UNIVERSITY OF CAPE TOWN

Self-Adapting Simulated Artificial Societies

by

Brandon Gower-Winter

A dissertation submitted in partial fulfillment for the
degree of Masters in Computer Science

in the
Faculty of Science
Department of Computer Science

November 2022

Declaration of Authorship

I, Brandon Gower-Winter, declare that this dissertation titled, 'Self-Adapting Simulated Artificial Societies' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 11 November 2022

Abstract

Agent-Based Models (ABM) are computational models that utilize autonomous agents to interact and adapt to the environments in which they occupy. They are used in fields ranging from Economics to Ecology. More recently, ABM are being used in Computational Archaeology to aid in explaining the complex social phenomena that gave rise to ancient societies all over the world.

Despite their potential, ABM are limited by the fact their agents are rarely adaptive despite adaptability often touted as one of Agent-Based Modelling's greatest strengths. In this work we remedy this by investigating whether Machine Learning (ML) algorithms can be used as adaptive mechanisms for Agent-based Models simulating complex social phenomena. We aim to do this by comparing ML agents, developed using Reinforcement Learning and two Evolutionary Algorithms as adaptive-mechanisms, to rule-based agents typically found in contemporary literature.

To achieve this, we create *NeoCOOP*, an Agent-Based Model designed to simulate the complex social phenomena that arise from resource sharing agents in ancient societies. By conducting scenario experimentation, we examined the adaptive capacity of our four agent-types by measuring their ability to maintain both population and resources levels in a virtual re-creation of Ancient Egypt during the Predynastic Period. Our results indicate that our ML agents (*Utility* and *IE*) perform better or on par with even complex rule-based agents (*Traditional* and *RBAdaptive*). The *IE* agent-type ranked first and was the most adaptive agent-type. The *Utility* and *RBAdaptive* agents jointly ranked second and the *Traditional* agent ranked last.

Overall, the findings of this work clearly show that adaptive-agents are more suited to modelling the dynamics of complex environments than their rule-based counterparts. More specifically, our results demonstrate that ML algorithms are particularly well suited as these adaptive mechanisms given that they not only allowed our agents to maintain high population and resource levels, they facilitated the emergence of additional emergent phenomena such as resource acquisition strategy specialization. It is our hope that the findings presented in this work pushes the state of the art such that future research endeavours seek to use truly adaptive-agents in their complex Archaeological ABM.

Acknowledgements

I would like to thank Tristan, Natelie, Eric, Ryan and Stephanie for their support during this research endeavour.

I would also like to thank my supervisor Associate Professor Geoff Nitschke for the guidance and constructive criticism he provided over the course of this dissertation.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ABM	**A**gent-**B**ased **M**odel
ACO	**A**nt-**C**olony **O**timization
ALIFE	**A**rtificial **Life**
ANN	**A**rtificial **N**eural **N**etwork
BDI	**B**elief **D**esire **I**ntention
CA	**C**ultural **A**lgorithm
EA	**E**volutionary **A**lgorithm
ECS	**E**ntity **C**omponent **S**ystem
GA	**G**enetic **A**lgorithm
GIS	**G**eographic **I**nformation **S**ystem
IE	**I**nformation **E**xchange
KIDS	**K**eep **I**t **D**escriptive **S**tupid
KISS	**K**eep **I**t **S**imple **S**tupid
ML	**M**achine **L**earning
NeoCOOP	**Neo**lithic **Coop**eration Model
ODD	**O**verview **D**esign-Concepts and **D**etail
PECS	**P**hysis **E**motion **C**ognition and **S**tatus
RL	**R**einforcement **L**earning
SES	**S**ociological-**E**cological **S**ystems
UML	**U**nified **M**odelling **L**anguage

Nomenclature

Ancient Egypt A Northeast African Ancient Civilization situated in the Nile Valley that formed around 3200-3100 BC.

Badarian Predynastic Egyptian archaeological culture. Preceding the Naqada culture, the Badarian culture provides some of the earliest direct evidence of agricultural practices in Upper Egypt.

Delta Large flat plain lying north of Cairo (ancient Memphis) and drained by the Nile river. Area also known as Lower Egypt.

Early Dynastic Synonym for Archaic Period, Dynasties 1 and 2.

Late Predynastic Synonym for Naqada II (Gerzean) Period.

Lower Egypt The Delta.

Maadi-Buto Predynastic Egyptian Culture from Lower Egypt. The Maadi-Buto Culture preceded the Naqada III culture that slowly dominated Lower Egypt towards the end of the Predynastic Period.

Naqada Chronology Periods in Predynastic Egyptian history. The Naqada Chronology has three main divisions (Naqada I/II/III) each with further subdivisions (A/B/C/D).

Naqada Culture Predynastic Egyptian archaeological culture named after the town of Naqada situated in Upper Egypt.

Neolithic revolution Term applied to the apparent rapid spread of a Neolithic lifestyle throughout the ancient world.

Neolithic Period when evidence of domestication (plants or animals) can be determined. In Egypt the Neolithic precedes the Predynastic.

Nome Pre-State Large political Upper Egyptian entity consisting of aggregations of proto-nomes.

Palaeolithic Old Stone Age, a general reference to that period prior to the domestication of plants and animals.

Pre-Nome Independent local village in Upper Egypt characterized by its political autonomy.

Predynastic Period that followed the Neolithic period in Egypt. Sometimes referred to as Prehistoric Egypt, this period consisted of the unification of both Lower and Upper Egypt under a single state identity. The Predynastic preceded the Early Dynastic Period.

Proto-Nome The first composite political units of Upper Egypt consisting of aggregations of previously autonomous local villages.

Protodynastic The Early Dynastic Period.

rand() Shorthand for sampling a random value between 0 and 1 from a uniform distribution.

Social complexity (stratification) Term used to describe a culture with multiple social classes, often used as a synonym for civilization; a socially complex and economically diverse culture.

The Upper Egyptian Proto-State An economic and political unit constructed from previous nome pre-states. Sometimes called the "Upper Egyptian commonwealth", this political entity was ideologically, economically, and militarily “glued” to the most powerful polity, likely situated in Hierakonpolis or Abydos.

Upper Egypt Southern Egypt, traditionally that area south of Cairo. When used with Middle Egypt, it refers to that area south of Asyut.

Valley Used in reference to the entire Egyptian Nile Valley, but can refer specifically to Upper Egypt, that area south of Cairo.

Several definitions taken from Brewer and Teeter’s "Egypt and the Egyptians" [11] and Anđelković’s "Political Organization of Egypt in the Predynastic Period" [12].

Chapter 1

Introduction

Agent-Based Models (ABM) are dynamic computational models that utilize autonomous agents capable of interacting with other agents and adapting to novel situations within the simulations they occupy [13]. ABM have been used in numerous research fields ranging from Chemistry [14] and Epidemiology [15] to Economics [16] and Archaeology [17] (See Figure 1.1 for a few visual references). ABM take a bottom-up approach to modelling. This makes them the accepted approach to modelling emergent behaviour, that is, the observed macro-behaviours of a system that result from the micro-behaviours of the agents themselves.

The nature in which ABM create emergent behaviour is of interest to those in the Social Sciences, particularly those concerned with the modelling of human behaviour. In contrast to their mathematical counterparts, ABM allow for the modelling of heterogeneous autonomous agents with bounded rationality. This makes them apt candidates for modelling complex social processes. This fact, and the conceptual accessibility of ABM, has led to the rapid adoption of ABM in the Social Sciences. This is especially true for the field of Archaeology [18] which, up until the development of ABM, did not have a method for observing the emergence of complex social phenomena through the lens of individual behaviours and interactions.

This is further demonstrated in Figure 1.2a where we see that archaeological finds can be viewed as snapshots into the past. Theories are then manifested to explain how these societies or communities transitioned from snapshot to another. For the most part, these theories are conjecture. However, ABM provide an empirical framework by which these

theories can be tested. Furthermore, theories are subject to the concept of Equifinality which states there may, in fact, be several theories which reasonably explain or produce the observed transition from archaeological snapshot to another. While ABM do not inherently solve this problem, researchers can develop an ABM for each of these theories and compare the results they produce against other ABM and the archaeological record [19] (See Figure 1.2b).

However, ABM are relatively new to the Social Sciences (Game theoretic cooperation dynamics are a notable exception to this) and, as such, are not without their limitations. These limitations can be broadly categorized as follows:

1. **ABM are black-boxes**. That is to say that the nature in which ABM are designed and implemented are often opaque with the reproducibility of any given ABM dependant on the descriptive prowess of its creators [20].

