarios of MST considered, the SAO obtained a better cost with a percentage improvement of 15.97% and 8.94% over PSO on the second and third scenario respectively. However, both SAO and PSO obtained the same cost in the first scenario. When compared with SDA, the SAO performed better with a percentage improvement of 3.03%, 20.67% and 14.14% on the three scenarios respectively.
5. In this thesis, an interactive graphical user interface (GUI) simulator in order to ease simulation of the test functions with the SAO was developed.

5.4 Limitations

Although, the aim of this research which is the development of an optimization algorithm inspired by the phenomenon of smell has been successfully achieved, however, it could not establish that all the molecules evaporating from a smell source are accounted for by the agent. Thus, it is assumed that the agent only makes its decision on the smell molecules it perceived.

5.5 Recommendations for Future Work

Though, the SAO parameters (such as: temperature, mass, iteration, step movement and olfaction capacity) can take constant values, the choice of these values plays a significant role in the general performance of the algorithm. Thus, the following areas are highlighted for consideration.

- 1) In practical situations, increase in temperature of gas molecules increases its evaporation and the velocity of the gas. Since the smell molecules in SAO are

considered as gas molecules, a method to adaptively select temperature can be considered. This method should be such that, the temperature has a decreasing value as the algorithm moves towards the optimum solution. This will enable the algorithm to converge faster and eventually terminate the optimization process when the minimum (i.e. zero) value of the temperature is attained.

- 2) In this thesis, all the gas molecules were assumed to have a fixed mass. Perhaps, in practical situations, that is not always the case. Making the mass of the gas molecule adaptive in SAO can be considered. For example, larger value of mass favours the exploitation capability of the agent while smaller value will favour exploration.
- 3) It has been stated that, every smell agent has a specific capacity of olfaction. If this capacity is known for a particular animal (agent), the developed SAO can easily be termed the animal (agent) based optimization. However, there is no known empirical study specifying the numerical values of olfaction capacity of agents. Since this capacity plays a significant role in the developed SAO, determining this parameter empirically for a specific agent can be considered.
- 4) The possibility of developing a novel algorithm using other sensory systems such as sense of test, sense of feel and sense of hearing can be considered. For example, the system of sense of smell and sense of taste are coordinated through the chemo-sensation process. Thus, if an algorithm can be developed using sense of smell, similar algorithm can also be developed using sense of taste.
- 5) The developed SOA can be hybridized or cascaded with similar computational intelligent algorithm such as PSO, SDA, ABC AFSA etc. for improved performance.

- 6) The developed SAO can also be applied to problems related other fields like, image and signal processing, power systems, sensor networks etc.

REFERENCES

- Abdechiri, M., Meybodi, M. R., & Bahrami, H. (2013). Gases Brownian motion optimization: an algorithm for optimization (GBMO). *Applied Soft Computing*, 13(5), 2932-2946.
- Abdelkader, M., Shaqura, M., Ghommem, M., Collier, N., Calo, V., & Claudel, C. (2014). *Optimal multi-agent path planning for fast inverse modeling in UAV-based flood sensing applications*. Paper presented at the 2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings.
- Abdullah, N. R. H., Musirin, I., & Othman, M. M. (2010, 23-24 June 2010). *Computational intelligence technique for solving power scheduling optimization problem*. Paper presented at the 2010 4th International Power Engineering and Optimization Conference (PEOCO).
- Abedinia, O., Amjady, N., & Ghasemi, A. (2016). A new metaheuristic algorithm based on shark smell optimization. *Complexity*, 21(5), 97-116.
- Achtelik, M. W., Lynen, S., Weiss, S., Chli, M., & Siegwart, R. (2014). Motion- and uncertainty-aware path planning for micro aerial vehicles. *Journal of Field Robotics*, 31(4), 676-698. doi: 10.1002/rob.21522
- Ahmadigorji, M., & Amjady, N. (2016). A multiyear DG-incorporated framework for expansion planning of distribution networks using binary chaotic shark smell optimization algorithm. *Energy*, 102, 199-215.
- Amoore, J. E., & Hautala, E. (1983). Odor as an aid to chemical safety: odor thresholds compared with threshold limit values and volatilities for 214 industrial chemicals in air and water dilution. *Journal of applied toxicology*, 3(6), 272-290.
- Awad, M. K., El-Shafei, M., Dimitriou, T., Rafique, Y., Baidas, M., & Alhusaini, A. (2017). Power-efficient routing for SDN with discrete link rates and size-limited flow tables: A tree-based particle swarm optimization approach. *International Journal of Network Management*.
- Axel, R. (2005). Scents and Sensitivity: A molecular Logic Olfactory Perception (Nobel Lecture). *Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim*, 44, 18. doi: DOI: 10.1002/anie.200501726
- Basu, M. (2015). Modified particle swarm optimization for nonconvex economic dispatch problems. *International Journal of Electrical Power & Energy Systems*, 69, 304-312.
- Becerra, I., Murrieta-Cid, R., Monroy, R., Hutchinson, S., & Laumond, J. P. (2015). Maintaining strong mutual visibility of an evader moving over the reduced visibility graph. *Autonomous Robots*. doi: 10.1007/s10514-015-9477-5
- Ben-Tal, A., & Nemirovski, A. (2001). *Lectures on modern convex optimization: analysis, algorithms, and engineering applications* (Vol. 2): Siam.
- Bentes, C., & Saotome, O. (2012). *Dynamic swarm formation with potential fields and A* path planning in 3D environment*. Paper presented at the Proceedings - 2012 Brazilian Robotics Symposium and Latin American Robotics Symposium, SBR-LARS 2012.
- Bezdek, J. C. (1994). What is computational intelligence? : USDOE Pittsburgh Energy Technology Center, PA (United States); Oregon State Univ., Corvallis, OR (United States). Dept. of Computer Science; Naval Research Lab., Washington, DC (United States); Electric Power Research Inst., Palo Alto, CA (United States); Bureau of Mines, Washington, DC (United States).
- Bezdek, J. C. (2013). The History, Philosophy and Development of Computational Intelligence (How a Simple Tune Became a Monster Hit). *Ch. 1 in*

Computational Intelligence, [Ed. Hisao Ishibuchi], in *Encyclopedia of Life Support Systems(EOLSS)*, Developed under the Auspices of the UNESCO, Eolss Publishers, Oxford ,UK, [<http://www.eolss.net>].

