Document Information

Analyzed document	C13358011_Ciaran_OLoughlin_MSc_thesisForEvaluation.final.pdf (D108993046)
Submitted	6/15/2021 9:31:00 PM
Submitted by	
Submitter email	C13358011@mytudublin.ie
Similarity	99%
Analysis address	emma.x.murphy.dit@analysis.urkund.com

Sources included in the report

SA	**TU Dublin City / C13358011_MSC_ASD.final.pdf** Document C13358011_MSC_ASD.final.pdf (D108754758) Submitted by: C13358011@mytudublin.ie Receiver: 455507.dit@analysis.urkund.com	46
SA	**TU Dublin City / MSc__ASD_Dissertation(2).pdf** Document MSc__ASD_Dissertation(2).pdf (D106726158) Submitted by: C13358011@mytudublin.ie Receiver: 455507.dit@analysis.urkund.com	4

Entire Document

96%	MATCHING BLOCK 1/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

Performance comparison between a distributed particle swarm algorithm and a centralised algorithm Ciar an O'Loughlin A dissertation submitted in partial ful lment of the requirements of Technological University Dublin for the degree of M.Sc. in Computing (TU060) Date Declaration I certify that this dissertation which I now submit for examination for the award of MSc in Computing (TU060), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work. This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and has not been submitted in whole or part for an award in any other Institute or University. The work reported on in this dissertation conforms to the principles and requirements of the Institute's guidelines for ethics in research. Signed: Ciar an O'Loughlin Date: 13 th June 2021 I Abstract Particle Swarm optimisation (PSO) is a particular form of swarm intelligence, which itself is an innovative intelligent paradigm for solving optimization problems. PSO is generally used to nd a global optimum in a single optimisation function. This typically occurs on one node(machine) but there has been a signi cant body of research into creating distributed implementations of the PSO algorithm. Such research has often focused on the creation and performance of the distributed implementation in an isolated manner or compared to di erent distributed algorithms. This research piece aims to bridge a gap in the existing literature, by testing a distributed implementation of a PSO algorithm against a centralised implementation, and investigating what, if any, gains there are to utilising a distributed implementation over a centralised implementation. The focus will primarily be on the time taken for the algorithm to successfully nd a global minimum to a speci c tness function, but other elements will be examined over the course of the study. Keywords: Swarm Intelligence, Particle Swarm Optimisation, PSO, Distributed Algorithms II Acknowledgments I would like to express my sincere thanks to my supervisor Dr. Edina Hatunic Webster, whose guidance and support helped me throughout my dissertation. A special thanks to my parents Eileen & Cormac O'L., my friends, colleagues and family. Without all their encouragement and support this dissertation would never have been possible. III Contents Declaration I Abstract II Acknowledgments III Contents IV List of Figures VIII List of Tables X List of Acronyms XII 1 Introduction 1 1.1 Background . . . . 1 1.2 Research Project/problem . . . . 2 1.3 Research Objectives . . . . 2 1.4 Research Methodologies . . . . 3 1.4.1 Based on Type: Primary vs Secondary . . . . 3 1.4.2 Based on Objective: Quantitative vs Qualitative . . . . 4 1.4.3 Based on Form: Exploratory vs Constructive vs Empirical . . . 4 1.4.4 Based on Reasoning: Deductive vs Inductive . . . . 5 1.5 Scope and Limitations . . . . 5 1.6 Document Outline . . . . 6 IV 2 Background Research 8 2.1 Introduction . . . . 8 2.2 PSO(Particle Swarm Optimisation) . . . . 8 2.2.1 What is PSO? . . . . 8 2.3 PSO Implementations . . . . 10 2.3.1 Stopping Criteria . . . . 11 2.3.2 Parameter Selection . . . . 12 2.3.3 Topology adjustments . . . . 12 2.3.4 Multi-swarm PSO . . . . 13 2.3.5 Distributed PSO . . . . 15 2.4 Evaluating Performance . . . . 18 2.4.1 Benchmarking . . . . 18 2.5 Research Gaps . . . . 18 2.6 Conclusion . . . . 19 3 Experiment design and methodology 20 3.1 Introduction . . . . 20 3.2 Aim of Research . . . . 20 3.3 Hypothesis . . . . 21 3.4 Design overview . . . . 21 3.4.1 Centralised PSO Design . . . . 23 3.4.2 Distributed PSO Design . . . . 25 3.5 Distributed environment . . . . 26 3.5.1 DigitalOcean . . . . 26 3.5.2 Container and Network Design . . . . 27 3.6 Data Design and Data Capture Overview . . . . 28 3.6.1 Data Aggregation . . . . 30 3.7 Design and Methodology Summary . . . . 30 4 Experiment Implementation 31 4.1 Introduction . . . . 31 V 4.2 PSO Implementations . . . . 31 4.2.1 Java . . . . 31 4.2.2 Spring Boot . . . . 32 4.2.3 Centralised Implementation . . . . 32 4.2.4 Distributed Implementation . . . . 37 4.2.5 Fitness Functions overview . . . . 41 4.3 Test Harness . . . . 42 4.3.1 Postman/Newman and NodeJs . . . . 42 4.3.2 Implementation . . . . 43 4.4 Data Aggregator Implementation . . . . 45 4.5 Implementation conclusion . . . . 47 5 Results, Evaluation and Discussion 48 5.1 Introduction . . . . 48 5.2 Results . . . . 48 5.2.1 Centralised Results . . . . 48 5.2.2 Distributed Results . . . . 52 5.2.3 Performance Comparison . . . . 56 5.3 Discussion . . . . 59 5.3.1 Results Evaluation . . . . 59 5.3.2 Strengths and Limitations . . . . 60 5.4 Conclusion . . . . 61 6 Conclusion 62 6.1 Research Overview . . . . 62 6.2 Problem De nition 63 6.3

PSO Particle Swarm Algorithm LTS Long Term Support CPU Computer processor unit RAM Random Access Memory TTS Time To Solution CSV Comma Separated Values JSON JavaScript Object Notation OTP Optimisation Tipping Point XII Chapter 1 Introduction 1.1 Background Swarm Intelligence (SI) is an innovative distributed intelligent paradigm for solving optimization problems that originally took its inspiration from the biological exam- ples by swarming, ocking and herding phenomena in vertebrates (Abraham, Guo, & Liu, 2006). Within the boundaries of swarm intelligence there are many di er- ent algorithms, all with di erent uses and capabilities. One particular algorithm is the particle swarm optimisation algorithm (PSO). PSO is a population-based search algorithm and is initialized with a population of random solutions, called particles (Shi, 2004). Particles are then arranged into a "swarm". Swarms allow the sharing of information between particles, the so-called "social" element of the algorithm. PSO as an algorithm was rst proposed by James Kennedy and Russell Eberhart in their 1995 paper "Particle Swarm Optimization" (Kennedy & Eberhart, 1995a). While not a new technique, PSO is still being expanded upon today, with modern uses ranging from simple algorithmic evaluation, to more complex robotics implementations. This is the basis of a single swarm PSO. It is possible to increase the number of particles in a swarm, but we can also increase the number of swarms. A multi-swarm PSO (Or a Cooperative Particle Swarm Optimisation algorithm, CPSO) model allows for "a signi cant increases in the solution diversity in CPSO-S algorithm, because of the many di erent members from di erent swarms" (Vandenbergh & Engelbrecht, 1 CHAPTER 1. INTRODUCTION 2004). Even when using a multi-swarm model, there will be hard limits on the number of particles and swarms a singular machine can generate. This is where a distributed model can help, with swarms being logically segregated across a network of machines. 1.2 Research Project/problem There is an upper limit to the amount of particles any one system can support in a PSO algorithm. At a certain point the algorithm will slow down and the time taken for it to complete an iteration will drastically increase. This will be exacerbated by running multiple swarms. To alleviate this problem, we can distribute the swarms onto di erent machines, which allows us to increase the total number of swarms and parti- cles available to us. However, this has a disadvantage as to evaluate results generated by the swarms, network communications will need to be established. Network connec- tions are inherently slower than connections and data transfers on a local machine, so the question this report aims to answers is: "At what point are performance gains in running a particle swarm optimisation algorithm in a distributed environment outweighed by the time lost in network com- munications between multiple swarms?" 1.3 Research Objectives The key objective of the research is to identify whether there exists a point whereby it is more e cient to run a particle swarm optimisation algorithm in a distributed manner over a centralised manner. To answer this question the following research objectives where identi ed: 1. Create a distributed and a centralised implementation of the PSO 2. Generate a result set for the centralised implementation, with varying inputs (example; di erent evaluation function, number of swarms and or particles etc.). Average out results with the same inputs 2 CHAPTER 1. INTRODUCTION 3. Generate a result set for the distributed implementation, with varying inputs (examples; di erent evaluation function, number of swarms and or particles etc.). Average out results with the same inputs 4. Cross comparison between the two results sets. Identify points at which dis- tributed had a faster response time, or other values that would make it preferable to a centralised implementation 5. Identify any limiting factors, signi cant points of interest in the data and identify future research 1.4 Research Methodologies The research can be classi ed based in a few di erent ways - 1.4.1 Based on Type: Primary vs Secondary Primary research refers to a collection of original data speci c to a particular re- search question generated during the project. When doing primary research, the researcher gathers information rst-hand rather than relying on available information in databases and other publications (Bouchrika, 2020). Secondary research instead focuses on collecting and summarizing existing data collections and results. This involves researching existing literature, published articles and analyzing the data produced from these articles to come to a new conclusion, or test a hypothesis. When doing secondary research, researchers use and analyze data from primary research sources (Bouchrika, 2020). The research type for this project can be de ned as primary research. Data sets needed to answer the research question will be generated during the course of the project. The data set will be unique, as no other study has sought to compare imple- mentation types of particle swarm optimisation algorithms. 3 CHAPTER 1. INTRODUCTION 1.4.2 Based on Objective: Quantitative vs Qualitative Quantitative research methods refer to collecting numerical data, data that can be used to measure variables, predict outcomes etc. Quantitative data is structured and statistical, using a grounded theory method that relies on data collection that is systematically analyzed. Quantitative research is a methodology that provides support when you need to draw general conclusions from your research and predict outcomes (McLeod, 2019). Qualitative research is fundamentally opposite to Quantitative research, as it relies on non-statistical and unstructured or semi-structured data. It is a methodology designed to collect non-numerical data to gain insights. It relies on data collected based on a research design that answers the question \why." 1 The research objective of this project can be de ned as quantitative research. This study will generate and examine structured data sets, comparing two particle swarm optimisation implementations, and drawing conclusions from those comparisons. 1.4.3 Based on Form: Exploratory vs Constructive vs Empir- ical Exploratory research refers to when researching a problem which has not been clearly de ned. Through exploratory research we can determine the best research design and data collection method. When conducting constructive research, a completely new approach, theory or model will be proposed. Constructive research adds a new contribution to the current body of research. Empirical research is a way of obtaining knowledge through direct observation or experience. Empirical research involves a process of de ning a hypothesis and then making predictions that can be tested using a suitable scienti c experiment. The study can be de ned as an empirical study. This study will de ne its hypoth- esis, test that hypothesis, examine the results from tests, and draw conclusions/pre- dictions from the data. Based on that data the hypothesis can then be accepted or 1 https://www.surveymonkey.com/mp/quantitative-vs-qualitative-research/ 4 CHAPTER 1. INTRODUCTION rejected, thus concluding the study. 1.4.4 Based on Reasoning: Deductive vs Inductive Deductive reasoning is a logical form of reasoning. Deductive begins with a general statement,

a hypothesis, and uses speci c logical steps to accept or reject the proposed hypothesis.

100%	MATCHING BLOCK 2/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

Inductive reasoning is the opposite of deductive reasoning. Inductive reasoning

creates a general statement based on

100%	MATCHING BLOCK 3/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

speci c observations. Basically, there is data, then conclusions are drawn from the data 2 For this study deductive reasoning will be used. The study will state a hypothesis, and attempt to validate that hypothesis though testing, generating data sets and drawing conclusions from those data sets. 1.5 Scope and Limitations The scope of this research is to identify potential points where a distributed imple- mentation of the particle swarm optimisation algorithm is faster than a centralised implementation. There are two main limiting factors to this research. The rst being that the implementations are only tested against speci c tness functions. There are many di erent tness functions, or optimisation problems, that can be applied to a particle swarm optimisation algorithm, and many of them will have di erent e ciency points. Some may gain a bene t from being distributed at di erent levels of swarms and particles, and others may not ever bene t from being distributed. For this reason, three di erent algorithms were chosen and tested against the two implementations. The second limiting factor is an environmental one. Computers with better qual- ity CPU (Computer Processor Unit) and a higher amount of RAM (Random Access Memory) will be able to support more swarms and particles. In order to account for 2

https://www.livescience.com/21569-deduction-vs-induction.html 5

CHAPTER 1. INTRODUCTION

82%	MATCHING BLOCK 4/50	SA	MSc__ASD_Dissertation(2).pdf (D106726158)

this the same machine type will be used across the distributed and centralised im- plementations. How this will be achieved is discussed in Chapter Three, Design and Implementation Overview. 1.6 Document Outline Chapter One Chapter One is a short introduction and few reasons why this research was conducted. Chapter

Two

90%	MATCHING BLOCK 5/50	SA	MSc__ASD_Dissertation(2).pdf (D106726158)

Chapter Two involves a comprehensive literature review. Some gaps in the existing research are discussed. Chapter

Three

92%	MATCHING BLOCK 6/50	SA	MSc__ASD_Dissertation(2).pdf (D106726158)

Chapter Three includes a detailed design of the experiment as well as the methodology behind it. The design of the testing systems and some implementation overviews are presented. Chapter Four Chapter Four contains a full discussion of the implementation details. Coded examples are provided to add further clarity to how the experiment was built. Environment and test implementations are also discussed. Chapter

Five

78%	MATCHING BLOCK 7/50	SA	MSc__ASD_Dissertation(2).pdf (D106726158)

Chapter Five presents the results obtained from running the experiment. Some obser- vation and discussion is presented around these results. 6 CHAPTER 1.

INTRODUCTION

98%	MATCHING BLOCK 8/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

Chapter Six Chapter Six is a nal discussion and conclusion of the research project, including future suggestions on the given problem as well as recommendations regarding accuracy of the experiment and its design 7 Chapter 2 Background Research 2.1 Introduction This Chapter provides a review of the literature available on particle swarm optimisa- tion algorithms, distributed implementation, various approaches adopted to solve the problem and evaluation metrics used for evaluating the results. The Chapter concludes with the gaps in the existing research and forms the objective for the research. 2.2 PSO(Particle Swarm Optimisation) 2.2.1 What is PSO? The original PSO algorithm was introduced in 1995 (Kennedy & Eberhart, 1995b), in the paper they discussed what the PSO algorithm is, and how to use it to optimise nonlinear functions. At its heart PSO is a population-based optimization technique where the population is referred to as a swarm and population members are referred to as particles. At the algorithm's inception a number of unique particles are created and given random positions within a D-dimensional space. Each particle can be considered a potential solution to the tness function. Particles then go through iterations, or epochs, where each particle calculates its own personal best evaluation value, then the global best value is calculated. Particles are encouraged to move closer to their 8 CHAPTER 2. BACKGROUND RESEARCH own personal best or the groups based on the algorithm's implementation. A particle moving closer towards the groups best will de ne a higher "social value", and moving closer to its own personal best will de ne a higher "cognitive value". Velocity can also be changed to reduce the variance in particle positions between epochs. Figure 2.1: Flow Diagram of Particle Swarm Optimisation Algorithm (D. Wang et al., 2017) We can see from the above explanation that PSO shares many elements of other arti cial intelligence types, such as evolutionary computation, and genetic algorithms. The particles are manipulated according to the following equations: V id = W V id + C l Rand() (P id * X id) + C 2 Rand() (P gd * X id) (2.1) X id = X id + V id (2.2) Where Cl and C2 are two positive constants, Rand() is a random function in the range [0,1] and w is the inertia weight (Shi & Eberhart, 1998). A particle's new velocity can be calculated using equation 2.1, using its previous velocity and the distances of its cur- rent position from its own best experience (position) and the group's best experience. Equation 2.2 calculates the particles new position after its updated velocity. 9 CHAPTER 2. BACKGROUND RESEARCH Fitness Function When talking about PSO we must also talk about a tness function. A tness function is a function that maps the values in particles to a real value that must reward those particles that are closer to the optimisation criterion. Essentially this is the function to nd the minimum or maximum value of. For instance, if we wished to nd the global minimum of the Beale Function, an optimisation test function, we could use PSO. Beale's function can be de ned as: f(x;y) = (1:5 * x + xy) 2 + (2:25 * x + xy 2) 2 + (2:625 * x + xy 3) 2 (2.3) Using this tness function the global minimum could be found, in this case it would equate out to be f(x) = 0 at x = (3; 0:5) (Bingham, n.d.). It

is

99%	MATCHING BLOCK 9/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

important to note that as PSO is a heuristic algorithm, its solution results are not necessarily optimal. PSO will nd an answer to a tness function, but it may not be the best answer to the function. In addition, some problems may have multiple global minimum values, in which case the PSO can only nd one answer at a time. 2.3 PSO Implementations Since James Kennedy and Russell Eberhart original paper there has been signi cant research into the original algorithm (Piotrowski, Napiorkowski, & Piotrowska, 2020; Bai, 2010; Imranª, Hashima, & Abd Khalidb, 2013; D. Wang et al., 2017) its applica- tions (Hereford, 2006; Beni, 2005; Blum & Li, n.d.; Raquel & Naval Jr, 2005) and its performance (Yin, Yu, Wang, & Wang, 2006; Kennedy, 1999). There is also a large body of research into the use of swarm algorithms in distributed environments (Akat & Gazi, 2008; Salza & Ferrucci, 2019; Peleg, 2005) and centralised environments (Trelea, 2003; Xie, Zhang, & Yang, 2003; Poli, Kennedy, & Blackwell, 2007), which will be discussed more in the following section. PSO has also been improved upon, with some newer implementations updating the topology or some other underlying principle of the algorithm, examples including SPSO, APSO, TRIBES, Cyber Swarm etc (Zhou 10 CHAPTER 2. BACKGROUND RESEARCH & Tan, 2009; Oca, Stutzle, Birattari, & Dorigo, 2009; Cooren, Clerc, & Siarry, 2009; Yin, Glover, Laguna, & Zhu, 2010). In terms of implementing PSO there have been multiple studies into the uses of swarm intelligence in robotics systems (S a, Nedjah, & Mourelle, 2016; Meng & Gan, 2008; Hereford, 2006). Some other studies also look at interesting use cases of PSO and what swarm intelligence can be applied to, such as scheduling systems and analysis of distributed systems (Li, Yang, Su, Lu, & Yu, 2019; Moradi & Fotuhi-Firuzabad, 2008; Nouiri, Bekrar, Jemai, Niar, & Ammari, 2015; Sahin, Yavuz, Arnavut, & Uluyol, 2007). 2.3.1 Stopping Criteria When utilising the PSO algorithm, it is important to identify how and when the algorithm should terminate. The simplest form of stopping criteria is implementing a max number of iterations/epochs check. This is a simple stopping criteria however, it carries with it a risk of stopping the algorithm before a global optimum value has been found. It also must be prede ned prior to the

algorithm's

98%	MATCHING BLOCK 10/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

inception, which leaves little leeway for change during runtime. An alternate to this approach is a check for "when the chance of achieving sig- ni cant improvements in further iterations is extremely low" (Bassimir, Schmitt, & Wanka, 2020). E ectively this is checking to see when the particles are no longer actively moving in the search space when they have settled at a particular answer. Within the literature there has been some research into more adaptive stopping cri- teria, for example Zielinski and Laur 2007 paper (Zielinski & Laur, 2007). In this paper a list of upper-limit-based and adaptive termination criteria is presented and tested against a real world problem, "optimizing a power allocation scheme for a Code Division Multiple Access (CDMA) system"(Zielinski & Laur, 2007). Both of these approaches o er a more dynamic and adaptive method of stopping the algorithm than the more straightforward maximum iteration check criteria. 11 CHAPTER 2. BACKGROUND RESEARCH 2.3.2 Parameter Selection As mentioned in earlier sections there are multiple input parameters for any PSO algorithm, the basic ones being inertia, cognitive and social weightings. Parameters are usually tuned for individual optimisation functions, but there are some basic selecting parameters that are normally used. When looking at the inertia weighting, according to Yan He et al. inertia should be kept within the following range w min = 0:4;w max = 0:9 (He, Ma, & Zhang, 2016). When it comes to both cognitive and social weightings, there is a little less consen- sus with some research saying both values should be equal to 2 (Kennedy & Eberhart, 1995a), and other literature mentioning that the values should range between 1 and 3 (Zhang, Ma, jun Wei, & feng Liang, 2014). It is also suggested that the sum of these constants should not exceed 3 i:es + c 3 (Kan & Jihong, 2012). What is conclusive across all the literature is that each optimisation problem will require its own ne tuning with all three values. 2.3.3 Topology adjustments Since its original inception, several adjustments have been proposed in literature to alter some implementation details around PSO. One reoccurring change is a change to the population topology or sociometry. It is well established that these factors play an important role in improving the performance of population-based optimization algo- rithms by enhancing population diversity when solving multiobjective and multimodal problems (Lynn, Ali, & Suganthan, 2018). Global version PSO (GPSO), which pri- oritises the GBest value (Global best value from all particles), and local version PSO (LPSO), which prioritises the LBest value (Local best value, which favours the best value from smaller subsections of the swarm) are two common neighbour topologies. G-PSO In the GBest population, the trajectory of each particle's search is in uenced by the best point found by any member of the entire population (Kennedy & Mendes, 2002). 12 CHAPTER 2. BACKGROUND RESEARCH What this means is that the particle uses its experience (the best position achieved so far) and uses the knowledge of the best particle in the swarm to in uence its movement strategy. It prioritises the global best value when updating its position. However, the drawback of this approach is that if the best particles are far from global optimum, the swarm can be trapped in a local optima since the swarm cannot explore other areas in the search space (Azab, Hady, & Hefny, 2016). L-PSO The LBest population allows each individual to be in uenced by some smaller number of adjacent members of the population array. Typically LBest neighbourhoods com- prise exactly two neighbours, one on each side: a ring lattice (Kennedy & Mendes, 2002). The same update function is used by particles in the two topologies, however LBest is generally considered to be the slower topology. Its advantage is that it

is

98%	MATCHING BLOCK 11/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

much less likely to get stuck in the local optima, and therefore nds the target value far more frequently. Figure 2.2: GBest(Left) and LBest(Right) sociometric patterns.(Kennedy & Mendes, 2002) 2.3.4 Multi-swarm PSO As mentioned previously there has been a wide body of research conducted into vari- ants of the original PSO algorithm. One such area has been in using multiple swarms with researchers nding some success with their implementations. One such imple- 13 CHAPTER 2. BACKGROUND RESEARCH mentation is the Multi-Swarm Particle Swarm Optimization (MSPSO). MSPO "is based on multiple swarms framework cooperating with the dynamic sub-swarm num- ber strategy (DNS), sub-swarm regrouping strategy (SRS), and purposeful detecting strategy (PDS)" (Xia, Gui, & Zhan, 2018). Using these strategies allows MSPSO to balance its exploration and the exploitation ability, resulting in good performance, in terms of solutions accuracy. It however does su er from an increased time complex- ity when compared to several other PSO implementations. Improved Particle Swarm Optimization Algorithm (

IPSO)