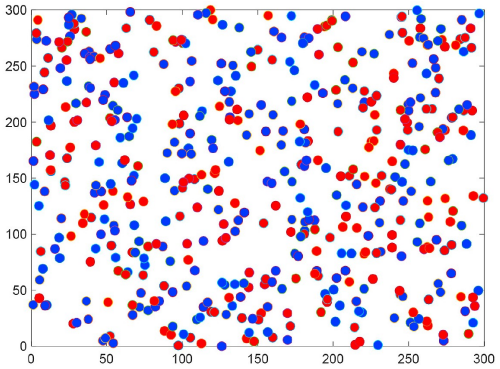
2. **ABM are unpredictable**. This is largely to do with the fact that ABM are constructed in a bottom-up manner [21]. Not only does this make them sensitive to different kinds of input but the ramifications of adding or removing particular systems are typically intangible until the model is run.

3. **ABM suffer at scale**. For every agent and system added, there is an additional computational cost. There is a practical upper-limit to this cost which "large-scale" ABM often reach [22]. This means that modellers often need to simplify their models or employ complex parallelization techniques in order to run their models in acceptable time-frames.

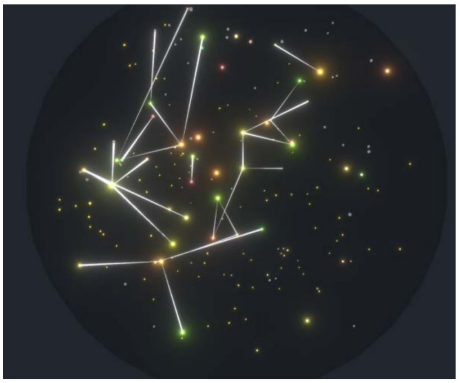
4. **ABM often lack adaptive mechanisms** despite adaptability being touted as one of ABM's biggest strengths [4, 13].

1.1 Motivations

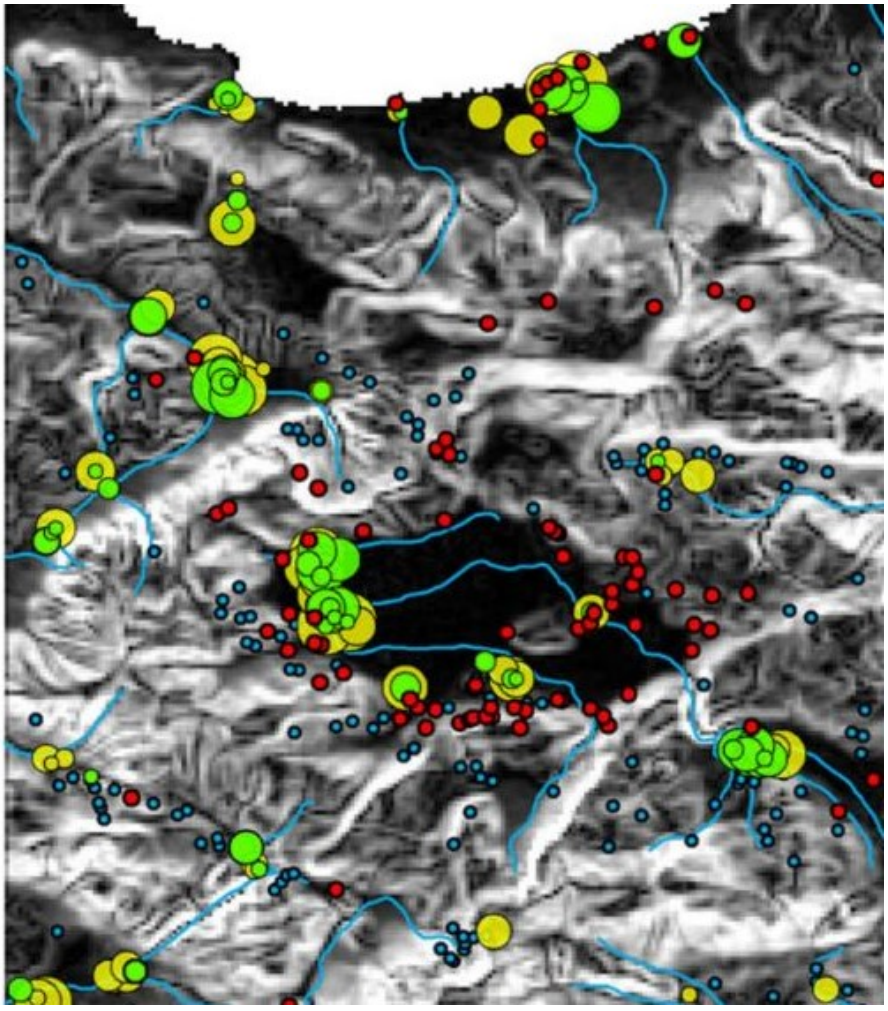
Of these four issues, research pertaining to adaptive-ABM is noticeably absent and therefore the primary focus of our research. The lack of adaptive-ABM and adaptive-ABM literature is actually quite surprising. In the context of Agent-Based Modelling, adaptability refers to the ability of agents to create new strategies about how to make decisions.



(A) A model of showcasing the transmissibility of a virus. Taken from Cuevas [23], the blue and red dots showcase susceptible and infected agents respectively.



(B) A visual representation of a social network of resource sharing agents. Taken from Molin et al. [24], green nodes represent agents with more resources while red nodes represent agents with no resources.



(C) A model showcasing settlement distribution of the ancient Minoan civilization on Crete Island. Taken from Chliaoutakis and Chalkiadakis [25], the green and yellow circles are settlements while the red dots represent actual archaeological dig sites.

FIGURE 1.1: Figures showcasing various ABM.

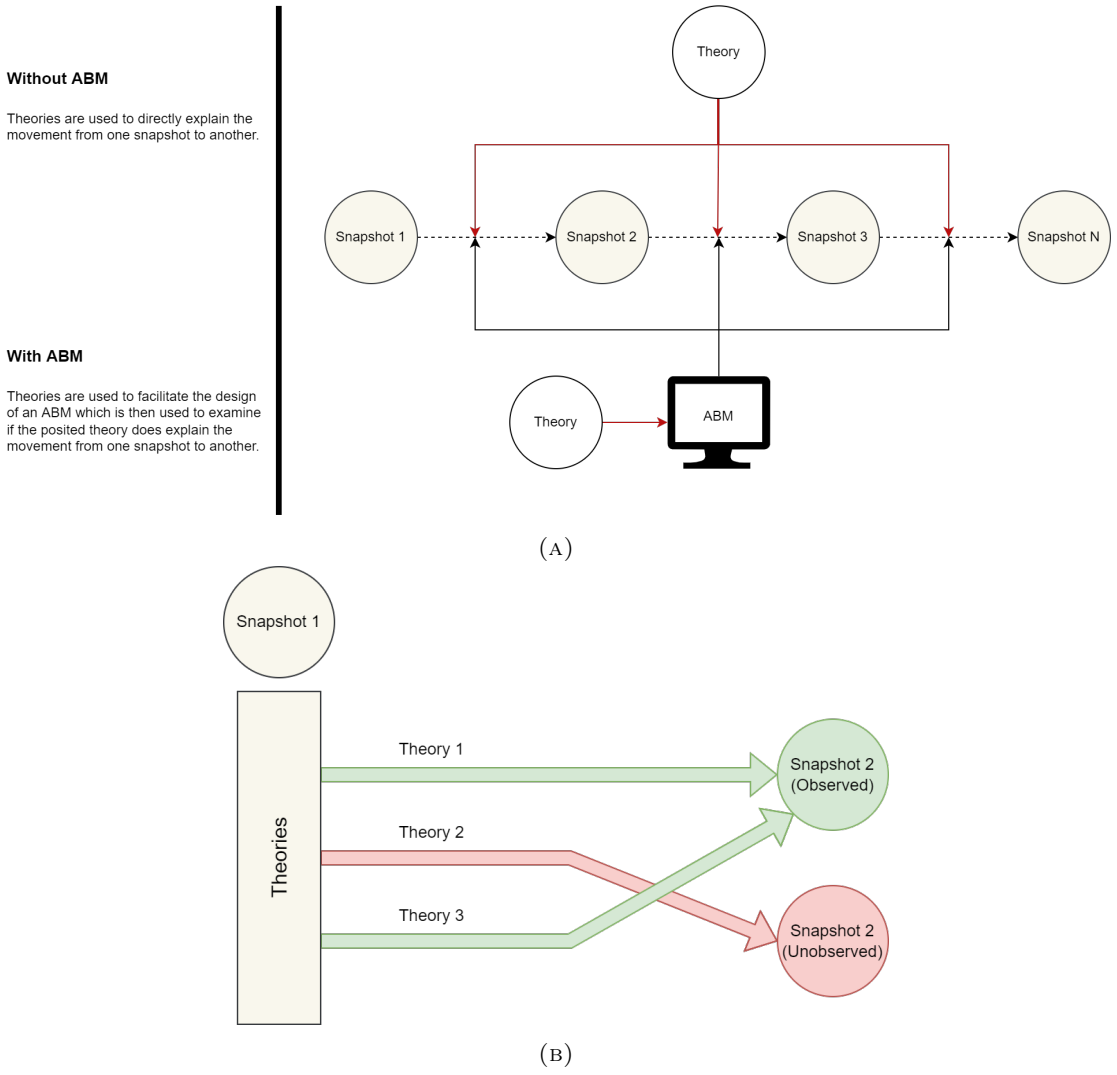


FIGURE 1.2: Figures of the various topics surrounding the usage of ABM in Computational Archaeology. (a) Demonstrates how ABM are used to provide empirical evidence to support theoretical constructions of snapshot transitions while (b) illustrates the concept of Equifinality which states that multiple theories (indicated in green) might explain the transition from one snapshot to another.