- Bindima, T., & Elias, E. (2017). A novel design and implementation technique for low complexity variable digital filters using multi-objective artificial bee colony optimization and a minimal spanning tree approach. *Engineering Applications of Artificial Intelligence*, 59, 133-147.
- Black, S. M. (2014). Chemistery of Smell (chem 1010). from Dixie State College of Utah.
- Bradshaw, J. Anthrozoology Institute, School of Biological Sciences, University of Southampton. Retrieved 21/11/2017, 2017, from <http://maxshouse.com/Database toc.htm>
- Breer, H. (2008). The Sense of Smell. *Annals of the New York Academy of Sciences*, 1126(1), 1-6.
- Chandra, V. S. S. (2016). Smell Detection Agent Based Optimization Algorithm. *Journal of The Institution of Engineers (India): Series B*, 97(3), 431-436. doi: 10.1007/s40031-014-0182-0
- Chapman, S., Cowling, T. G., & Burnett, D. (1970). *The mathematical theory of non-uniform gases: an account of the kinetic theory of viscosity, thermal conduction and diffusion in gases*: Cambridge university press.
- Chen, L., Liu, C., Shi, H., & Gao, B. (2013). *New robot planning algorithm based on improved artificial potential field*. Paper presented at the Proceedings - 3rd International Conference on Instrumentation and Measurement, Computer, Communication and Control, IMCCC 2013.
- Chen, Y.-P., Li, Y., Wang, G., Zheng, Y.-F., Xu, Q., Fan, J.-H., & Cui, X.-T. (2017). A novel bacterial foraging optimization algorithm for feature selection. *Expert Systems with Applications*, 83, 1-17.
- Cheng, C., Zhu, D., Sun, B., Chu, Z., Nie, J., & Zhang, S. (2015). *Path planning for autonomous underwater vehicle based on artificial potential field and velocity synthesis*. Paper presented at the Canadian Conference on Electrical and Computer Engineering.
- Chung, C. J. (1997). *Knowledge-based approaches to self-adaptation in cultural algorithms*. (Ph.D Dessertation), Wayne State University, Unpublished.
- Chung, C. J., & Reynolds, R. G. (1998). CAEP: An Evolution-Based Tool for Real-Valued Function Optimization Using Cultural Algorithms. *International Journal on Artificial Intelligence Tool*, 7(3), 60.
- Chung, H. Y., Hou, C. C., & Liu, S. C. (2013). *Automatic navigation of a wheeled mobile robot using particle swarm optimization and fuzzy control*. Paper presented at the IEEE International Symposium on Industrial Electronics.
- Correa, J. E. (2005). The dog's sense of smell. *Alabama Cooperative Extension System, Alabama A&M*.
- Crystal, D. (2004). *The Cambridge encyclopedia of the English language*: Ernst Klett Sprachen.
- Delaite, A., & Pesant, G. (2017). *Counting Weighted Spanning Trees to Solve Constrained Minimum Spanning Tree Problems*. Paper presented at the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems.
- Delice, Y., Aydoğan, E. K., Özcan, U., & İlkay, M. S. (2017). A modified particle swarm optimization algorithm to mixed-model two-sided assembly line balancing. *Journal of Intelligent Manufacturing*, 28(1), 23-36.

- Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1), 29-41.
- Dorigo, M., & Stützle, T. (2003). The ant colony optimization metaheuristic: Algorithms, applications, and advances *Handbook of metaheuristics* (pp. 250-285): Springer.
- Doty, R. L. (2017). Psychophysical Testing of Human Olfactory Function *Springer Handbook of Odor* (pp. 59-60): Springer.
- Doty, R. L., Applebaum, S., & Zusho, H., R.Gregg . . (1985). Sex differences in odor identification ability: A cross-cultural analysis. *Neuropsychologia.*, 23 (5), 17. doi: doi:10.1016/0028-3932(85)90067-3.
- Eberhart, R., & Kennedy, J. (1995). *New optimizer using particle swarm theory*. Paper presented at the Proceedings of the 1995 6th International Symposium on Micro Machine and Human Science, Piscataway, NJ, United States Nagoya, Jpn.
- Elsaesser, R., & Paysan, J. (2007). The sense of smell, its signaling pathways, and the dichotomy of cilia and microvillus in olfactory sensory cells. *BMC Neuroscience*, 8.
- Elyas, S. H., Mandal, P., Haque, A. U., Giani, A., & Tseng, T.-L. B. (2014). *A new hybrid optimization algorithm for solving economic load dispatch problem with valve-point effect*. Paper presented at the North American Power Symposium (NAPS), 2014.
- English, D. O. (2007). Oxford English dictionary online: JSTOR.
- Flinn, J., Ortiz, H. S. C., & Yuan, S. (2016). *A Secure Routing Scheme for Networks with Unknown or Dynamic Topology using A-star Algorithm*. Paper presented at the Proceedings of the International Conference on Security and Management (SAM).
- Fox, C. A., McKinley, W. A., & Magoun, H. W. (1944). An Oscillographic Study of Olfactory System of Cats. *Journal of Neurophysiology*, 7(1), 1-16.
- Furton, K., & Myers, L. (2001). The scientific foundation and efficacy of the use of canines as chemical detectors for explosives. *Talanta*, 3(53), 13.
- Graham, R. L., & Hell, P. (1985). On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1), 43-57.
- Habib, N., Purwanto, D., & Soeprijanto, A. (2016, 28-30 July 2016). *Mobile robot motion planning by point to point based on modified ant colony optimization and Voronoi diagram*. Paper presented at the 2016 International Seminar on Intelligent Technology and Its Applications (ISITIA).
- Hamdani, E. H., & Døving, K. B. (2007). The functional organization of the fish olfactory system. *Progress in Neurobiology*, 82(2), 80-86. doi: <https://doi.org/10.1016/j.pneurobio.2007.02.007>
- Hansen, N. (2006). Compilation of results on the 2005 CEC benchmark function set. *Online, May*.
- Held, M., & Karp, R. M. (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6), 1138-1162.
- Jamil, M., & Yang, X.-S. (2013). A Literature Survey of Benchmark Functions For Global Optimization Problems. *International Journal of Mathematical Modeling and Numerical Optimization*, 4(2), 47. doi: 10.1504/IJMMNO.2013.055204
- Jiang, W., Shi, Y., & Zhao, W. (2017). Modified FOA Applied to Parameter Extraction of Flux-Gate Core. *Journal of Sensors*, 2017.

- Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of global optimization*, 39(3), 459-471.
- Lakkaraju, H., Kamar, E., Caruana, R., & Horvitz, E. (2017). Identifying Unknown Unknowns in the Open World: Representations and Policies for Guided Exploration.
- lei Li, X. (2001). *Jixi an Ian*. "Artificial Fish-swarm Algorithm: Bottom-up Optimization Model". Paper presented at the Trans Annual Meeting of Chinese Process Systems Engineering Society.
- Lei., L. X., Shao, Z. J., & Qian, J. X. (2002). Optimizing method based on autonomous animats: Fish-swarm Algorithm. *Xitong Gongcheng Lilun yu Shijian/System Engineering Theory and Practice*, 22(11), 32.
- Li, X., Tang, K., Omidvar, M. N., Yang, Z., Qin, K., & China, H. (2013). Benchmark functions for the CEC 2013 special session and competition on large-scale global optimization. *gene*, 7, 33.
- Linda, B. B. (2005). Unraveling the sense of smell (Nobel lecture). *Angewandte Chemie International Edition*, 44(38), 6128-6140.
- Luan, X.-Y., Li, Z.-P., & Liu, T.-Z. (2016). A novel attribute reduction algorithm based on rough set and improved artificial fish swarm algorithm. *Neurocomputing*, 174, 522-529.
- Lynch, P. J. (2006). Head anatomy with olfactory nerve. Olfactory bulb labeled in English. Retrieved 05/06/2018, 2018, from https://commons.wikimedia.org/wiki/File:Head_olfactory_nerve_-_olfactory_bulb_en.png
- Ma, S., Li, X., & Cai, Y. (2017). Delimiting the urban growth boundaries with a modified ant colony optimization model. *Computers, Environment and Urban Systems*, 62, 146-155. doi: <https://doi.org/10.1016/j.compenvurbsys.2016.11.004>
- Ma, Y., Zheng, G., Perruquetti, W., & Qiu, Z. (2015). Local path planning for mobile robots based on intermediate objectives. *Robotica*, 33(4), 1017-1031. doi: 10.1017/S0263574714000186
- Mac, T. T., Copot, C., Tran, D. T., & De Keyser, R. (2017). A Hierarchical Global Path Planning Approach for Mobile Robots based on Multi-Objective Particle Swarm Optimization. *Applied Soft Computing*.
- Malarvizhi, K., & Kumar, M. (2015, 26-27 Feb. 2015). *Particle Swarm Optimization tuned BELBIC controller for 8/6 SRM operation*. Paper presented at the Electronics and Communication Systems (ICECS), 2015 2nd International Conference on.
- Mandal, P., Barai, R. K., Maitra, M., & Roy, S. (2013). *Path planning of autonomous mobile robot: A new approach*. Paper presented at the 7th International Conference on Intelligent Systems and Control, ISCO 2013.
- Marques, L., Nunes, U., & de Almeida, A. T. (2006). Particle swarm-based olfactory guided search. *Autonomous Robots*, 20(3), 277-287.
- Martin, C. W. (2017). Effects of macrophyte-specific olfactory cues on fish preference patterns. *Aquatic Ecology*, 51(1), 159-165.
- Masehian, E., & Amin-Naseri, M. (2004). A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning. *Journal of Field Robotics*, 21(6), 275-300.
- Mashreghi, A., & King, V. (2017). *Time-communication trade-offs for minimum spanning tree construction*. Paper presented at the Proceedings of the 18th International Conference on Distributed Computing and Networking.