99%	MATCHING BLOCK 12/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

is another mutli-swarm technique that "adopts a new mutation operator and a new method that congregates some neighbouring individuals to form multiple sub-populations in order to lead particles to explore new search space" (Zheng et al., 2007). Subdividing out the population into multiple neighborhoods, or swarms allows the algorithm to divide up the problem space, and thereby improve performance. Utilising the mutation factor also allowed the implementation to "en- hance the e ciency of advantageous direction of ying particles, so particles can y to feasible region more quickly and more e ciently". Figure 2.3: IPSO multi-populations being formed .(Zheng et al., 2007) 14 CHAPTER 2. BACKGROUND RESEARCH 2.3.5 Distributed PSO dPSO As mentioned in Chapter One, there will be a limit to the number of particles and swarms any one computer can support, the limiting factor being the amount of memory or computational power. To avoid this bottleneck there has been a signi cant body of research into creating a distributed implementation of the PSO algorithm. One such area of research is in robotics, using PSO as a search algorithm distributed across many particles, in reality each robot acts as a particle in a PSO algorithm. In James M. Hereford paper, "A Distributed Particle Swarm Optimization Algorithm for Swarm Robotic Applications" a distributed PSO algorithm is proposed, which he calls the dPSO (Hereford, 2006). Listing 2.1 shows the pseudo code used to implement the algorithm. Listing 2.1: Herefords dPSO Code Flow 1 pbest = -1; gbest = -1; % Initialize pbest and gbest 2 While (target not found or time not expired) 3 % Make measurement and update, if necessary 4 meas = make_measurement(); 5 if (meas < pbest) % Update pbest, if true 6 Up_pbest(); %Update pbest value and location 7 if (meas < gbest) % Update gbest, if true 8 Set gflag; % set a flag 9 Up_gbest(); %Update gbest value and location 10 end 11 end 12 Move(); % Move bot based on PSO update equation 13 % For simulation, constrain bot movement 14 % Two conditions: 15 % (1) Magnitude of velocity must be > Vmax 16 % (2) Direction must be within in max turn angle 15 CHAPTER 2. BACKGROUND RESEARCH 17 If (new gbest found) %broadcast new gbest value 18 Broadcast(gbest); 19 end 20 If (gbest is global gbest) %broadcast gbest location 21 Broadcast(gbest_location); 22 end 23 End while In the paper Hereford designed the algorithm for robotic operators using search algorithms to nd speci c targets within a search space. From the pseudo code we can see that once a new GBest value has been found by an operator, that best value is broadcast to all particles in the swarm. This creates a truly distributed model with no reliance on a central operator to process the results of all the particles. Particles must maintain an active list of all particles in the swarm in order to broadcast the new global best value. If a new particle(robot) is added to the swarm, all particles must be updated to be made aware of that new particle. In his results he found a good deal of success, with the robots nding their target 99% of the time. Additionally, he found that as you increase the number of particles in the swarm, the time to nd the solution considerably reduces. Figure 2.4: dPSO Time to nd Target (Hereford, 2006) 16 CHAPTER 2. BACKGROUND RESEARCH AGLDPSO In Wang et als. paper "Adaptive Granularity Learning Distributed Particle Swarm Optimization for Large-Scale Optimization" a distributed implementation is proposed, called Adaptive granularity learning distributed particle

swarmoptimization (AGLDPSO) (

Z.-J. Wang et al., 2020). AGLDPSO uses a master/slave relationship when creating distributed particles, which is signi cantly di erent to Herefords distributed model, whereby all particles are peers within a swarm, there is no orchestration or master controller. Each particle, or robot was aware of all other robots in the swarm and up- dated accordingly. In AGLDPSO, a master acts as the intermediary and updates all particles in the population. This works very well for their implementation but creates a network bottleneck that dSPO altogether avoids. When tested against multiple other optimisation algorithms the authors found that AGLDPSO "achieves a promising and satisfying performance when solving the large-scale optimization problems.". Figure 2.5: Flow Diagram of AGLDSPO (Z.-J. Wang et al., 2020) 17 CHAPTER 2. BACKGROUND RESEARCH 2.4 Evaluating Performance 2.4.1 Benchmarking Benchmarking, also referred to as "best practice benchmarking" or "process bench- marking", is a method of conducting necessary actions and activities in order to eval- uate how a given piece of software or a system is going to perform once it is deployed and used in the eld (Jetmarov a, 2011). This is evaluated from a user's perspec- tive, and is typically assessed in terms of throughput, stimulus-response time, or some combination of the two. (Vokolos & Weyuker, 1998). It is also a way of assessing the systems availability. Meaning that whenever the system undergoes high levels of stress, be it increasing the number of processed requests (throughput), or high resource consumption (machine resources), the system still processes these requests or events in acceptable numbers (Vokolos & Weyuker, 1998). Usually benchmarking involves predicting how a technology will behave once it's used in regular everyday life. For instance, benchmarking a web server will allow organisations to predict how well it will cope with increased web tra c. These predictions allow organisations to antici- pate potential limitation and can lead them to a plan to overcome those limitations. Utilising a benchmarking evaluation strategy, the study will compare the two pro- posed PSO implementations, providing insights into the performance of a distributed implementation. 2.5 Research Gaps As outlined above there has been a large amount of research in the area of Swarm intelligence over the past 20 years. We can see in the J. F. Schutte et. al article they had a comprehensive cover of using parallel PSO (Schutte, Reinbolt, Fregly, Haftka, & George, 2004), however, to limit network connection time they used a parallel processor for running the algorithm. This would certainly cut down on networking connection times, however it cannot be described as a truly distributed algorithm. S. Burak Akat and Veysel Gazi focused much more on the distributed aspect of the 18 CHAPTER 2. BACKGROUND RESEARCH algorithm, but no e ort was made at running comparisons between the distributed version and a centralised version of the algorithm (Akat & Gazi, 2008). James M. Hereford again focused on the distributed aspect of the algorithm, with no relative benchmarks comparing it to a centralised version. (Hereford, 2006) The gap identi ed for this research is the performance bene ts of running a PSO algorithm in distributed environments vs centralised environments. As you can see from the above there has been plenty of research into crafting and testing distributed implementations of the PSO algorithm. However there has been no published literature testing the PSO algorithm in distributed environments vs a centralised environment and comparing the results, with an emphasis on nding a point where a distributed implementation is more time e cient than a centralised implementation. 2.6 Conclusion In this Chapter an overview of PSO was presented, along with some more in-depth research into parameter selection and di erent PSO implementations were discussed. multi-swarm PSO was discussed at a high level and two distinct distributed PSO implementations were presented and discussed. Research gaps were identi ed and discussed, along with the evaluation metric that will be used in this study. The next Chapter will focus on the experiment design and methodology. It will also cover the aim of the research and provide a basic overview of the experiment implementation details. 19 Chapter 3 Experiment design and methodology 3.1 Introduction This Chapter presents the design used to craft the experiment and the methodologies used to test the results from that experiment. The Chapter will cover the aim of the research, the design and implementation of the experiment, data output design and will conclude with design conclusions. 3.2 Aim of Research The aim of this paper is to nd out if a point exists when scaling a PSO algorithm that it becomes more e cient to run a distributed algorithm instead of a centralised algorithm. Increasing the number of active particles in a PSO algorithm can lead to a decline in performance. As for each iteration or epoch the machine must dedicate more and more computational power to recalculate particles positions and best evalua- tions. In a distributed model this burden can be divided across a number of machines, however in order to calculate the GBest value network communications must occur between machines, thus forming the swarm. Network communications are inherently slower than local connection communications, however at high levels of active particles 20 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY this speed decrease may be alleviated by the increased computational power available. 3.3 Hypothesis Null Hypothesis H 0 : The time taken to nd a stabilised result from using a distributed network of swarms never exceeds the time lost from the network communications needed to update swarms of each other's presence and activities. Alternate Hypothesis H 1 : The increased speed to solution time from using a distributed network of swarms exceeds the time lost from the network communications needed to update swarms of each other's presence and activities once the number of particles and/or swarms has reached a su cient level. 3.4 Design overview In order to create the data required to prove/disprove the hypothesis two code bases will need to be created. One code base will need to handle generating the data required for a centralised PSO, and the other will need to generate the data from a distributed PSO implementation. Each will need to be able to run the same tness functions and operate nearly identically, except the distributed implementation, which will need to call other PSO nodes to run swarms rather than running them on the local system. The coded system will need to be run multiple times with the same inputs in order to account for the random element inherent in the PSO algorithm. After each run, the system will need to output all generated data and a data aggregator will need to sort, average and output all the data in order for it to be evaluated. 21 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY Figure 3.1: Experiment Design Flow Chart 22 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY Figure 3.1 displays the ow of the experiment. First the environment type will be chosen, then from that point the required tness function will be chosen. To give greater breadth to the results, three di erent tness functions will be used with varying degrees of complexity. Once the tness function has been chosen, the swarms and par- ticles can be initialised with the function implementation. In the case of the centralised implementation this will be the same machine, in the decentralised implementation swarm nodes will be contacted and initialised remotely. Once each environment type has completed its run and found a stable answer, the implementations will return a data response object. At this point the data aggregator will pull in the results from both implementations, with corresponding con guration details, and generate a data output le useful for data comparison. This data le will be loaded into a data tabulation tool, whereby the results can be inspected, and comparisons can be drawn up. At this point the study will be able to draw conclusions on whether the alternate hypothesis can be accepted or rejected. 3.4.1 Centralised PSO Design Expanding on the design overview, the centralised implementation will need to perform several steps in order to generate the required results set. Firstly, it will initialise and create a parameterised number of swarms, each with a parameterised number of particles. At this point the rst iteration will begin for all swarms. This will all occur on the one node. The iteration will follow the same format as what was discussed in section 2.2.1. Each swarm will attempt to nd its GBest value. All swarms will then report the GBest value. At this point the controller class, which will also be the class that instantiates the swarms, will try and calculate if a "settled" GBest value has been found. What this means is that at a certain point the algorithm will settle on a particular value, the global minimum. This will be the optimum solution to the tness function. In this context settled means that for a certain number of iterations the same GBest value will be returned across all swarms. So, the system will need to record past values and compare against returned values to see if the same value has 23 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY been returned for multiple iterations in a row. When this has occurred the domain controller will calculate the time to arrive at this optimal solution, and the number of iterations it took to nd that value, this being the Time To Solution (TTS) value. The basic ow of centralised implementation is shown below in gure 3.2. Figure 3.2: Flow Diagram of Centralised Particle Swarm Algorithm 24 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY 3.4.2 Distributed PSO Design Figure 3.3: Flow Diagram of Distributed Particle Swarm Algorithm As discussed in the overview, the distributed implementation will function similarly to the centralised implementation to keep di erences in results to a minimum. The only functional di erence will be that swarms will be logically separated across multiple ma- chines in the distributed implementation. Where the centralised implementation will be running a domain controller, and all swarm instances, the distributed implementa- tion will feature a domain controller on a single node, calling out to other networked 25 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY nodes, that will

start up

100%	MATCHING BLOCK 14/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

and iterate individual swarms with a parameterised number of particles. Once an iteration ends the node will communicate back to the domain controller, which will collect responses from all active swarms, and check to see if the swarms have "settled" on the parameterised global minimum value. If they have the domain will stop the nodes from running any further iterations, collect resulting val- ues,

i.e.

99%	MATCHING BLOCK 15/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

number of iterations and TTS values. Network time will be accounted for in the TTS value. 3.5 Distributed environment In order to create a distributed environment for the distributed implementation for this project a cloud provider was used. Using these cloud providers, a virtual machine can be created and initialised, allowing the end user or developer to install custom software and applications. In this instance it allows the distributed implementation and centralised implementation to be deployed and run in a containerised environ- ment. Snapshots were used to ensure consistency between the two implementations and between the nodes of the distributed implementation. To facilitate this a num- ber of di erent providers were considered, including Amazon Web Services (AWS), Google cloud and DigitalOcean. Each had a containerised virtual machine product that allowed the deployment of custom applications, along with unique domain names and other internet connection utilities, however only one provider was required and DigitalOcean was chosen as the provider for this project. 3.5.1 DigitalOcean DigitalOcean is an Infrastructure as a Service (IaaS) platform for software developers. It allows developers to create containers in a simple and quick manner. To deploy DigitalOcean's IaaS environment, developers launch a virtual machine instance, which DigitalOcean calls a "droplet". This is containerised, and a number of operating systems can be chosen to initialise the doplet. The OS's available to developer include: Ubuntu, CentOS, Debian, Fedora, CoreOS or FreeBSD. DigitalOcean also o er the 26 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY ability to choose the geographic region the droplet will be created in, and some other con guration options such as the amount of dedicated RAM, number/quality of CPU, and the size of the on-board hard drive. Once initialised the developer may install their own custom software or use some prepared packages from DigitalOcean to help speed along deployment. Once the droplet has been created developers have the ability to monitor activity from their browser using DigitalOcean's control panel. They also have the ability to update the domain name, and create a Virtual Private Network (VPN) for the droplet. Snapshots of the droplet can be taken at any time and used to recreate it if required. DigitalOcean is a very a ordable option among its peers, with basic containers costing at minimum 0.00744$ an hour, working out at 5$ a month. 1 For that price DigitalOcean provide a container with one gigabyte of RAM, one core CPU and twenty gigabytes of on-board storage. 3.5.2 Container and Network Design Utilizing the containerisation service previously mentioned, a number of droplets will be created. Droplets will contain the implementation code in order to act as a domain controller and as a swarm node. However each node will only ever act as a domain controller or a swarm node. This design can be seen in gure 3.4. When running the distributed implementation, the domain controller node will be called using the same request as the non-distributed method, however instead of an integer de ning the number of swarms to be used, a list of IP address will be passed in. These IP addresses will correspond with the IP addresses for the droplets running swarm nodes. The domain controller will then connect with each of these droplets, initialise the swarm node with the passed in con guration values, begin iterating and collecting responses from all connected swarm nodes. 1 https://www.digitalocean.com/pricing 27 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY Figure 3.4: Container Design Overview This design does feature a core aw in that it creates a central dependency in the domain controller. However in order to keep the results as consistent and in line with the centralised implementation, the domain controller will be the same droplet size as all other swarm nodes. 3.6 Data Design and Data Capture Overview For the purpose of this project the following variables will be recorded for comparison. 28 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY Variable Description Data Type TTS Time taken to get to solution Integer N-Particles Number of active Particles Integer N-Swarm Number of active Swarms. Integer Environment-Type Distributed or Centralised String Iterations Total number of iterations/epochs to reach a solution Integer nalGroupBest Final GBest from all swarms Integer averageBestX Average best X value across all swarms Integer averageBestY Average best Y value across all swarms Integer Table 3.1: Data output variables Starting with the rst variable in table 3.1, the time variable, this will represent the amount of time it took the speci c test to calculate an answer, essentially this is the TTS value. This will be recorded in milliseconds, and it will be calculated by the domain controller. Starting from the time it initialises the swarms in either the distributed implementation or the centralised implementation and ending when either implementation arrives at a settled answer. The number of particles and swarms will be speci ed in the request when running the algorithm but will also be recorded as part of the output for ease of comparison of results. Again, environment type will be speci ed in the request and recorded in the output for comparison. The iterations variable will be used to record the number of iterations or epochs it took to reach a settled solution. This will also include the de ned number of iterations for a result to be considered settled, which will in ate the total number of iterations, however as this a ects all test runs it will have no signi cant impact on the nal values. FinalGroupBest value will record the nal GBest answer, from all swarms. All iterations should come to the same value as the tness functions used will contain only one global minimum. However it was recorded to see if there was any variation in the results between implementation types. Similarly with average best X and Y values, all implementation

88%	MATCHING BLOCK 16/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

types should conclude with the same values, but they were recorded to see if there was any variation. This data will be outputted to a JSON(JavaScript Object Notation) le at the end of 29

CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY each test run. The JSON le will be named in the following convention: >Function-Name<_>SwarmNumber<_>ParticleNumber<.results.json 3.6.1 Data Aggregation As mentioned in section 3.4 Design overview, due to the random element of the PSO algorithm, an average of results with the same parameters will need to be taken. Therefore multiple data output les will be generated for tests with the same inputs. In combination with multiple tests with varied numbers of particles and swarms, a large number of output les will be generated. To deal with this a data aggregation system will need to be created. This aggregation system will load in results with the same input variables, create an average values for the TTS variable, number of iterations, nalGroupBest, and averageBestX/Y values. Values will then be outputted into a single data le, with results from the distributed and centralised implementation being directly compared. 3.7 Design and Methodology Summary This Chapter presented an overview of the design and methodology used for this re- search. It discussed the overarching design of the experiment, with subsequent sections going into a more detailed design of speci c elements of the research. Data output and capture was also discussed, with a data output design presented in some detail. The next Chapter will focus in on the implementation detail, providing coded examples of the design presented in this Chapter. 30 Chapter 4 Experiment Implementation 4.1 Introduction This Chapter will expand on the previous Chapter and cover the implementation details in more depth. Coding examples will be provided for more clarity on how the two PSO implementations function. Additionally, the test harness and data loader that were created for this experiment will be expanded upon. Coded examples of each will be given. 4.2 PSO Implementations 4.2.1 Java For the purpose of this study Java will be used as the implementation language. Java is a mature language, with its initial 1.0 release in 1995. Since then, it has gone through multiple iterations and major version changes. It has been largely adopted by enterprise organisations due to its stability and feature-rich ecosystem. Java was chosen due to the "deploy anywhere" model that allows for applications to be run on any operating system that supports a JVM(Java Virtual Machine). This will be a bene t when the application is deployed to a distributed environment. Other 31 CHAPTER 4. EXPERIMENT IMPLEMENTATION advantages of using Java are its quite performant 1 2 , has a large set of supported libraries and extensions, along with a very active community and a wide range of supporting tools and documentation. 4.2.2 Spring Boot Spring Boot is an open source Java-based framework, which includes many utilities and libraries that allows a developer to quickly create a "production-grade" 3 application. It is developed and maintained by the Pivotal team. Spring boot is con gured with an embedded servlet container, contains many auto-con guration features for ease of development, and provides plenty of additional tools for crafting microservices. Spring Boot was used in this experiment because of its mature and stable nature, and the ability to easily create REST(Representational State Transfer) based services. Due to the distributed nature of the experiment, REST API calls were used to facilitate the messaging between swarm nodes and the domain controller. More on this will be discussed in the following sections. Spring Boot allows developers to quickly con gure REST API's and is reasonably performant during runtime 4 . Spring Boot also allows developers to easily con gure a number of di erent logging tools and frameworks to aid in debugging runtime errors and exceptions, which was a requirement during the development phase of this experiment. 4.2.3 Centralised Implementation Using Spring Boot a basic rest controller was created, whereby the swarm could be initialised and run by calling one HTTP POST endpoint with a request object that contains the con gurable values. Listing 4.1 shows the coded example of this. 1 @PostMapping("/runSwarm") 2 public Response runCentralisedSwarm(@RequestBody Request request) throws Exception f 3 return centralisedService.runSwarm(request); 1 https://www.infoq.com/presentations/java-8-13-performance/ 2 https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go 3 https://spring.io/projects/spring-boot 4 https://spring.io/blog/2018/12/12/how-fast-is-spring 32 CHAPTER 4. EXPERIMENT IMPLEMENTATION 4 g Listing 4.1: Centralised Rest Controller Once the endpoint in the rest controller receives a request, it will call out to a service method which will initialise the swarms and particles. This then begins running the algorithm. Listing 4.2 is an extract of how these swarms are initialised in the service method. The swarm and particle objects are abstracted out of the service class, so that the distributed service class can reuse the same methods, with some modi cations. The service class also handles generating the response object with all necessary data points discussed in Chapter Three. 1 List>Swarm< multiSwarm = new ArrayList><(); 2 ... 3 /∗ Initialize all swarms and particle with supplied con guration ∗/ 4 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 5 multiSwarm.add(utils.createSwarm(swarmId, request.getNumberOfParticles(), request. ,! getCon gVariables())); 6 g Listing 4.2: Centralised Swarm initialisation Listing 4.3 shows an extract of the swarm iteration code in the centralised service. Once the swarms are initialized the service class moves onto iterating through each swarm. We can see in the extract this is done through a while loop. In the while loop each iteration of all swarms is done through a for loop. Each iteration of the while loop counts as a full iteration for all swarms. In the listing we can also see the code records a start time and a stop time. Once completed the time between entering the service class and completing the while loop is calculated. This is used to accurately calculate the time taken only for the PSO code to run, rather than the time to get a response from the rest controller. This is returned as the TTS value. 1 Instant start = Instant.now(); 2 ... 3 while(!targetFound) f 4 ... 5 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 33 CHAPTER 4. EXPERIMENT IMPLEMENTATION 6 Result result = utils.runSwarmIteration(multiSwarm.get(swarmId), stepCount); 7 ... 8 g 9 ... 10 stepCount++; 11 g 12 Instant nish = Instant.now(); 13 long timeElapsed = Duration.between(start, nish).toMillis(); 14 ... Listing 4.3: Centralised Iteration Method Within the main while loop the code contains the stopping criteria. Listing 4.4 shows an extract of this code. The code loops through each of the

swarm's re-

98% **MATCHING BLOCK 18/50** SA C13358011_MSC_ASD.final.pdf (D108754758)

sponses and each of their responses are compared to a previous value, and if the exact same global best has been found for the con gured number of times (The TARGET_NUMBER_OF_EXACT_ANSWERS variable) then the while loop forcibly ends. This is done through the use of an array list. If the array list is not null, the code checks if the array list contains the current swarms GBest value. If it does, the value is added to the array list, and the code checks the size of the array list, and if it is the same as the TARGET_NUMBER_OF_EXACT_ANSWERS variable then the while loop ends. If the value isn't present, then the array list resets itself to ensure all swarms return the exact same answer the same number of times. If the step count i.e. the number of iterations reaches a certain threshold (The MAX_NUMBER_OF_STEPS variable) the while loop is also terminated to prevent the system from becoming trapped in an unending loop. This is assumed to be the worst-case scenario and is not expected to regularly happen. 1 /∗ Check if the same answer has been found multiple times ∗/ 2 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 3 if (settledList.size() == 0) f 4 settledList.add(bestEvaluation[swarmId]); 5 g else f 6 if (settledList.contains(bestEvaluation[swarmId])) f 7 settledList.add(bestEvaluation[swarmId]); 34 CHAPTER 4. EXPERIMENT IMPLEMENTATION 8

99%	MATCHING BLOCK 19/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

g else f 9 settledList = new ArrayList><(); 10 g 11 g 12 if (settledList.size() == TARGET NUMBER OF EXACT ANSWERS) f 13 TARGET NUMBER OF EXACT ANSWERS); 14 targetFound = true; 15 g 16 g Listing 4.4: Centralised Stopping Criteria Swarm Creation and Initialisation Listing 4.5 has an extract of the swarm creation method, createSwarm. From this method we can see how it creates a tness function based on an enum ("An enum type is a special data type that enables for a variable to be a set of prede ned con- stants" 5) passed in, allowing for the system to be extended with new tness functions with a low amount of code change. Fitness functions are de ned in the request sent into the service, and it initialises the swarm with the number of particles and other con guration values. 1 public Swarm createSwarm(int swarmId, int numberOfParticles, Con gVariables con gVariables) ,! throws Exception f 2 ... 3 if(con gVariables.getFunctionType().equals(FunctionType.BOOTHS FUNCTION))f 4 function = new BoothsFunction(); 5 g 6 ... 7 return new Swarm(numberOfParticles, con gVariables, function); 8 g Listing 4.5: Swarm creation Listing 4.6 shows how the swarm object is initialised. Best position and best evaluation are both set to the positive in nity, as this PSO is trying to nd the global minimum of 5 https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html 35 CHAPTER 4. EXPERIMENT IMPLEMENTATION optimisation functions. If instead the object was to nd global maximums this would need to be set to negative in nity. Inertia, cognitive and social parameters are set using con gured values. Particles are initialised within the function de ned in 4.5. 1 public Swarm (int numberOfParticles, Con gVariables con gVariables, Function function) f 2 double in nity = Double.POSITIVE INFINITY; 3 bestPosition = new Vector(in nity, in nity, in nity); 4 bestEval = Double.POSITIVE INFINITY; 5 this.particles = this.initialiseParticles(numberOfParticles, function); 6 this.inertia = con gVariables.getInertia(); 7 this.cognitive = con gVariables.getInertia(); 8 this.social = con gVariables.getSocial(); 9 g Listing 4.6: Swarm Initialisation Swarm Iteration Each swarm iteration is triggered by the service class calling it. Within the swarm object there is a run swarm method, which iterates through each particle, updating its personal best and the GBest value. Once that is completed the swarm updates all particles velocity and then moves each particle to its new position based on that velocity. A response object is created based on the iteration/epoch number including the global best value and best X/Y values. Listing 4.7 shows this method in detail. 1 public Result runSwarm(int epoch) f 2 for (Particle particle : particles) f 3 particle.updatePersonalBest(); 4 updateGlobalBest(particle); 5 g 6 7 for (Particle particle : particles) f 8 updateVelocity(particle, this.inertia, this.cognitive, this.social); 9 particle.updatePosition(); 10 g 11 12 return createResultObject(epoch); 36 CHAPTER 4. EXPERIMENT IMPLEMENTATION 13 g Listing 4.7: Swarm Iteration The swarm, particle and vector models, along with the swarm velocity update function were adapted from LazoCoder's Particle-Swarm-Optimization implementation 6 . The codebase was heavily adapted in order to support the multi-swarm and distributed implementations, along with the required output data. 4.2.4 Distributed Implementation Similar to the centralised implementation listing 4.8 shows the REST controller for the distributed implementation. Unlike the centralised implementation there are three endpoints. When running the algorithm from the beginning, the "/distributedImple- mentation" endpoint is used. The node that this endpoint was hit with will become the domain controller and will call out to all other swarm nodes established in the network. The other two endpoints present are used by the system to run the algorithm. The endpoint "/initialiseSwarm" initialises the swarm with passed in variables. The code retains a copy of the swarm in its initialised state through the use of global variables. Then the domain controller will call the endpoint "/runSwarmIteration" which will take the in memory swarm object and run one iteration, returning the response values from it to the calling domain controller. 1 @PostMapping("/initialiseSwarm") 2 public Boolean initialiseSwarm(@RequestBody DistributedRequest request) throws Exception f 3 return distibutedService.setInitialisedSwarm(request.getSwarmId(), request. ,! getNumberOfParticles(), request.getCon gVariables()); 4 g 5 6 @PostMapping("/runSwarmIteration") 7 public Result runSwarmIteration(@RequestBody DistributedRequest request) f 8 return distibutedService.runSwarmIteration(request.getEpoch()); 6 Particle-Swarm-Optimization codebase: https://github.com/LazoCoder/Particle-Swarm -Optimization 37 CHAPTER 4. EXPERIMENT IMPLEMENTATION 9 g 10 11 @PostMapping("/distributedImplementation") 12 public Response distributedImplementation(@RequestBody DistributedRequest request) f 13 return distibutedService.runDistributedCallingSystem(request.getNumberOfParticles(), ,! request.getBaseUrls(), request.getCon gVariables()); 14 g Listing 4.8: Distributed Rest Controller Similar to the centralised implementation, the distributed rest controller calls out to a service class, which initiates all the swarms and runs the iteration steps. Listing 4.9 is an extract of that initialisation part of this class. We can see this di ers to the centralised implementation in that it uses an entirely di erent method to initialise the swarms. This method is used to call out to the remote nodes to initialise the swarm's there. 1 for(int swarmId=0; swarmId > numberOfSwarms; swarmId++)f 2 ... 3 callInitialiseSwarms(swarmId, baseUrls.get(swarmId), distributedRequest); 4 g Listing 4.9: Distributed Initialisation Method Listing 4.10 shows an extract of the iteration loop code for the distributed implementa- tions. There is more required of the code here than in the centralised implementation. Here the code must craft a request to send to each swarm node. Each request will contain all con guration values and the swarm ID, which will need to be returned by the swarm node to identify it when formatting the results response object. Again, the while loop persists until a target number of the same answers has been recorded, or the step count exceeds a speci ed threshold. Once the while loop is complete the timer stops, and the TTS is recorded and a response object is formed. The response object for the distributed implementation is the exact same as the centralised imple- mentation. This means the stopping criteria will function identically to the extract found in gure 4.4. 38 CHAPTER 4. EXPERIMENT IMPLEMENTATION 1 while(!targetFound) f 2 ... 3 for(int swarmId = 0; swarmId>numberOfSwarms; swarmId++) f 4 DistributedRequest distributedRequest = new DistributedRequest(); 5 distributedRequest.setSwarmId(swarmId); 6 distributedRequest.setCon gVariables(con gVariables); 7 Result result = callRunSwarm(swarmId, baseUrls.get(swarmId) ,distributedRequest); 8 multiSwarmResults.get(swarmId).getResults().add(result); 9 bestEvaluation[swarmId] = result.getBest(); 10 g Listing 4.10: Distributed Iteration Loop Rest calls to swarm nodes As seen in the above section, the service class calls out to an additional method to handle the REST call for initialising the swarms. Listing 4.11 shows this method, and in it we can see it calls out to the swarm node based on the passed in URL. The endpoint, port and HTTP protocol will be the same for all the nodes, so this is hardcoded in the method. The method passes the con guration details to the swarm node and returns a Boolean to acknowledge whether the initialisation was successful. If any swarm was not successful, the entire system aborts the run. This would be considered the worst case scenario and is not expected to happen frequently. 1 private Boolean callInitialiseSwarms(int swarmId, String baseUrl, DistributedRequest ,! distributedRequest) f 2 String fullUrl = "http://" + baseUrl + ":8080/swarm/initialiseSwarm"; 3 ... 4 HttpEntity>DistributedRequest< request = new HttpEntity><(distributedRequest); 5 try f 6 ResponseEntity>Boolean< responseEntity = restTemplate.exchange(fullUrl, HttpMethod. ,! POST, request, Boolean.class); 7 return responseEntity.getBody(); 8 g catch (Exception ex) f 9 return false; 10 g 39 CHAPTER 4. EXPERIMENT IMPLEMENTATION 11 g Listing 4.11: Rest call to initialise swarm Listing 4.12 shows the code snippet for calling the swarm nodes to run a single iter- ation. Similar to the initialise method it calls out to the swarm using a xed URL scheme, only amending the base URL according to the swarm node. It does however expect a response back that details the iteration number, TTS, global best variable and best X and Y variables. This response will be added to the general response object returned once the test run is complete. If an exception occurs during the test run the whole run is aborted, again this is a worst-case scenario and is not expected to happen frequently. 1 private Result callRunSwarm(int swarmId, String baseUrl, DistributedRequest distributedRequest ,!) f 2 String fullUrl = "http://" + baseUrl + ":8080/swarm/runSwarmIteration"; 3 log.info("Calling endpoint fg to run swarm with id fg", fullUrl, swarmId); 4 HttpEntity>DistributedRequest< request = new HttpEntity><(distributedRequest); 5 try f 6 ResponseEntity>Result< responseEntity = restTemplate.exchange(fullUrl, HttpMethod. ,! POST, request, Result.class); 7 return responseEntity.getBody(); 8 g catch (Exception ex) f 9 ... 10 throw new Exception("Error calling service to iterate swarm", ex); 11 g 12 g Listing 4.12: Rest call to run swarm iteration On the swarm nodes receiving end the distributed service has simple methods for using the exact same initialisation and iteration methods as the centralised implementation. As mentioned previously this was done in order to keep the PSO algorithm code di er- ences between the two implementations to an absolute minimum. The only di erence being that the swarm state is kept in memory once the method has completed through the use of global variables. Listing 4.13 shows how this works for initialisation. The 40 CHAPTER 4. EXPERIMENT IMPLEMENTATION swarm is initialised and stored in memory through the global variable initialisedSwarm. Each swarm iteration simply references this global variable and uses the same iteration command as the centralised implementation, shown in listing 4.7. 1 private Swarm initialisedSwarm; 2 public boolean setInitialisedSwarm(int swarmId, int numberOfParticles, Con gVariables 3 con gVariables) throws Exception f 4 this.initialisedSwarm = 5 utils.createSwarm(swarmId, numberOfParticles, con gVariables); 6 .... 7 g Listing 4.13: Distributed swarm initialisation 4.2.5 Fitness Functions overview In order to thoroughly test the hypothesis three tness functions were chosen. The three functions ranged in computational complexity in order to give a greater breadth of results to draw conclusions from. All functions have only one global minimum and all are two dimensional only. The rst optimisation function chosen was the Easom function, equation 4.1 represents this 7 . The easom optimisation function has a global minimum value of -1 where X and Y are equal to . f(x;y) = *cos(x)cos(y)exp(*(x *) 2 * (y *) 2) (4.1) The next equation chosen was the Beale's function, equation 4.2 represents this 8 . Beale'

s

Curiginal

96%	MATCHING BLOCK 20/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

function has a global minimum value of 0 where X is equal to 3 and Y is equal to 0.5. f(x;y) = (1:5 * x + xy) 2 + (2:25 * x + xy 2) 2 + (2:625 * x + xy 3) 2 (4.2) The nal equation chosen was the Booth's function, equation 4.3 represents this 9 . 7

100%	MATCHING BLOCK 22/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

http://www.sfu.ca/ ~ ssurjano/easom.html 8 http://www.sfu.ca/ ~ ssurjano/beale.html 9 http://www.sfu.ca/ ~ ssurjano/booth.html 41 CHAPTER 4. EXPERIMENT IMPLEMENTATION

Booth's

function

has a global minimum value of 0 where X is equal to 1 and Y is equal to 3.