With regards to the modelling of complex social processes, adaptability, both evolutionary and individually, is a fundamentally human trait [26] that a traditional rule-based ABM is rarely capable of achieving.

Despite the lack of adaptive-mechanisms in ABM literature, Machine Learning (ML) has been proposed as a potential solution to this problem. Machine Learning, the programming of computers to optimize a criterion using example data or past experience [27], is capable of introducing adaptive mechanisms by which agents, and the ABM, can learn and adapt [4]. These techniques can be applied at two different scales. Either they are applied to the population of agents, in the case of Genetic and Cultural Algorithms [28],

or they are applied to the agent itself, in the case Reinforcement Learning [29]. Additionally, ML techniques can also be used to parameter tune [30] as well as reduce the complexity of ABM [31].

In order to effectively evaluate the benefits brought upon by the inclusion of ML adaptive-mechanisms, a sufficiently complex contextual backdrop is needed. The motivation for this statement are two-fold. Firstly, a "Toy-box" model [32] is unlikely to illuminate the deficiencies of rule-based agents simply due to the fact that the environment would be too simple. Secondly, ABM are slowly becoming more data-oriented which has resulted in an explosion of model complexity (See Chapter 2). Thus, it would be appropriate to illustrate the benefits of adaptive-agents in a state-of-the-art environment.

Given this prerequisite, we have elected to model the formation of the Ancient Egyptian state during the Predynastic period. The Predynastic period marks the cultural transformation of the Egyptian people from hunter gatherers to agricultural villages [11]. What were the factors that caused this transition and how do these factors compare to other Neolithic transitions? Several natural and social factors have been attributed to this transition. The interaction of these factors and the complexity of human behaviour undoubtedly makes this entire period consist of a number of complex social processes which our proposed adaptive-ABM is poised to model.

1.2 Research Questions

In short, it is through the utilization of ML as adaptive-mechanisms and a model of Egypt during the Predynastic period (created using both archaeological data and and assumptions drawn from the data itself) that we seek to answer the following research questions:

1. Do agents using machine learning techniques as adaptive mechanisms, exhibit greater adaptive capacity (recovery and resistance of population and resource levels) [33] than traditional, rule-based, agents when placed into sufficiently adversarial environments (A model of Egypt during the Predynastic period)?

2. Are our adaptive-agents capable of producing new emergent behaviour (such as polity cycling [34] or strategy specialization [35]) that the traditional, rule-based, agents could not?

3. Using both the traditional and adaptive-agents, what insights do they provide with regards to the Predynastic agricultural revolution as it relates to the presence of the Nile floodplain [12] and desertification [36]?

There is no universal measure for Adaptability (or Adaptive Capacity in vulnerability literature). As noted by Engle [37], adaptability is really a combination of a system's perceived vulnerability and resilience. If a system is vulnerable, its adaptive capacity is reduced. Conversely, if a system's resilience is high, its adaptive capacity is high. To objectively measure the adaptability of our agent designs, we use two metrics used in resilience research [33]:

1. **Recovery:** The time taken for a variable to return to its original value after a disturbance.

2. **Resistance:** The change of a variable after a disturbance event.

An agent with greater adaptive capacity will be able to resist stress to a greater degree and, in the case were it is affected by said stress, recover faster. Recovery and resistance require the monitoring of specified data-points produced by each agent type. These simple data-points are derived directly from the properties of the model and agents themselves. The properties of interest are *population* and *resources* which are commonly used to measure culture and population dynamics in both general adaptation and resilience research (See Heckbert [38], Molin et al. [24] and Schlüter and Pahl-Wostl [39]).

The second type of data we are interested in observing are composite properties. Composite properties are derived from model or agent properties. While interesting composite properties cannot necessarily be predetermined due to the black-box nature in which ABM produce emergent behaviour, some composite properties we may be interested in include *population inequality* (a predictable composite property) and *settlement centrality* [40] (an unpredictable emergent property). Should our adaptive-agents facilitate emergent behaviour not observable in the rule-based agents (Research Question 2), composite properties will be used to understand why that was the case.

1.3 Contributions

The primary contribution of our work is a comprehensive demonstration of the capabilities of ML techniques (specifically Reinforcement Learning for individual adaptation and a Genetic and Cultural Algorithm for generational adaptation) as adaptive-mechanisms compared to traditional, rule-based, mechanisms in complex (realistic) environments. Additionally, this work contributes the following:

- The design and development of *NeoCOOP* (See Section 3.4) an ABM that simulates the complex social phenomena that arise from Neolithic-inspired resource sharing agents. We also demonstrate that *NeoCOOP* operates under a wide-range of parameter values in both Toy-box [41–43] and realistic (See Sections 4.4 and 4.5) environments showing its scalability and applicability as a modelling tool. In achieving this, we also demonstrate the applicability of the Entitiy-Component-System (ECS) design pattern (See Section 3.1) for developing adaptive-ABM.
- Given that our primary case study is that of Predynastic Egypt (See Section 3.3), our work empirically investigates the effects the natural factors of desertification [36] and the presence the Nile floodplain [12] had on the formation of Ancient Egypt during the Predynastic Period. Specifically, we investigate emergent inequality (See Section 4.6.3), settlement density (See Section 4.6.2) and the rate of agricultural adoption (See Section 4.6.1) through the lens of these aforementioned natural factors.

1.4 Outline

The dissertation is presented as follows: Chapter 2 provides an extensive review of background, related and relevant work. Chapter 3 details the development process of *NeoCOOP*, the ABM developed to study the research questions presented in this work. Chapter 4 outlines the experiment design process and the results produced by said experiments. This Chapter also includes an analysis of the outputs produced by all of the agent-types investigated in this work. Chapter 5 provides a discussion of the research conducted in this work. The dissertation concludes with Chapter 6 which summarizes our findings and highlights avenues for future work.

Chapter 2

Background and Related Work

In this chapter, we seek to explore the current state of ABM literature with a particular focus on the Social Sciences and the modelling of complex social phenomena. Our primary motivation for the review is predicated on the fact the ABM research in the Social Sciences has seen an uptick in popularity over the past couple years and we thought it pertinent to study not only the reason for the surge in popularity, but also the current limitations with the state-of-the-art.

Early on in this review process, we found that despite being touted as a core benefit of ABM, most ABM do not incorporate adaptive-mechanisms. Given this, we expanded our review to also include methods for incorporating adaptive-mechanisms into ABM. Machine Learning was identified as the most promising and our findings are presented below.

Section 2.1.1 introduces the types of ABM, Section 2.1.2 discusses the concept of the "Agent" in Agent-Based Modelling and Section 2.1.3 defines the types of environments that agents occupy. Additionally, we highlight some of the core concepts that underpin Agent-Based Modelling in the Social Sciences (Section 2.1.4), their limitations (Section 2.1.5) and we discuss the current frameworks for conducting and implementing ABM research (Section 2.1.6). Section 2.2 provides an extensive overview of Machine Learning and its relationship to ABMs. The chapter concludes with a a review of the start of the art (Section 2.3) and a final discussion (Section 2.4) that includes our own recommendations for how the state-of-the-art can be improved.

2.1 Agent-Based Modelling

Agent-Based Models (ABM) are dynamic computational models that utilize autonomous agents capable of interacting with other agents and adapting to novel situations within the virtual environments they occupy [13]. Sometimes referred to as Individual-Based Models, ABM are complex adaptive systems capable of producing emergent behaviour from simple rule-sets [44]. There are several comparable computational models such as Dynamic Decision Networks, Cellular Automata and Microsimulations, however, ABM emphasize the emergence of complex, multi-directional, interactions facilitated by individual decision making [45].