- Miller, C. C. (1924). The Stokes-Einstein law for diffusion in solution. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 106(740), 724-749.
- Momin, J., & Yang, X. S. (2013). A literature survey of benchmark functions for global optimisation problems. *International Journal of Mathematical Modelling and Numerical Optimisation*, 4(2), 150-194.
- Morrison, J. (2014). Human nose can detect 1 trillion odours (Publication no. doi:10.1038/nature.2014.14904). Retrieved 29-June-2017, from Nature International Weekly Journal of Science
- Mu'azu, M. B. (2006). *Forecasting and Modeling Statistical Phenomena using Neurofuzzy Logic: A Case Study of Rainfall Forecasting for Zaria*. (PhD.), Ahmadu Bello Univeristy, Unpublished.
- Mu'azu, M. B. (2016) Computational Intelligence: from the Tradition to the Inovative in Search for the Optimal Solution. Vol. 2. Nigeria: Ahmadu Bello University Press Limited, Zaria Kaduna State.
- Nešetřil, J., Milková, E., & Nešetřilová, H. (2001). Otakar Borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete mathematics*, 233(1-3), 3-36.
- Ni, J., Wang, K., Huang, H., Wu, L., & Luo, C. (2016). *Robot path planning based on an improved genetic algorithm with variable length chromosome*. Paper presented at the Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2016 12th International Conference on.
- Ninomiya, K., Kapadia, M., Shoulson, A., Garcia, F., & Badler, N. (2015). Planning approaches to constraint-aware navigation in dynamic environments. *Computer Animation and Virtual Worlds*, 26(2), 119-139. doi: 10.1002/cav.1622
- Norouzi, N., Sadegh-Amalnick, M., & Tavakkoli-Moghaddam, R. (2016). Modified particle swarm optimization in a time-dependent vehicle routing problem: minimizing fuel consumption. *Optimization Letters*, 1-14.
- Pan, Q.-K., Sang, H.-Y., Duan, J.-H., & Gao, L. (2014). An improved fruit fly optimization algorithm for continuous function optimization problems. *Knowledge-Based Systems*, 62, 69-83.
- Pan, W.-T. (2012). A new fruit fly optimization algorithm: taking the financial distress model as an example. *Knowledge-Based Systems*, 26, 69-74.
- Pan, W. T. (2011). A New Fruit Fly Optimization Algorithm: Taking the Financial Distress Model as an Example. *Knowledge-Based Systems*, 26, 69-74 doi: 10.1016
- Park, J.-B., Lee, K.-S., Shin, J.-R., & Lee, K. Y. (2005). A particle swarm optimization for economic dispatch with nonsmooth cost functions. *IEEE Transactions on Power systems*, 20(1), 34-42.
- Passino, K. M. (2002). Biomimicry of bacterial foraging for distributed optimization and control. *Control Systems, IEEE*, 22(3), 52-67.
- Poole, D., & Mackworth, A. (2010). *Artificial Intelligence: Foundations of Computational Agents* (Second ed.). Creative Commons Attribution-Noncommercial-No Derivative Works 2.5 Canada License: Cambridge University Press.
- Porter, J., Craven, B., Khan, R. M., Chang, S.-J., Kang, I., Judkewitz, B., Volpe, J., Settles, G., & Sobel, N. (2007). Mechanisms of scent-tracking in humans. *Nature neuroscience*, 10(1), 27.

- Pradhan, M., Roy, P. K., & Pal, T. (2016). Grey wolf optimization applied to economic load dispatch problems. *International Journal of Electrical Power & Energy Systems*, 83, 325-334.
- Pugh, J., & Martinoli, A. (2007). *Inspiring and modeling multi-robot search with particle swarm optimization*. Paper presented at the Swarm Intelligence Symposium, 2007. SIS 2007. IEEE.
- Ranaldi, A. (2011). Do Vibrating Molecules Give Us Our Sense of Smell. Retrieved 21 June 2017, 2017
- Rashid, R., Perumal, N., Elamvazuthi, I., Tageldeen, M. K., Khan, M. A., & Parasuraman, S. (2016). *Mobile robot path planning using Ant Colony Optimization*. Paper presented at the Robotics and Manufacturing Automation (ROMA), 2016 2nd IEEE International Symposium on.
- Rathore, C., & Roy, R. (2014). A novel modified GBMO algorithm based static transmission network expansion planning. *International Journal of Electrical Power & Energy Systems*, 62, 519-531.
- Richtmyer, R. D., & Burdorf, C. (1981). *Principles of advanced mathematical physics* (Vol. 2): Springer.
- Salawudeen, A. T. (2015). *Development of an Improved Cultural Artificial Fish Swarm Algorithm with Crossover*. (Master of Science Thesis), Ahmadu Bello University Zaria, Nigeria., Kubani. (25)
- Salgueiro, R., de Almeida, A., & Oliveira, O. (2017). New genetic algorithm approach for the min-degree constrained minimum spanning tree. *European Journal of Operational Research*, 258(3), 877-886.
- Seet, B.-C., Liu, G., Lee, B.-S., Foh, C.-H., Wong, K.-J., & Lee, K.-K. (2004). *A-STAR: A mobile ad hoc routing strategy for metropolis vehicular communications*. Paper presented at the International Conference on Research in Networking.
- Sheng, Y., Qin, Z., & Shi, G. (2017). Minimum spanning tree problem of uncertain random network. *Journal of Intelligent Manufacturing*, 28(3), 565-574.
- Siddiqui, A., & Hojjat, N. (2013). *Computational Intelligence: Synergies of Fuzzy Logic, Neural Networks and Evolutionary Computing*. USA: Joun Wiley & Son.
- Singh, A. (2009). An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem. *Applied Soft Computing*, 9(2), 625-631.
- Smell-(n.d). (2017). Smell. Retrieved June 28, 2017, from <http://www.dictionary.com/browse/smell>
- Smith, D. (2016). *Olfactory enrichment in Amur leopards*. Western Illinois University.
- Sookoian, S., Burgueño, A., Gianotti, T. F., Marillet, G., & Pirola, C. J. (2011). Odor perception between heterosexual partners: its association with depression, anxiety, and genetic variation in odorant receptor OR7D4. *Biological psychology*, 86(3), 153-157.
- Stevenson, R. J. (2009). An initial evaluation of the functions of human olfaction. *Chemical senses*, 35(1), 3-20.
- Sudhakara, P., & Ganapathy, V. (2016). Trajectory Planning of a Mobile Robot using Enhanced A-Star Algorithm. *Indian Journal of Science and Technology*, 9(41).
- Sundaram, K. M., Kumar, R. S., Krishnakumar, C., & Sugavanam, K. (2016). Fuzzy Logic and Firefly Algorithm based Hybrid System for Energy Efficient Operation of Three Phase Induction Motor Drives. *Indian Journal of Science and Technology*, 9(1).
- Syufy, F. (2017, 10/03/17). Amazing Facts About Cats Sense of Smell. Retrieved 22-11-2017, 2017, from <https://www.thespruce.com/sense-of-smell-552117>