100%	MATCHING BLOCK 21/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

f(x;y) = (x + 2y * 7) 2 + (2x + y * 5) 2 (4.3) 4.3 Test Harness As described in Chapter Three, a test harness was created in order to test the PSO implementations. Separating out the testing element from the code implementation allowed for greater exibility in testing and allowed for tests to be updated inde- pendently of the code deployments to the containers within DigitalOcean. The test harness was designed to be exible with its parameters, along with being able to run multiple tests while saving the responses from each test run into a results folder. 4.3.1 Postman/Newman and NodeJs Figure 4.1: Postman Test In order to accomplish that goal Postman was chosen to create the tests with, and Newman was used as the test runner. Postman is a "scalable API testing tool that quickly integrates into CI/CD pipelines." 10 . Postman allows you to easily de ne a test 10

100%	MATCHING BLOCK 23/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

https://www.guru99.com/postman-tutorial.html 42 CHAPTER 4. EXPERIMENT IMPLEMENTATION

98%	MATCHING BLOCK 24/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

for a remote API HTTP endpoint, specifying headers, body parameter, and display the responses from that API. Postman comes equipped with plenty of development tools, for example it supports environment selection and parameters, allowing developers to substitute values based on the environment they have chosen. Figure 4.1 shows an image of a Postman test opened in the Postman application. In the gure it can be seen how the URL can be parameterised using the curly brackets operators {{Parameter_Name}}, which can also be applied to body variables. Postman can organise a series of tests into what it calls a test collection, which can be exported to JSON les and saved locally. Newman is the command line implementation of Postman. Newman can take these JSON test collections and run them in a batch format. Newman runs on NodeJs and is installed through NPM. NodeJs is a "an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications." 11 . NPM (Node Package Manager) is a command line package manager tool that allows developers to download and install javascript libraries to their local development environment. While Newman is mainly described as a CLI tool, it has a full API available for developers to use through node. Utilising this API allows for developers to create programs utilising Postman test suites, while being able to programmatically change parameters as required. 4.3.2 Implementation The test harness was split into two main methods for looping, one for the centralised run, and one for the distributed run, however they only di er in one aspect. When adding additional

swarm's

100%	MATCHING BLOCK 25/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

the distributed implementation builds up a list of IP ad- dress corresponding to the swarm nodes. The centralised implementation only adds the required number of swarms to the request. Listing 4.14 shows how this loop func- tions. This is speci cally the centralised method, however the distributed method is nearly identical except for some minor con guration changes. The harness rst iter- ates through each tness function i.e. Beale'

s, Booth's and Easom's function. For 11 https://nodejs.org/en/about/ 43

CHAPTER 4. EXPERIMENT IMPLEMENTATION

96%	MATCHING BLOCK 26/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

each tness function test set, the number of swarms is gradually increased from two to ten, and the total number of particles across all swarms in each test increases from 100,000 to 1,000,000. So, for example when the test is for two swarms, each swarm will increase from 50,000 particles to 100,000, to 150,000 etc. until the total number of particles is 1,000,000. Each of these tests will keep the same con guration values and will be run three times so that an average across results may be obtained. The harness is con gured to wait for each request to return a response object before mov- ing onto the next request, to avoid overloading the droplets. Newman by default runs asynchronously, so the harness was con gured to check every second if a le with the expected test name was written to the disk, and if it has move on, if not keep looping. 1 async function centralisedRun() f 2 for(var functionId = 0; functionId>= functionNames.length-1; functionId ++) f 3 for (var swarmNumber = 1; swarmNumber >= maxNumberSwarms; swarmNumber++) f 4 particleIncrement = 1000; 5 for (var particleNumber = particleStartNumber; particleNumber >= ,! maxNumberParticles; particleNumber += particleIncrement) f 6 runNewmanCode("NON DISTRIB", functionNames[functionId], "Remote", ,! swarmNumber, particleNumber, baseUrlStringsNonDistributed, ,! nonDistribEndpoint); 7 var moveOntoNextTest = false; 8 while (!moveOntoNextTest) f 9 moveOntoNextTest = checkIfFilesExist("NON DISTRIB", functionNames[,! functionId], swarmNumber, particleNumber); 10 await new Promise(resolve =< setTimeout(resolve, 1000)); 11 g 12 g 13 g 14 g 15 g Listing 4.14: Test Harness Main Loop Listing 4.15 shows the Newman node implementation code. The implementation allows for a lot of reuse, by accepting parameters for changing the PSO implementation type, number of swarms/particles and tness function type. The code uses the function 44

CHAPTER 4. EXPERIMENT IMPLEMENTATION

99%	MATCHING BLOCK 27/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

name to load up the correct Postman collection type, there is only one test in each collection. Utilising the Newman API, and correctly loading up the test collection and environment variables le, the method calls out to the remote domain controller and records the responses. Errors are gracefully handled through Newman's inbuilt error handling system. This is repeated until all tests have been run across all tness functions. 1 function runNewmanCode(envName, testCollectionName, environmentName, numberOfSwarms, ,! numberOfParticles, baseUrlsSting, endpoint) f 2 newman.run(f 3 collection: require('./Requests/' + testCollectionName + '.postman collection.json'), 4 environment: require('./EnvVariables/' + environmentName + '.postman environment.json ,! '), 5 reporters: 'cli', 6 globalVar: [f "key":"NumberOfSwarms", "value":numberOfSwarms g, f"key":"endpoint", " ,! value": endpointg, f "key":"NumberOfParticles", "value": numberOfParticlesg, f " ,! key":"BaseUrls", "value": baseUrlsStingg] 7 g).on('request', function (error, args) f 8 fs.writeFile(dir + testCollectionName + " " + numberOfSwarms + " " + ,! numberOfParticles + '.result.json', args.response.stream, function (error) 9 g 10 g); 11 g Listing 4.15: Newman Implementation Code Once the method receives a response back from the service, it will use the function name, number of particles/swarms to write out the response to a le with those pa- rameters. Those parameters will be used in its name, allowing each response to be uniquely named, and avoiding overwriting the output le with each new response. 4.4 Data Aggregator Implementation As mentioned in Chapter Three, multiple test iterations will be run to get averages from the results. In this experiment each function was tested three times for the two 45

CHAPTER 4. EXPERIMENT IMPLEMENTATION

99% **MATCHING BLOCK 28/50** **SA** C13358011_MSC_ASD.final.pdf (D108754758)

implementation types, with swarms increasing from one to ten, and particles in all swarms increasing from 100,000 to 1,000,000. In order to aggregate all the results a simple data aggregation tool was created. The data aggregator needed to load data from each of these three tests runs, get an average value for the following values: Number of iterations, TTS and nal GBest. NodeJs was used to load the data and generate the average values and the system writes the result out as a CSV (Comma Separated Value) le. Listing 4.16 shows the code extract for this. In the listing we can see there is a set header string that will always be outputted at the beginning of the program, this is simply for ease of reading when the CSV is imported into excel. The program iterates through each function, swarm and particle number, then loads the JSON output le. Once it loads the result le for all three test runs, for both implementations it calculates the centralised av- erages and creates an output string in the CSV format. Once the centralised output string has been generated the program generates the same output string for the dis- tributed implementation. The output string is built up for the same tness function tests, and then outputted to the le system. 1 var outputString = "SWARM NUMBER,PARTICLE NUMBER,CENTRALISED ITERATIONS, ,! CENTRALISED TIME TO SOLUTION,CENTRALISED FINAL GROUP BEST, ,! DISTRIBUTED ITERATIONS,DISTRIBUTED TIME TO SOLUTION, ,! DISTRIBUTED FINAL GROUP BESTnn"; 2 3 for (var swarmNumber = 1; swarmNumber >= numberOfSwarms; swarmNumber++) f 4 for (var particleNumber = 1000; particleNumber >= maxNumberParticles; particleNumber ,! += particleIncrement) f 5 .... 6 let centralised RUN1 = fs.readFileSync(leUrlCentralised RUN1); 7 ... 8 let centralised data RUN1 = JSON.parse(centralised RUN1); 9 ... 10 var averageCentralisedTTS = Math.round((centralised data RUN1.timeToSolution + ,! centralised data RUN2.timeToSolution + centralised data RUN3.timeToSolution) ,! /3); 11 ... 46 CHAPTER 4. EXPERIMENT IMPLEMENTATION 12 13 outputString = outputString + swarmNumber + "," + particleNumber + "," + ,! averageCentralisedIterations + "," + averageCentralisedTTS + "," + ,! averageCentralisedGbest + ","; 14 ..... 15 g Listing 4.16: Data aggregation Implementation 4.5 Implementation conclusion This Chapter covered in depth how the PSO code implementations were developed, with a speci c focus towards coded examples. Each implementation was discussed in detail, along with how the distributed implementation speci cally calls swarm nodes on the network. The test harness and data aggregation tools

were

99% **MATCHING BLOCK 29/50** **SA** C13358011_MSC_ASD.final.pdf (D108754758)

also covered in depth, with plenty of code extracts supplied. Testing implementation was discussed in some detail. The next Chapter will cover this in more detail along with the results obtained from the tests. 47 Chapter 5 Results, Evaluation and Discussion 5.1 Introduction This Chapter discusses the results obtained from running the experiment described in detail in Chapters Three and Four. Results from the centralised and distributed implementations will be presented and compared. As discussed in Chapter Three, several tness functions were tested in this experiment. This Chapter will expand on each for both implementations, with a primary focus on the TTS value. This Chapter will also evaluate the results, with some discussion taking place. 5.2 Results 5.2.1 Centralised Results Chapter Three and Four described the process of running the experiment in detail, in this section the results from those experiment runs will be presented. Each tness function will be examined, and results will be presented. All results are an average of three tests runs in order to compensate for the random element in PSO. Results were compared in a like for like manner, meaning when we compare the number of particles between a two swarm test and a ten swarm test, we are comparing the total number of particles in the test, not how many particles on each swarm. To further expand on 48 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION that example, when comparing 100,000 particle tests, we look at there being 50,000 particles on each swarm in the two swarm test, and 10,000 particles on each swarm in the ten swarm tests. This goes for both the centralised and distributed implementation tests. Easom's Function The rst function to cover is the centralised Easom test runs. Figure 5.1 shows the results when looking at the TTS values. In the results we can see that as the number of particles increases, the TTS increases at a disproportionate rate. TTS for each swarm tends to increase at the same rate, except for the ten swarm test runs, where the TTS is slightly longer at the beginning, but increases at a much greater degree closer to the 1,000,000 particles tests. Figure 5.1: Centralised TTS Easom'

s

97%	MATCHING BLOCK 30/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

Results Figure 5.2 shows the number of epochs or iterations taken to reach a solution for each of the swarms. All swarms display a downward trend in the number of iterations it takes to reach a solution, with the two and four swarms coming in at the lowest number of iterations at the maximum number of particles, one million particles. From the gure it can be seen that the number of iterations, while on a downward trend, can have a lot of variance as the particles increase in the tests, i.e. in the two swarm test the average iterations jumps from 114 iterations for 600,000, to 120 iterations for 700,000 particles. As the swarms increase the variance between particle increments 49 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION decreases. Figure 5.2: Centralised Iterations Easom's Results Booth's Function Looking at the Booth'

s

94%	MATCHING BLOCK 31/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

function results in gure 5.3 we see it closely follow the Easom's function results, in that the general trend on the results looks to be at an exponential increase. Once again, the test runs where ten swarms are used show a slight increase in the TTS at lower numbers of particles, and a large increase towards the one million particles run. Figure 5.3: Centralised TTS Booth's Results Figure 5.4 shows the results of the average iterations obtained by each swarm in testing. Similar to the Easom'

s

96%	MATCHING BLOCK 32/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

results the gure showcases a downward trend in the number of iterations as particles increase. Again, the results show that the lower number of 50 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION swarms, two and four swarms, obtained a lower number of iterations at the higher number of particles. Figure 5.4: Centralised Iterations

Booth's Results Beale's Function Looking at the Beale'

s

100%	MATCHING BLOCK 33/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

TTS results in gure 5.5, the same general trends from Easom'

s and Booth's

94%	MATCHING BLOCK 34/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

function can be seen, with the general trend looking to be exponential. Again, the higher number of swarms tends to end up with a higher TTS as particles increase. Figure 5.5: Centralised TTS Beale's Results Figure 5.6 shows the number of epochs each swarm took to get to a solution. Once again, the general trend is that the number of epochs decreases with the increased number of particles, with some variations between the particle increments. Similar to 51 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION Booth's and Beale's function the lower number of swarms, two and four swarms had, on average, a lower number of epochs to reach a solution. Figure 5.6: Centralised Iterations Beale'

s

97%	MATCHING BLOCK 35/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

Results 5.2.2 Distributed Results This section examines the results from the distributed implementation tests, using the same format as the centralised results section. Each tness functions results will be presented, primarily focusing on the TTS and number of Epochs to solution, along with a small amount of commentary on these results. Easom's Function Looking at the Easom'

s

99%	MATCHING BLOCK 36/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

function rst, gure 5.7 shows the TTS results. TTS appears to linearly increase with the number of particles across all number of swarms. At the upper end of the experiment bounds, with 1,000,000 particles the two swarm tests appear to increase at a great rate than the four, eight and ten swarm tests, all of which appear to continue the same rate of increase in TTS. There are some variations in this linear increase, most notably the eight swarm tests show a large decrease in TTS at the 800,000 particle point, but otherwise the variations are relatively minor. 52 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION Figure 5.7: Distributed TTS Easom's Results Figure 5.8 shows the number of epochs to solution results. From that we can see the general trend is a linear decrease, however it is a very slight decrease. The two swarm and ten swarm tests show the least amount of change in results, with the two swarm tests decreasing by two epochs from start to nish, and the ten swarm tests decreasing by three epochs. The eight and four swarm tests showed a greater deal of variance, with the eight and four swarm tests decreasing by nine iterations. Figure 5.8: Distributed Iterations Easom'

s Results Booth's Function Looking next at the Booth's

96%	MATCHING BLOCK 37/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

function gure 5.9 shows the TTS results. Looking at the results shows similar trends as the Easom's function results, with the overall trend appearing to be a linear increase in TTS as the number of particles increases. Two 53 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION swarm tests once again appear to increase at a more aggressive rate towards the upper particle numbers. The ten swarm tests show an overall higher TTS than any other number of swarms. Eight swarm tests showed a large decrease in TTS at the 800,000 particles interval but resumes the same trend at the next interval. Figure 5.9: Distributed TTS Booth's Results Figure 5.10 shows the epochs to solution results. Unlike Easom's results, the results are a bit more mixed. The two and ten swarm tests start o at a lower number of epochs at the beginning of the tests, with both starting at 108 iterations and ending at 117 iterations for ten swarms, and 114 iterations for the two swarm tests. Both the four and eight swarm tests show a more linear decrease, from start to nish, however both show a great deal of variance at various intervals in the testing. Figure 5.10: Distributed Iterations

Booth's

91%	MATCHING BLOCK 38/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

Results 54 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION Beale's Function Finally, looking at gure 5.11 we can see the TTS results for the Beale's function. Again, this follows the same trends as Booth's and Easom's functions, however its even less erratic, with little to no variance in any of the swarm tests. Again, the two swarm test runs show signs of it increasing at more than a linear increase at the higher number of particles, 900,000 and 1,000,000 particle tests. Figure 5.11: Distributed TTS Beale'

s

100%	MATCHING BLOCK 39/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

Results Looking at gure 5.12 we can see that once again the results show a great deal of variance when it comes to the number of epochs to reach a solution. Similar to the

Booth's

94%	MATCHING BLOCK 40/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

function results, swarms two and ten start at a much lower number of epochs, but at the higher number of particles have much more similar results to the eight and four swarm results. Eight and four swarm results show a more gradual and consistent decline in epochs as the number of particles increases. 55 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION Figure 5.12: Distributed Iterations Beale's Results 5.2.3 Performance Comparison Figure 5.13 displays a direct comparison between the distributed implementation of the Easom'

s

100%	MATCHING BLOCK 41/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

TTS results, and the centralised TTS results. This clearly displays the general trend of the centralised implementation towards an exponential increase, whereas the distributed implementation more closely follows a linear increase in TTS. Figure 5.13: Performance Comparison Easom'

s

90%	MATCHING BLOCK 42/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

TTS Looking more closely at gure 5.13 we can see the two implementations seem to diverge in TTS increase rate at around 500,000 particles. This is more or less the same case in both Beale's, gure 5.14 and Booth's function gure 5.15. In the graphs we can see 56 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION

the

89%	MATCHING BLOCK 43/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

red line denotes this tipping point, Optimisation Tipping Point (OTP). In all three results it shows the distributed implementation increasing at a linear rate, while the centralised implementations begin to increase exponentially after the 500,000 particles tests. There are other patterns observable in all three results, including the same tendency for two swarm tests in the centralised implementation to increase at a more rapid rate than other number of swarms at the higher number of particles tests. Figure 5.14: Booth's TTS comparison Figure 5.15: Beale's TTS comparison All three functions display a large increase in TTS at the early stage of the experiment, with 100,000 particles. Table 5.1 compares these results. In the case of the Beale'

s 57

CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION

84%	MATCHING BLOCK 44/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

function, on average, the distributed implementation was 50% slower at nding a result. Easom's function had an even greater increase, taking 61% more time to nd a solution, and Booth's function was even bigger again, with it taking 82% more time in nding a solution. TTS - 2 Swarms TTS - 4 Swarms TTS - 8 Swarms TTS - 10 Swarms Beale's -Centralised 4205 3842 4775 4417 Beale's -Distributed 5291 5630 7153 7860

Booth's -Centralised 2910 3243 3634 2805 Booth's -

Distributed 4331 4846 6197 7541 Easom's -

77%	MATCHING BLOCK 45/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

Centralised 2364 2297 2313 2388 Easom's -Distributed 3551 3287 3925 4385 Table 5.1: TTS Results for 100,000 Particles Conversely there are some signi cant gains in TTS for the distributed implementation at the highest number of particles. Table 5.2 compares these results for 1,000,000 particle tests. For the Beale function it was found that the distributed implementation was on average 145% faster than the centralised implementation. Similar performance gains were found with both Booth's and Easom's function, with Booth's being 152% faster, and Easom's being 123% faster. TTS - 2 Swarms TTS - 4 Swarms TTS - 8 Swarms TTS - 10 Swarms Beale's -Centralised 112258 119800 112751 160899 Beale's -Distribributed 56502 48906 49696 51248 Booth's -Centralised 102709 103364 102856 131270 Booth's -Distribributed 48322 39878 36861 49330 Easom's -Centralised 50887 51263 52918 78160 Easom's -Distribributed 29327 23923 24623 26300 Table 5.2: TTS Results for 1,000,000 Particles When looking at the number of iterations between the two implementations there is very little di erence. Looking at gures; 5.16, 5.17, 5.18, we can see the same general trends in all three tness functions. There is an overall gradual decline in iterations as particles increase, with some large variations, especially at the lower end of the 58 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION particles. Overall, the distributed implementations showed more variations than the centralised implementations. Figure 5.16: Performance Comparison Easom's Epoch Figure 5.17: Booth's Epoch comparison Figure 5.18: Beale's Epoch comparison 5.3 Discussion 5.3.1 Results Evaluation From the performance comparison section, we can see there is a clear point in all three tness functions whereby the distributed implementation displays a better TTS than the centralised implementation. Interestingly, OTP occurs for all three functions at more or less the same point. Once the number of particles across all swarms reaches 500,000 or greater, the distributed implementation begins to be a more e cient implementation than the centralised implementation. Once the particles reach one 59

CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION

98%	MATCHING BLOCK 46/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

million all three functions had a greater than 100% decreases in TTS compared to the corresponding centralised tests. This clearly answers the research question, that there is in fact a point whereby it is more e cient to run a distributed PSO implementation over a centralised implemen- tation. However, this must be caveated by the fact that these results are only a guide, a di erent tness function may have a di erent e ciency point. Additionally, di erent implementation languages, or PSO implementation methods may also have di erent e ciency points. We can therefore conclude that the alternate hypothesis has been proven, and there is a point where a distributed implementation is more time e cient than a centralised implementation. In terms of di erences between the number of iterations taken before nding a solu- tion, between the two implementations, this study found no di erence between the distributed implementation and the centralised implementation. There was a notice- able decline in the number of iterations across both implementations as the number of particles increased, and this was evident across all three tness functions. When looking at other resulting data points, such as the average X/Y value, and nal GBest, this study found no di erence between the implementation, and no di erences as the number of particles increased. The number of particles used in this study can be con- sidered excessive, as the number of particles required to correctly nd a solution to any of the tness functions was signi cantly lower than what was used in this study. However, as matching the trend seen in both implementations, the lower number of particles required much more iterations to nd a "settled" solution. 5.3.2 Strengths and Limitations The key strength to this research is the consistency of the results across the three tness functions. All three tness functions displayed an OTP at around 500,000 particles, which gives a very clear answer to the research question. This also give a strong starting position for future research, as there is a clear answer as to whether the research should use a distributed or centralised model, based on the number of particles required for the research. 60

CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION

97%	MATCHING BLOCK 47/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

The limitation within in this however, is that this research piece was only conducted with one speci c environment. Using a machine with a greater level of RAM or a higher quality CPU may produce di erent results. Additionally, di erent languages may have di erent OTP's, even with the same tness functions. Finally, while these results strongly point toward 500,000 thousand particles as being the OTP, this is really only relevant for functions with a similar complexity to the three chosen for this study. Using a more complex or signi cantly simpler tness function will likely result it a di erent OTP than the one found in this study. As mentioned in the Results discussion, to reach OTP, an excessive amount of particles was required. In the normal usage of PSO, 1000 particles or less would be more than adequate to nd the global minimum of a tness function. The only advantage of using such extreme levels of particles is to reduce the number of iterations required to reach a solution. 5.4 Conclusion This Chapter covered in-depth the results obtained from running the experiment de- scribed in chapters three and four. Each implementations results were presented, sub- divided by the individual tness function tested. Commentary was provided on these results, with some notable trends highlighted. Distributed implementation results and centralised results were also compared together, highlighting some interesting trends within the results, including a tendency for centralised TTS results to increase expo- nentially, whereas the distributed TTS results appeared to increase at a more linear rate. The research question was answered, and the alternate hypothesis was accepted, in that there is a point whereby a distributed PSO implementation becomes more e cient than a centralised implementation. It was noted that across both implemen- tations there was a general trend of decreasing the total number of epochs required to reach a solution as the number of particles increased. The next Chapter will conclude this research, with an overview of what has been accomplished in this experiment, evaluation of the results found from the experiment, some proposed future work and recommendations will be discussed. 61 Chapter 6 Conclusion This Chapter summarises the entire research project, reviewing the research objective and presents the answer to the research question. Finally, the contributions, impact, future work and recommendations are presented and discussed. 6.1 Research Overview This study aimed to nd empirical evidence that there was a point whereby it becomes more e cient to run a particle swarm optimisation algorithm in a distributed manner over a centralised implementation. Chapter One introduced the research topic and some basic overview of the research piece. Chapter Two presented research into the PSO topic, and identi ed a gap in the literature around PSO, that being that no existing literature focused on the point where a distributed PSO implementation is more time e cient than a centralised implementation. This gap was used to form the basis of this research topic. Chapter Three and Four discussed the experiment design and implementation, with some details provided on how to run the experiment and how the resulting data set was obtained. Finally, Chapter Five concluded with an overview of the results obtained and some interesting trends seen within the results data set. 62 CHAPTER 6. CONCLUSION 6.2 Problem De nition The aim of the research was to nd a point where a distributed particle implementa- tion became more time e cient than a centralised implementation, once the number of swarms or particles hits a particular level. This formed the research question "At what point are performance gains in running a particle swarm optimisation algorithm in a distributed environment outweighed by the time lost in network communications between multiple swarms?". From this research question the following research objec- tives were formed. 1. Create a distributed and centralised implementation of the PSO 2. Generate a result set for the centralised implementation, with varying inputs (example: di erent evaluation function, number of swarms and or particles etc.). Average out results with the same inputs 3. Generate a result set for the distributed implementation, with varying inputs (examples: di erent evaluation function, number of swarms and or particles etc.). Average out results with the same inputs 4. Cross comparison between the two result sets. Identify points at which dis- tributed had a faster response time, or other values that would make it preferable to a centralised implementation 5. Identify any limiting factors, signi cant points of interest in the data and identify future research Stemming from the research question the null and alternate hypothesis were formed. Null Hypothesis H 0 : The time taken to nd a stabilised result from using a distributed network of swarms never exceeds the time lost from the network communications needed to update swarms of each other's presence and activities. 63 CHAPTER 6. CONCLUSION Alternate Hypothesis H 1 : The increased speed to solution time from using a distributed network of swarms exceeds the time lost from the network communications needed to update swarms of each other's presence and activities once the number of particles and/or swarms has reached a su cient level. 6.3 Design/Experimentation, Evaluation & Re- sults For this research piece two PSO implementations were created, one for a centralised approach, and one for a distributed approach. Both implementations were deployed into a cloud container, with all nodes within the distributed implementation matching the centralised implementation. As both implementations were designed with a cloud environment model in mind, a REST endpoint was created to trigger running swarms, based on passed in con guration options. Options that could be passed in included: number of swarms, number of particles, social weighting, inertia weighting, cognitive weighting and tness function type. Utilising this REST endpoint a series of tests where created to generate the required output data set. Each test was repeated three times, with the same con guration options passed in, allowing the results to be aver- aged across the three runs. Once all the results had been aggregated and averaged, a general results le was created in a CSV format. This format worked quite well, with failure rarely happening, and when they did, the program was designed to fail gracefully, without returning malformed results. Utilising REST connections between distributed nodes allowed for rapid development and testing, however it may have added additional time overheads to the nal TTS result. Additionally, even after averaging the results across multiple runs, there still appeared to be some outliers in the results, most notable in the results for the number of iterations taken to reach a solution. When reviewing the results, it was clearly observable that there was in fact, a point 64 CHAPTER 6. CONCLUSION where a distributed swarm implementation outperformed a centralised implementation in terms of TTS. This does however only occur when the number of particles reaches an incredibly high OTP. The results pointed to

the OTP

94%	MATCHING BLOCK 48/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

being around 500,000 particles, with this being roughly the same across three di erent tness functions. We can therefore con rm that the alternate hypothesis is correct, and there is a point whereby the TTS when using a distributed network of swarms exceeds the time lost from the network communications. Indeed, at the highest level of particles used in the experiment, 1,000,000 particles, the distributed implementation displayed a greater than 100% reduction in TTS when compared to the centralised implementation 6.4 Contributions and Impact The main contribution of this research piece is to ll in the research gap posed in Chapter Two. This research piece focused in on benchmarking two PSO implementa- tions, a distributed and centralised implementation, in order to asses if there was an OTP whereby a distributed implementation is more e cient than a centralised imple- mentation. Additionally, an original distributed and centralised PSO implementation was developed for this project that could aid in future work. Where this could have the most impact is for within a more practical experiment of a PSO, where very high levels of particles are required, possibly leaning towards more of a simulation problem than an optimisation problem. Chapter Two presented and cited many forms of robotics or simulated robotic PSO experiments, this research piece may help future simulated robotic experiments by highlighting the threshold where it becomes more time e cient to run a distributed im- plementation over a centralised implementation. Researchers may be able to

pre-empt

99%	MATCHING BLOCK 49/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

the need to distribute their simulated PSO implementation, rather than benchmarking their system themselves. 65 CHAPTER 6. CONCLUSION 6.5 Future Work & Recommendations • This work could be extended by adapting the implementation into alternate languages or frameworks. It was mentioned previously that utilising REST end- points may have incurred additional overheads when calculating the TTS for the distributed implementations. Utilising a di erent network communication method may nd additional gains to TTS than highlighted in this report. Addi- tionally, implementing the code in a di erent language may provide a di erent OTP than the one noted in this study. • Testing more tness functions to see if the OTP remains true. Additionally some more complex simulation problems could be tested to nd variations in the OTP. • Running the same experiment against more powerful machines, in terms of RAM and CPU may highlight more trends in how performant a distributed PSO im- plementations is. 66 References Abraham, A., Guo, H., & Liu, H. (2006). Swarm intelligence: Foundations, perspec- tives and applications. Swarm Intelligent Systems Studies in Computational Intelli- gence, 3{25. doi: 10.1007/978-3-540-33869-7 1 Akat, S. B., & Gazi, V. (2008). Decentralized asynchronous particle swarm optimiza- tion. 2008 IEEE Swarm Intelligence Symposium. doi: 10.1109/sis.2008.4668304 Azab, S. S., Hady, M. F., & Hefny, H. A. (2016). Local best particle swarm optimiza- tion for partitioning data clustering. 2016 12th International Computer Engineering Conference (ICENCO). doi: 10.1109/icenco.2016.7856443 Bai, Q. (2010). Analysis of particle swarm optimization algorithm. Computer and Information Science, 3(1). doi: 10.5539/cis.v3n1p180 Bassimir, B., Schmitt, M., & Wanka, R. (2020). Self-adaptive potential-based stop- ping criteria for particle swarm optimization with forced moves. Swarm Intelligence, 14(4), 285{311. Beni, G. (2005). From swarm intelligence to swarm robotics. Swarm Robotics Lecture Notes in Computer Science, 1{9. doi: 10.1007/978-3-540-30552-1 1 Bingham, D. (n.d.). Virtual library of simulation experiments:. Retrieved from https://www.sfu.ca/ ~ ssurjano/beale.html Blum, C., & Li, X. (n.d.). Swarm intelligence in optimization. Natural Computing Series Swarm Intelligence, 43{85. doi: 10.1007/978-3-540-74089-6 2 67 REFERENCES Bouchrika, I. (2020, Aug). Primary research vs secondary research: De nitions, di erences, and examples. Retrieved from https://www.guide2research.com/ research/primary-research-vs-secondary-research Cooren, Y., Clerc, M., & Siarry, P. (2009). Performance evaluation of tribes, an adaptive particle swarm optimization algorithm. Swarm Intelligence, 3(2), 149{178. He, Y., Ma, W. J., & Zhang, J. P. (2016). The parameters selection of pso algorithm in uencing on performance of fault diagnosis. In Matec web of conferences (Vol. 63, p. 02019). Hereford, J. (2006). A distributed particle swarm optimization algorithm for swarm robotic applications. 2006 IEEE International Conference on Evolutionary Compu- tation. doi: 10.1109/cec.2006.1688510 Imranª, M., Hashima, R., & Abd Khalidb, N. E. (2013). An overview of particle swarm optimization variants. Procedia Engineering, 53, 491{496. Jetmarov a, B. (2011). Comparison of best practice benchmarking models. Problems of Management in the 21st Century, 2, 76. Kan, W., & Jihong, S. (2012). The convergence basis of particle swarm optimization. In 2012 international conference on industrial control and electronics engineering (pp. 63{66). Kennedy, J. (1999). Small worlds and mega-minds: e ects of neighborhood topology on particle swarm performance. Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). doi: 10.1109/cec.1999.785509 Kennedy, J., & Eberhart, R. (1995a). Particle swarm optimization. IEEE Interna- tional Conference on Neural Networks (ICNN). Kennedy, J., & Eberhart, R. (1995b). Particle swarm optimization. In Proceedings of icnn'95-international conference on neural networks (Vol. 4, pp. 1942{1948). 68 REFERENCES Kennedy, J., & Mendes, R. (2002). Population structure and particle swarm perfor- mance. In Proceedings of the 2002 congress on evolutionary computation. cec'02 (cat. no. 02th8600) (Vol. 2, pp. 1671{1676). Li, H., Yang, D., Su, W., Lu, J., & Yu, X. (2019). An overall distribution particle swarm optimization mppt algorithm for photovoltaic system under partial shading. IEEE Transactions on Industrial Electronics, 66(1), 265{275. doi: 10.1109/tie.2018 .2829668 Lynn, N., Ali, M. Z., & Suganthan, P. N. (2018). Population topologies for par- ticle swarm optimization and di erential evolution. Swarm and Evolutionary Com- putation, 39, 24-35. Retrieved from https://www.sciencedirect.com/science/ article/pii/S2210650217308805 doi: https://doi.org/10.1016/j.swevo.2017.11 .002 McLeod, S. (2019). Qualitative vs quantitative research: Simply psychology. Retrieved from https://www.simplypsychology.org/qualitative-quantitative.html Meng, Y., & Gan, J. (2008). A distributed swarm intelligence based algorithm for a cooperative multi-robot construction task. 2008 IEEE Swarm Intelligence Sympo- sium. doi: 10.1109/sis.2008.4668296 Moradi, A., & Fotuhi-Firuzabad, M. (2008). Optimal switch placement in distribution systems using trinary particle swarm optimization algorithm. IEEE Transactions on Power Delivery, 23(1), 271{279. doi: 10.1109/tpwrd.2007.905428 Nouiri, M., Bekrar, A., Jemai, A., Niar, S., & Ammari, A. C. (2015). An e ective and distributed particle swarm optimization algorithm for exible job-shop scheduling problem. Journal of Intelligent Manufacturing, 29(3), 603{615. doi: 10.1007/s10845 -015-1039-3 Oca, M. M. D., Stutzle, T., Birattari, M., & Dorigo, M. (2009). Frankenstein's pso: A composite particle swarm optimization algorithm. IEEE Transactions on Evolutionary Computation, 13(5), 1120{1132. doi: 10.1109/tevc.2009.2021465 69

REFERENCES Peleg, D. (2005). Distributed coordination algorithms for mobile robot swarms: New directions and challenges. Distributed Computing { IWDC 2005 Lecture Notes in Computer Science, 1{12. doi: 10.1007/11603771 1 Piotrowski, A. P., Napiorkowski, J. J., & Piotrowska, A. E. (2020). Population size in particle swarm optimization. Swarm and Evolutionary Computation, 58, 100718. doi: 10.1016/j.swevo.2020.100718 Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. Swarm intelligence, 1(1), 33{57. Raquel, C. R., & Naval Jr, P. C. (2005). An e ective use of crowding distance in mul- tiobjective particle swarm optimization. In Proceedings of the 7th annual conference on genetic and evolutionary computation (pp. 257{264). Sahin, F., Yavuz, M. C ., Arnavut, Z., & Uluyol, O. (2007). Fault diagnosis for airplane engines using bayesian networks and distributed particle swarm optimization. Parallel Computing, 33(2), 124{143. Salza, P., & Ferrucci, F. (2019). Speed up genetic algorithms in the cloud using software containers. Future Generation Computer Systems, 92, 276{289. doi: 10 .1016/j.future.2018.09.066 Schutte, J. F., Reinbolt, J. A., Fregly, B. J., Haftka, R. T., & George, A. D. (2004). Parallel global optimization with the particle swarm algorithm. International journal for numerical methods in engineering, 61(13), 2296{2315. Shi, Y. (2004, Feb). Particle swarm optimization. IEEE Neural Networks Society. Shi, Y., & Eberhart, R. C. (1998). Parameter selection in particle swarm optimiza- tion. In International conference on evolutionary programming (pp. 591{600). S a, A. O. D., Nedjah, N., & Mourelle, L. D. M. (2016). Distributed e cient localiza- tion in swarm robotic systems using swarm intelligence algorithms. Neurocomputing, 172, 322{336. doi: 10.1016/j.neucom.2015.03.099 70 REFERENCES Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. Information Processing Letters, 85(6), 317{325. doi: 10 .1016/s0020-0190(02)00447-7 Vandenbergh, F., & Engelbrecht, A. (2004). A cooperative approach to parti- cle swarm optimization. IEEE Transactions on Evolutionary Computation, 8(3), 225{239. doi: 10.1109/tevc.2004.826069 Vokolos, F. I., & Weyuker, E. J. (1998). Performance testing of software systems. In Proceedings of the 1st international workshop on software and performance (pp. 80{87). Wang, D., Tan, D., & Liu, L. (2017). Particle swarm optimization algorithm: an overview. Soft Computing, 22(2), 387{408. doi: 10.1007/s00500-016-2474-6 Wang, Z.-J., Zhan, Z.-H., Kwong, S., Jin, H., & Zhang, J. (2020). Adaptive gran- ularity learning distributed particle swarm optimization for large-scale optimization. IEEE transactions on cybernetics. Xia, X., Gui, L., & Zhan, Z.-H. (2018). A multi-swarm particle swarm optimization algorithm based on dynamical topology and purposeful detecting. Applied Soft Com- puting, 67, 126-140. Retrieved from https://www.sciencedirect.com/science/ article/pii/S1568494618301017 doi: https://doi.org/10.1016/j.asoc.2018.02.042 Xie, X.-f., Zhang, W.-j., & Yang, Z.-l. (2003). Overview of particle swarm optimiza- tion. Control and Decision, 18(2), 129{134. Yin, P.-Y., Glover, F., Laguna, M., & Zhu, J.-X. (2010). Cyber swarm algorithms{ improving particle swarm optimization using adaptive memory strategies. European Journal of Operational Research, 201(2), 377{389. Yin, P.-Y., Yu, S.-S., Wang, P.-P., & Wang, Y.-T. (2006). A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. Computer Standards & Interfaces, 28(4), 441{450. doi: 10.1016/j.csi.2005.03.005 71 REFERENCES Zhang, W., Ma, D., jun Wei, J., & feng Liang, H. (2014). A parameter selection strategy for particle swarm optimization based on particle positions. Expert Systems with Applications, 41(7), 3576-3584. Retrieved from https://www.sciencedirect .com/science/article/pii/S0957417413008932 doi: https://doi.org/10.1016/j .eswa.2013.10.061 Zheng, J., Wu, Q., & Song, W. (2007). An improved particle swarm algorithm for solving nonlinear constrained optimization problems. In Third international confer- ence on natural computation (icnc 2007) (Vol. 4, pp. 112{117). Zhou, Y., & Tan, Y. (2009). Gpu-based parallel particle swarm optimization. In 2009 ieee congress on evolutionary computation (pp. 1493{1500). Zielinski, K., & Laur, R. (2007). Stopping criteria for a constrained single-objective particle swarm optimization algorithm. Informatica, 31(1). 72 Appendix A Code Snippets Listing A.1 shows and example data output produced for the booths tness function from a distributed and centralised PSO implementation. 1 { 2 "finalGroupBest": 0.0, 3 "averageBestX": 1.0000000000000003, 4 "averageBestY": 3.0, 5 "iterations": 151, 6 "timeToSolution": 1303, 7 "resultsList": [{ 8 "swarmId": 0, 9 "results": [{ 10 "step": 0, 11 "best": 20.288124656313526, 12 "bestPositionX": 0.0, 13 "bestPositionY": 0.0 14 }] 15 }] 16 } Listing A.1: Data output example 73 APPENDIX A. CODE SNIPPETS Listing A.2 shows an extract of the data aggregation code. Code to retrieve the results from the le system, and aggregate and average them is visible. Code to output the resulting text string is omitted, but is a straightforward process of creating a text le using the variable outputString. 1 var outputString = "SWARM NUMBER,PARTICLE NUMBER,CENTRALISED ITERATIONS, ,! CENTRALISED TIME TO SOLUTION,CENTRALISED FINAL GROUP BEST, ,! DISTRIBUTED ITERATIONS,DISTRIBUTED TIME TO SOLUTION, ,! DISTRIBUTED FINAL GROUP BESTnn"; 2 3 for (var swarmNumber = 1; swarmNumber >= numberOfSwarms; swarmNumber++) f 4 for (var particleNumber = 1000; particleNumber >= maxNumberParticles; particleNumber ,! += particleIncrement) f 5 var leUrlCentralised RUN1 = './Results/NON DISTRIB/RUN 1/' + functionName + '/' ,! + functionName + ' ' + swarmNumber + ' ' + particleNumber + '.result.json'; 6 .... 7 var leUrlDistributed RUN1 = './Results/DISTRIB/RUN 1/' + functionName + '/' + ,! functionName + ' ' + swarmNumber + ' ' + particleNumber + '.result.json'; 8 .... 9 let centralised RUN1 = fs.readFileSync(leUrlCentralised RUN1); 10 ... 11 let centralised data RUN1 = JSON.parse(centralised RUN1); 12 ... 13 var averageCentralisedIterations = Math.round((centralised data RUN1.iterations + ,! centralised data RUN2.iterations + centralised data RUN3.iterations)/3);

14 var averageCentralisedTTS = Math.round((centralised data RUN1.timeToSolution + ,! centralised data RUN2.timeToSolution + centralised data RUN3.timeToSolution) ,! /3); 15 var averageCentralisedGbest = Math.round((centralised data RUN1. nalGroupBest + ,! centralised data RUN2. nalGroupBest + centralised data RUN3. nalGroupBest) ,! /3); 16 ... 17 18 outputString = outputString + swarmNumber + "," + particleNumber + "," + ,! averageCentralisedIterations + "," + averageCentralisedTTS + "," + ,! averageCentralisedGbest + ","; 74 APPENDIX A. CODE SNIPPETS 19 ..... 20 /∗ 21 Generate the same output string for distributed test run data 22 ∗/ 23 g 24 /∗ 25 Write outputString to le system. 26 ∗/ Listing A.2: Data aggregation Implementation Listing A.3 shows an extract of the Newman API implementation code. Code to load the request le and environment variables can be seen, along with the code to write the result to the le system the format discussed in Chapter Three. 1 function runNewmanCode(envName, testCollectionName, environmentName, numberOfSwarms, ,! numberOfParticles, baseUrlsSting, endpoint) f 2 newman.run(f 3 collection: require('./Requests/' + testCollectionName + '.postman collection.json'), 4 environment: require('./EnvVariables/' + environmentName + '.postman environment.json ,! '), 5 reporters: 'cli', 6 globalVar: [f "key":"NumberOfSwarms", "value":numberOfSwarms g, f"key":"endpoint", " ,! value": endpointg, f "key":"NumberOfParticles", "value": numberOfParticlesg, f " ,! key":"BaseUrls", "value": baseUrlsStingg] 7 g).on('request', function (error, args) f 8 ... 9 else f 10 ... 11 fs.writeFile(dir + testCollectionName + " " + numberOfSwarms + " " + ,! numberOfParticles + '.result.json', args.response.stream, function (error) f 12 ... 13 g); 14 g 15 g); 16 g Listing A.3: Newman Implementation Code 75 APPENDIX A. CODE SNIPPETS Listing A.4 shows a partial extract of the distributed implementation service class. The swarm looping code can be seen, along with the stopping criteria code and the response formatting methods. 1 public Response runDistributedCallingSystem(int numberOfParticles, List>String< baseUrls, ,! Con gVariables con gVariables) f 2 Instant start = Instant.now(); 3 4 for(int swarmId=0; swarmId > numberOfSwarms; swarmId++)f 5 ... 6 callInitialiseSwarms(swarmId, baseUrls.get(swarmId), distributedRequest); 7 g 8 9 ... 10 11 while(!targetFound) f 12 double[] bestEvaluation = new double[numberOfSwarms]; 13 for(int swarmId = 0; swarmId>numberOfSwarms; swarmId++) f 14 DistributedRequest distributedRequest = new DistributedRequest(); 15 distributedRequest.setNumberOfParticles(numberOfParticles); 16 distributedRequest.setSwarmId(swarmId); 17 distributedRequest.setCon gVariables(con gVariables); 18 Result result = callRunSwarm(swarmId, baseUrls.get(swarmId) ,distributedRequest); 19 multiSwarmResults.get(swarmId).getResults().add(result); 20 bestEvaluation[swarmId] = result.getBest(); 21 g 22 23 for(int swarmId = 0; swarmId>numberOfSwarms; swarmId++) f 24 if (settledList.size() == 0) f 25 settledList.add(bestEvaluation[swarmId]); 26 g else f 27 if (settledList.contains(bestEvaluation[swarmId])) f 28 settledList.add(bestEvaluation[swarmId]); 29 g else f 30 settledList = new ArrayList><(); 31 g 32 g 76 APPENDIX A. CODE SNIPPETS 33 34 if (settledList.size() == TARGET NUMBER OF EXACT ANSWERS) f 35 ... 36 targetFound = true; 37 g 38 g 39 if(stepCount == MAX NUMBER OF STEPS)f 40 targetFound =true; 41 g 42 43 stepCount++; 44 g 45 46 Instant nish = Instant.now(); 47 long timeElapsed = Duration.between(start, nish).toMillis(); 48 49 return utils.createResponse(multiSwarmResults, stepCount, 50 timeElapsed, numberOfSwarms); 51 g Listing A.4: Distributed Service Class Listing A.5 shows a partial extract of the centralised implementation service class. The swarm looping code can be seen, along with the stopping criteria code. 1 public Response runSwarm(Request request) throws Exception f 2 Instant start = Instant.now(); 3 ... 4 List>Swarm< multiSwarm = new ArrayList><(); 5 ... 6 /∗ Initialize all swarms and particle with supplied con guration ∗/ 7 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 8 multiSwarm.add(utils.createSwarm(swarmId, request.getNumberOfParticles(), request. ,! getCon gVariables())); 9 g 10 11 while(!targetFound) f 12 ... 77 APPENDIX A. CODE SNIPPETS 13 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 14 Result result = utils.runSwarmIteration(multiSwarm.get(swarmId), stepCount); 15 ... 16 g 17 18 /∗ Check if the same answer has been found multiple times ∗/ 19 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); 20 swarmId++) f 21 if (settledList.size() == 0) f 22 settledList.add(bestEvaluation[swarmId]); 23 g else f 24 if (settledList.contains(bestEvaluation[swarmId])) f 25 settledList.add(bestEvaluation[swarmId]); 26 g else f 27 settledList = new ArrayList><(); 28 g 29 g 30 31 if (settledList.size() == TARGET NUMBER OF EXACT ANSWERS) f 32 TARGET NUMBER OF EXACT ANSWERS); 33 targetFound = true; 34 g 35 g 36 37 if(stepCount == MAX NUMBER OF STEPS)f 38 targetFound =true; 39 g 40 41 stepCount++; 42 g 43 Instant nish = Instant.now(); 44 long timeElapsed = Duration.between(start, nish).toMillis(); 45 ... 46 g Listing A.5: Centralised Service Class 78 Appendix B Data Results Table B.1 shows the full averaged result set for the centralised Easom's function TTS. Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 2364 2297 2313 2388 200000 4578 4449 4313 4783 300000 6681 6731 6694 8033 400000 9776 9779 9749 11264 500000 12875 13835 13176 16012 600000 19400 18945 18825 21732 700000 24188 23920 23905 27103 800000 28344 27520 30373 33907 900000 36510 38057 36939 47329 1000000 50887 51263 52918 78160 Table B.1: Centralised Easom'

s

93%	MATCHING BLOCK 50/50	SA	C13358011_MSC_ASD.final.pdf (D108754758)

Results Table B.2 shows the full averaged result set for the centralised Booth's function TTS. 79 APPENDIX B. DATA RESULTS Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 2910 3243 3634 2805 200000 6160 6802 6481 5986 300000 9857 10398 10425 10185 400000 14798 15180 15161 15400 500000 21596 21013 21246 23378 600000 34828 33524 34117 36270 700000 43639 42629 42271 45328 800000 52655 51940 55802 58774 900000 70949 70174 69936 83288 1000000 102709 103364 102856 131270 Table B.2: Centralised Booth's TTS Table B.3 shows the full averaged result set for the centralised Beale's function TTS. Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 4205 3842 4775 4417 200000 8828 8676 9276 9294 300000 14003 13934 14489 14964 400000 19742 19301 20336 22167 500000 27418 27981 27615 33411 600000 39291 40426 40594 45841 700000 52039 51825 49939 56124 800000 60327 61134 65653 70674 900000 78404 81726 79632 98879 1000000 112258 119800 112751 160899 Table B.3: Centralised Beale's TTS Table B.4 shows the full averaged result set for the distributed Easom's function TTS. 80 APPENDIX B. DATA RESULTS Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 3551 3287 3925 4385 200000 2545 5154 6217 5449 300000 7217 7583 8228 8556 400000 9145 7434 9961 11039 500000 12404 11663 12298 13091 600000 14614 14158 14661 15548 700000 17662 16690 17383 17736 800000 20410 19008 14769 20474 900000 24803 21444 22707 22619 1000000 29327 23923 24623 26300 Table B.4: Distributed Easom's TTS Table B.5 shows the full averaged result set for the distributed Booth's function TTS. Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 4331 4846 6197 7541 200000 4092 8195 8942 9957 300000 11497 11190 11917 15068 400000 15046 11951 14992 19667 500000 19193 19601 18699 24308 600000 23894 22650 22266 28513 700000 27431 28039 26392 33023 800000 33195 31968 21680 38411 900000 39787 35334 34058 43588 1000000 48322 39878 36861 49330 Table B.5: Distributed Booth's TTS Table B.6 shows the full averaged result set for the distributed Beale's function TTS. 81 APPENDIX B. DATA RESULTS Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 5291 5630 7153 7860 200000 7370 10308 10988 10223 300000 13213 13959 15164 16299 400000 18240 19389 20080 20567 500000 23625 24069 24213 25084 600000 29258 27959 28980 29629 700000 35314 33992 33915 33713 800000 40600 39246 39852 39174 900000 49349 45128 44175 44859 1000000 56502 48906 49696 51248 Table B.6: Distributed Beale's TTS 82

Hit and source - focused comparison, Side by Side

Submitted text	As student entered the text in the submitted document.
Matching text	As the text appears in the source.