Due to their broad applicability and relative infancy in some fields, ABM do not have a universally accepted formal definition. A typical ABM usually consists of a set of agents, placed onto an artificial environment, behaving in accordance with their rule-sets and the current state of the environment [46]. Similarly, the constitution of an agent or an environment is equally as broad in scope as the ABM themselves.

2.1.1 Types of Models

As noted by Gilbert [47], ABM come in several different flavours. Namely:

2.1.1.1 Scale-Models

The real world is simultaneously rich in detail and vast is scale. This makes modelling the real world both impractical and impossible. Scale models serve to reduce both the level of detail and the complexity of a real-world model such that its implementation and execution is practical yet informative. Scale models may also be the result of, up or down, scaling some other model. A common example of model scaling in ABM is agent abstraction, the process of taking a smaller unit of agents (individuals) and grouping them as one singular agent that represents their collective decision making (e.g. household). It is worth noting that scaling an ABM can be particularly tricky given that predicting the consequences of model scaling is impossible.

2.1.1.2 Idealized-Models

Target models may be incredibly complex, consisting of a number of defining characteristics and systems. However, a particular researcher may only be interested in studying one of these characteristics or systems. Idealized models solve this problem by exaggerating some of the characteristics or systems in order to simplify the model. For example, a stock market model may assume that stock exchange is instantaneous or a model simulating the transportation of a particular product may assume that no accidents occur along any given route. Idealization may have a negative effect on the validity of a model's results, however, if used appropriately, idealized models can reduce the complexity of a target model while maintaining the validity of its results.

2.1.1.3 Analogical-Models

These models are based on an analogy drawn between a target phenomena and some better understood phenomena. These models may be successful, such as the computational model of the mind [48], but their validity is solely dependent on the adequacy of the analogy.

This is by no means an exhaustive list nor does it capture that most ABM are some combination of these sub-categories, however, it does highlight why ABM are so broadly applicable. In addition to broad applicability, ABM have three distinct advantages over other computational models. Firstly, ABM take a bottom-up approach to modelling. This makes ABM the accepted approach to modelling emergent behaviour [49]. Emergent behaviour is described as the observation of phenomena, such as self-organization and chaos, unfamiliar to classical sciences [21]. These phenomena are observed in ABM through the macro-behaviours of the system in response to the micro-behaviours of the agents themselves. Secondly, ABM are easier to conceptualize and are often a natural method for describing real-world problems. It is typically easier to describe complex behaviour using ABM than it is for most traditional models using differential equations. Lastly, the ever-popular Object-Oriented Programming paradigm naturally compliments the bottom-up approach of ABM with its emphasis on representing autonomous units as models/sub-models that are aggregated into an executable program [21].

2.1.2 What is an Agent?

The concept of an "Agent" in Agent-Based Modelling has not been universally agreed upon. An agent can be intelligent but it does not have to be. An agent may be an individual, a household or even a system that makes decisions. The broad applicability of ABM does not do them any favours either as the concept of an "Agent" may differ depending on the field of research. Fortunately, a number of features are prevalent in all agent definitions. Crooks and Castle [45] identified the commonalities among these definitions as follows:

- **Autonomy:** Agents are autonomous units that are capable of processing information as well as exchanging information with other agents in order to make independent decisions. They are free to interact with other agents. Although the simulation may restrict this if appropriate. (Prohibiting information exchange over vast distances for example)
- **Heterogeneity:** The agents are all different individuals and the notion of the average agent is redundant and although agents may form groups, this should be the result of a bottom-up amalgamation rather than forced grouping.
- **Active:** The agents are considered active because they influence the simulation. This is further defined:
  - **Pro-active/Goal-directed:** Agents are defined as having a set of goals. They should behave in such a way as to achieve those goals.
  - **Reactive/Perceptive:** The Agents can be aware of the changes in the environment. They can also be provided with prior-knowledge of possible obstacles and other entities.
  - **Bounded Rationality:** Agents should behave rationally within the context of their bounded rationality. Their rationality can be altered through their heterogeneity as well as their perceptive abilities.
  - **Interactive/Communicative:** While all Agents may not communicate extensively, they should all have the ability to.
  - **Mobility:** The Agents should be able to roam the environment. This, in combination with the agent's intelligence and ability to interact, can create complex situations.

- **Adaption/Learning:** Agents may also be adaptive such that they become complex adaptive systems. Adaptation may consist of memory or learning and can be implemented at both the individual and population levels.

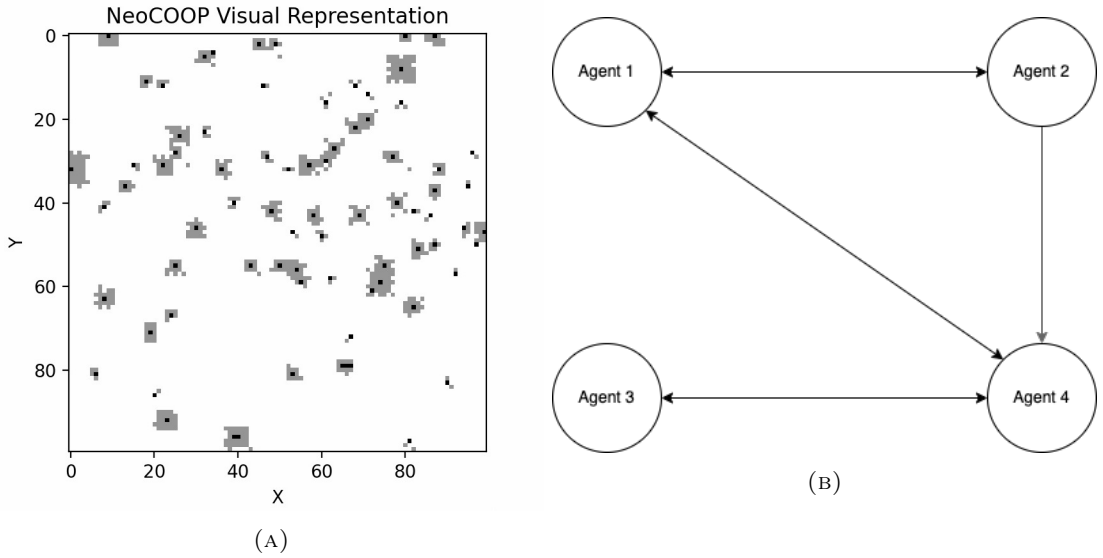
In a practical sense, Agents can be seen as social actors programmed to act within the environment they occupy [47]. A typical Agent within in an ABM will consist of a unique identifier, some number of characteristics (age, wealth and so on) and a set of behaviours that define how the agent acts in its environment. Some simulations may consist of many different types of agents each with their own set of characteristics and behaviours. Some of these agents may be animated (move freely about the environment) while others may be stationary.

2.1.3 Environments

In a similar vein to agents, the environment of an ABM is equally as undefined. Loosely speaking, the primary purpose of the environment is to be a medium upon which agents act to achieve their goals through agent-to-agent and agent-to-environment interactions [45]. An environment may be neutral (void of characteristics) or as detailed as the agents themselves. Given the popularity of ABM in both the Natural and Social sciences, a typical ABM environment represents a geographical space. The resolution/fidelity of this space is largely problem dependent but ranges from a simple 2D grid-world [46] to recreations of real-world geographical locations with multiple GIS-layers [25]. ABM that have environments which represent a geographical locations are know as *spatially-explicit* [47]. Although less common, some ABM elect to represent their environment using some other feature such as a knowledge space or relationship graphs. These types of ABM are known as *spatially-implicit*.

2.1.4 Agent-Based Modelling in the Social Sciences

While well established within the Natural Sciences, the use of ABM in the Social Sciences only came into prominence in the 1990s [50] with Epstein describing them as "changing the face of Social Science" [51]. When compared to traditional Mathematical models, ABM offer agent heterogeneity for modelling different views within a social environment, the ability to model more complex social conditions and adaptive agents with bounded



(A) (B)

FIGURE 2.1: A figure showing the different types of ABM environments. (A) showcases a spatially-explicit grid-world environment where black pixels represent settlements, grey pixels represent farmland and white pixels represent uninhabited land. (B) showcases a spatially-implicit relationship graph where agents must be connected if they are to interact with each other. These relationships may be bidirectional ($Agent1 <-> Agent4$) or unidirectional ($Agent2 -> Agent4$)

.

rationality [52]. ABM are considered a "generative" approach to the Social Sciences, that is to say that ABM seek to explain the emergence of complex social processes through the local interactions of heterogeneous autonomous agents. These local interactions "grow" or "generate" the observed macro-behaviours and if the agents are cognitively plausible, behave under plausible rule-sets and generate the macro-behaviours in a plausible time frame, the local interactions can then be viewed as possible progenitors of the observed macro-behaviours. The generative approach to modelling makes ABM apt candidates for modelling emergent social processes such as human mobility [15] and the formation of hierarchical structures [53].