- Taizhi, L., & Maoyan, F. (2017). A smooth local path planning algorithm based on modified visibility graph. *Modern Physics Letters B*, 1740091.
- Tang, K., Xiao, X., Wu, J., Yang, J., & Luo, L. (2017). An improved multilevel thresholding approach based modified bacterial foraging optimization. *Applied Intelligence*, 46(1), 214-226.
- Tang, K., Yáo, X., Suganthan, P. N., MacNish, C., Chen, Y.-P., Chen, C.-M., & Yang, Z. (2007). Benchmark functions for the CEC'2008 special session and competition on large scale global optimization. *Nature Inspired Computation and Applications Laboratory, USTC, China*.
- Tijani, S. A., & Mua'zu, M. (2015). *Stabilization of inverted pendulum system using intelligent Linear Quadratic Regulator controller*. Paper presented at the 2015 7th International Joint Conference on Computational Intelligence (IJCCI).
- Touhara, K. (2014). Odor and Pheromone Molecules, Receptors, and Behavioral Responses *The Olfactory System* (pp. 19-38): Springer.
- Tsai, H.-C. (2017). Unified particle swarm delivers high efficiency to particle swarm optimization. *Applied Soft Computing*, 55, 371-383.
- Turabieh, H., & Abdullah, S. (2011) A hybrid fish swarm optimisation algorithm for solving examination timetabling problems. & S. P. A. EnginSoft (Vol. Ed.): *Vol. 6683 LNCS. 5th International Conference on Learning and Intelligent Optimization, LION 2011* (pp. 539-551). Rome.
- Vogt, J. (2017). *Kinetic Theory Exam Survival Guide: Physical Chemistry* (pp. 147-173): Springer.
- Wang, S., Yang, J., Liu, G., Du, S., & Yan, J. (2016). Multi-objective path finding in stochastic networks using a biogeography-based optimization method. *Simulation*, 92(7), 637-647.
- Wang, X., Shi, Y., Ding, D., & Gu, X. (2016). Double global optimum genetic algorithm–particle swarm optimization-based welding robot path planning. *Engineering Optimization*, 48(2), 299-316.
- Wang, X. Q., Dou, A. X., Wang, L., Yuan, X. X., Ding, X., & Zhang, W. (2015). RS-based assessment of seismic intensity of the 2013 Lushan, Sichuan, China MS7.0 earthquake. *Chinese Journal of Geophysics (Acta Geophysica Sinica)*, 58(1), 163-171. doi: 10.6038/cjg20150114
- Wang, Y., Ma, J., & Wang, Y. (2017). *Application of ant colony algorithm in path planning of the data center room robot*. Paper presented at the AIP Conference Proceedings.
- Wheeler, S. (1990). Movement of large gas bubbles in unsaturated fine-grained sediments. *Marine Georesources & Geotechnology*, 9(2), 113-129.
- Winston, P. H. (1992). *Artificial Intelligence* (3 ed.): Addison-Wesley.
- Wolpert, D. H., & Macready, W. G. (1995). No free lunch theorems for search: Technical Report SFI-TR-95-02-010, Santa Fe Institute.
- Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1), 67-82.
- Wu, Y., Gao, X. Z., & Zenger, K. (2011). *Knowledge-based Artificial Fish-Swarm algorithm*. Paper presented at the 18th IFAC World Congress, Milano.
- Wysocki, C. J., & Beauchamp, G. K. (1984). Ability to smell androstenone is genetically determined. *Proceedings of the National Academy of Sciences*, 81(15), 4899-4902.
- Xian, S., Zhang, J., Xiao, Y., & Pang, J. (2017). A novel fuzzy time series forecasting method based on the improved artificial fish swarm optimization algorithm. *Soft Computing*, 1-11.

- Xiang, L. S., Yu-Rong, Z., & Lin, W. (2017). An effective fruit fly optimization algorithm with hybrid information exchange and its applications. *International Journal of Machine Learning and Cybernetics*, 1-26.
- Xing, B., & Gao, W.-J. (2014). *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms* (1 ed. Vol. 62). Swizerland: Springer international
- Yang, B., Sang, X., Xing, S., Cui, H., Yan, B., Yu, C., Dou, W., & Xiao, L. (2016). *A-star algorithm based path planning for the glasses-free three-dimensional display system*. Paper presented at the SPIE/COS Photonics Asia.
- Yang, X.-S. (2010). Firefly algorithm, stochastic test functions and design optimisation. *International Journal of Bio-Inspired Computation*, 2(2), 78-84.
- Yao, J., Lin, C., Xie, X., Wang, A. J., & Hung, C.-C. (2010). *Path planning for virtual human motion using improved A* star algorithm*. Paper presented at the Information Technology: New Generations (ITNG), 2010 Seventh International Conference on.
- Zhang, Y., Guan, G., & Pu, X. (2016). The Robot Path Planning Based on Improved Artificial Fish Swarm Algorithm. *Mathematical Problems in Engineering*, 2016.
- Zhou, G., & Gen, M. (1999). Genetic algorithm approach on multi-criteria minimum spanning tree problem. *European Journal of Operational Research*, 114(1), 141-152.
- Zhu, G.-Y., & Zhang, W.-B. (2017). Optimal foraging algorithm for global optimization. *Applied Soft Computing*, 51, 294-313.
- Zhu, H., Wang, J., & Li, J. (2013). *A novel potential field method for path planning of mobile robot*. Paper presented at the 2013 25th Chinese Control and Decision Conference, CCDC 2013.
- Ziaei, Z., Oftadeh, R., & Mattila, J. (2014). *Global path planning with obstacle avoidance for omnidirectional mobile robot using overhead camera*. Paper presented at the 2014 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2014.
- Zimmermann, H. J. (1999, Jul 1999). *From fuzzy set theory to computational intelligence-special European experiences*. Paper presented at the Intelligent Processing and Manufacturing of Materials, 1999. IPMM '99. Proceedings of the Second International Conference on.