1/50	SUBMITTED TEXT	4839 WORDS	96%	MATCHING TEXT	4839 WORDS

Performance comparison between a distributed particle swarm algorithm and a centralised algorithm Ciar an O'Loughlin A dissertation submitted in partial ful lment of the requirements of Technological University Dublin for the degree of M.Sc. in Computing (TU060) Date Declaration I certify that this dissertation which I now submit for examination for the award of MSc in Computing (TU060), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work. This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and has not been submitted in whole or part for an award in any other Institute or University. The work reported on in this dissertation conforms to the principles and requirements of the Institute's guidelines for ethics in research. Signed: Ciar an O'Loughlin Date: 13 th June 2021 I Abstract Particle Swarm optimisation (PSO) is a particular form of swarm intelligence, which itself is an innovative intelligent paradigm for solving optimization problems. PSO is generally used to nd a global optimum in a single optimisation function. This typically occurs on one node(machine) but there has been a signi cant body of research into creating distributed implementations of the PSO algorithm. Such research has often focused on the creation and performance of the distributed implementation in an isolated manner or compared to di erent distributed algorithms. This research piece aims to bridge a gap in the existing literature, by testing a distributed implementation of a PSO algorithm against a centralised implementation, and investigating what, if any, gains there are to utilising a distributed implementation over a centralised implementation. The focus will primarily be on the time taken for the algorithm to successfully nd a global minimum to a speci c tness function, but other elements will be examined over the course of the study. Keywords: Swarm Intelligence, Particle Swarm Optimisation, PSO, Distributed Algorithms II Acknowledgments I would like to express my sincere thanks to my supervisor Dr. Edina Hatunic Webster, whose guidance and support helped me throughout my dissertation. A special thanks to my parents Eileen & Cormac O'L., my friends, colleagues and family. Without all their encouragement and support this dissertation would never have been possible. III Contents Declaration I Abstract II Acknowledgments III Contents IV List of Figures VIII List of Tables X List of Acronyms XII 1 Introduction 1 1.1 Background . . . . 1 1.2 Research Project/problem . . . . 2 1.3 Research Objectives . . . . 2 1.4 Research Methodologies . . . .

List of Acronyms PSO Particle Swarm
Algorithm LTS Long Term Support CPU Computer
processor unit RAM Random Access Memory TTS Time To
Solution CSV Comma Separated Values JSON JavaScript
Object Notation OTP Optimisation Tipping Point XII
Chapter 1 Introduction 1.1 Background Swarm Intelligence
(SI) is an innovative distributed intelligent paradigm for
solving optimization problems that originally took its
inspiration from the biological exam- ples by swarming,
ocking and herding phenomena in vertebrates (Abraham,
Guo, & Liu, 2006). Within the boundaries of swarm
intelligence there are many di er- ent algorithms, all with
di erent uses and capabilities. One particular algorithm is
the particle swarm optimisation algorithm (PSO). PSO is a
population-based search algorithm and is initialized with a
population of random solutions, called particles (Shi,
2004). Particles are then arranged into a "swarm". Swarms
allow the sharing of information between particles, the
so-called "social" element of the algorithm. PSO as an
algorithm was rst proposed by James Kennedy and
Russell Eberhart in their 1995 paper "Particle Swarm
Optimization" (Kennedy & Eberhart, 1995a). While not a
new technique, PSO is still being expanded upon today,
with modern uses ranging from simple algorithmic
evaluation, to more complex robotics implementations.
This is the basis of a single swarm PSO. It is possible to
increase the number of particles in a swarm, but we can
also increase the number of swarms. A multi-swarm PSO
(Or a Cooperative Particle Swarm Optimisation algorithm,
CPSO) model allows for "a signi cant increases in the
solution diversity in CPSO-S algorithm, because of the
many di erent members from di erent swarms"
(Vandenbergh & Engelbrecht, 1 CHAPTER 1.

INTRODUCTION 2004). Even when using a multi-swarm model, there will be hard limits on the number of particles and swarms a singular machine can generate. This is where a distributed model can help, with swarms being logically segregated across a network of machines. 1.2 Research Project/problem There is an upper limit to the amount of particles any one system can support in a PSO algorithm. At a certain point the algorithm will slow down and the time taken for it to complete an iteration will drastically increase. This will be exacerbated by running multiple swarms. To alleviate this problem, we can distribute the swarms onto di erent machines, which allows us to increase the total number of swarms and parti- cles available to us. However, this has a disadvantage as to evaluate results generated by the swarms, network communications will need to be established. Network connec- tions are inherently slower than connections and data transfers on a local machine, so the question this report aims to answers is: "At what point are performance gains in running a particle swarm optimisation algorithm in a distributed environment outweighed by the time lost in network com- munications between multiple swarms?" 1.3 Research Objectives The key objective of the research is to identify whether there exists a point whereby it is more e cient to run a particle swarm optimisation algorithm in a distributed manner over a centralised manner. To answer this question the following research objectives where identi ed: 1. Create a distributed and a centralised implementation of the PSO 2. Generate a result set for the centralised implementation, with varying inputs (example; di erent evaluation function, number of swarms and or particles etc.). Average out results with the same inputs 2 CHAPTER 1. INTRODUCTION 3. Generate a result set for the distributed implementation, with varying inputs (examples; di erent evaluation function, number of swarms and or particles etc.). Average out results with the same inputs 4. Cross comparison between the two results sets. Identify points at which dis- tributed had a faster response time, or other values that would make it preferable to a centralised implementation 5. Identify any limiting factors, signi cant points of interest in the data and identify future research 1.4 Research Methodologies The research can be classi ed based in a few di erent ways - 1.4.1 Based on Type: Primary vs Secondary Primary research refers to a collection of original data speci c to a particular re- search question generated during the project. When doing primary research, the researcher gathers information rst-hand rather than relying on available information in databases and other publications (Bouchrika, 2020). Secondary research instead focuses on collecting and summarizing existing data collections and results. This involves researching existing literature, published articles and analyzing the data produced from these articles to come to a new conclusion, or test a hypothesis. When doing secondary research, researchers use and analyze data from primary research sources (Bouchrika, 2020). The research type for this project can be de ned as primary research. Data sets needed to answer the research question will be generated during the course of the project. The data set will be unique, as no other study has sought to compare imple- mentation types of particle swarm optimisation algorithms. 3 CHAPTER 1. INTRODUCTION 1.4.2 Based on Objective: Quantitative vs Qualitative Quantitative research methods refer to collecting numerical data, data that can be used to measure variables, predict outcomes etc. Quantitative data is structured and statistical, using a grounded theory method that relies on data collection that is systematically analyzed. Quantitative research is a methodology that provides support when you need to draw general conclusions from your research and predict outcomes (McLeod, 2019). Qualitative research is fundamentally opposite to Quantitative research, as it relies on non-statistical and unstructured or semi-structured data. It is a methodology designed to collect non-numerical data to gain insights. It relies on data collected based on a research design that answers the question \why." 1 The research objective of this project can be de ned as quantitative research. This study will generate and examine structured data sets, comparing two particle swarm optimisation implementations, and drawing conclusions from those comparisons. 1.4.3 Based on Form: Exploratory vs Constructive vs Empir- ical Exploratory research refers to when researching a problem which has not been clearly de ned. Through exploratory research we can determine the best research design and data collection method. When conducting constructive research, a completely new approach, theory or model will be proposed. Constructive research adds a new contribution to the current body of research. Empirical research is a way of obtaining knowledge through direct observation or experience. Empirical research involves a process of de ning a hypothesis and then making predictions that can be tested using a suitable scienti c experiment. The study can be de ned as an empirical study. This study will de ne its hypoth- esis, test that hypothesis, examine the results from tests, and draw conclusions/pre- dictions from the data. Based on that data the hypothesis can then be accepted or 1 https://www.surveymonkey.com/mp/quantitative-vs-qualitative-research/ 4 CHAPTER 1. INTRODUCTION rejected, thus concluding the study. 1.4.4 Based on Reasoning: Deductive vs Inductive Deductive reasoning is a logical form of reasoning. Deductive begins with a general statement,

SA C13358011_MSC_ASD.final.pdf (D108754758)

2/50	SUBMITTED TEXT	10 WORDS	100%	MATCHING TEXT	10 WORDS

Inductive reasoning is the opposite of deductive reasoning. Inductive reasoning

SA C13358011_MSC_ASD.final.pdf (D108754758)

3/50	SUBMITTED TEXT	202 WORDS	100%	MATCHING TEXT	202 WORDS

speci c observations. Basically, there is data, then conclusions are drawn from the data 2 For this study deductive reasoning will be used. The study will state a hypothesis, and attempt to validate that hypothesis though testing, generating data sets and drawing conclusions from those data sets. 1.5 Scope and Limitations The scope of this research is to identify potential points where a distributed imple- mentation of the particle swarm optimisation algorithm is faster than a centralised implementation. There are two main limiting factors to this research. The rst being that the implementations are only tested against speci c tness functions. There are many di erent tness functions, or optimisation problems, that can be applied to a particle swarm optimisation algorithm, and many of them will have di erent e ciency points. Some may gain a bene t from being distributed at di erent levels of swarms and particles, and others may not ever bene t from being distributed. For this reason, three di erent algorithms were chosen and tested against the two implementations. The second limiting factor is an environmental one. Computers with better qual- ity CPU (Computer Processor Unit) and a higher amount of RAM (Random Access Memory) will be able to support more swarms and particles. In order to account for 2

SA C13358011_MSC_ASD.final.pdf (D108754758)

4/50	SUBMITTED TEXT	44 WORDS	82%	MATCHING TEXT	44 WORDS

this the same machine type will be used across the distributed and centralised im- plementations. How this will be achieved is discussed in Chapter Three, Design and Implementation Overview. 1.6 Document Outline Chapter One Chapter One is a short introduction and few reasons why this research was conducted. Chapter

SA MSc__ASD_Dissertation(2).pdf (D106726158)

5/50	SUBMITTED TEXT	15 WORDS	90%	MATCHING TEXT	15 WORDS

Chapter Two involves a comprehensive literature review. Some gaps in the existing research are discussed. Chapter

SA MSc__ASD_Dissertation(2).pdf (D106726158)

6/50	SUBMITTED TEXT	56 WORDS	92%	MATCHING TEXT	56 WORDS

Chapter Three includes a detailed design of the experiment as well as the methodology behind it. The design of the testing systems and some implementation overviews are presented. Chapter Four Chapter Four contains a full discussion of the implementation details. Coded examples are provided to add further clarity to how the experiment was built. Environment and test implementations are also discussed. Chapter

SA MSc__ASD_Dissertation(2).pdf (D106726158)

7/50	SUBMITTED TEXT	22 WORDS	78%	MATCHING TEXT	22 WORDS

Chapter Five presents the results obtained from running the experiment. Some obser- vation and discussion is presented around these results. 6 CHAPTER 1.

SA MSc__ASD_Dissertation(2).pdf (D106726158)

8/50	SUBMITTED TEXT	558 WORDS	98%	MATCHING TEXT	558 WORDS

Chapter Six Chapter Six is a nal discussion and conclusion of the research project, including future suggestions on the given problem as well as recommendations regarding accuracy of the experiment and its design 7 Chapter 2 Background Research 2.1 Introduction This Chapter provides a review of the literature available on particle swarm optimisa- tion algorithms, distributed implementation, various approaches adopted to solve the problem and evaluation metrics used for evaluating the results. The Chapter concludes with the gaps in the existing research and forms the objective for the research. 2.2 PSO(Particle Swarm Optimisation) 2.2.1 What is PSO? The original PSO algorithm was introduced in 1995 (Kennedy & Eberhart, 1995b), in the paper they discussed what the PSO algorithm is, and how to use it to optimise nonlinear functions. At its heart PSO is a population-based optimization technique where the population is referred to as a swarm and population members are referred to as particles. At the algorithm's inception a number of unique particles are created and given random positions within a D-dimensional space. Each particle can be considered a potential solution to the tness function. Particles then go through iterations, or epochs, where each particle calculates its own personal best evaluation value, then the global best value is calculated. Particles are encouraged to move closer to their 8 CHAPTER 2. BACKGROUND RESEARCH own personal best or the groups based on the algorithm's implementation. A particle moving closer towards the groups best will de ne a higher "social value", and moving closer to its own personal best will de ne a higher "cognitive value". Velocity can also be changed to reduce the variance in particle positions between epochs. Figure 2.1: Flow Diagram of Particle Swarm Optimisation Algorithm (D. Wang et al., 2017) We can see from the above explanation that PSO shares many elements of other arti cial intelligence types, such as evolutionary computation, and genetic algorithms. The particles are manipulated according to the following equations: V id = W V id + C l Rand() (P id * X id) + C 2 Rand() (P gd * X id) (2.1) X id = X id + V id (2.2) Where Cl and C2 are two positive constants, Rand() is a random function in the range [0,1] and w is the inertia weight (Shi & Eberhart, 1998). A particle's new velocity can be calculated using equation 2.1, using its previous velocity and the distances of its cur- rent position from its own best experience (position) and the group's best experience. Equation 2.2 calculates the particles new position after its updated velocity. 9 CHAPTER 2. BACKGROUND RESEARCH Fitness Function When talking about PSO we must also talk about a tness function. A tness function is a function that maps the values in particles to a real value that must reward those particles that are closer to the optimisation criterion. Essentially this is the function to nd the minimum or maximum value of. For instance, if we wished to nd the global minimum of the Beale Function, an optimisation test function, we could use PSO. Beale's function can be de ned as: f(x;y) = (1:5 * x + xy) 2 + (2:25 * x + xy 2) 2 + (2:625 * x + xy 3) 2 (2.3) Using this tness function the global minimum could be found, in this case it would equate out to be f(x) = 0 at x = (3; 0:5) (Bingham, n.d.). It

SA C13358011_MSC_ASD.final.pdf (D108754758)

9/50	SUBMITTED TEXT	355 WORDS	99%	MATCHING TEXT	355 WORDS

important to note that as PSO is a heuristic algorithm, its solution results are not necessarily optimal. PSO will nd an answer to a tness function, but it may not be the best answer to the function. In addition, some problems may have multiple global minimum values, in which case the PSO can only nd one answer at a time. 2.3 PSO Implementations Since James Kennedy and Russell Eberhart original paper there has been signi cant research into the original algorithm (Piotrowski, Napiorkowski, & Piotrowska, 2020; Bai, 2010; Imranª, Hashima, & Abd Khalidb, 2013; D. Wang et al., 2017) its applica- tions (Hereford, 2006; Beni, 2005; Blum & Li, n.d.; Raquel & Naval Jr, 2005) and its performance (Yin, Yu, Wang, & Wang, 2006; Kennedy, 1999). There is also a large body of research into the use of swarm algorithms in distributed environments (Akat & Gazi, 2008; Salza & Ferrucci, 2019; Peleg, 2005) and centralised environments (Trelea, 2003; Xie, Zhang, & Yang, 2003; Poli, Kennedy, & Blackwell, 2007), which will be discussed more in the following section. PSO has also been improved upon, with some newer implementations updating the topology or some other underlying principle of the algorithm, examples including SPSO, APSO, TRIBES, Cyber Swarm etc (Zhou 10 CHAPTER 2. BACKGROUND RESEARCH & Tan, 2009; Oca, Stutzle, Birattari, & Dorigo, 2009; Cooren, Clerc, & Siarry, 2009; Yin, Glover, Laguna, & Zhu, 2010). In terms of implementing PSO there have been multiple studies into the uses of swarm intelligence in robotics systems (S a, Nedjah, & Mourelle, 2016; Meng & Gan, 2008; Hereford, 2006). Some other studies also look at interesting use cases of PSO and what swarm intelligence can be applied to, such as scheduling systems and analysis of distributed systems (Li, Yang, Su, Lu, & Yu, 2019; Moradi & Fotuhi-Firuzabad, 2008; Nouiri, Bekrar, Jemai, Niar, & Ammari, 2015; Sahin, Yavuz, Arnavut, & Uluyol, 2007). 2.3.1 Stopping Criteria When utilising the PSO algorithm, it is important to identify how and when the algorithm should terminate. The simplest form of stopping criteria is implementing a max number of iterations/epochs check. This is a simple stopping criteria however, it carries with it a risk of stopping the algorithm before a global optimum value has been found. It also must be prede ned prior to the

SA C13358011_MSC_ASD.final.pdf (D108754758)

inception, which leaves little leeway for change during runtime. An alternate to this approach is a check for "when the chance of achieving sig- ni cant improvements in further iterations is extremely low" (Bassimir, Schmitt, & Wanka, 2020). E ectively this is checking to see when the particles are no longer actively moving in the search space when they have settled at a particular answer. Within the literature there has been some research into more adaptive stopping cri- teria, for example Zielinski and Laur 2007 paper (Zielinski & Laur, 2007). In this paper a list of upper-limit-based and adaptive termination criteria is presented and tested against a real world problem, "optimizing a power allocation scheme for a Code Division Multiple Access (CDMA) system"(Zielinski & Laur, 2007). Both of these approaches o er a more dynamic and adaptive method of stopping the algorithm than the more straightforward maximum iteration check criteria. 11 CHAPTER 2. BACKGROUND RESEARCH 2.3.2 Parameter Selection As mentioned in earlier sections there are multiple input parameters for any PSO algorithm, the basic ones being inertia, cognitive and social weightings. Parameters are usually tuned for individual optimisation functions, but there are some basic selecting parameters that are normally used. When looking at the inertia weighting, according to Yan He et al. inertia should be kept within the following range w min = 0:4;w max = 0:9 (He, Ma, & Zhang, 2016). When it comes to both cognitive and social weightings, there is a little less consen- sus with some research saying both values should be equal to 2 (Kennedy & Eberhart, 1995a), and other literature mentioning that the values should range between 1 and 3 (Zhang, Ma, jun Wei, & feng Liang, 2014). It is also suggested that the sum of these constants should not exceed 3 i:es + c 3 (Kan & Jihong, 2012). What is conclusive across all the literature is that each optimisation problem will require its own ne tuning with all three values. 2.3.3 Topology adjustments Since its original inception, several adjustments have been proposed in literature to alter some implementation details around PSO. One reoccurring change is a change to the population topology or sociometry. It is well established that these factors play an important role in improving the performance of population-based optimization algo- rithms by enhancing population diversity when solving multiobjective and multimodal problems (Lynn, Ali, & Suganthan, 2018). Global version PSO (GPSO), which pri- oritises the GBest value (Global best value from all particles), and local version PSO (LPSO), which prioritises the LBest value (Local best value, which favours the best value from smaller subsections of the swarm) are two common neighbour topologies. G-PSO In the GBest population, the trajectory of each particle's search is in uenced by the best point found by any member of the entire population (Kennedy & Mendes, 2002). 12 CHAPTER 2. BACKGROUND RESEARCH What this means is that the particle uses its experience (the best position achieved so far) and uses the knowledge of the best particle in the swarm to in uence its movement strategy. It prioritises the global best value when updating its position. However, the drawback of this approach is that if the best particles are far from global optimum, the swarm can be trapped in a local optima since the swarm cannot explore other areas in the search space (Azab, Hady, & Hefny, 2016). L-PSO The LBest population allows each individual to be in uenced by some smaller number of adjacent members of the population array. Typically LBest neighbourhoods com- prise exactly two neighbours, one on each side: a ring lattice (Kennedy & Mendes, 2002). The same update function is used by particles in the two topologies, however LBest is generally considered to be the slower topology. Its advantage is that it

SA C13358011_MSC_ASD.final.pdf (D108754758)

11/50	SUBMITTED TEXT	148 WORDS	98%	MATCHING TEXT	148 WORDS

much less likely to get stuck in the local optima, and therefore nds the target value far more frequently. Figure 2.2: GBest(Left) and LBest(Right) sociometric patterns. (Kennedy & Mendes, 2002) 2.3.4 Multi-swarm PSO As mentioned previously there has been a wide body of research conducted into vari- ants of the original PSO algorithm. One such area has been in using multiple swarms with researchers nding some success with their implementations. One such imple- 13 CHAPTER 2. BACKGROUND RESEARCH mentation is the Multi-Swarm Particle Swarm Optimization (MSPSO). MSPO "is based on multiple swarms framework cooperating with the dynamic sub-swarm num- ber strategy (DNS), sub-swarm regrouping strategy (SRS), and purposeful detecting strategy (PDS)" (Xia, Gui, & Zhan, 2018). Using these strategies allows MSPSO to balance its exploration and the exploitation ability, resulting in good performance, in terms of solutions accuracy. It however does su er from an increased time complex- ity when compared to several other PSO implementations. Improved Particle Swarm Optimization Algorithm (

SA C13358011_MSC_ASD.final.pdf (D108754758)

12/50	SUBMITTED TEXT	536 WORDS	99%	MATCHING TEXT	536 WORDS

is another mutli-swarm technique that "adopts a new mutation operator and a new method that congregates some neighbouring individuals to form multiple sub-populations in order to lead particles to explore new search space" (Zheng et al., 2007). Subdividing out the population into multiple neighborhoods, or swarms allows the algorithm to divide up the problem space, and thereby improve performance. Utilising the mutation factor also allowed the implementation to "en- hance the e ciency of advantageous direction of ying particles, so particles can y to feasible region more quickly and more e ciently". Figure 2.3: IPSO multi-populations being formed . (Zheng et al., 2007) 14 CHAPTER 2. BACKGROUND RESEARCH 2.3.5 Distributed PSO dPSO As mentioned in Chapter One, there will be a limit to the number of particles and swarms any one computer can support, the limiting factor being the amount of memory or computational power. To avoid this bottleneck there has been a signi cant body of research into creating a distributed implementation of the PSO algorithm. One such area of research is in robotics, using PSO as a search algorithm distributed across many particles, in reality each robot acts as a particle in a PSO algorithm. In James M. Hereford paper, "A Distributed Particle Swarm Optimization Algorithm for Swarm Robotic Applications" a distributed PSO algorithm is proposed, which he calls the dPSO (Hereford, 2006). Listing 2.1 shows the pseudo code used to implement the algorithm. Listing 2.1: Herefords dPSO Code Flow 1 pbest = -1; gbest = -1; % Initialize pbest and gbest 2 While (target not found or time not expired) 3 % Make measurement and update, if necessary 4 meas = make_measurement(); 5 if (meas < pbest) % Update pbest, if true 6 Up_pbest(); %Update pbest value and location 7 if (meas < gbest) % Update gbest, if true 8 Set gflag; % set a flag 9 Up_gbest(); %Update gbest value and location 10 end 11 end 12 Move(); % Move bot based on PSO update equation 13 % For simulation, constrain bot movement 14 % Two conditions: 15 % (1) Magnitude of velocity must be > Vmax 16 % (2) Direction must be within in max turn angle 15 CHAPTER 2. BACKGROUND RESEARCH 17 If (new gbest found) %broadcast new gbest value 18 Broadcast(gbest); 19 end 20 If (gbest is global gbest) %broadcast gbest location 21 Broadcast(gbest_location); 22 end 23 End while In the paper Hereford designed the algorithm for robotic operators using search algorithms to nd speci c targets within a search space. From the pseudo code we can see that once a new GBest value has been found by an operator, that best value is broadcast to all particles in the swarm. This creates a truly distributed model with no reliance on a central operator to process the results of all the particles. Particles must maintain an active list of all particles in the swarm in order to broadcast the new global best value. If a new particle(robot) is added to the swarm, all particles must be updated to be made aware of that new particle. In his results he found a good deal of success, with the robots nding their target 99% of the time. Additionally, he found that as you increase the number of particles in the swarm, the time to nd the solution considerably reduces. Figure 2.4: dPSO Time to nd Target (Hereford, 2006) 16 CHAPTER 2. BACKGROUND RESEARCH AGLDPSO In Wang et als. paper "Adaptive Granularity Learning Distributed Particle Swarm Optimization for Large-Scale Optimization" a distributed implementation is proposed, called Adaptive granularity learning distributed particle

SA C13358011_MSC_ASD.final.pdf (D108754758)