Perhaps the most famous ABM within the social sciences, *Sugarscape* [46] simulated the survival of sugar consuming agents on a 2D grid. At each coordinate on the grid, there was a sugar level to denote the amount of sugar available in that cell. Agents, who needed to consume sugar in order to live, traversed the grid freely in search for sugar. *Sugarscape* allowed for multiple sugar replenishment strategies as well as agent heterogeneity through varied metabolism and view distances. Simple alterations in the agent decision making process and varied sugar replenishment strategies allowed *Sugarscape* to simulate wealth inequality and population carrying capacity dynamics to name a few.

Although ABM have been used to explain the emergence of a variety of social phenomena, when does an ABM sufficiently explain the emergence of some observed macro-behavior(s)? If a particular model cannot repeatedly generate a target macro-behaviour then the model fails to explain it. This is known as "generative sufficiency" and can be summarized by the saying "If you didn't grow it, you didn't explain it" [54]. It is worth mentioning that successfully simulating some macro-behaviour does not necessarily explain it either. The mechanisms by which the macro-behaviours are generated need to be critically examined. In other words, successfully modelling a macro-behaviour only provides a possible or candidate explanation and further investigative work needs to be done in order to prove the work empirically. This by no means indicates that ABM are useless, in fact, Epstein [55] argues that it is the very ability of ABM to empirically disprove the emergence of some target macro-behaviour(s) through the interactions of some micro-behaviours that qualifies Agent-Based Modelling as a useful scientific instrument.

2.1.4.1 Agent-Based Modelling in Archaeology

The ability to computationally validate the emergence of social phenomena is, unsurprisingly, of particular interest to those in the Social Sciences. This statement is even more appropriate to those in the field of Archaeology. Within last decade, there has been a rapid adoption of Agent-Based Modelling within archaeological literature [20]. Lake [56] attributes this to the conceptual accessibility of ABM. Firstly ABM are capable of producing a wide variety of highly-digestible outputs. For example, virtual reality recreations of ancient civilizations [57] and, more commonly, two-dimensional mappings of household distributions [58] and artefact deposits [18]. Lake [56] further notes that ABM are generally more comprehensible to non-specialists. Secondly, ABM do not enforce a particular method of rule specification. Rules can be specified mathematically but, they can also be described algorithmically. Algorithmic rules are typically similar to informal models when compared to mathematical rules. Thirdly, ABM offer great flexibility in what can be modelled. Models can involve different types of agent interactions (direct or indirect), offer a variety of environment models such as topological networks and two-dimensional abstractions of real world locations all the while allowing agents to make decisions from either a global or individual model of the environment.

Lastly, agents are often modelled as individuals or households and while ABM are scale agnostic, the ethnographic-scale of a typical archaeological ABM aligns with the notion that Archaeology is about the study of the behaviour of people in the past. Barton [59] notes the attractiveness of such systems that view individuals as the drivers of social dynamics. Furthermore, the fragmented nature of the archaeological record means that archaeologists are often left connecting particular behaviours and an expected archaeological pattern intuitively or in some cases experimentally. ABM offer a quantitative method for connecting processes and patterns for building knowledge about the past [60].

2.1.5 Limitations of Agent-Based Models

ABM are by no means the "Swiss Army Knife" of computational modelling. In fact, ABM are subject to a fair number of limitations both practical and theoretical in nature. These limitations are categorized below (It should be noted that the categories presented are by no means exhaustive but rather a list of commonly mentioned grievances with the current state of ABM in academic literature).

2.1.5.1 ABM Suffer at Scale

As is true for every computer program, there is a practical upper-limit to the amount of computational resources that can be used. ABM are no different. For every agent, parameter and behavioural rule, there is a computational cost. ABM that push the boundaries of this practical limit either through the sheer number of agents within the model or through their complexity are know as "Large Scale" ABM [22]. "Large Scale" ABM are particularly desirable because they allow us to examine agent-to-agent interaction and emergent behaviour produced from those interactions in more realistic and complex environments. However, as ABM reach the limits of serial computing, hardware and software solutions such as vector computers and GPU programming are needed [22]. This in itself is no small endeavour, the act of parallelizing an ABM introduces new problems such as memory management, thread management as well as recovery management strategies [61].

Alternative techniques that involve simplifying the model are also available. Firstly, one can simply reduce the number of agents within the model. This method involves

no model restructuring but does assume that the dynamics present within the smaller model would be similar to those present in a large environment [22]. The second method utilizes Super-Individuals. Super-Individuals are agents that represent a collection of agents. For example, a household that represents a group of individuals is considered a Super-Individual. Although this method often requires that models be restructured, it is a relatively simple process that has the potential to retain the dynamics present in the original model. It is worth mentioning that this method is particularly inappropriate when the grouping of agents does not match the spatial context of the model itself [62]. Other techniques such as surrogate modelling can be used to speed up the execution time of a "Large-Scale" ABM. Although, these techniques almost always come at the cost of complexity of both the model and the results it produces.

2.1.5.2 ABM are Black Boxes

While their inputs and outputs are comprehensible, the opaque nature of an ABM's implementation is very rarely understood by anyone other than its creator(s). To combat this, researchers have adopted the Overview, Design-Concepts and Detail (ODD) protocol [63–65]. This protocol seeks to quickly provide readers with the focus, resolution and complexity of an agent-based model. Once a reader has read the ODD description of a model, they should be able to, in any object-orientated programming language, be able to create a skeleton program that implements the described agent-based model. The implementation of the ODD protocol not only makes the inner-workings of an ABM transparent, it also makes the work reproducible. A core concept in the scientific method but rarely present in current ABM research [20]. The ODD protocol has been updated by Müller et al. [66] to include a *Decision* making component. This component is designed to describe the decision making process of human-like agents. The ODD+D protocol is only applicable when modelling human behaviour. Alternatively, the Unified Modelling Language (UML), provides a set of standardized diagrams capable of describing an agent's composition, environment, interactions and relationships. UML is rarely used to describe ABM in the Social Sciences, although examples do exist [67].

As Manzo and Matthew [68] note, the lack of reproducible ABM is also in part due to the lack of generally available software packages. While packages, such as *NetLogo*[1] ,

[1] *NetLogo* is a popular Multi-Agent modelling environment available at: `https://ccl.northwestern.edu/netlogo/`

do exist, the computer code needed to implement an ABM is largely dependent on the observed phenomena and the hypotheses formalized. To solve this problem a number of methodologies such a KISS, BDI and PECS, which are discussed in Section 2.1.6, have been proposed. There have also been calls to publish future ABM under open-source licenses in an attempt to make ABM research both transparent and easier to reproduce [1, 68].

2.1.5.3 ABM are Unpredictable

The unpredictability of ABM largely has to do with complexity of the model itself. The stochastic nature of ABM requires that they be validated over multiple runs. This is further exacerbated by the number of parameters under consideration, of which there is a practical upper-limit [49]. This is particularly true when it comes to mimicking complex social processes. The abundance of parameter combinations possible in models with heterogeneous agents in combination with the stochastic nature of the model itself often produce results that are both too large and too difficult to describe [68]. To avoid this, systems or processes may need to be simplified. A process that effects ABM in an unpredictable manner [21]. Additionally, it is often argued that the emergent behaviour exhibited in ABM are rarely found in the empirical world. This is due to the bottom-up nature in which ABM are constructed which tends to make them sensitive to different kinds of input [21]

Since the ramifications of simplifying a model are unpredictable, how does one go about deciding what should or should not be included within the model? Techniques like pattern-oriented modelling [69] aid us in designing for optimal model complexity. However, the unfortunate reality is that the search for optimal model complexity largely remains a trial and error process. However, as Dean et al. [70] note, "failures are likely to be as informative as successes because they illuminate deficiencies of explanation and indicate potentially fruitful new research approaches".

2.1.5.4 ABM often Lack Adaptability

Adaptability is often touted as one of ABM's biggest strengths [13]. This is particularly interesting given that few models actually use adaptive-mechanisms when it comes to

modelling agent behaviour [4]. Adaptability in ABM literature refers to the ability of agents to not only modify the actions they take but, also the strategies they use to determine which actions to take [4]. Adaptive ABM are better suited to producing emergent behaviour as well as overcoming the rigid structure of traditional ABM, exploring unanticipated parameter distributions when the model was created [71]. The caveat being that introducing these adaptive-mechanisms is not entirely straightforward either as it requires that researchers introduce other, often Machine Learning, techniques to their model which in itself increases the complexity of the model and may even require that a model be reworked to support these techniques.