APPENDIX A
MATLAB CODE FOR SAO

```

% _____
% _____
% Smell Agent Optimization (SAO) source codes Version v1.0           %
%
% Developed in MATLAB R2017a                                         %
%
% Author: Salawudeen Ahmed Tijani                                    %
%
% e-Mail: tasalawudeen@abu.edu.ng                                   %
%
% Credits: Prof. Muhammad Bashir Mua'azu                            %
%         Dr. Yusuf Abubakar Sha'aban                             %
%         Dr. Adedokun Emmanuel Adewale                           %
%
% _____
% _%
function SAO_run
clc;close all;
format long
tic
N=50;%input('Provide the Number of Smell Molecules:= ');%INITIAL POPULATION
clc
T=3;%Temperature of gas molecules.
fitness=input('Press 1 to Select Ackley [D=2]\nPress 2 to Select Adjiman [D=2]\nPress
3 to Select Alpin F1 [D=2]\nPress 4 to Select Beale [D=2]\nPress 5 to Select Bird
[D=2]\nPress 6 to Select Bohachevsky [D=2]\nPress 7 to Select Boot [D=2]\nPress 8 to
Select Box [D=3]\nPress 9 to Select Bukin F6 [D=2]\nPress 10 to Select Colville
[D=4]\nPress 11 to Select CM [D=2]\nPress 12 to Select Cross-in-Tray [D=2]\nPress 13
to Select Dejong F4 [D=2]\nPress 14 to Select Easom [D=2]\nPress 15 to Select Egg
Crate [D=2]\nPress 16 to Select Ellipsoid [D=30]\nPress 17 to Select Goldstein Price
[D=2]\nPress 18 to Select Griewank [D=30]\nPress 19 to Select Holder Table 1
[D=2]\nPress 20 to Select Kowalik [D=4]\nPress 21 to Select LM F1 [D=2]\nPress 22
to Select Michalewicz [D=5]\nPress 23 to Select Matyas [D=2]\nPress 24 to Select
Mishra [D=30]\nPress 25 to Select McComic [D=3]\nPress 26 to Select Neumaier F3
[D=2]\nPress 27 to Select Quadratic [D=30]\nPress 28 to Select Rastrigin [D=5]\nPress
29 to Select Rosenbrock [D=2]\nPress 30 to Select Sphere [D=30]\nPress 31 to Select
Styblinkis Tang [D=2]\nPress 32 to Select Step [D=5]\nPress 33 to Select Sal
[D=30]\nPress 34 to Select Schaffer [D=2]\nPress 35 to Select Shchweifel [D=2]\nPress
36 to Select Subert F1 [D=5]\nPress 37 to Select Watson [D=6]\nPress 38 to Select
Yang F3 [D=2]\nPress 39 to Select Zettl [D=2]\n n = ');
% itr=300;%input('Provide the value of iteration:= ');
D=input('Provide the Search Dimension:= ');%Number of decision variables or problem
dimension.
K=1.38064852*10^(-23);% This is the Boltzmann constant
run=3000; % maximum epoch
SN=2.5;% this is the step movement
%model=CModel();

```

```

SAO_Obj=@(x) MyCost(n,x)
lb=zeros(1,D);% Search lower bound
ub=ones(1,D);% search upper bound
m=2.4;% This is the mass of the molecules.
% Initial population (position) of the smell molecules as follows
tic
for k=1:run

    for i=1:N
        for j=1:D
            molecules(i,j)=lb(j)+rand()*(ub(j)-lb(j));% Define the initial positions of the smell
molecules.
        end
    end
% molecules=rand(N,D);
v=molecules*0.1;
molecules=molecules+v;% Initial population of SAO
for i=1:N
    y(i)=SAO_Obj(fitness,molecules(i,:));% Evaluate the fitness of the initial smell
molecules
end
[ymin,index]=min(y);% Obtain the fitness of the best molecule
x_agent=molecules(index,:);% Determine the agent
olf=ymin/(sum(y)/N);% Determine the Olfaction capacity of the agent.
% iteration=0;
% while iteration<=itr

% Implementing the sniffing mode
for i=1:N
    for j=1:D
        % Update the molecular Velocity
        v(i,j)=(v(i,j)+rand*sqrt(3*K*T/m));
    end
end
% perform sniffing
for i=1:N
    for j=1:D
        molecules(i,j)=molecules(i,j)+v(i,j);
    end
end
for i=1:N
    ys(i)=SAO_Obj(fitness,molecules(i,:));
end
[ysmin,sindex]=min(ys);
xs_agent=molecules(sindex,:);
[ysmax,sidx]=max(y);
x_worst=molecules(sidx,:);% Determine the position of worst smell molecule
if ysmin<ymin
    % xs_agent=molecules(sindex,:);
    ymin=ysmin;

```

```

end
%%
%Evaluate the Trailing mode
for i=1:N
    for j=i:D
        molecules(i,j)=molecules(i,j)+rand*olf*(x_agent(1,j)-abs(molecules(i,j)))...
            -rand*olf*(x_worst(1,j)-abs(molecules(i,j)));
    end
end
%Make sure no smell molecules escape the boundary
for i=1:N
    for j=1:D
        if molecules(i,j)<lb(j)
            molecules(i,j)=lb(j);
        elseif molecules(i,j)>ub(j)
            molecules(i,j)=ub(j);
        end
    end
end
%Evaluate the fitness of the Trailing mode
for i=1:N
    yt(i)=SAO_Obj(fitness,molecules(i,:));
end
[ytmin,tindex]=min(yt);
%%
% Compare the fitness of the trailing mode and the sniffing mode
% and implement the random mode
for i=1:N
    for j=1:D
        if yt(i) < ys(i)
            Best_Molecule(i,j)=yt(1,1);
        elseif yt(i) > ys(i)
            molecules(i,j)=molecules(i,j)+rand()*SN;
            molecules(i,j)=molecules(i,j)+(v(i,j)+rand*sqrt(3*K*T/m));
            molecules(i,j)=molecules(i,j)+rand*olf*(x_agent(1,j)-
                abs(molecules(i,j)))...
                -rand*olf*(x_worst(1,j)-abs(molecules(i,j)));
        end
    end
end
for i=1:N
    ybest(i)=SAO_Obj(fitness,molecules(i,:));
end
[SmellObject,Position]=min(ybest);
% if iteration==1
% disp(sprintf('Iteration Best particle Objective fun'));
% end
% disp(sprintf('%8g %8g %8.4f',iteration,Position,SmellObject));
Object(k)=sort(SmellObject,'descend');
disp(['Iteration ' num2str(k) ': Smell Object = ' num2str(Object(k))]);

```

```
% iteration=iteration+1;
% end
end
disp('The Smell Object is Obtained as: ');
Best_Object=sort(Object,'descend')
disp('The Object Evapourating the Smell is')
Smell_Object=min(Best_Object)
figure(1)
plot(Best_Object,'b','LineWidth',2)
grid on
figure(2)
semilogy(Best_Object,'k','LineWidth',2);
grid on
title('Optimization process','fontsize',12)
xlabel('Iteration Number','fontsize',12);ylabel('Smell','fontsize',12);
toc
```

APPENDIX B
MATLAB CODE FOR BENCHMARK FUNCTIONS

```

function Y=MyCost(n,x)
[m,d]=size(x);
if n==1
    %Evaluate Ackley
    a=0;
    b=0;
    for i=1:d
        a=a+x(:,i).^2;
        b=b+cos(2*pi.*x(:,i));
    end
    Y=-20*exp((-1/5)*(1/d)*sqrt((1/d)*a))-exp((1/d)*b)+20+exp(1);
elseif n==2
    % Evaluate Adjiman
    Z=cos(x(:,1)).*sin(x(:,2))-x(:,1)./(x(:,2).^2+1);
    Y=Z/100;
elseif n==3
    % Evaluate Alpin F1
    W=0;
    for i=1:d
        W=W+abs(x(:,i).*sin(x(:,i))+0.1*x(:,i));
    end
    Y=W;
elseif n==4
    % Evaluate Beale
    Y=(1.5-x(:,1)+(x(:,1).*(x(:,2))))).^2+(2.25-x(:,1)+(x(:,1).*(x(:,2)).^2)).^2+...
        (2.625-x(:,1)+(x(:,1).*(x(:,2)).^3)).^2;
elseif n==5
    % Evaluate Bird
    Y=sin(x(:,2)).*(exp(1-cos(x(:,1))).^2)+cos(x(:,1)).*(exp(1-sin(x(:,2))).^2)...
        +(x(:,1)+x(:,2))).^2;
elseif n==6
    % Evaluate Bohachevsky
    W=0;
    for i=1:d-1
        W=W+x(:,i).^2+2.*x(:,i+1).^2-0.3.*cos(3.*pi.*x(:,i+1))-
            0.4.*cos(4.*pi.*x(:,i+1))+0.7;
    end
    Y=W;
elseif n==7
    % Evaluate Booth
    Y=(x(:,1)+(2.*x(:,2))-7).^2+(2.*x(:,1))+x(:,2)+5).^2;
elseif n==8
    % Evaluate Box
    W=0;
    for i=1:d
        g=exp(-0.1.*(i+1)).*x(:,1)-exp(-0.1.*(i+1)).*x(:,2)-((exp(-0.1.*(i+1))))-exp(-
            (i+1)).*x(:,3));
    end
    Y=W;
end