Z.-J. Wang et al., 2020). AGLDPSO uses a master/slave relationship when creating distributed particles, which is signi cantly di erent to Herefords distributed model, whereby all particles are peers within a swarm, there is no orchestration or master controller. Each particle, or robot was aware of all other robots in the swarm and up- dated accordingly. In AGLDPSO, a master acts as the intermediary and updates all particles in the population. This works very well for their implementation but creates a network bottleneck that dSPO altogether avoids. When tested against multiple other optimisation algorithms the authors found that AGLDPSO "achieves a promising and satisfying performance when solving the large-scale optimization problems.". Figure 2.5: Flow Diagram of AGLDSPO (Z.-J. Wang et al., 2020) 17 CHAPTER 2. BACKGROUND RESEARCH 2.4 Evaluating Performance 2.4.1 Benchmarking Benchmarking, also referred to as "best practice benchmarking" or "process bench- marking", is a method of conducting necessary actions and activities in order to eval- uate how a given piece of software or a system is going to perform once it is deployed and used in the eld (Jetmarov a, 2011). This is evaluated from a user's perspec- tive, and is typically assessed in terms of throughput, stimulus-response time, or some combination of the two. (Vokolos & Weyuker, 1998). It is also a way of assessing the systems availability. Meaning that whenever the system undergoes high levels of stress, be it increasing the number of processed requests (throughput), or high resource consumption (machine resources), the system still processes these requests or events in acceptable numbers (Vokolos & Weyuker, 1998). Usually benchmarking involves predicting how a technology will behave once it's used in regular everyday life. For instance, benchmarking a web server will allow organisations to predict how well it will cope with increased web tra c. These predictions allow organisations to antici- pate potential limitation and can lead them to a plan to overcome those limitations. Utilising a benchmarking evaluation strategy, the study will compare the two pro- posed PSO implementations, providing insights into the performance of a distributed implementation. 2.5 Research Gaps As outlined above there has been a large amount of research in the area of Swarm intelligence over the past 20 years. We can see in the J. F. Schutte et. al article they had a comprehensive cover of using parallel PSO (Schutte, Reinbolt, Fregly, Haftka, & George, 2004), however, to limit network connection time they used a parallel processor for running the algorithm. This would certainly cut down on networking connection times, however it cannot be described as a truly distributed algorithm. S. Burak Akat and Veysel Gazi focused much more on the distributed aspect of the 18 CHAPTER 2. BACKGROUND RESEARCH algorithm, but no e ort was made at running comparisons between the distributed version and a centralised version of the algorithm (Akat & Gazi, 2008). James M. Hereford again focused on the distributed aspect of the algorithm, with no relative benchmarks comparing it to a centralised version. (Hereford, 2006) The gap identi ed for this research is the performance bene ts of running a PSO algorithm in distributed environments vs centralised environments. As you can see from the above there has been plenty of research into crafting and testing distributed implementations of the PSO algorithm. However there has been no published literature testing the PSO algorithm in distributed environments vs a centralised environment and comparing the results, with an emphasis on nding a point where a distributed implementation is more time e cient than a centralised implementation. 2.6 Conclusion In this Chapter an overview of PSO was presented, along with some more in-depth research into parameter selection and di erent PSO implementations were discussed. multi-swarm PSO was discussed at a high level and two distinct distributed PSO implementations were presented and discussed. Research gaps were identi ed and discussed, along with the evaluation metric that will be used in this study. The next Chapter will focus on the experiment design and methodology. It will also cover the aim of the research and provide a basic overview of the experiment implementation details. 19 Chapter 3 Experiment design and methodology 3.1 Introduction This Chapter presents the design used to craft the experiment and the methodologies used to test the results from that experiment. The Chapter will cover the aim of the research, the design and implementation of the experiment, data output design and will conclude with design conclusions. 3.2 Aim of Research The aim of this paper is to nd out if a point exists when scaling a PSO algorithm that it becomes more e cient to run a distributed algorithm instead of a centralised algorithm. Increasing the number of active particles in a PSO algorithm can lead to a decline in performance. As for each iteration or epoch the machine must dedicate more and more computational power to recalculate particles positions and best evalua- tions. In a distributed model this burden can be divided across a number of machines, however in order to calculate the GBest value network communications must occur between machines, thus forming the swarm. Network communications are inherently slower than local connection communications, however at high levels of active particles 20 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY this speed decrease may be alleviated by the increased computational power available. 3.3 Hypothesis Null Hypothesis H 0 : The time taken to nd a stabilised result from using a distributed network of swarms never exceeds the time lost from the network communications needed to update swarms of each other's presence and activities. Alternate Hypothesis H 1 : The increased speed to solution time from using a distributed network of swarms exceeds the time lost from the network communications needed to update swarms of each other's presence and activities once the number of particles and/or swarms has reached a su cient level. 3.4 Design overview In order to create the data required to prove/disprove the hypothesis two code bases will need to be created. One code base will need to handle generating the data required for a centralised PSO, and the other will need to generate the data from a distributed PSO implementation. Each will need to be able to run the same tness functions and operate nearly identically, except the distributed implementation, which will need to call other PSO nodes to run swarms rather than running them on the local system. The coded system will need to be run multiple times with the same inputs in order to account for the random element inherent in the PSO algorithm. After each run, the system will need to output all generated data and a data aggregator will need to sort, average and output all the data in order for it to be evaluated. 21 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY Figure 3.1: Experiment Design Flow Chart 22 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY Figure 3.1 displays the ow of the experiment. First the environment type will be chosen, then from that point the required tness function will be chosen. To give greater breadth to the results, three di erent tness functions will be used with varying degrees of complexity. Once the tness function has been chosen, the swarms and par- ticles can be initialised with the function implementation. In the case of the centralised implementation this will be the same machine, in the decentralised implementation swarm nodes will be contacted and initialised remotely. Once each environment type has completed its run and found a stable answer, the implementations will return a data response object. At this point the data aggregator will pull in the results from both implementations, with corresponding con guration details, and generate a data output le useful for data comparison. This data le will be loaded into a data tabulation tool, whereby the results can be inspected, and comparisons can be drawn up. At this point the study will be able to draw conclusions on whether the alternate hypothesis can be accepted or rejected. 3.4.1 Centralised PSO Design Expanding on the design overview, the centralised implementation will need to perform several steps in order to generate the required results set. Firstly, it will initialise and create a parameterised number of swarms, each with a parameterised number of particles. At this point the rst iteration will begin for all swarms. This will all occur on the one node. The iteration will follow the same format as what was discussed in section 2.2.1. Each swarm will attempt to nd its GBest value. All swarms will then report the GBest value. At this point the controller class, which will also be the class that instantiates the swarms, will try and calculate if a "settled" GBest value has been found. What this means is that at a certain point the algorithm will settle on a particular value, the global minimum. This will be the optimum solution to the tness function. In this context settled means that for a certain number of iterations the same GBest value will be returned across all swarms. So, the system will need to record past values and compare against returned values to see if the same value has 23 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY been returned for multiple iterations in a row. When this has occurred the domain controller will calculate the time to arrive at this optimal solution, and the number of iterations it took to nd that value, this being the Time To Solution (TTS) value. The basic ow of centralised implementation is shown below in gure 3.2. Figure 3.2: Flow Diagram of Centralised Particle Swarm Algorithm 24 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY 3.4.2 Distributed PSO Design Figure 3.3: Flow Diagram of Distributed Particle Swarm Algorithm As discussed in the overview, the distributed implementation will function similarly to the centralised implementation to keep di erences in results to a minimum. The only functional di erence will be that swarms will be logically separated across multiple ma- chines in the distributed implementation. Where the centralised implementation will be running a domain controller, and all swarm instances, the distributed implementa- tion will feature a domain controller on a single node, calling out to other networked 25 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY nodes, that will

SA C13358011_MSC_ASD.final.pdf (D108754758)

14/50	SUBMITTED TEXT	60 WORDS	100%	MATCHING TEXT	60 WORDS

and iterate individual swarms with a parameterised number of particles. Once an iteration ends the node will communicate back to the domain controller, which will collect responses from all active swarms, and check to see if the swarms have "settled" on the parameterised global minimum value. If they have the domain will stop the nodes from running any further iterations, collect resulting val- ues,

SA C13358011_MSC_ASD.final.pdf (D108754758)

15/50	SUBMITTED TEXT	885 WORDS	99%	MATCHING TEXT	885 WORDS

number of iterations and TTS values. Network time will be accounted for in the TTS value. 3.5 Distributed environment In order to create a distributed environment for the distributed implementation for this project a cloud provider was used. Using these cloud providers, a virtual machine can be created and initialised, allowing the end user or developer to install custom software and applications. In this instance it allows the distributed implementation and centralised implementation to be deployed and run in a containerised environ- ment. Snapshots were used to ensure consistency between the two implementations and between the nodes of the distributed implementation. To facilitate this a num- ber of di erent providers were considered, including Amazon Web Services (AWS), Google cloud and DigitalOcean. Each had a containerised virtual machine product that allowed the deployment of custom applications, along with unique domain names and other internet connection utilities, however only one provider was required and DigitalOcean was chosen as the provider for this project. 3.5.1 DigitalOcean DigitalOcean is an Infrastructure as a Service (IaaS) platform for software developers. It allows developers to create containers in a simple and quick manner. To deploy DigitalOcean's IaaS environment, developers launch a virtual machine instance, which DigitalOcean calls a "droplet". This is containerised, and a number of operating systems can be chosen to initialise the doplet. The OS's available to developer include: Ubuntu, CentOS, Debian, Fedora, CoreOS or FreeBSD. DigitalOcean also o er the 26 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY ability to choose the geographic region the droplet will be created in, and some other con guration options such as the amount of dedicated RAM, number/quality of CPU, and the size of the on-board hard drive. Once initialised the developer may install their own custom software or use some prepared packages from DigitalOcean to help speed along deployment. Once the droplet has been created developers have the ability to monitor activity from their browser using DigitalOcean's control panel. They also have the ability to update the domain name, and create a Virtual Private Network (VPN) for the droplet. Snapshots of the droplet can be taken at any time and used to recreate it if required. DigitalOcean is a very a ordable option among its peers, with basic containers costing at minimum 0.00744$ an hour, working out at 5$ a month. 1 For that price DigitalOcean provide a container with one gigabyte of RAM, one core CPU and twenty gigabytes of on-board storage. 3.5.2 Container and Network Design Utilizing the containerisation service previously mentioned, a number of droplets will be created. Droplets will contain the implementation code in order to act as a domain controller and as a swarm node. However each node will only ever act as a domain controller or a swarm node. This design can be seen in gure 3.4. When running the distributed implementation, the domain controller node will be called using the same request as the non-distributed method, however instead of an integer de ning the number of swarms to be used, a list of IP address will be passed in. These IP addresses will correspond with the IP addresses for the droplets running swarm nodes. The domain controller will then connect with each of these droplets, initialise the swarm node with the passed in con guration values, begin iterating and collecting responses from all connected swarm nodes. 1 https://www.digitalocean.com/pricing 27 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY Figure 3.4: Container Design Overview This design does feature a core aw in that it creates a central dependency in the domain controller. However in order to keep the results as consistent and in line with the centralised implementation, the domain controller will be the same droplet size as all other swarm nodes. 3.6 Data Design and Data Capture Overview For the purpose of this project the following variables will be recorded for comparison. 28 CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY Variable Description Data Type TTS Time taken to get to solution Integer N-Particles Number of active Particles Integer N-Swarm Number of active Swarms. Integer Environment-Type Distributed or Centralised String Iterations Total number of iterations/epochs to reach a solution Integer nalGroupBest Final GBest from all swarms Integer averageBestX Average best X value across all swarms Integer averageBestY Average best Y value across all swarms Integer Table 3.1: Data output variables Starting with the rst variable in table 3.1, the time variable, this will represent the amount of time it took the speci c test to calculate an answer, essentially this is the TTS value. This will be recorded in milliseconds, and it will be calculated by the domain controller. Starting from the time it initialises the swarms in either the distributed implementation or the centralised implementation and ending when either implementation arrives at a settled answer. The number of particles and swarms will be speci ed in the request when running the algorithm but will also be recorded as part of the output for ease of comparison of results. Again, environment type will be speci ed in the request and recorded in the output for comparison. The iterations variable will be used to record the number of iterations or epochs it took to reach a settled solution. This will also include the de ned number of iterations for a result to be considered settled, which will in ate the total number of iterations, however as this a ects all test runs it will have no signi cant impact on the nal values. FinalGroupBest value will record the nal GBest answer, from all swarms. All iterations should come to the same value as the tness functions used will contain only one global minimum. However it was recorded to see if there was any variation in the results between implementation types. Similarly with average best X and Y values, all

SA C13358011_MSC_ASD.final.pdf (D108754758)

16/50	SUBMITTED TEXT	33 WORDS	88%	MATCHING TEXT	33 WORDS

types should conclude with the same values, but they were recorded to see if there was any variation. This data will be outputted to a JSON(JavaScript Object Notation) le at the end of 29

SA C13358011_MSC_ASD.final.pdf (D108754758)

CHAPTER 3. EXPERIMENT DESIGN AND METHODOLOGY each test run. The JSON le will be named in the following convention: >Function-Name<_>SwarmNumber<_>ParticleNumber<.results.json 3.6.1 Data Aggregation As mentioned in section 3.4 Design overview, due to the random element of the PSO algorithm, an average of results with the same parameters will need to be taken. Therefore multiple data output les will be generated for tests with the same inputs. In combination with multiple tests with varied numbers of particles and swarms, a large number of output les will be generated. To deal with this a data aggregation system will need to be created. This aggregation system will load in results with the same input variables, create an average values for the TTS variable, number of iterations, nalGroupBest, and averageBestX/Y values. Values will then be outputted into a single data le, with results from the distributed and centralised implementation being directly compared. 3.7 Design and Methodology Summary This Chapter presented an overview of the design and methodology used for this re- search. It discussed the overarching design of the experiment, with subsequent sections going into a more detailed design of speci c elements of the research. Data output and capture was also discussed, with a data output design presented in some detail. The next Chapter will focus in on the implementation detail, providing coded examples of the design presented in this Chapter. 30 Chapter 4 Experiment Implementation 4.1 Introduction This Chapter will expand on the previous Chapter and cover the implementation details in more depth. Coding examples will be provided for more clarity on how the two PSO implementations function. Additionally, the test harness and data loader that were created for this experiment will be expanded upon. Coded examples of each will be given. 4.2 PSO Implementations 4.2.1 Java For the purpose of this study Java will be used as the implementation language. Java is a mature language, with its initial 1.0 release in 1995. Since then, it has gone through multiple iterations and major version changes. It has been largely adopted by enterprise organisations due to its stability and feature-rich ecosystem. Java was chosen due to the "deploy anywhere" model that allows for applications to be run on any operating system that supports a JVM(Java Virtual Machine). This will be a bene t when the application is deployed to a distributed environment. Other 31 CHAPTER 4. EXPERIMENT IMPLEMENTATION advantages of using Java are its quite performant 1 2 , has a large set of supported libraries and extensions, along with a very active community and a wide range of supporting tools and documentation. 4.2.2 Spring Boot Spring Boot is an open source Java-based framework, which includes many utilities and libraries that allows a developer to quickly create a "production-grade" 3 application. It is developed and maintained by the Pivotal team. Spring boot is con gured with an embedded servlet container, contains many auto-con guration features for ease of development, and provides plenty of additional tools for crafting microservices. Spring Boot was used in this experiment because of its mature and stable nature, and the ability to easily create REST(Representational State Transfer) based services. Due to the distributed nature of the experiment, REST API calls were used to facilitate the messaging between swarm nodes and the domain controller. More on this will be discussed in the following sections. Spring Boot allows developers to quickly con gure REST API's and is reasonably performant during runtime 4 . Spring Boot also allows developers to easily con gure a number of di erent logging tools and frameworks to aid in debugging runtime errors and exceptions, which was a requirement during the development phase of this experiment. 4.2.3 Centralised Implementation Using Spring Boot a basic rest controller was created, whereby the swarm could be initialised and run by calling one HTTP POST endpoint with a request object that contains the con gurable values. Listing 4.1 shows the coded example of this. 1 @PostMapping("/runSwarm") 2 public Response runCentralisedSwarm(@RequestBody Request request) throws Exception f 3 return centralisedService.runSwarm(request); 1 https://www.infoq.com/presentations/java-8-13-performance/ 2 https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go 3 https://spring.io/projects/spring-boot 4 https://spring.io/blog/2018/12/12/how-fast-is-spring 32 CHAPTER 4. EXPERIMENT IMPLEMENTATION 4 g Listing 4.1: Centralised Rest Controller Once the endpoint in the rest controller receives a request, it will call out to a service method which will initialise the swarms and particles. This then begins running the algorithm. Listing 4.2 is an extract of how these swarms are initialised in the service method. The swarm and particle objects are abstracted out of the service class, so that the distributed service class can reuse the same methods, with some modi cations. The service class also handles generating the response object with all necessary data points discussed in Chapter Three. 1 List>Swarm< multiSwarm = new ArrayList><(); 2 ... 3 /∗ Initialize all swarms and particle with supplied con guration ∗/ 4 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 5 multiSwarm.add(utils.createSwarm(swarmId, request.getNumberOfParticles(), request. ,! getCon gVariables())); 6 g Listing 4.2: Centralised Swarm initialisation Listing 4.3 shows an extract of the swarm iteration code in the centralised service. Once the swarms are initialized the service class moves onto iterating through each swarm. We can see in the extract this is done through a while loop. In the while loop each iteration of all swarms is done through a for loop. Each iteration of the while loop counts as a full iteration for all swarms. In the listing we can also see the code records a start time and a stop time. Once completed the time between entering the service class and completing the while loop is calculated. This is used to accurately calculate the time taken only for the PSO code to run, rather than the time to get a response from the rest controller. This is returned as the TTS value. 1 Instant start = Instant.now(); 2 ... 3 while(!targetFound) f 4 ... 5 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 33 CHAPTER 4. EXPERIMENT IMPLEMENTATION 6 Result result = utils.runSwarmIteration(multiSwarm.get(swarmId), stepCount); 7 ... 8 g 9 ... 10 stepCount++; 11 g 12 Instant nish = Instant.now(); 13 long timeElapsed = Duration.between(start, nish).toMillis(); 14 ... Listing 4.3: Centralised Iteration Method Within the main while loop the code contains the stopping criteria. Listing 4.4 shows an extract of this code. The code loops through each of the

SA C13358011_MSC_ASD.final.pdf (D108754758)

18/50	SUBMITTED TEXT	202 WORDS	98%	MATCHING TEXT	202 WORDS

sponses and each of their responses are compared to a previous value, and if the exact same global best has been found for the con gured number of times (The TARGET_NUMBER_OF_EXACT_ANSWERS variable) then the while loop forcibly ends. This is done through the use of an array list. If the array list is not null, the code checks if the array list contains the current swarms GBest value. If it does, the value is added to the array list, and the code checks the size of the array list, and if it is the same as the TARGET_NUMBER_OF_EXACT_ANSWERS variable then the while loop ends. If the value isn't present, then the array list resets itself to ensure all swarms return the exact same answer the same number of times. If the step count i.e. the number of iterations reaches a certain threshold (The MAX_NUMBER_OF_STEPS variable) the while loop is also terminated to prevent the system from becoming trapped in an unending loop. This is assumed to be the worst-case scenario and is not expected to regularly happen. 1 /⋆ Check if the same answer has been found multiple times ⋆/ 2 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 3 if (settledList.size() == 0) f 4 settledList.add(bestEvaluation[swarmId]); 5 g else f 6 if (settledList.contains(bestEvaluation[swarmId])) f 7 settledList.add(bestEvaluation[swarmId]); 34 CHAPTER 4. EXPERIMENT IMPLEMENTATION 8

SA C13358011_MSC_ASD.final.pdf (D108754758)

19/50	SUBMITTED TEXT	1469 WORDS	99%	MATCHING TEXT	1469 WORDS

g else f 9 settledList = new ArrayList><(); 10 g 11 g 12 if (settledList.size() == TARGET NUMBER OF EXACT ANSWERS) f 13 TARGET NUMBER OF EXACT ANSWERS); 14 targetFound = true; 15 g 16 g Listing 4.4: Centralised Stopping Criteria Swarm Creation and Initialisation Listing 4.5 has an extract of the swarm creation method, createSwarm. From this method we can see how it creates a tness function based on an enum ("An enum type is a special data type that enables for a variable to be a set of prede ned con- stants" 5) passed in, allowing for the system to be extended with new tness functions with a low amount of code change. Fitness functions are de ned in the request sent into the service, and it initialises the swarm with the number of particles and other con guration values. 1 public Swarm createSwarm(int swarmId, int numberOfParticles, Con gVariables con gVariables) ,! throws Exception f 2 ... 3 if(con gVariables.getFunctionType().equals(FunctionType.BOOTHS FUNCTION))f 4 function = new BoothsFunction(); 5 g 6 ... 7 return new Swarm(numberOfParticles, con gVariables, function); 8 g Listing 4.5: Swarm creation Listing 4.6 shows how the swarm object is initialised. Best position and best evaluation are both set to the positive in nity, as this PSO is trying to nd the global minimum of 5 https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html 35 CHAPTER 4. EXPERIMENT IMPLEMENTATION optimisation functions. If instead the object was to nd global maximums this would need to be set to negative in nity. Inertia, cognitive and social parameters are set using con gured values. Particles are initialised within the function de ned in 4.5. 1 public Swarm (int numberOfParticles, Con gVariables con gVariables, Function function) f 2 double in nity = Double.POSITIVE INFINITY; 3 bestPosition = new Vector(in nity, in nity, in nity); 4 bestEval = Double.POSITIVE INFINITY; 5 this.particles = this.initialiseParticles(numberOfParticles, function); 6 this.inertia = con gVariables.getInertia(); 7 this.cognitive = con gVariables.getInertia(); 8 this.social = con gVariables.getSocial(); 9 g Listing 4.6: Swarm Initialisation Swarm Iteration Each swarm iteration is triggered by the service class calling it. Within the swarm object there is a run swarm method, which iterates through each particle, updating its personal best and the GBest value. Once that is completed the swarm updates all particles velocity and then moves each particle to its new position based on that velocity. A response object is created based on the iteration/epoch number including the global best value and best X/Y values. Listing 4.7 shows this method in detail. 1 public Result runSwarm(int epoch) f 2 for (Particle particle : particles) f 3 particle.updatePersonalBest(); 4 updateGlobalBest(particle); 5 g 6 7 for (Particle particle :

particles) f 8 updateVelocity(particle, this.inertia, this.cognitive, this.social); 9 particle.updatePosition(); 10 g 11 12 return createResultObject(epoch); 36 CHAPTER 4. EXPERIMENT IMPLEMENTATION 13 g Listing 4.7: Swarm Iteration The swarm, particle and vector models, along with the swarm velocity update function were adapted from LazoCoder's Particle-Swarm-Optimization implementation 6 . The codebase was heavily adapted in order to support the multi-swarm and distributed implementations, along with the required output data. 4.2.4 Distributed Implementation Similar to the centralised implementation listing 4.8 shows the REST controller for the distributed implementation. Unlike the centralised implementation there are three endpoints. When running the algorithm from the beginning, the "/distributedImple- mentation" endpoint is used. The node that this endpoint was hit with will become the domain controller and will call out to all other swarm nodes established in the network. The other two endpoints present are used by the system to run the algorithm. The endpoint "/initialiseSwarm" initialises the swarm with passed in variables. The code retains a copy of the swarm in its initialised state through the use of global variables. Then the domain controller will call the endpoint "/runSwarmIteration" which will take the in memory swarm object and run one iteration, returning the response values from it to the calling domain controller. 1 @PostMapping("/initialiseSwarm") 2 public Boolean initialiseSwarm(@RequestBody DistributedRequest request) throws Exception f 3 return distibutedService.setInitialisedSwarm(request.getSwarmId( ), request. ,! getNumberOfParticles(), request.getCon gVariables()); 4 g 5 6 @PostMapping("/runSwarmIteration") 7 public Result runSwarmIteration(@RequestBody DistributedRequest request) f 8 return distibutedService.runSwarmIteration(request.getEpoch()); 6 Particle-Swarm-Optimization codebase: https://github.com/LazoCoder/Particle-Swarm - Optimization 37 CHAPTER 4. EXPERIMENT IMPLEMENTATION 9 g 10 11 @PostMapping("/distributedImplementation") 12 public Response distributedImplementation(@RequestBody DistributedRequest request) f 13 return distibutedService.runDistributedCallingSystem(request.get NumberOfParticles(), ,! request.getBaseUrls(), request.getCon gVariables()); 14 g Listing 4.8: Distributed Rest Controller Similar to the centralised implementation, the distributed rest controller calls out to a service class, which initiates all the swarms and runs the iteration steps. Listing 4.9 is an extract of that initialisation part of this class. We can see this di ers to the centralised implementation in that it uses an entirely di erent method to initialise the swarms. This method is used to call out to the remote nodes to initialise the swarm's there. 1 for(int swarmId=0; swarmId > numberOfSwarms; swarmId++)f 2

... 3 callInitialiseSwarms(swarmId, baseUrls.get(swarmId), distributedRequest); 4 g Listing 4.9: Distributed Initialisation Method Listing 4.10 shows an extract of the iteration loop code for the distributed implementa- tions. There is more required of the code here than in the centralised implementation. Here the code must craft a request to send to each swarm node. Each request will contain all con guration values and the swarm ID, which will need to be returned by the swarm node to identify it when formatting the results response object. Again, the while loop persists until a target number of the same answers has been recorded, or the step count exceeds a speci ed threshold. Once the while loop is complete the timer stops, and the TTS is recorded and a response object is formed. The response object for the distributed implementation is the exact same as the centralised imple- mentation. This means the stopping criteria will function identically to the extract found in gure 4.4. 38 CHAPTER 4. EXPERIMENT IMPLEMENTATION 1 while(!targetFound) f 2 ... 3 for(int swarmId = 0; swarmId>numberOfSwarms; swarmId++) f 4 DistributedRequest distributedRequest = new DistributedRequest(); 5 distributedRequest.setSwarmId(swarmId); 6 distributedRequest.setCon gVariables(con gVariables); 7 Result result = callRunSwarm(swarmId, baseUrls.get(swarmId) ,distributedRequest); 8 multiSwarmResults.get(swarmId).getResults().add(result); 9 bestEvaluation[swarmId] = result.getBest(); 10 g Listing 4.10: Distributed Iteration Loop Rest calls to swarm nodes As seen in the above section, the service class calls out to an additional method to handle the REST call for initialising the swarms. Listing 4.11 shows this method, and in it we can see it calls out to the swarm node based on the passed in URL. The endpoint, port and HTTP protocol will be the same for all the nodes, so this is hardcoded in the method. The method passes the con guration details to the swarm node and returns a Boolean to acknowledge whether the initialisation was successful. If any swarm was not successful, the entire system aborts the run. This would be considered the worst case scenario and is not expected to happen frequently. 1 private Boolean callInitialiseSwarms(int swarmId, String baseUrl, DistributedRequest ,! distributedRequest) f 2 String fullUrl = "http://" + baseUrl + ":8080/swarm/initialiseSwarm"; 3 ... 4 HttpEntity>DistributedRequest< request = new HttpEntity><(distributedRequest); 5 try f 6 ResponseEntity>Boolean< responseEntity = restTemplate.exchange(fullUrl, HttpMethod. ,! POST, request, Boolean.class); 7 return responseEntity.getBody(); 8 g catch (Exception ex) f 9 return false; 10 g 39 CHAPTER 4. EXPERIMENT IMPLEMENTATION 11 g Listing 4.11: Rest call to initialise swarm Listing 4.12 shows the code snippet for calling the swarm nodes to run a single iter- ation. Similar to the initialise method it calls out to the swarm using a xed URL scheme, only amending the base URL according to the swarm node. It does however expect a response back that details the iteration number, TTS, global best variable and best X and Y variables. This response will be added to the general response object returned once the test run is complete. If an exception occurs during the test run the whole run is aborted, again this is a worst-case scenario and is not expected to happen frequently. 1 private Result callRunSwarm(int swarmId, String baseUrl, DistributedRequest distributedRequest ,!) f 2 String fullUrl = "http://" + baseUrl + ":8080/swarm/runSwarmIteration"; 3 log.info("Calling endpoint fg to run swarm with id fg", fullUrl, swarmId); 4 HttpEntity>DistributedRequest< request = new HttpEntity><(distributedRequest); 5 try f 6 ResponseEntity>Result< responseEntity = restTemplate.exchange(fullUrl, HttpMethod. ,! POST, request, Result.class); 7 return responseEntity.getBody(); 8 g catch (Exception ex) f 9 ... 10 throw new Exception("Error calling service to iterate swarm", ex); 11 g 12 g Listing 4.12: Rest call to run swarm iteration On the swarm nodes receiving end the distributed service has simple methods for using the exact same initialisation and iteration methods as the centralised implementation. As mentioned previously this was done in order to keep the PSO algorithm code di er- ences between the two implementations to an absolute minimum. The only di erence being that the swarm state is kept in memory once the method has completed through the use of global variables. Listing 4.13 shows how this works for initialisation. The 40 CHAPTER 4. EXPERIMENT IMPLEMENTATION swarm is initialised and stored in memory through the global variable initialisedSwarm. Each swarm iteration simply references this global variable and uses the same iteration command as the centralised implementation, shown in listing 4.7. 1 private Swarm initialisedSwarm; 2 public boolean setInitialisedSwarm(int swarmId, int numberOfParticles, Con gVariables 3 con gVariables) throws Exception f 4 this.initialisedSwarm = 5 utils.createSwarm(swarmId, numberOfParticles, con gVariables); 6 .... 7 g Listing 4.13: Distributed swarm initialisation 4.2.5 Fitness Functions overview In order to thoroughly test the hypothesis three tness functions were chosen. The three functions ranged in computational complexity in order to give a greater breadth of results to draw conclusions from. All functions have only one global minimum and all are two dimensional only. The rst optimisation function chosen was the Easom function, equation 4.1 represents this 7 . The easom optimisation function has a global minimum value of -1 where X and Y are equal to . f(x;y) = *cos(x)cos(y)exp(*(x *) 2 * (y *) 2) (4.1) The next equation chosen was the Beale's function, equation 4.2 represents this 8 . Beale'