2.1.6 Frameworks

In every ABM the rules of interaction between entities and the environment must be grounded in a theoretical framework or academic consensus. The implementation of theoretical models forces one to acknowledge the model's assumptions both intentional and not. Romonowska [1] describes these frameworks as having three separate phases (See Figure 2.2). Most deliberation occurs over the conceptual phase as the latter two phases, technical and dissemination, focus on general best practices. That is not to say that the two latter phases are without criticism, but rather that their points of contention have little to do with the simulation of social processes. The conceptual phase has two predominant approaches. The KISS (Keep-it-Simple-Stupid) approach and the KIDS (Keep-it-Descriptive-Stupid) approach. The KISS approach, as highlighted by Cioffi-Revilla [72], is a top-down approach whereby a researcher or student starts with a simple model M_0 and iteratively builds to some final model M_f. The paper describes the requirements of M_0 such as specifying the questions to be answered by the model and identifying the minimal set of social entities to name a few. Cioffi-Revilla further describes the process of evolving the model (adding features, making features more elaborate) and defines the final model M_f as a model that can sufficiently answer the questions the modeller sought to answer as well as have the capability to undergo peer-review and sensitivity analysis. The KIDS approach is a bottom-up approach that starts with a highly sophisticated model and proceeds to simplify and/or remove systems or entities until an appropriately complex model exists. ABM designs tend to favour the KISS approach as the top-down, general to specific, approach reduces the likelihood of introducing errors into a simulation as well as reducing the complexity of the results

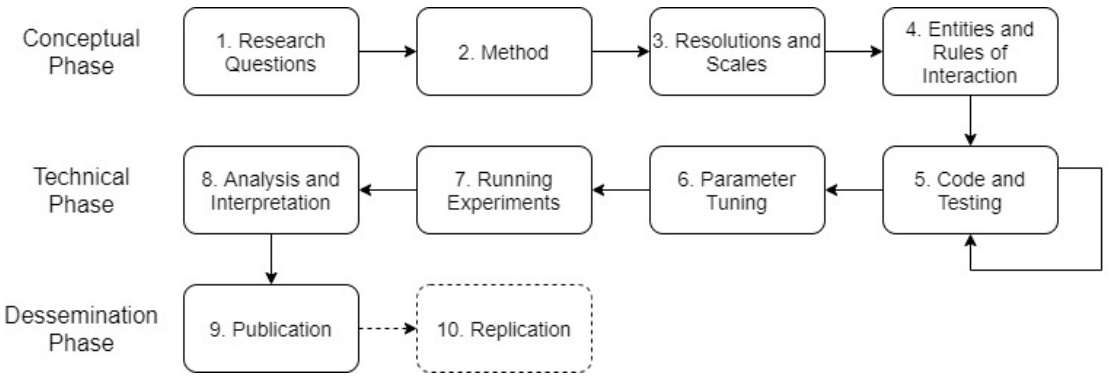


FIGURE 2.2: Romanowska's [1] framework for ABM development

produced. This may seem counterintuitive to the bottom-up nature in which ABM produce emergent behaviour, however, simpler and correct models allow for researchers to analyze the emergent behaviour of their models with the confidence that the observed results are, in fact, valid in the first place.

Within the context of modelling human behaviour, Srbljinović and Škunca [50] note that, due to the extraordinarily complex nature of social processes, the goal of an ABM is not one of prediction but rather one of formalizing and developing new theories. Kennedy [9] highlighted two modelling approaches that have seen varying levels of success. They are detailed as follows:

2.1.6.1 Mathematical

These approaches simplify complex behaviours into a series of simple mathematical rules. Often supported by a random number generator, variables and threshold values are introduced to facilitate agent decision making. These models can be further described as Dynamic Models [73] whereby the rate of change of a variable can be directly affected by the value of the variable itself. Algorithm 1 illustrates an example of simple threshold mathematical model. These models normally follow a KISS approach as the simple mathematical rules can be aggregated to form a more complex model of agent behaviour. While mathematical models are sufficient when the modeller is only concerned about implementing a few agent behaviours, they often fail to capture nuance and complexity of most social processes.

Algorithm 1: A Simple Threshold Mathematical Model for Modelling Human Behaviour (Adapted from Kennedy [9])

```
1 def decide(hunger, starvingThreshold, hungryThreshold):
2     if hunger < starvingThreshold then
3         Die();
4     else if hunger < hungryThreshold then
5         SearchForFood();
6     else
7         Wander();
8 return
```

2.1.6.2 Conceptual

There are two prevalent conceptual approaches for modelling complex social processes. They are the belief, desire and intention (BDI) approach and the physis, emotion, cognition and status (PECS) approach. Due to the conceptual nature of these approaches, they may follow either a KISS or a KIDS approach. However, the equivocal nature of the BDI framework encourages the KISS approach while the comparatively comprehensive PECS framework encourages a KIDS approach.

The first approach, BDI [74], models an individual as having a set of beliefs, their world as they perceive it, a set of desires (what they want to achieve within the world), a set of intentions, and a set of possible actions to take after some deliberation. A decision tree is constructed to represent the sets of beliefs, desires and intents. From this the agents actions are determined and acted upon. Although BDI is appropriate for a great number of circumstances, it is often too general to be considered anything other than a purely conceptual framework.

Alternatively, PECS [2, 3] is a more specific framework that seeks to address the generality of BDI. The framework consists of three horizontal layers which can be seen in Figure 2.3a. The first layer contains the sensor and perception components. These components are responsible for relaying input information to the other components. Interestingly the perception component can be used to manipulate the "raw" input data to include biases, disabilities or any other factors that might affect how the agent might perceive the environment. The second layer is the "meat" of the framework and maintains the internal state of the agent. The Physis component models the physical attributes of the agent (age, injuries). These attributes can be affected by vegetative processes or by other agents. The Emotion component is used solely for maintain the emotional state

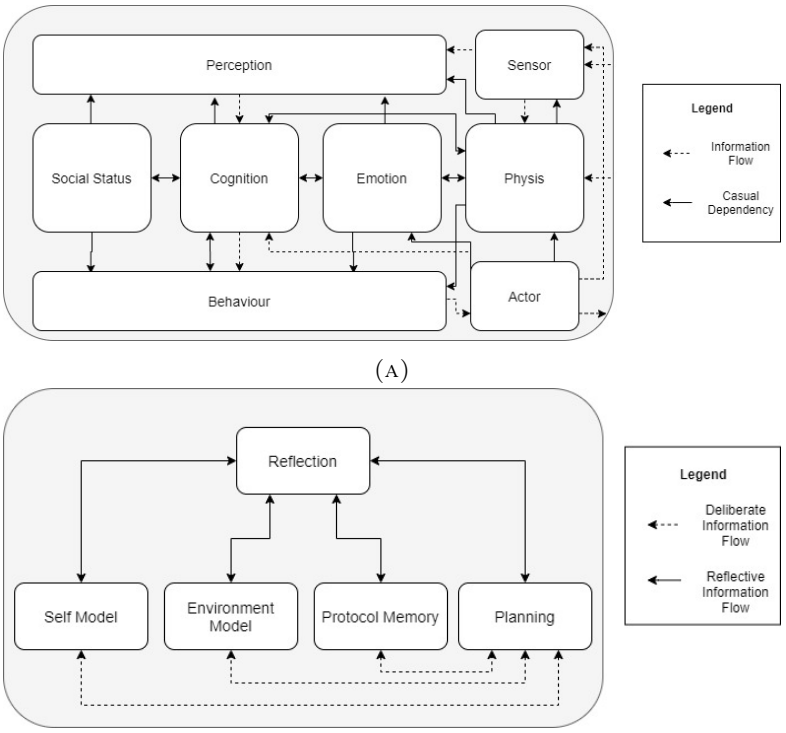


FIGURE 2.3: A figure showing the PECS [2, 3] Model (A) and a closer look at the Cognition component (B)

of an agent. The emotional state of an agent is highly important in creating believable behaviours as the emotional state of an agent may not only affect how it perceives information, but also how it makes decisions. The Cognitive Component can be further broken down as seen in Figure 2.3b. The Self component contains information about the agent's knowledge about its internal state. The Environment component contains the agent's view of the world. The protocol component contains information about an agent's past actions and plans. The Planning component contains the agent's future plans. The Reflection component is responsible for changing the states of the four other Cognition sub-components. It uses the other sub-components to provide the agent with an updated view of the world as well as strategies that best suit the agent given its perceived view of the world.

The advantage of PECS is that allows for the explanation of behavioural patterns in a plausible manner, however, the framework noticeably lacks any implementation details regarding the external to internal transformations of world parameters to agent parameters as well as the order in which components should be prioritized or integrated.