```

```

    W=W+g.^2;
end
Y=W;
elseif n==9
% Evaluate Bukin F6
Y=100.*sqrt(abs(x(:,1)-0.01.*x(:,2).^2))+0.01.*abs(x(:,2)+10);
elseif n==10
% Evaluate Colville
Y=100.*(x(:,1)-x(:,2)).^2+(x(:,1)-1).^2+(x(:,3)-1).^2+90.*(x(:,3).^2-x(:,4)).^2;
elseif n==11
% Cosine Mixture
a=0;
b=0;
for i=1:d
    a=a+cos(5*pi.*(x(:,i)));
    b=b+x(:,i).^2;
    W=a+b;
end
Y=-0.1.*W;
elseif n==12
% Evaluate Cross-in-tray
Y=-0.0001*(abs(sin(x(:,1)).*sin(x(:,2))).*exp(abs(100-
sqrt(x(:,1).^2+x(:,2).^2)/pi))+1).^0.1);
elseif n==13
% Evaluate Dejong F4
W=0;
for i=1:d
    W=W+i*x(:,i).^4;
end
Y=W;
elseif n==14
% Evaluate Easom
Y=-cos(x(:,1)).*cos(x(:,2)).*exp(-(x(:,1)-pi).^2-(x(:,2)-pi).^2);
elseif n==15
% Evaluate Egg Crate
Y=x(:,1).^2+x(:,2).^2+25*((sin(x(:,1))).^2+(sin(x(:,2))).^2);
elseif n==16
% Evaluate Ellipsoid
W=0;
for i=1:d
    W=W+i^2.*x(:,i).^2;
end
Y=W;
elseif n==17
% Evaluate Goldstein Price
% a=(1+((x(:,1)+x(:,2)+1).^2).*(19-14.*x(:,1)+3.*x(:,1).^2-14.*x(:,2)+...
% 16.*x(:,1).*x(:,2)+3.*x(:,2).^2));
% b=(30+((2.*x(:,1)-3.*x(:,2)).^2).*(18-32.*x(:,1)+12.*x(:,1).^2+48.*x(:,2)-
36.*x(:,1).*x(:,2)+27.*x(:,2).^2));
% Y=a.*b;

```

```

% Y=goldsteinprice(x);
a=(x(:,1)+x(:,2)+1).^2;
b=(19-14.*x(:,1)+3.*(x(:,1).^2)-14.*x(:,2)+6.*x(:,1).*x(:,2)+3.*(x(:,2).^2));
W=1+a.*b;
c=(30+(2.*x(:,1)-3.*x(:,2)).^2);
f=(18-32.*x(:,1)+12.*x(:,1).^2+48.*x(:,2)-36.*x(:,1).*x(:,2)+27.*x(:,2).^2);
Q=c.*f;
Y=W.*Q;

elseif n==18
% Evaluate Griewank
% a=0;
% for i=1:d
% a=a+x(:,i).^2;
% end
% b=0;
% for i=1:d
% b=b.*cos(x(:,i)/sqrt(i))+1;
% end
% Y=(1/4000).*a+b;
Y=griewank(x);
elseif n==19
% Evaluate Holder Table F1
% a=(1-(sqrt(x(:,1).^2-x(:,2).^2))/pi);
% b=sin(x(:,1)).*cos(x(:,2));
% Y=-1.*abs(a.*exp(abs(b)));
Y=holdertable(x);
% Y=-abs(cos(x(:,1)).*cos(x(:,2)).*exp(abs(1-(x(:,1)+x(:,2)).^(0.5/pi)))));
elseif n==20
% Evaluate Kowalik
a=[0.1957; 0.1947; 0.1735; 0.1600; 0.0844; 0.0627;0.0456; 0.0342;...
0.0323; 0.0235; 0.0246]';
b=[4;2;1;1/2;1/4;1/6; 1/8;1/10;1/12;1/14;1/16]';
W=0;
for i=1:d
W=W+(a(:,i)-(x(:,1)).*(b(:,i).^2+b(:,i).*x(:,2))./(b(:,i).^+b(:,i).*x(:,3)+x(:,4))))).^2;
end
Y=W;
elseif n==21
% Evaluate Levi & Montelvo F1
n=d;
for i = 1:n;
z(i) = 1+(x(i)-1)/4;
end
s = sin(pi*z(1))^2;
for i = 1:n-1
s = s+(z(i)-1)^2*(1+10*(sin(pi*z(i)+1))^2);
end
Y = s+(z(n)-1)^2*(1+(sin(2*pi*z(n))))^2);
elseif n==22

```

```

% Evaluate Michalwicz
W=0;
for i=1:d
    W=sin(x(:,1)).*sin(i*x(:,i).^2/pi).^2*d;
end
Y=-W;
elseif n==23
% Evaluate Matyas

Y=0.26*(x(:,1).^2+x(:,2).^2)-0.48*x(:,1).*x(:,2);

elseif n==24
% Evaluate Mishra
a=0;
% aa=0;
for i=1:d-1
    a=a+x(:,1);
end
aa=d-a;
b=0;
for j=1:d-1
    b=b+x(:,j);
end
W=abs((1+d-b).^aa);
Y=W;
elseif n==25
% Evaluate McCormick
Y=mccormick(x);% sin(x(:,1)+x(:,2))+(x(:,1)+x(:,2)).^2-(3/2).*x(:,1)+(5/2).*x(:,2)+1;
elseif n==26
% Evaluate Neumaier F3
a=0;
for i=1:d
    a=a+(x(:,i)-1).^2;
end
b=0;
for i=2:d
    b=x(:,i).*x(:,i-1);
end
Y=a-b;
elseif n==27
% Evaluate Quadratic
Y=-3803.84-138.08*x(:,1)-
232.92*x(:,2)+128.08*x(:,1).^2+203.64*x(:,2).^2+182.25*x(:,1).*x(:,2);
elseif n==28
% Evaluate Rastrigin
% W=0;
% for i=1:d
%     W=W+x(:,i).^2-10*cos(2*pi.*x(:,i));
% end
% Y=10*d+W;

```

```

Y=rastrigin(x);
elseif n==29
% Evaluate Rosenbrock
W=0;
for i=1:d-1
    W=W+100*((x(:,i+1)-x(:,i)).^2).^2+(x(:,i)-1).^2;
end
% W=rosenbrock(x);
Y=W;
elseif n==30
% Evaluate Sphere
W=0;
for i=d
    W=W+x(:,i).^2;
end
Y=W;
elseif n==31
% Evaluate Styblinski's Tang
W=0;
for i=1:d
    W=W+(x(:,i).^4-16.*x(:,i).^2+5.*x(:,i));
end
Y=W.*0.5;
elseif n==32
% Evaluate Step
W=0;
for i=1:d
    W=W+(floor(x(:,i)+0.5)).^2;
end
Y=W;
elseif n==33
% Evaluate Sal
a=0;
b=0;
for i=1:d
    a=x(:,i).^2;
    b=x(:,i).^2;
end
Y=1-cos(2*pi.*sqrt(a))+0.1*sqrt(b);
elseif n==34
% Evaluate Schaffer
w=0;
for i=1:d-1
    w=w+((x(i).^2+x(i+1).^2).^5).*(sin(50.*(x(i).^2+x(i+1).^2).^0.1)).^2;
end
Y=w;
elseif n==35
% Evaluate Schwefel
W=0;
for i=1:d