SA C13358011_MSC_ASD.final.pdf (D108754758)

Curiginal

20/50	SUBMITTED TEXT	65 WORDS	96%	MATCHING TEXT	65 WORDS

function has a global minimum value of 0 where X is equal to 3 and Y is equal to 0.5. f(x;y) = (1:5 * x + xy) 2 + (2:25 * x + xy 2) 2 + (2:625 * x + xy 3) 2 (4.2) The nal equation chosen was the Booth's function, equation 4.3 represents this 9 . 7

SA C13358011_MSC_ASD.final.pdf (D108754758)

22/50	SUBMITTED TEXT	15 WORDS	100%	MATCHING TEXT	15 WORDS

http://www.sfu.ca/ ~ ssurjano/easom.html 8 http://www.sfu.ca/ ~ ssurjano/beale.html 9 http://www.sfu.ca/ ~ ssurjano/booth.html 41 CHAPTER 4. EXPERIMENT IMPLEMENTATION

SA C13358011_MSC_ASD.final.pdf (D108754758)

21/50	SUBMITTED TEXT	146 WORDS	100%	MATCHING TEXT	146 WORDS

f(x;y) = (x + 2y * 7) 2 + (2x + y * 5) 2 (4.3) 4.3 Test Harness As described in Chapter Three, a test harness was created in order to test the PSO implementations. Separating out the testing element from the code implementation allowed for greater exibility in testing and allowed for tests to be updated inde- pendently of the code deployments to the containers within DigitalOcean. The test harness was designed to be exible with its parameters, along with being able to run multiple tests while saving the responses from each test run into a results folder. 4.3.1 Postman/Newman and NodeJs Figure 4.1: Postman Test In order to accomplish that goal Postman was chosen to create the tests with, and Newman was used as the test runner. Postman is a "scalable API testing tool that quickly integrates into CI/CD pipelines." 10 . Postman allows you to easily de ne a test 10

SA C13358011_MSC_ASD.final.pdf (D108754758)

23/50	SUBMITTED TEXT	5 WORDS	100%	MATCHING TEXT	5 WORDS

https://www.guru99.com/postman-tutorial.html 42 CHAPTER 4. EXPERIMENT IMPLEMENTATION

SA C13358011_MSC_ASD.final.pdf (D108754758)

24/50	SUBMITTED TEXT	243 WORDS	98%	MATCHING TEXT	243 WORDS

for a remote API HTTP endpoint, specifying headers, body parameter, and display the responses from that API. Postman comes equipped with plenty of development tools, for example it supports environment selection and parameters, allowing developers to substitute values based on the environment they have chosen. Figure 4.1 shows an image of a Postman test opened in the Postman application. In the gure it can be seen how the URL can be parameterised using the curly brackets operators {{Parameter_Name}}, which can also be applied to body variables. Postman can organise a series of tests into what it calls a test collection, which can be exported to JSON les and saved locally. Newman is the command line implementation of Postman. Newman can take these JSON test collections and run them in a batch format. Newman runs on NodeJs and is installed through NPM. NodeJs is a "an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications." 11 . NPM (Node Package Manager) is a command line package manager tool that allows developers to download and install javascript libraries to their local development environment. While Newman is mainly described as a CLI tool, it has a full API available for developers to use through node. Utilising this API allows for developers to create programs utilising Postman test suites, while being able to programmatically change parameters as required. 4.3.2 Implementation The test harness was split into two main methods for looping, one for the centralised run, and one for the distributed run, however they only di er in one aspect. When adding additional

SA C13358011_MSC_ASD.final.pdf (D108754758)

25/50	SUBMITTED TEXT	66 WORDS	100%	MATCHING TEXT	66 WORDS

the distributed implementation builds up a list of IP ad- dress corresponding to the swarm nodes. The centralised implementation only adds the required number of swarms to the request. Listing 4.14 shows how this loop func- tions. This is speci cally the centralised method, however the distributed method is nearly identical except for some minor con guration changes. The harness rst iter- ates through each tness function i.e. Beale'

SA C13358011_MSC_ASD.final.pdf (D108754758)

26/50 **SUBMITTED TEXT** 260 WORDS **96%** **MATCHING TEXT** 260 WORDS

each tness function test set, the number of swarms is gradually increased from two to ten, and the total number of particles across all swarms in each test increases from 100,000 to 1,000,000. So, for example when the test is for two swarms, each swarm will increase from 50,000 particles to 100,000, to 150,000 etc. until the total number of particles is 1,000,000. Each of these tests will keep the same con guration values and will be run three times so that an average across results may be obtained. The harness is con gured to wait for each request to return a response object before mov- ing onto the next request, to avoid overloading the droplets. Newman by default runs asynchronously, so the harness was con gured to check every second if a le with the expected test name was written to the disk, and if it has move on, if not keep looping. 1 async function centralisedRun() f 2 for(var functionId = 0; functionId>= functionNames.length-1; functionId ++) f 3 for (var swarmNumber = 1; swarmNumber >= maxNumberSwarms; swarmNumber++) f 4 particleIncrement = 1000; 5 for (var particleNumber = particleStartNumber; particleNumber >= ,! maxNumberParticles; particleNumber += particleIncrement) f 6 runNewmanCode("NON DISTRIB", functionNames[functionId], "Remote", ,! swarmNumber, particleNumber, baseUrlStringsNonDistributed, ,! nonDistribEndpoint); 7 var moveOntoNextTest = false; 8 while (!moveOntoNextTest) f 9 moveOntoNextTest = checkIfFilesExist("NON DISTRIB", functionNames[,! functionId], swarmNumber, particleNumber); 10 await new Promise(resolve =< setTimeout(resolve, 1000)); 11 g 12 g 13 g 14 g 15 g Listing 4.14: Test Harness Main Loop Listing 4.15 shows the Newman node implementation code. The implementation allows for a lot of reuse, by accepting parameters for changing the PSO implementation type, number of swarms/particles and tness function type. The code uses the function 44

SA C13358011_MSC_ASD.final.pdf (D108754758)

27/50	SUBMITTED TEXT	226 WORDS	99%	MATCHING TEXT	226 WORDS

name to load up the correct Postman collection type, there is only one test in each collection. Utilising the Newman API, and correctly loading up the test collection and environment variables le, the method calls out to the remote domain controller and records the responses. Errors are gracefully handled through Newman's inbuilt error handling system. This is repeated until all tests have been run across all tness functions. 1 function runNewmanCode(envName, testCollectionName, environmentName, numberOfSwarms, ,! numberOfParticles, baseUrlsSting, endpoint) f 2 newman.run(f 3 collection: require('./Requests/' + testCollectionName + '.postman collection.json'), 4 environment: require('./EnvVariables/' + environmentName + '.postman environment.json ,! '), 5 reporters: 'cli', 6 globalVar: [f "key":"NumberOfSwarms", "value":numberOfSwarms g, f"key":"endpoint", " ,! value": endpointg, f "key":"NumberOfParticles", "value": numberOfParticlesg, f " ,! key":"BaseUrls", "value": baseUrlsStingg] 7 g).on('request', function (error, args) f 8 fs.writeFile(dir + testCollectionName + " " + numberOfSwarms + " " + ,! numberOfParticles + '.result.json', args.response.stream, function (error) 9 g 10 g); 11 g Listing 4.15: Newman Implementation Code Once the method receives a response back from the service, it will use the function name, number of particles/swarms to write out the response to a le with those pa- rameters. Those parameters will be used in its name, allowing each response to be uniquely named, and avoiding overwriting the output le with each new response. 4.4 Data Aggregator Implementation As mentioned in Chapter Three, multiple test iterations will be run to get averages from the results. In this experiment each function was tested three times for the two 45

SA C13358011_MSC_ASD.final.pdf (D108754758)

Curiginal

28/50	SUBMITTED TEXT	365 WORDS	99%	MATCHING TEXT	365 WORDS

implementation types, with swarms increasing from one to ten, and particles in all swarms increasing from 100,000 to 1,000,000. In order to aggregate all the results a simple data aggregation tool was created. The data aggregator needed to load data from each of these three tests runs, get an average value for the following values: Number of iterations, TTS and nal GBest. NodeJs was used to load the data and generate the average values and the system writes the result out as a CSV (Comma Separated Value) le. Listing 4.16 shows the code extract for this. In the listing we can see there is a set header string that will always be outputted at the beginning of the program, this is simply for ease of reading when the CSV is imported into excel. The program iterates through each function, swarm and particle number, then loads the JSON output le. Once it loads the result le for all three test runs, for both implementations it calculates the centralised av- erages and creates an output string in the CSV format. Once the centralised output string has been generated the program generates the same output string for the dis- tributed implementation. The output string is built up for the same tness function tests, and then outputted to the le system. 1 var outputString = "SWARM NUMBER,PARTICLE NUMBER,CENTRALISED ITERATIONS, ,! CENTRALISED TIME TO SOLUTION,CENTRALISED FINAL GROUP BEST, ,! DISTRIBUTED ITERATIONS,DISTRIBUTED TIME TO SOLUTION, ,! DISTRIBUTED FINAL GROUP BESTnn"; 2 3 for (var swarmNumber = 1; swarmNumber >= numberOfSwarms; swarmNumber++) f 4 for (var particleNumber = 1000; particleNumber >= maxNumberParticles; particleNumber ,! += particleIncrement) f 5 .... 6 let centralised RUN1 = fs.readFileSync(leUrlCentralised RUN1); 7 ... 8 let centralised data RUN1 = JSON.parse(centralised RUN1); 9 ... 10 var averageCentralisedTTS = Math.round((centralised data RUN1.timeToSolution + ,! centralised data RUN2.timeToSolution + centralised data RUN3.timeToSolution) ,! /3); 11 ... 46 CHAPTER 4. EXPERIMENT IMPLEMENTATION 12 13 outputString = outputString + swarmNumber + "," + particleNumber + "," + ,! averageCentralisedIterations + "," + averageCentralisedTTS + "," + ,! averageCentralisedGbest + ","; 14 ..... 15 g Listing 4.16: Data aggregation Implementation 4.5 Implementation conclusion This Chapter covered in depth how the PSO code implementations were developed, with a speci c focus towards coded examples. Each implementation was discussed in detail, along with how the distributed implementation speci cally calls swarm nodes on the network. The test harness and data aggregation tools

SA C13358011_MSC_ASD.final.pdf (D108754758)

29/50	SUBMITTED TEXT	332 WORDS	99%	MATCHING TEXT	332 WORDS

also covered in depth, with plenty of code extracts supplied. Testing implementation was discussed in some detail. The next Chapter will cover this in more detail along with the results obtained from the tests. 47 Chapter 5 Results, Evaluation and Discussion 5.1 Introduction This Chapter discusses the results obtained from running the experiment described in detail in Chapters Three and Four. Results from the centralised and distributed implementations will be presented and compared. As discussed in Chapter Three, several tness functions were tested in this experiment. This Chapter will expand on each for both implementations, with a primary focus on the TTS value. This Chapter will also evaluate the results, with some discussion taking place. 5.2 Results 5.2.1 Centralised Results Chapter Three and Four described the process of running the experiment in detail, in this section the results from those experiment runs will be presented. Each tness function will be examined, and results will be presented. All results are an average of three tests runs in order to compensate for the random element in PSO. Results were compared in a like for like manner, meaning when we compare the number of particles between a two swarm test and a ten swarm test, we are comparing the total number of particles in the test, not how many particles on each swarm. To further expand on 48 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION that example, when comparing 100,000 particle tests, we look at there being 50,000 particles on each swarm in the two swarm test, and 10,000 particles on each swarm in the ten swarm tests. This goes for both the centralised and distributed implementation tests. Easom's Function The rst function to cover is the centralised Easom test runs. Figure 5.1 shows the results when looking at the TTS values. In the results we can see that as the number of particles increases, the TTS increases at a disproportionate rate. TTS for each swarm tends to increase at the same rate, except for the ten swarm test runs, where the TTS is slightly longer at the beginning, but increases at a much greater degree closer to the 1,000,000 particles tests. Figure 5.1: Centralised TTS Easom'

SA C13358011_MSC_ASD.final.pdf (D108754758)

30/50	SUBMITTED TEXT	128 WORDS	97%	MATCHING TEXT	128 WORDS

Results Figure 5.2 shows the number of epochs or iterations taken to reach a solution for each of the swarms. All swarms display a downward trend in the number of iterations it takes to reach a solution, with the two and four swarms coming in at the lowest number of iterations at the maximum number of particles, one million particles. From the gure it can be seen that the number of iterations, while on a downward trend, can have a lot of variance as the particles increase in the tests, i.e. in the two swarm test the average iterations jumps from 114 iterations for 600,000, to 120 iterations for 700,000 particles. As the swarms increase the variance between particle increments 49 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION decreases. Figure 5.2: Centralised Iterations Easom's Results Booth's Function Looking at the Booth'

SA C13358011_MSC_ASD.final.pdf (D108754758)

31/50	SUBMITTED TEXT	80 WORDS	94%	MATCHING TEXT	80 WORDS

function results in gure 5.3 we see it closely follow the Easom's function results, in that the general trend on the results looks to be at an exponential increase. Once again, the test runs where ten swarms are used show a slight increase in the TTS at lower numbers of particles, and a large increase towards the one million particles run. Figure 5.3: Centralised TTS Booth's Results Figure 5.4 shows the results of the average iterations obtained by each swarm in testing. Similar to the Easom'

SA C13358011_MSC_ASD.final.pdf (D108754758)

32/50	SUBMITTED TEXT	49 WORDS	96%	MATCHING TEXT	49 WORDS

results the gure showcases a downward trend in the number of iterations as particles increase. Again, the results show that the lower number of 50 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION swarms, two and four swarms, obtained a lower number of iterations at the higher number of particles. Figure 5.4: Centralised Iterations

SA C13358011_MSC_ASD.final.pdf (D108754758)

33/50	SUBMITTED TEXT	12 WORDS	100%	MATCHING TEXT	12 WORDS

TTS results in gure 5.5, the same general trends from Easom'

SA C13358011_MSC_ASD.final.pdf (D108754758)

34/50	SUBMITTED TEXT	104 WORDS	94%	MATCHING TEXT	104 WORDS

function can be seen, with the general trend looking to be exponential. Again, the higher number of swarms tends to end up with a higher TTS as particles increase. Figure 5.5: Centralised TTS Beale's Results Figure 5.6 shows the number of epochs each swarm took to get to a solution. Once again, the general trend is that the number of epochs decreases with the increased number of particles, with some variations between the particle increments. Similar to 51 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION Booth's and Beale's function the lower number of swarms, two and four swarms had, on average, a lower number of epochs to reach a solution. Figure 5.6: Centralised Iterations Beale'

SA C13358011_MSC_ASD.final.pdf (D108754758)

35/50	SUBMITTED TEXT	51 WORDS	97%	MATCHING TEXT	51 WORDS

Results 5.2.2 Distributed Results This section examines the results from the distributed implementation tests, using the same format as the centralised results section. Each tness functions results will be presented, primarily focusing on the TTS and number of Epochs to solution, along with a small amount of commentary on these results. Easom's Function Looking at the Easom'

SA C13358011_MSC_ASD.final.pdf (D108754758)

36/50	SUBMITTED TEXT	193 WORDS	99%	MATCHING TEXT	193 WORDS

function rst, gure 5.7 shows the TTS results. TTS appears to linearly increase with the number of particles across all number of swarms. At the upper end of the experiment bounds, with 1,000,000 particles the two swarm tests appear to increase at a great rate than the four, eight and ten swarm tests, all of which appear to continue the same rate of increase in TTS. There are some variations in this linear increase, most notably the eight swarm tests show a large decrease in TTS at the 800,000 particle point, but otherwise the variations are relatively minor. 52 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION Figure 5.7: Distributed TTS Easom's Results Figure 5.8 shows the number of epochs to solution results. From that we can see the general trend is a linear decrease, however it is a very slight decrease. The two swarm and ten swarm tests show the least amount of change in results, with the two swarm tests decreasing by two epochs from start to nish, and the ten swarm tests decreasing by three epochs. The eight and four swarm tests showed a greater deal of variance, with the eight and four swarm tests decreasing by nine iterations. Figure 5.8: Distributed Iterations Easom'

SA C13358011_MSC_ASD.final.pdf (D108754758)

37/50	SUBMITTED TEXT	189 WORDS	96%	MATCHING TEXT	189 WORDS

function gure 5.9 shows the TTS results. Looking at the results shows similar trends as the Easom's function results, with the overall trend appearing to be a linear increase in TTS as the number of particles increases. Two 53 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION swarm tests once again appear to increase at a more aggressive rate towards the upper particle numbers. The ten swarm tests show an overall higher TTS than any other number of swarms. Eight swarm tests showed a large decrease in TTS at the 800,000 particles interval but resumes the same trend at the next interval. Figure 5.9: Distributed TTS Booth's Results Figure 5.10 shows the epochs to solution results. Unlike Easom's results, the results are a bit more mixed. The two and ten swarm tests start o at a lower number of epochs at the beginning of the tests, with both starting at 108 iterations and ending at 117 iterations for ten swarms, and 114 iterations for the two swarm tests. Both the four and eight swarm tests show a more linear decrease, from start to nish, however both show a great deal of variance at various intervals in the testing. Figure 5.10: Distributed Iterations

SA C13358011_MSC_ASD.final.pdf (D108754758)

38/50	SUBMITTED TEXT	78 WORDS	91%	MATCHING TEXT	78 WORDS

Results 54 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION Beale's Function Finally, looking at gure 5.11 we can see the TTS results for the Beale's function. Again, this follows the same trends as Booth's and Easom's functions, however its even less erratic, with little to no variance in any of the swarm tests. Again, the two swarm test runs show signs of it increasing at more than a linear increase at the higher number of particles, 900,000 and 1,000,000 particle tests. Figure 5.11: Distributed TTS Beale'

SA C13358011_MSC_ASD.final.pdf (D108754758)

39/50	SUBMITTED TEXT	33 WORDS	100%	MATCHING TEXT	33 WORDS

Results Looking at gure 5.12 we can see that once again the results show a great deal of variance when it comes to the number of epochs to reach a solution. Similar to the

SA C13358011_MSC_ASD.final.pdf (D108754758)

40/50	SUBMITTED TEXT	74 WORDS	94%	MATCHING TEXT	74 WORDS

function results, swarms two and ten start at a much lower number of epochs, but at the higher number of particles have much more similar results to the eight and four swarm results. Eight and four swarm results show a more gradual and consistent decline in epochs as the number of particles increases. 55 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION Figure 5.12: Distributed Iterations Beale's Results 5.2.3 Performance Comparison Figure 5.13 displays a direct comparison between the distributed implementation of the Easom'

SA C13358011_MSC_ASD.final.pdf (D108754758)

41/50	SUBMITTED TEXT	35 WORDS	100%	MATCHING TEXT	35 WORDS

TTS results, and the centralised TTS results. This clearly displays the general trend of the centralised implementation towards an exponential increase, whereas the distributed implementation more closely follows a linear increase in TTS. Figure 5.13: Performance Comparison Easom'

SA C13358011_MSC_ASD.final.pdf (D108754758)

42/50	SUBMITTED TEXT	54 WORDS	90%	MATCHING TEXT	54 WORDS

TTS Looking more closely at gure 5.13 we can see the two implementations seem to diverge in TTS increase rate at around 500,000 particles. This is more or less the same case in both Beale's, gure 5.14 and Booth's function gure 5.15. In the graphs we can see 56 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION

SA C13358011_MSC_ASD.final.pdf (D108754758)

43/50	SUBMITTED TEXT	108 WORDS	89%	MATCHING TEXT	108 WORDS

red line denotes this tipping point, Optimisation Tipping Point (OTP). In all three results it shows the distributed implementation increasing at a linear rate, while the centralised implementations begin to increase exponentially after the 500,000 particles tests. There are other patterns observable in all three results, including the same tendency for two swarm tests in the centralised implementation to increase at a more rapid rate than other number of swarms at the higher number of particles tests. Figure 5.14: Booth's TTS comparison Figure 5.15: Beale's TTS comparison All three functions display a large increase in TTS at the early stage of the experiment, with 100,000 particles. Table 5.1 compares these results. In the case of the Beale'

SA C13358011_MSC_ASD.final.pdf (D108754758)

44/50	SUBMITTED TEXT	70 WORDS	84%	MATCHING TEXT	70 WORDS

function, on average, the distributed implementation was 50% slower at nding a result. Easom's function had an even greater increase, taking 61% more time to nd a solution, and Booth's function was even bigger again, with it taking 82% more time in nding a solution. TTS - 2 Swarms TTS - 4 Swarms TTS - 8 Swarms TTS - 10 Swarms Beale's -Centralised 4205 3842 4775 4417 Beale's - Distributed 5291 5630 7153 7860

SA C13358011_MSC_ASD.final.pdf (D108754758)

45/50	SUBMITTED TEXT	290 WORDS	77%	MATCHING TEXT	290 WORDS

Centralised 2364 2297 2313 2388 Easom's -Distributed 3551 3287 3925 4385 Table 5.1: TTS Results for 100,000 Particles Conversely there are some signi cant gains in TTS for the distributed implementation at the highest number of particles. Table 5.2 compares these results for 1,000,000 particle tests. For the Beale function it was found that the distributed implementation was on average 145% faster than the centralised implementation. Similar performance gains were found with both Booth's and Easom's function, with Booth's being 152% faster, and Easom's being 123% faster. TTS - 2 Swarms TTS - 4 Swarms TTS - 8 Swarms TTS - 10 Swarms Beale's - Centralised 112258 119800 112751 160899 Beale's - Distribributed 56502 48906 49696 51248 Booth's - Centralised 102709 103364 102856 131270 Booth's - Distribributed 48322 39878 36861 49330 Easom's - Centralised 50887 51263 52918 78160 Easom's - Distribributed 29327 23923 24623 26300 Table 5.2: TTS Results for 1,000,000 Particles When looking at the number of iterations between the two implementations there is very little di erence. Looking at gures; 5.16, 5.17, 5.18, we can see the same general trends in all three tness functions. There is an overall gradual decline in iterations as particles increase, with some large variations, especially at the lower end of the 58 CHAPTER 5. RESULTS, EVALUATION AND DISCUSSION particles. Overall, the distributed implementations showed more variations than the centralised implementations. Figure 5.16: Performance Comparison Easom's Epoch Figure 5.17: Booth's Epoch comparison Figure 5.18: Beale's Epoch comparison 5.3 Discussion 5.3.1 Results Evaluation From the performance comparison section, we can see there is a clear point in all three tness functions whereby the distributed implementation displays a better TTS than the centralised implementation. Interestingly, OTP occurs for all three functions at more or less the same point. Once the number of particles across all swarms reaches 500,000 or greater, the distributed implementation begins to be a more e cient implementation than the centralised implementation. Once the particles reach one 59

SA C13358011_MSC_ASD.final.pdf (D108754758)

46/50	SUBMITTED TEXT	346 WORDS	98%	MATCHING TEXT	346 WORDS

million all three functions had a greater than 100% decreases in TTS compared to the corresponding centralised tests. This clearly answers the research question, that there is in fact a point whereby it is more e cient to run a distributed PSO implementation over a centralised implemen- tation. However, this must be caveated by the fact that these results are only a guide, a di erent tness function may have a di erent e ciency point. Additionally, di erent implementation languages, or PSO implementation methods may also have di erent e ciency points. We can therefore conclude that the alternate hypothesis has been proven, and there is a point where a distributed implementation is more time e cient than a centralised implementation. In terms of di erences between the number of iterations taken before nding a solu- tion, between the two implementations, this study found no di erence between the distributed implementation and the centralised implementation. There was a notice- able decline in the number of iterations across both implementations as the number of particles increased, and this was evident across all three tness functions. When looking at other resulting data points, such as the average X/Y value, and nal GBest, this study found no di erence between the implementation, and no di erences as the number of particles increased. The number of particles used in this study can be con- sidered excessive, as the number of particles required to correctly nd a solution to any of the tness functions was signi cantly lower than what was used in this study. However, as matching the trend seen in both implementations, the lower number of particles required much more iterations to nd a "settled" solution. 5.3.2 Strengths and Limitations The key strength to this research is the consistency of the results across the three tness functions. All three tness functions displayed an OTP at around 500,000 particles, which gives a very clear answer to the research question. This also give a strong starting position for future research, as there is a clear answer as to whether the research should use a distributed or centralised model, based on the number of particles required for the research. 60

SA C13358011_MSC_ASD.final.pdf (D108754758)