2.2 Machine Learning and Agent-Based Models

The utilization of Machine Learning (ML) and Evolutionary Computing (EC) in ABM has piqued the interest of many within the last decade and while it is by no means invasive in ABM literature, integrating ML and ABM is of particular interest for three reasons:

1. It allows researchers to make use of adaptive mechanisms within their ABM framework [4].
2. It can be used to abstract complex ABM while retaining the complexity of the results/behaviours produced [75].
3. Certain techniques, like genetic algorithms, are particularly useful when researchers need to quickly explore and understand the parameter space of their ABM [76].

In this section we seek to explore these three topics while simultaneously providing an overview of various ML algorithms that are potentially useful when developing ABM. Section 2.2.1 introduces the concept of using ML as adaptive-mechanisms, Section 2.2.2 describes the utilization of ML techniques for complexity reduction and Section 2.2.3 highlights ML techniques as ABM parameter tuning mechanisms. We also provide an overview of surrogate modelling (Section 2.2.4), Reinforcement Learning (Section 2.2.5), Genetic Algorithm (Section 2.2.6) and Cultural Algorithm (Section 2.2.7) techniques and detail how they are typically used in ABM literature.

2.2.1 Machine Learning for Adapting Agent Behaviour

Adaptive-behaviour ABM do not simply refer to ABM in which agents can make numerous decisions, but rather to ABM in which agents are capable of dynamically creating new strategies about which decisions they should make. Adaptive-behaviour ABM are typically harder to create. ML offers a surprisingly compatible solution to this problem. In most ABM, agents often possess an internal model, their view of the world, which they use to guide themselves in an effort to meet a particular goal. Similarly, ML techniques are just as concerned with creating internal models to aid in the creation of adaptive behaviour [4].

However, not all ML techniques are created equally and careful consideration must be taken to ensure that a specific ML technique appropriately represents an agent's learning behaviour or if ML is appropriate for the problem at all. With regards to modelling individual and collective behaviour, Reinforcement Learning, Genetic Algorithms and, by extension, Cultural Algorithms are appropriate [77] while other techniques, such as Neural Networks, are often reserved for abstracting complex ABM. Techniques may also be combined to further encourage adaptive behaviour. Tang [78] demonstrated this by using Reinforcement Learning and Genetic Algorithms to simulate the migratory patterns of Elk in Yellowstone Park. The Reinforcement Learning component served as the learning component of the Elk while the Genetic Algorithm served as a cooperative mechanism for transferring migratory patterns. This combination of ML techniques allowed for the ABM to more accurately capture the complex adaptive systems of Yellowstone Park.

Fortunately, integrating ML techniques into agent decision making is relatively simple in concept. Rand [4] describes the typical algorithmic structure of an adaptive-behaviour ABM as follows:

1. Initialize the system
2. Observe the state of the world
3. Update your internal model of the world
4. Take Action
5. Go to step 2 until time is up

When integrating a ML technique into an ABM, this procedure can be viewed as two separate cycles (One for the ABM and one the ML technique), however, the integration of the two cycles is largely problem dependent. Figure 2.4 is taken from Rand's paper and illustrates a "model refinement" implementation found in ABM that utilize ML techniques as substitutes for rule/probability-based decision making. Additionally, a parameter tuning ML technique, such as genetic algorithms, would operate on the ABM rather than the agents themselves.

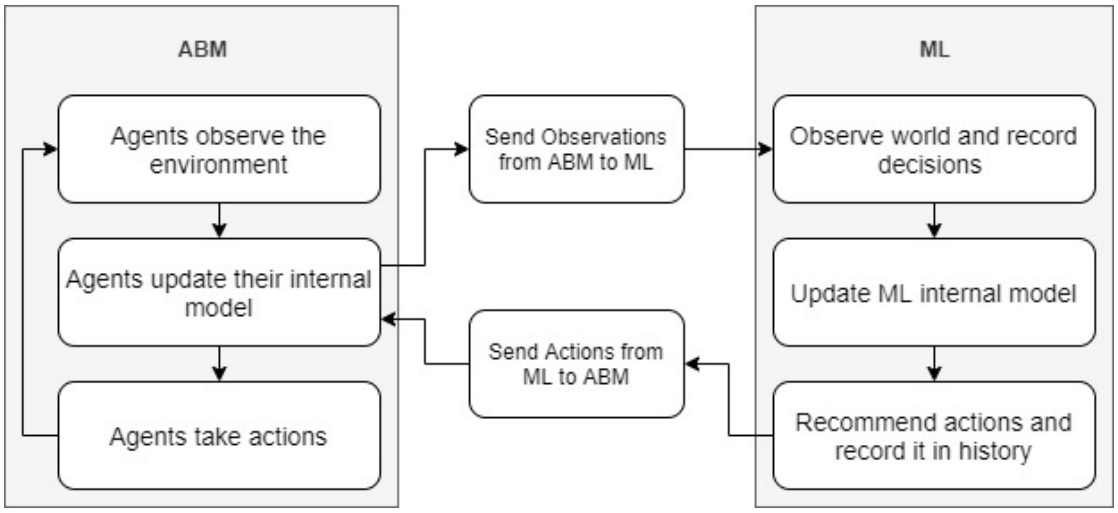


FIGURE 2.4: A "model refinement" design for a ML integrated adaptive ABM as described by Rand [4]

2.2.2 Complexity Reduction

ABM suffer at scale [22]. For every parameter, rule, system and agent introduced into the model, there is a computational cost. This is also true for robustness testing, parameter sensitivity analyses and with procedures that seek to validate the results produced by an ABM. Van der Hoog [75] defines this as the computational intractability of Agent-based Modelling. Traditional solutions to this problem involve reducing the number of rules, parameters or agents within the system or opting to use high-performance computing (HPC) [22]. While both solutions are appropriate, neither are ideal. With regards to the former solution, complexity reduction through simplification is effective in reducing the computational cost of a model, however, the black-box nature of ABM makes the subsequent decline in model fidelity due to complexity reduction unpredictable. With regards to the latter solution, the use of HPC allows modellers to retain model fidelity while reducing the "time-to-completion". HPC has been successfully applied to a number of "large-scale" ABM frameworks in the past [79, 80] with Collier and North [81] noting the "embarrassingly parallel" nature of some ABM allow for multiple simulation runs across a number of different processors simultaneously.

HPC provides a solution for reducing the time it takes to run simulations, however, there is an added financial cost associated with inclusion of additional processing units and while this may not necessarily be of concern to those who use ABM for theoretical work, it is of utmost concern to those who seek to use ABM practically. This is particularly

true for economic ABM where models need to maintain a high-level of precision while being cost effective [75]. ML offers a solution that meets both these conditions.

Surrogate Learning is a ML technique whereby a "meta-model" is trained to approximate the relationship between the inputs and outputs of a different, computationally expensive, model [82]. With regards to ABM, surrogate learning refers to the training of a "meta-model" that approximates the inputs and outputs of an ABM. There are two approaches [75]: (1) Micro-Emulation and (2) Macro-emulation. Micro-emulation refers to the replacing of the decision making component of the agents with a "meta-model", typically an artificial neural network, trained to emulate the decision making process of an agent. The "meta-model" is fed the inputs a typical agent would receive and predicts which action an agent would make given the input conditions. This process is repeated across all, or some, of the agents for the duration of the simulation. Macro-emulation seeks to remove the need to run the ABM at all. The "meta-model" is trained to take the output of the ABM at timestep *t-1*, as input, to predict the output an ABM at timestep *t*. This processes is repeated, with the output predicted at timestep t being used to predict the output of timestep *t+1*, until the desired number of iterations are reached. Comparatively, both micro and macro-emulation serve similar purposes. They reduce the complexity of an ABM while retaining the fidelity of the original model and they are computationally inexpensive compared to a typical "large-scale" ABM and are thus more cost-effective. Furthermore, Lamperti et al. [82] do note that macro-emulation is particularly useful in parameter tuning.

2.2.3 Parameter Tuning

Parameter tuning is a critical step in the development of ABM. Unfortunately, even relatively simple ABM can be characterized by a large number of parameters [76]. This makes parameter tuning, the exploration and understanding of the parameter space, a tedious endeavor. Additionally, ABM parameters are notoriously sensitive to change with desirable emergent behaviour only occurring in a small subset of parameter values.

ML techniques can be used to aid in this process. As mentioned in Section 2.2.2, macro-emulation is particularly useful in exploring the parameter space as the "meta-model" surrogates are computationally inexpensive and are capable of exploring a large subset

of the parameter space in a short period of time. However, the predictive capabilities of the surrogate are largely dependent on the amount of time spent training it [82].