```

```

    W=W+x(:,i).*sin(sqrt(abs(x(:,i))));
end
Y=W;
elseif n==36
% Evaluate Subert F1
s1=0;
s2=0;
for i = 1:d;
    s1 = s1+i*cos((i+1)*x(1)+i);
    s2 = s2+i*cos((i+1)*x(2)+i);
end
Y = s1*s2;
elseif n==37
% Evaluate Watson
for i=0:29
    a=i/29;
    b=0;
    for j=0:4

        b=b+((j-1).*a.*x(j+1));
    end
    c=0;
    for k=0:5
        c=(c+a.*x(k+1)).^2-1;
    end
    Z=(b+c).^2;
end
Y=Z+x(:,1).^2;
elseif n==38
% % Evaluate Yang F2
% Beta=15;
% m=5;
% a=0;
% b=0;
% for i=1:d
%     a=a+x(:,i);
%     b=b+x(:,i).^2;
% end
% c=0;
% for i=1:d
%     c=c.*(cos(x(:,i)).^2);
% end
% Y=(exp(-a)./(Beta^(2*m))-2.*exp(b).*c);
a=0;
b=0;
for i=1:d
    a=a+abs(x(:,1));
end
for i=1:d
    b=b+sin(x(:,i)).^2;

```

```
end
Y=a.*exp(-b);
elseif n==39
% Evaluate Zirilli
Y=0.25.*x(:,1).^4-0.5.*x(:,1).^2+0.1.*x(:,1)+0.5.*x(:,2).^2;
% Y=0.25.*x(:,1)+(x(:,1).^2-2.*x(:,1)+x(:,2).^2).^2;
end
```

APPENDIX C

Path Planning Cost Function

```

function [z, sol]=MyCost(sol1,model)
    sol=PseSol(sol1,model);
    beta=100;
    z=sol.L*(1+beta*sol.Viol);
end
function model=CModel()
% Source path
x_s=rand;
y_s=rand;
% Target of Destination
a1=3.5;
b1=4.5;
x_t=b1-rand(b1-a1);
b2=6;
a2=5;
yt=b2-rand(b2-a2);
xobst=[1.5 4.0 1.2 2.65 0];
yobst=[4.5 3.0 1.5 1.0 2.5];
robst=[1.5 1.0 0.8 0.6 0.6];
n=3;
x_min=-10;
x_max= 10;
y_min=-10;
y_max= 10;
model.x_s=x_s;
function sol2= PseSol(sol1,model)
x=sol1.x;
y=sol1.y;
x_s=model.x_s;
y_s=model.y_s;
x_t=model.x_t;
y_t=model.y_t;
xobst=model.xobst;
yobst=model.yobst;
robst=model.robst;
XS=[x_s x x_t];
YS=[y_s y y_t];
k=numel(XS);
TS=linspace(0,1,k);
tt=linspace(0,1,100);
x_x=spline(TS,XS,tt);
y_y=spline(TS,YS,tt);
dx=diff(xx);
dy=diff(yy);
L=sum(sqrt(dx.^2+dy.^2));
nobst = numel(xobst); % Number of Obstacles

```

```

Viol = 0;
for k=1:nobst
d=sqrt((x_x-xobst(k)).^2+(y_y-yobst(k)).^2);
v=max(1-d/robst(k),0);
Viol=Viol+mean(v);
end
sol.TS=TS;
sol.XS=XS;
sol.YS=YS;
sol.tt=tt;
sol.x_x=x_x;
sol.y_y=y_y;
sol.dx=dx;
sol.dy=dy;
sol.L=L;
sol.Viol=Viol
sol.IsFeas=(Viol==0);
end
function PlotSol(sol,model)
% This Function plot the solution
x_s=model.x_s;
y_s=model.y_s;
x_t=model.x_t;
y_t=model.y_t;
xobst=model.xobst;
yobst=model.yobst;
robst=model.robst;
XS=sol.XS;
YS=sol.YS;
x_x=sol.x_x;
y_y=sol.y_y;
theta=linspace(0,2*pi,100);
for k=1: numel(xobs)
fill(xobst(k)+robst(k)*cos(theta),yobst(k)+robst(k)*sin(theta),[0.7 0.7 0.4]);
hold on;
end
plot(x_x, y_y,'k','LineWidth',2);

plot(XS,YS,'wo');
plot(x_s,y_s,'rp','MarkerSize',18,'MarkerFaceColor','b','LineWidth',1.5);
plot(x_t,y_t,'bs','MarkerSize',14,'MarkerFaceColor','g','LineWidth',2);

hold off;
grid on;
axis equal;
xlabel('x-coordinate search')
ylabel('y-coordinate search')
end

```

APPENDIX D

MST Cost Function

```

function model=CModel()
%X=[14 42 50 7 36 87 96 50 11 21 36 14 81 39 11 59 78 89 11 55];
%Y=[63 52 93 83 31 97 6 4 10 43 93 82 71 24 74 9 81 46 36 45];
% X=[14 42 50 7 36 87 96 50 11 21 36 14 81 39 11 59 78 89 11 55 65 75 30 75 20 50
85 95 45 85];
%Y=[63 52 93 83 31 97 6 4 10 43 93 82 71 24 74 9 81 46 36 45 20 55 65 75 22 80 55
57 45 40];
X=[14 42 50 7 36 87 96 50 11 21 36 14 81 39 11];
Y=[63 52 93 83 31 97 6 4 10 43 93 82 71 24 74];
n=numel(X);
d=zeros(n,n)
for i=1:n
    for j=i+1:n
        d(i,j)=sqrt((X(i)-X(j))^2+(Y(i)-Y(j))^2);
        d(j,i)=d(i,j);
    end
end
model.n=n;
model.X=X;
model.Y=Y;
model.d=d;
end
function A=CMatFrmVec(x)
k=numel(x);
n=(sqrt(8*k+1)+1)/2;
A=zeros(n,n);
c=0;
for i=1:n
    for j=i+1:n
        c=c+1;
        A(i,j)=x(c);
        A(j,i)=x(c);
    end
end
end
function q= CalDconcty(A)
n=size(A,1);
L=logical((eye(n)+A)^n);
q=1-mean(L(:));
end
function [z, sol]=MyCost(xhat,model)
x=double(xhat>=0.5);
d=model.d;
A= CMatFrmVec (x);
q= CalDconcty (A);
AD=A.*d;
alpha=5*sum(d(:));
SumAD=sum(AD(:));

```

```
z=SumAD+alpha*q;  
sol.A=A;  
sol.q=q;  
sol.SumAD=SumAD;  
sol.z=z;  
sol.IsFeasible=(q==0);
```

```
end
```