47/50	SUBMITTED TEXT	1057 WORDS	97%	MATCHING TEXT	1057 WORDS

The limitation within in this however, is that this research piece was only conducted with one speci c environment. Using a machine with a greater level of RAM or a higher quality CPU may produce di erent results. Additionally, di erent languages may have di erent OTP's, even with the same tness functions. Finally, while these results strongly point toward 500,000 thousand particles as being the OTP, this is really only relevant for functions with a similar complexity to the three chosen for this study. Using a more complex or signi cantly simpler tness function will likely result it a di erent OTP than the one found in this study. As mentioned in the Results discussion, to reach OTP, an excessive amount of particles was required. In the normal usage of PSO, 1000 particles or less would be more than adequate to nd the global minimum of a tness function. The only advantage of using such extreme levels of particles is to reduce the number of iterations required to reach a solution. 5.4 Conclusion This Chapter covered in-depth the results obtained from running the experiment de- scribed in chapters three and four. Each implementations results were presented, sub- divided by the individual tness function tested. Commentary was provided on these results, with some notable trends highlighted. Distributed implementation results and centralised results were also compared together, highlighting some interesting trends within the results, including a tendency for centralised TTS results to increase expo- nentially, whereas the distributed TTS results appeared to increase at a more linear rate. The research question was answered, and the alternate hypothesis was accepted, in that there is a point whereby a distributed PSO implementation becomes more e cient than a centralised implementation. It was noted that across both implemen- tations there was a general trend of decreasing the total number of epochs required to reach a solution as the number of particles increased. The next Chapter will conclude this research, with an overview of what has been accomplished in this experiment, evaluation of the results found from the experiment, some proposed future work and recommendations will be discussed. 61 Chapter 6 Conclusion This Chapter summarises the entire research project, reviewing the research objective and presents the answer to the research question. Finally, the contributions, impact, future work and recommendations are presented and discussed. 6.1 Research Overview This study aimed to nd empirical evidence that there was a point whereby it becomes more e cient to run a particle swarm optimisation algorithm in a distributed manner over a centralised implementation. Chapter One introduced the research topic and some basic overview of the research piece. Chapter Two presented research into the PSO topic, and identi ed a gap in the literature around PSO, that being that no existing literature focused on the point where a distributed PSO implementation is more time e cient than a centralised implementation. This gap was used to form the basis of this research topic. Chapter Three and Four discussed the experiment design and implementation, with some details provided on how to run the experiment and how the resulting data set was obtained. Finally, Chapter Five concluded with an overview of the results obtained and some interesting trends seen within the results data set. 62 CHAPTER 6. CONCLUSION 6.2 Problem De nition The aim of the research was to nd a point where a distributed particle implementa- tion became more time e cient than a centralised implementation, once the number of swarms or particles hits a particular level. This formed the research question "At what point are performance gains in running a particle swarm optimisation algorithm in a distributed environment outweighed by the time lost in network communications between multiple swarms?". From this research question the following research objec- tives were formed. 1. Create a distributed and centralised implementation of the PSO 2. Generate a result set for the centralised implementation, with varying inputs (example: di erent evaluation function, number of swarms and or particles etc.). Average out results with the same inputs 3. Generate a result set for the distributed implementation, with varying inputs (examples: di erent evaluation function, number of swarms and or particles etc.). Average out results with the same inputs 4. Cross comparison between the two result sets. Identify points at which dis- tributed had a faster response time, or other values that would make it preferable to a centralised implementation 5. Identify any limiting factors, signi cant points of interest in the data and identify future research Stemming from the research question the null and alternate hypothesis were formed. Null Hypothesis H 0 : The time taken to nd a stabilised result from using a distributed network of swarms never exceeds the time lost from the network communications needed to update swarms of each other's presence and activities. 63 CHAPTER 6. CONCLUSION Alternate Hypothesis H 1 : The increased speed to solution time from using a distributed network of swarms exceeds the time lost from the network communications needed to update swarms of each other's presence and activities once the number of particles and/or swarms has reached a su cient level. 6.3 Design/Experimentation, Evaluation & Re- sults For this research piece two PSO implementations were created, one for a centralised approach, and one for a distributed approach. Both implementations were deployed into a cloud container, with all nodes within the distributed implementation matching the centralised implementation. As both implementations were designed with a cloud environment model in mind, a REST endpoint was created to trigger running swarms, based on passed in con guration options. Options that could be passed in included: number of swarms, number of particles, social weighting, inertia weighting, cognitive weighting and tness function type. Utilising this REST endpoint a series of tests where created to generate the required output data set. Each test was repeated three times, with the same con guration options passed in, allowing the results to be aver- aged across the three runs. Once all the results had been aggregated and averaged, a general results le was created in a CSV format. This format worked quite well, with failure rarely happening, and when they did, the program was designed to fail gracefully, without returning malformed results. Utilising REST connections between distributed nodes allowed for rapid development and testing, however it may have added additional time overheads to the nal TTS result. Additionally, even after averaging the results across multiple runs, there still appeared to be some outliers in the results, most notable in the results for the number of iterations taken to reach a solution. When reviewing the results, it was clearly observable that there was in fact, a point 64 CHAPTER 6. CONCLUSION where a distributed swarm implementation outperformed a centralised implementation in terms of TTS. This does however only occur when the number of particles reaches an incredibly high OTP. The results pointed to

SA C13358011_MSC_ASD.final.pdf (D108754758)

48/50	SUBMITTED TEXT	228 WORDS	94%	MATCHING TEXT	228 WORDS

being around 500,000 particles, with this being roughly the same across three di erent tness functions. We can therefore con rm that the alternate hypothesis is correct, and there is a point whereby the TTS when using a distributed network of swarms exceeds the time lost from the network communications. Indeed, at the highest level of particles used in the experiment, 1,000,000 particles, the distributed implementation displayed a greater than 100% reduction in TTS when compared to the centralised implementation 6.4 Contributions and Impact The main contribution of this research piece is to ll in the research gap posed in Chapter Two. This research piece focused in on benchmarking two PSO implementa- tions, a distributed and centralised implementation, in order to asses if there was an OTP whereby a distributed implementation is more e cient than a centralised imple- mentation. Additionally, an original distributed and centralised PSO implementation was developed for this project that could aid in future work. Where this could have the most impact is for within a more practical experiment of a PSO, where very high levels of particles are required, possibly leaning towards more of a simulation problem than an optimisation problem. Chapter Two presented and cited many forms of robotics or simulated robotic PSO experiments, this research piece may help future simulated robotic experiments by highlighting the threshold where it becomes more time e cient to run a distributed im- plementation over a centralised implementation. Researchers may be able to

SA C13358011_MSC_ASD.final.pdf (D108754758)

the need to distribute their simulated PSO implementation, rather than benchmarking their system themselves. 65 CHAPTER 6. CONCLUSION 6.5 Future Work & Recommendations • This work could be extended by adapting the implementation into alternate languages or frameworks. It was mentioned previously that utilising REST end- points may have incurred additional overheads when calculating the TTS for the distributed implementations. Utilising a di erent network communication method may nd additional gains to TTS than highlighted in this report. Addi- tionally, implementing the code in a di erent language may provide a di erent OTP than the one noted in this study. • Testing more tness functions to see if the OTP remains true. Additionally some more complex simulation problems could be tested to nd variations in the OTP. • Running the same experiment against more powerful machines, in terms of RAM and CPU may highlight more trends in how performant a distributed PSO im- plementations is. 66 References Abraham, A., Guo, H., & Liu, H. (2006). Swarm intelligence: Foundations, perspec- tives and applications. Swarm Intelligent Systems Studies in Computational Intelli- gence, 3{25. doi: 10.1007/978-3-540-33869-7 1 Akat, S. B., & Gazi, V. (2008). Decentralized asynchronous particle swarm optimiza- tion. 2008 IEEE Swarm Intelligence Symposium. doi: 10.1109/sis.2008.4668304 Azab, S. S., Hady, M. F., & Hefny, H. A. (2016). Local best particle swarm optimiza- tion for partitioning data clustering. 2016 12th International Computer Engineering Conference (ICENCO). doi: 10.1109/icenco.2016.7856443 Bai, Q. (2010). Analysis of particle swarm optimization algorithm. Computer and Information Science, 3(1). doi: 10.5539/cis.v3n1p180 Bassimir, B., Schmitt, M., & Wanka, R. (2020). Self-adaptive potential-based stop- ping criteria for particle swarm optimization with forced moves. Swarm Intelligence, 14(4), 285{311. Beni, G. (2005). From swarm intelligence to swarm robotics. Swarm Robotics Lecture Notes in Computer Science, 1{9. doi: 10.1007/978-3-540-30552-1 1 Bingham, D. (n.d.). Virtual library of simulation experiments:. Retrieved from https://www.sfu.ca/ ~ ssurjano/beale.html Blum, C., & Li, X. (n.d.). Swarm intelligence in optimization. Natural Computing Series Swarm Intelligence, 43{85. doi: 10.1007/978-3-540-74089-6 2 67 REFERENCES Bouchrika, I. (2020, Aug). Primary research vs secondary research: De nitions, di erences, and examples. Retrieved from https://www.guide2research.com/ research/primary-research-vs-secondary-research Cooren, Y., Clerc, M., & Siarry, P. (2009). Performance evaluation of tribes, an adaptive particle swarm optimization algorithm. Swarm Intelligence, 3(2), 149{178.

He, Y., Ma, W. J., & Zhang, J. P. (2016). The parameters selection of pso algorithm in uencing on performance of fault diagnosis. In Matec web of conferences (Vol. 63, p. 02019). Hereford, J. (2006). A distributed particle swarm optimization algorithm for swarm robotic applications. 2006 IEEE International Conference on Evolutionary Compu- tation. doi: 10.1109/cec.2006.1688510 Imranª, M., Hashima, R., & Abd Khalidb, N. E. (2013). An overview of particle swarm optimization variants. Procedia Engineering, 53, 491{496. Jetmarov a, B. (2011). Comparison of best practice benchmarking models. Problems of Management in the 21st Century, 2, 76. Kan, W., & Jihong, S. (2012). The convergence basis of particle swarm optimization. In 2012 international conference on industrial control and electronics engineering (pp. 63{66). Kennedy, J. (1999). Small worlds and mega-minds: e ects of neighborhood topology on particle swarm performance. Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). doi: 10.1109/cec.1999.785509 Kennedy, J., & Eberhart, R. (1995a). Particle swarm optimization. IEEE Interna- tional Conference on Neural Networks (ICNN). Kennedy, J., & Eberhart, R. (1995b). Particle swarm optimization. In Proceedings of icnn'95-international conference on neural networks (Vol. 4, pp. 1942{1948). 68 REFERENCES Kennedy, J., & Mendes, R. (2002). Population structure and particle swarm perfor- mance. In Proceedings of the 2002 congress on evolutionary computation. cec'02 (cat. no. 02th8600) (Vol. 2, pp. 1671{1676). Li, H., Yang, D., Su, W., Lu, J., & Yu, X. (2019). An overall distribution particle swarm optimization mppt algorithm for photovoltaic system under partial shading. IEEE Transactions on Industrial Electronics, 66(1), 265{275. doi: 10.1109/tie.2018 .2829668 Lynn, N., Ali, M. Z., & Suganthan, P. N. (2018). Population topologies for par- ticle swarm optimization and di erential evolution. Swarm and Evolutionary Com- putation, 39, 24-35. Retrieved from https://www.sciencedirect.com/science/ article/pii/S2210650217308805 doi: https://doi.org/10.1016/j.swevo.2017.11 .002 McLeod, S. (2019). Qualitative vs quantitative research: Simply psychology. Retrieved from https://www.simplypsychology.org/qualitative-quantitative.html Meng, Y., & Gan, J. (2008). A distributed swarm intelligence based algorithm for a cooperative multi-robot construction task. 2008 IEEE Swarm Intelligence Sympo- sium. doi: 10.1109/sis.2008.4668296 Moradi, A., & Fotuhi-Firuzabad, M. (2008). Optimal switch placement in distribution systems using trinary particle swarm optimization algorithm. IEEE Transactions on Power Delivery, 23(1), 271{279. doi: 10.1109/tpwrd.2007.905428 Nouiri, M., Bekrar, A., Jemai, A., Niar, S., & Ammari, A. C. (2015). An e ective and distributed particle swarm optimization algorithm for exible job-shop scheduling problem. Journal of Intelligent

Manufacturing, 29(3), 603{615. doi: 10.1007/s10845 -015-1039-3 Oca, M. M. D., Stutzle, T., Birattari, M., & Dorigo, M. (2009). Frankenstein's pso: A composite particle swarm optimization algorithm. IEEE Transactions on Evolutionary Computation, 13(5), 1120{1132. doi: 10.1109/tevc.2009.2021465 69 REFERENCES Peleg, D. (2005). Distributed coordination algorithms for mobile robot swarms: New directions and challenges. Distributed Computing { IWDC 2005 Lecture Notes in Computer Science, 1{12. doi: 10.1007/11603771 1 Piotrowski, A. P., Napiorkowski, J. J., & Piotrowska, A. E. (2020). Population size in particle swarm optimization. Swarm and Evolutionary Computation, 58, 100718. doi: 10.1016/j.swevo.2020.100718 Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. Swarm intelligence, 1(1), 33{57. Raquel, C. R., & Naval Jr, P. C. (2005). An e ective use of crowding distance in mul- tiobjective particle swarm optimization. In Proceedings of the 7th annual conference on genetic and evolutionary computation (pp. 257{264). Sahin, F., Yavuz, M. C ., Arnavut, Z., & Uluyol, O. (2007). Fault diagnosis for airplane engines using bayesian networks and distributed particle swarm optimization. Parallel Computing, 33(2), 124{143. Salza, P., & Ferrucci, F. (2019). Speed up genetic algorithms in the cloud using software containers. Future Generation Computer Systems, 92, 276{289. doi: 10 .1016/j.future.2018.09.066 Schutte, J. F., Reinbolt, J. A., Fregly, B. J., Haftka, R. T., & George, A. D. (2004). Parallel global optimization with the particle swarm algorithm. International journal for numerical methods in engineering, 61(13), 2296{2315. Shi, Y. (2004, Feb). Particle swarm optimization. IEEE Neural Networks Society. Shi, Y., & Eberhart, R. C. (1998). Parameter selection in particle swarm optimiza- tion. In International conference on evolutionary programming (pp. 591{600). S a, A. O. D., Nedjah, N., & Mourelle, L. D. M. (2016). Distributed e cient localiza- tion in swarm robotic systems using swarm intelligence algorithms. Neurocomputing, 172, 322{336. doi: 10.1016/j.neucom.2015.03.099 70 REFERENCES Trelea, I. C. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. Information Processing Letters, 85(6), 317{325. doi: 10 .1016/s0020-0190(02)00447-7 Vandenbergh, F., & Engelbrecht, A. (2004). A cooperative approach to parti- cle swarm optimization. IEEE Transactions on Evolutionary Computation, 8(3), 225{239. doi: 10.1109/tevc.2004.826069 Vokolos, F. I., & Weyuker, E. J. (1998). Performance testing of software systems. In Proceedings of the 1st international workshop on software and performance (pp. 80{87). Wang, D., Tan, D., & Liu, L. (2017). Particle swarm optimization algorithm: an overview. Soft Computing, 22(2), 387{408. doi: 10.1007/s00500-016-2474-6 Wang, Z.-J., Zhan, Z.-H., Kwong, S., Jin, H., & Zhang, J. (2020). Adaptive gran- ularity learning distributed particle swarm optimization for

large-scale optimization. IEEE transactions on
cybernetics. Xia, X., Gui, L., & Zhan, Z.-H. (2018). A multi-
swarm particle swarm optimization algorithm based on
dynamical topology and purposeful detecting. Applied
Soft Com- puting, 67, 126-140. Retrieved from
https://www.sciencedirect.com/science/
article/pii/S1568494618301017 doi:
https://doi.org/10.1016/j.asoc.2018.02.042 Xie, X.-f.,
Zhang, W.-j., & Yang, Z.-l. (2003). Overview of particle
swarm optimiza- tion. Control and Decision, 18(2),
129{134. Yin, P.-Y., Glover, F., Laguna, M., & Zhu, J.-X.
(2010). Cyber swarm algorithms{ improving particle
swarm optimization using adaptive memory strategies.
European Journal of Operational Research, 201(2),
377{389. Yin, P.-Y., Yu, S.-S., Wang, P.-P., & Wang, Y.-T.
(2006). A hybrid particle swarm optimization algorithm for
optimal task assignment in distributed systems. Computer
Standards & Interfaces, 28(4), 441{450. doi:
10.1016/j.csi.2005.03.005 71 REFERENCES Zhang, W., Ma,
D., jun Wei, J., & feng Liang, H. (2014). A parameter
selection strategy for particle swarm optimization based
on particle positions. Expert Systems with Applications,
41(7), 3576-3584. Retrieved from
https://www.sciencedirect
.com/science/article/pii/S0957417413008932 doi:
https://doi.org/10.1016/j .eswa.2013.10.061 Zheng, J., Wu,
Q., & Song, W. (2007). An improved particle swarm
algorithm for solving nonlinear constrained optimization
problems. In Third international confer- ence on natural
computation (icnc 2007) (Vol. 4, pp. 112{117). Zhou, Y., &
Tan, Y. (2009). Gpu-based parallel particle swarm
optimization. In 2009 ieee congress on evolutionary
computation (pp. 1493{1500). Zielinski, K., & Laur, R.
(2007). Stopping criteria for a constrained single-objective
particle swarm optimization algorithm. Informatica, 31(1).
72 Appendix A Code Snippets Listing A.1 shows and
example data output produced for the booths tness
function from a distributed and centralised PSO
implementation. 1 { 2 "finalGroupBest": 0.0, 3
"averageBestX": 1.0000000000000003, 4 "averageBestY":
3.0, 5 "iterations": 151, 6 "timeToSolution": 1303, 7
"resultsList": [{ 8 "swarmId": 0, 9 "results": [{ 10 "step": 0, 11
"best": 20.288124656313526, 12 "bestPositionX": 0.0, 13
"bestPositionY": 0.0 14 }] 15 }] 16 } Listing A.1: Data output
example 73 APPENDIX A. CODE SNIPPETS Listing A.2
shows an extract of the data aggregation code. Code to
retrieve the results from the le system, and aggregate and
average them is visible. Code to output the resulting text
string is omitted, but is a straightforward process of
creating a text le using the variable outputString. 1 var
outputString = "SWARM NUMBER,PARTICLE
NUMBER,CENTRALISED ITERATIONS, ,! CENTRALISED
TIME TO SOLUTION,CENTRALISED FINAL GROUP BEST, ,!
DISTRIBUTED ITERATIONS,DISTRIBUTED TIME TO
SOLUTION, ,! DISTRIBUTED FINAL GROUP BESTnn"; 2 3

for (var swarmNumber = 1; swarmNumber >= numberOfSwarms; swarmNumber++) f 4 for (var particleNumber = 1000; particleNumber >= maxNumberParticles; particleNumber ,! += particleIncrement) f 5 var leUrlCentralised RUN1 = './Results/NON DISTRIB/RUN 1/' + functionName + '/' ,! + functionName + ' ' + swarmNumber + ' ' + particleNumber + '.result.json'; 6 .... 7 var leUrlDistributed RUN1 = './Results/DISTRIB/RUN 1/' + functionName + '/' + ,! functionName + ' ' + swarmNumber + ' ' + particleNumber + '.result.json'; 8 .... 9 let centralised RUN1 = fs.readFileSync(leUrlCentralised RUN1); 10 ... 11 let centralised data RUN1 = JSON.parse(centralised RUN1); 12 ... 13 var averageCentralisedIterations = Math.round((centralised data RUN1.iterations + ,! centralised data RUN2.iterations + centralised data RUN3.iterations)/3); 14 var averageCentralisedTTS = Math.round((centralised data RUN1.timeToSolution + ,! centralised data RUN2.timeToSolution + centralised data RUN3.timeToSolution) ,! /3); 15 var averageCentralisedGbest = Math.round((centralised data RUN1. nalGroupBest + ,! centralised data RUN2. nalGroupBest + centralised data RUN3. nalGroupBest) ,! /3); 16 ... 17 18 outputString = outputString + swarmNumber + "," + particleNumber + "," + ,! averageCentralisedIterations + "," + averageCentralisedTTS + "," + ,! averageCentralisedGbest + ","; 74 APPENDIX A. CODE SNIPPETS 19 ..... 20 /* 21 Generate the same output string for distributed test run data 22 */ 23 g 24 /* 25 Write outputString to le system. 26 */ Listing A.2: Data aggregation Implementation Listing A.3 shows an extract of the Newman API implementation code. Code to load the request le and environment variables can be seen, along with the code to write the result to the le system the format discussed in Chapter Three. 1 function runNewmanCode(envName, testCollectionName, environmentName, numberOfSwarms, ,! numberOfParticles, baseUrlsSting, endpoint) f 2 newman.run(f 3 collection: require('./Requests/' + testCollectionName + '.postman collection.json'), 4 environment: require('./EnvVariables/' + environmentName + '.postman environment.json ,! '), 5 reporters: 'cli', 6 globalVar: [f "key":"NumberOfSwarms", "value":numberOfSwarms g, f"key":"endpoint", " ,! value": endpointg, f "key":"NumberOfParticles", "value": numberOfParticlesg, f " ,! key":"BaseUrls", "value": baseUrlsStingg] 7 g).on('request', function (error, args) f 8 ... 9 else f 10 ... 11 fs.writeFile(dir + testCollectionName + " " + numberOfSwarms + " " + ,! numberOfParticles + '.result.json', args.response.stream, function (error) f 12 ... 13 g); 14 g 15 g); 16 g Listing A.3: Newman Implementation Code 75 APPENDIX A. CODE SNIPPETS Listing A.4 shows a partial extract of the distributed implementation service class. The swarm looping code can be seen, along with the stopping criteria code and the response formatting methods. 1 public Response runDistributedCallingSystem(int numberOfParticles, List>String< baseUrls, ,! Con gVariables con gVariables) f 2 Instant start = Instant.now(); 3 4 for(int swarmId=0; swarmId > numberOfSwarms; swarmId++)f 5 ... 6 callInitialiseSwarms(swarmId, baseUrls.get(swarmId), distributedRequest); 7 g 8 9 ... 10 11 while(!targetFound) f 12 double[] bestEvaluation = new double[numberOfSwarms]; 13 for(int swarmId = 0; swarmId>numberOfSwarms; swarmId++) f 14 DistributedRequest distributedRequest = new DistributedRequest(); 15 distributedRequest.setNumberOfParticles(numberOfParticles); 16 distributedRequest.setSwarmId(swarmId); 17 distributedRequest.setCon gVariables(con gVariables); 18 Result result = callRunSwarm(swarmId, baseUrls.get(swarmId) ,distributedRequest); 19 multiSwarmResults.get(swarmId).getResults().add(result); 20 bestEvaluation[swarmId] = result.getBest(); 21 g 22 23 for(int swarmId = 0; swarmId>numberOfSwarms; swarmId++) f 24 if (settledList.size() == 0) f 25 settledList.add(bestEvaluation[swarmId]); 26 g else f 27 if (settledList.contains(bestEvaluation[swarmId])) f 28 settledList.add(bestEvaluation[swarmId]); 29 g else f 30 settledList = new ArrayList><(); 31 g 32 g 76 APPENDIX A. CODE SNIPPETS 33 34 if (settledList.size() == TARGET NUMBER OF EXACT ANSWERS) f 35 ... 36 targetFound = true; 37 g 38 g 39 if(stepCount == MAX NUMBER OF STEPS)f 40 targetFound =true; 41 g 42 43 stepCount++; 44 g 45 46 Instant nish = Instant.now(); 47 long timeElapsed = Duration.between(start, nish).toMillis(); 48 49 return utils.createResponse(multiSwarmResults, stepCount, 50 timeElapsed, numberOfSwarms); 51 g Listing A.4: Distributed Service Class Listing A.5 shows a partial extract of the centralised implementation service class. The swarm looping code can be seen, along with the stopping criteria code. 1 public Response runSwarm(Request request) throws Exception f 2 Instant start = Instant.now(); 3 ... 4 List>Swarm< multiSwarm = new ArrayList><(); 5 ... 6 /* Initialize all swarms and particle with supplied con guration */ 7 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 8 multiSwarm.add(utils.createSwarm(swarmId, request.getNumberOfParticles(), request. ,! getCon gVariables())); 9 g 10 11 while(!targetFound) f 12 ... 77 APPENDIX A. CODE SNIPPETS 13 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); swarmId++) f 14 Result result = utils.runSwarmIteration(multiSwarm.get(swarmId), stepCount); 15 ... 16 g 17 18 /* Check if the same answer has been found multiple times */ 19 for(int swarmId = 0; swarmId>request.getNumberOfSwarms(); 20 swarmId++) f 21 if (settledList.size() == 0) f 22 settledList.add(bestEvaluation[swarmId]); 23 g else f 24 if (settledList.contains(bestEvaluation[swarmId])) f 25 settledList.add(bestEvaluation[swarmId]); 26 g else f 27 settledList = new ArrayList><(); 28 g 29 g 30 31 if (settledList.size() == TARGET NUMBER OF EXACT ANSWERS) f 32 TARGET NUMBER OF EXACT ANSWERS); 33 targetFound = true; 34 g 35 g 36 37 if(stepCount == MAX NUMBER OF STEPS)f 38 targetFound =true; 39 g 40 41 stepCount++; 42 g 43 Instant nish = Instant.now(); 44 long timeElapsed = Duration.between(start, nish).toMillis(); 45 ... 46 g Listing A.5: Centralised Service Class 78 Appendix B Data Results Table B.1 shows the full averaged result set for the centralised Easom's function TTS. Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 2364 2297 2313 2388 200000 4578 4449 4313 4783 300000 6681 6731 6694 8033 400000 9776 9779 9749 11264 500000 12875 13835 13176 16012 600000 19400 18945 18825 21732 700000 24188 23920 23905 27103 800000 28344 27520 30373 33907 900000 36510 38057 36939 47329 1000000 50887 51263 52918 78160 Table B.1: Centralised Easom'

SA C13358011_MSC_ASD.final.pdf (D108754758)

50/50	SUBMITTED TEXT	343 WORDS	93%	MATCHING TEXT	343 WORDS

Results Table B.2 shows the full averaged result set for the centralised Booth's function TTS. 79 APPENDIX B. DATA RESULTS Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 2910 3243 3634 2805 200000 6160 6802 6481 5986 300000 9857 10398 10425 10185 400000 14798 15180 15161 15400 500000 21596 21013 21246 23378 600000 34828 33524 34117 36270 700000 43639 42629 42271 45328 800000 52655 51940 55802 58774 900000 70949 70174 69936 83288 1000000 102709 103364 102856 131270 Table B.2: Centralised Booth's TTS Table B.3 shows the full averaged result set for the centralised Beale's function TTS. Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 4205 3842 4775 4417 200000 8828 8676 9276 9294 300000 14003 13934 14489 14964 400000 19742 19301 20336 22167 500000 27418 27981 27615 33411 600000 39291 40426 40594 45841 700000 52039 51825 49939 56124 800000 60327 61134 65653 70674 900000 78404 81726 79632 98879 1000000 112258 119800 112751 160899 Table B.3: Centralised Beale's TTS Table B.4 shows the full averaged result set for the distributed Easom's function TTS. 80 APPENDIX B. DATA RESULTS Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 3551 3287 3925 4385 200000 2545 5154 6217 5449 300000 7217 7583 8228 8556 400000 9145 7434 9961 11039 500000 12404 11663 12298 13091 600000 14614 14158 14661 15548 700000 17662 16690 17383 17736 800000 20410 19008 14769 20474 900000 24803 21444 22707 22619 1000000 29327 23923 24623 26300 Table B.4: Distributed Easom's TTS Table B.5 shows the full averaged result set for the distributed Booth's function TTS. Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 4331 4846 6197 7541 200000 4092 8195 8942 9957 300000 11497 11190 11917 15068 400000 15046 11951 14992 19667 500000 19193 19601 18699 24308 600000 23894 22650 22266 28513 700000 27431 28039 26392 33023 800000 33195 31968 21680 38411 900000 39787 35334 34058 43588 1000000 48322 39878 36861 49330 Table B.5: Distributed Booth's TTS Table B.6 shows the full averaged result set for the distributed Beale's function TTS. 81 APPENDIX B. DATA RESULTS Particle Number TTS-2 Swarms TTS-4 Swarms TTS-8 Swarms TTS-10 Swarms 100000 5291 5630 7153 7860 200000 7370 10308 10988 10223 300000 13213 13959 15164 16299 400000 18240 19389 20080 20567 500000 23625 24069 24213 25084 600000 29258 27959 28980 29629 700000 35314 33992 33915 33713 800000 40600 39246 39852 39174 900000 49349 45128 44175 44859 1000000 56502 48906 49696 51248 Table B.6: Distributed Beale's TTS 82

SA C13358011_MSC_ASD.final.pdf (D108754758)

[illegible]