APPENDIX E MATLAB CODE OF SAO SOFTWARE

```
function varargout = SAO_GUI(varargin)
clc
%SAO_GUI M-file for SAO_GUI.fig
%   SAO_GUI, by itself, creates a new SAO_GUI or raises the existing
%   singleton*.
%
%   H = SAO_GUI returns the handle to a new SAO_GUI or the handle to
%   the existing singleton*.
%
%   SAO_GUI('Property','Value',...) creates a new SAO_GUI using the
%   given property value pairs. Unrecognized properties are passed via
%   varargin to SAO_GUI_OpeningFcn. This calling syntax produces a
%   warning when there is an existing singleton*.
%
%   SAO_GUI('CALLBACK') and SAO_GUI('CALLBACK',hObject,...) call the
%   local function named CALLBACK in SAO_GUI.M with the given input
%   arguments.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one
%   instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help SAO_GUI

% Last Modified by GUIDE v2.5 13-Dec-2017 22:58:24

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',    mfilename, ...
                  'gui_Singleton', gui_Singleton, ...
                  'gui_OpeningFcn', @SAO_GUI_OpeningFcn, ...
                  'gui_OutputFcn', @SAO_GUI_OutputFcn, ...
                  'gui_LayoutFcn', [], ...
                  'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```

% --- Executes just before SAO_GUI is made visible.
function SAO_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   unrecognized PropertyName/PropertyValue pairs from the
%            command line (see VARARGIN)

% Choose default command line output for SAO_GUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes SAO_GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = SAO_GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

function cost_Callback(hObject, eventdata, handles)
% hObject    handle to cost (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cost as text
%        str2double(get(hObject,'String')) returns contents of cost as a double
input=str2double(get(hObject,'String'));
if isempty(input)
    set(hObject,'String',0)
end

% --- Executes during object creation, after setting all properties.
function cost_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cost (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.

```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function N_Callback(hObject, eventdata, handles)
% hObject    handle to N (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of N as text
%        str2double(get(hObject,'String')) returns contents of N as a double
% --- Executes during object creation, after setting all properties.
function N_CreateFcn(hObject, eventdata, handles)
% hObject    handle to N (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function D_Callback(hObject, eventdata, handles)
% hObject    handle to D (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of D as text
%        str2double(get(hObject,'String')) returns contents of D as a double
% --- Executes during object creation, after setting all properties.
function D_CreateFcn(hObject, eventdata, handles)
% hObject    handle to D (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function Temp_Callback(hObject, eventdata, handles)
% hObject    handle to Temp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Temp as text

```

```

%    str2double(get(hObject,'String')) returns contents of Temp as a double
% --- Executes during object creation, after setting all properties.
function Temp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Temp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function m_Callback(hObject, eventdata, handles)
% hObject    handle to m (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of m as text
%    str2double(get(hObject,'String')) returns contents of m as a double
% --- Executes during object creation, after setting all properties.
function m_CreateFcn(hObject, eventdata, handles)
% hObject    handle to m (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%    See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```

```

function K_Callback(hObject, eventdata, handles)
% hObject    handle to K (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of K as text
%    str2double(get(hObject,'String')) returns contents of K as a double
% --- Executes during object creation, after setting all properties.
function K_CreateFcn(hObject, eventdata, handles)
% hObject    handle to K (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

```

```

% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function run_Callback(hObject, eventdata, handles)
% hObject handle to run (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of run as text
% str2double(get(hObject,'String')) returns contents of run as a double
% --- Executes during object creation, after setting all properties.
function run_CreateFcn(hObject, eventdata, handles)
% hObject handle to run (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function SM_Callback(hObject, eventdata, handles)
% hObject handle to SM (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of SM as text
% str2double(get(hObject,'String')) returns contents of SM as a double
% --- Executes during object creation, after setting all properties.
function SM_CreateFcn(hObject, eventdata, handles)
% hObject handle to SM (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function olf_Callback(hObject, eventdata, handles)
% hObject handle to olf (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of olf as text
% str2double(get(hObject,'String')) returns contents of olf as a double

```

```

% --- Executes during object creation, after setting all properties.
function olf_CreateFcn(hObject, eventdata, handles)
% hObject   handle to olf (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%   See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
% --- Executes on selection change in Answer

% --- Executes on button press in pushbutton1.

function Result_Callback(hObject, eventdata, handles)
% hObject   handle to Result (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Result as text
%   str2double(get(hObject,'String')) returns contents of Result as a double
% --- Executes during object creation, after setting all properties.
function Result_CreateFcn(hObject, eventdata, handles)
% hObject   handle to Result (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%   See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function pushbutton1_Callback(hObject, eventdata, handles)
% hObject   handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)

content1 = get(handles.cost,'Value');

switch content1
    case 1
        fitness = 1;
    case 2
        fitness = 2;
    case 3
        fitness = 3;

```

case 4
 fitness = 4;
case 5
 fitness = 5;
case 6
 fitness = 6;
case 7
 fitness = 7;
case 8
 fitness = 8;
case 9
 fitness = 9;
case 10
 fitness = 10;
case 11
 fitness = 11;
case 12
 fitness = 12;
case 13
 fitness = 13;
case 14
 fitness = 14;
case 15
 fitness = 15;
case 16
 fitness = 16;
case 17
 fitness = 17;
case 18
 fitness = 18;
case 19
 fitness = 19;
case 20
 fitness = 20;
case 21
 fitness = 21;
case 22
 fitness = 22;
case 23
 fitness = 23;
case 24
 fitness = 24;
case 25
 fitness = 25;
case 26
 fitness = 26;
case 27
 fitness = 27;
case 28
 fitness = 28;

```

case 29
    fitness = 29;
case 30
    fitness = 30;
case 31
    fitness = 31;
case 32
    fitness = 32;
case 33
    fitness = 33;
case 34
    fitness = 34;
case 35
    fitness = 35;
case 36
    fitness = 36;
case 37
    fitness = 37;
case 38
    fitness = 38;
case 39
    fitness = 39;
otherwise
end
end

N=str2num(get(handles.N, 'String'));%input('Provide the Number of Smell Molecules:=
');%INITIAL POPULATION
clc
T=str2num(get(handles.Temp, 'String'));% Temperature of gas molecules.
D=str2num(get(handles.D, 'String'));% Number of decision variables or problem
dimension.
K=1.38064852*10^(-23);% This is the boltzman constant
run=str2num(get(handles.run, 'String'));
itr=0.3*run;%input('Provide the value of iteration:= ');
SN=str2num(get(handles.SM, 'String'));
lb=zeros(1,D);
ub=ones(1,D);
m=str2num(get(handles.m, 'String'));%This is the mass of the molecules.
olf=str2num(get(handles.olf, 'String'));% ymin/(sum(y)/N);% Determine the Olfaction
capacity of the agent.
% Initial population (position) of the gas molecules as follows
tic
Y=SAO_Obj(fitness,molecules)
t = ['Iteration ' num2str(k) ': Smell Object = ' num2str(Object(k))];
set(handles.Result, 'String', t);
pause(0.1);
% iteration=iteration+1;
% end
end
Best_Object=sort(Object,'descend');

```

```

Smell_Object=min(Best_Object);
t1 = ['The Object Evapourating the Smell is: ' num2str(Smell_Object)];
set(handles.Result, 'String', t1);
set(handles.listbox2, 'String', num2str(Best_Object.));
plot(Best_Object,'b','LineWidth',2)
xlabel('Iteration Number','fontsize',12);ylabel('Smell','fontsize',12);
grid on
figure(1)
plot(Best_Object,'b','LineWidth',2)
grid on
figure(2)
semilogy(Best_Object,'k','LineWidth',2);
grid on
title('Optimization process','fontsize',12)
xlabel('Iteration Number','fontsize',12);ylabel('Smell','fontsize',12);
t2 = ['The Elapsed Time is: ' num2str(toc) ' secs'];
set(handles.edit15, 'String', t2);

% --- Executes on selection change in listbox2.
function listbox2_Callback(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns listbox2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox2

% --- Executes during object creation, after setting all properties.
function listbox2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit15_Callback(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit15 as text
%        str2double(get(hObject,'String')) returns contents of edit15 as a double

```

```
% --- Executes during object creation, after setting all properties.
function edit15_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```