Alternatively, guided search heuristics have also shown promise with Genetic Algorithms having shown the most promise [76]. Guided search heuristics traverse the parameter space in accordance with an optimization function defined by the modeller *a priori*. An optimization function defines some metric for determining the "distance" or "closeness" between specific outputs, produced by a parameter subset, and the desired output(s). Guided search heuristics have a distinct advantage over macro-emulation in that they do not have a training phase. This is particularly useful when the modeller has some knowledge about the behaviour of the model. Conversely, careful consideration should be taken in identifying an optimization function. The choice of optimization function may drastically affect the types of solutions produced by the ABM. Stonedahl and Wilensky [30] demonstrated the usefulness of guided search heuristics by producing multiple flocking behaviours in boids by varying an optimization function. They were able to produce convergence, non-convergence, volatility and the emergence of v-shapes within the same ABM.

2.2.4 Surrogate Modelling

Surrogate Modelling seeks to utilize surrogates ("meta-models") to approximate the results produced by a more sophisticated and resource intensive computational model [83]. They are typically used to approximate the outputs of a computationally expensive tasks, parameter tune, filter noisy data and data mine [83]. In Agent-based Modelling, surrogate modelling is used primarily for complexity reduction.

Artificial Neural Networks (ANN) are the primary ML technique used in ABM surrogate modelling for both macro and micro emulation. ANNs are, as their name suggests, computational networks which attempt to simulate the decision making process of a network of neurons in a biological organism's nervous system [84]. An ANN (See Figure 2.5) traditionally consists of three layers: The input layer, the hidden layer and the output layer, although it is possible to have an ANN with multiple hidden layers. The neurons (nodes) in the aforementioned layers are connected by edges that have a weight associated with them. The weight of an edge denotes its strength and in tandem with the input values, or output values of a previous layer, are fed into an activation function

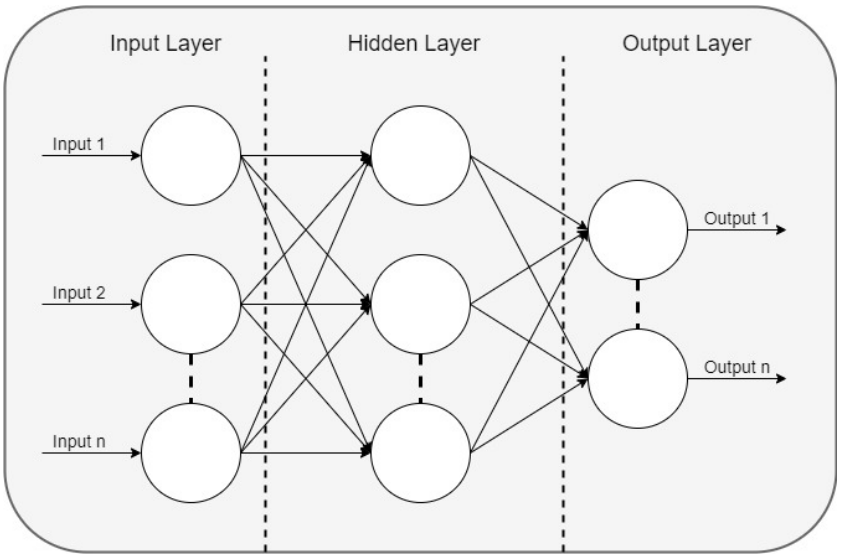


FIGURE 2.5: A feed-forward Artificial Neural Network with one hidden layer.

which determines the output of a neuron. This process runs sequentially through the network with the output values of a previous layer being used as the inputs of the next.

In order for an ANN to accurately approximate a given function, a training algorithm is needed. A training algorithm typically alters the weights of an ANN, although modifying the internal structure of the ANN is a possibility [85]. Some training algorithms, popular in ABM, include back-propagation [84] and Genetic Algorithms [86].

An attempt to discuss all the ANN variants would warrant its own thesis. For ABM, traditional ANNs usually suffice, however, Recurrent Neural Networks (RNN), Neural Networks where the output of the neurons in one or more layers are fed back into the network in subsequent epochs, are particularly interesting in that they allow Neural Networks to model long-term dependencies [75]. Additionally, Deep Reinforcement Learning is discussed in Section 2.2.5.

Yi et al. [87] investigated numerous Surrogate Modelling techniques (ANNs, Support-Vector Machines and Kriging) as calibration tools for crowd models. In their experiments the surrogates were tasked with predicting whether the offspring, produced by a genetic algorithm, would produce desirable crowding behaviour. They found ANNs to perform the best with a correlation coefficient of 0.7598. As discussed by Bonabeau [16], an ABM was created, by the Bios Group, for the National Association of Security Dealers Automated Quotation (NASDAQ) Stock Market in order to understand the effects certain modifications, a reduction in tick rate for example, would have on the stock market.

Their ABM employed a multitude of learning algorithms, including ANNs, to generate a variety of agent strategies for buying and selling shares. Their ABM produced some unexpected results. Namely, their ABM suggested that a reduction in the market's tick size would, consequently, reduce the market's ability to perform price discovery. Lastly, Xu et al. [31] compared ABM and Surrogate Modelling, specifically ANNs, in a simulation of residential customer electricity demand under varying price rates. Both techniques had distinct advantages and disadvantages. Namely, the ABM was sensitive to the customer behavioural data it was based on while the ANN was incapable of separating load demand into groups or appliances. They further note that ABM are more suitable for research and planning while ANNs perform better when predicting trends over entire regions.

2.2.5 Reinforcement Learning

Reinforcement Learning (RL) is a class of algorithms that involve learning how to map actions to rewards so as to maximize a numerical reward signal [10]. Simply put, RL agents are not told what to learn. They must discover it for themselves. RL is a balance between exploration and exploitation, that is, agents must balance the act of obtaining immediate rewards, through exploitation, with the possibility of discovering greater rewards through exploration.

In Agent-based modelling, Q-learning [88] is the most popular form of RL and is typically implemented to introduce adaptive-behaviour. In Q-learning (See Algorithm 2), an agent tries an action at a given state and is rewarded or punished. After repeating this procedure over a number of episodes, the agent learns an optimal policy, a set of state-action pairs that result in the most reward. The learning rate α and the discount factor γ are the variables that determine how an agent learns in a given environment. The learning rate is the factor at which new information overrides old information. A high learning rate ($\alpha = 1.0$) means that an agent will only ever learn from the most recent information while a low learning rate ($\alpha = 0.0$) means that an agent will learn nothing at all. The discount factor determines how valuable future rewards are. A high discount factor ($\gamma = 1.0$) means that an agent will value future rewards while a low discount factor ($\gamma = 0.0$) means that an agent will result in a agent only valuing immediate rewards.

Algorithm 2: The Q-learning Algorithm as outlined by Sutton and Barto [10].

Initialize $Q(s,a), \forall s \in S, a \in A(s)$ and $Q(terminal-state, \cdot)$
for *each episode* **do**
    Initialize S
    for *each step of episode until S is terminal* **do**
        Choose A from S using policy derived from Q(eg. ϵ-greedy)
        Take action A, Observe R, S$'$
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma max_a Q(S',a) - Q(S,A)]$
        $S \leftarrow S'$
    end
end

Zhang and Bhattacharyya [89] used Q-learning to calibrate agent-based supply network models. In their ABM, the Q-Learning algorithm successfully calibrated the supply network models to an optimal RPC (re-order point coefficient) value identified by a two-way sweep of the parameter space. They viewed this as a strong demonstration of Q-learning's effectiveness at calibrating agent-based supply networks. Jalalimanesh et al. [90] used multi-objective Q-learning (MDQ-learning) to identify multiple optimal agent strategies for administering radiotherapy to cancerous tumours while minimizing the duration of the tumour therapy period as well as minimizing the side effects of radiotherapy on healthy cells. The results of their experiment were acceptable, although they note the simplicity of the ABM itself, specifically the tumour growth process, could be further elaborated in future research. Finally, Dreyfus-León [29] integrated Deep Q-learning to model fisherman search behaviour. In real world tasks, Q-learning suffers from the curse of dimensionality whereby agents are required to learn efficient environment representations from extraordinarily high-dimensional sensory inputs. Deep Q-learning [91] solves this problem by introducing a neural network to predict rewards rather than storing them in a traditional Q-table. The network is trained according to some learning algorithm, like back-propagation, and the network becomes adept at accurately predicting the rewards for sets of state-action pairs. In Dreyfus-León's model the fishermen were controlled by two neural networks, one that decided whether to stay or move in a toroidal grid-world while the second network controlled the fisherman's movement within the grid cell. The agents were rewarded according to the number of fish that they caught and were investigated in a number of scenarios with varying fish distributions (uniform and non-uniform). Dreyfus-León found that the ABM was capable of producing fishing strategies similar to those found in the real world.

Algorithm 3: Algorithmic Structure of an Evolutionary Algorithm

Input: MaxGenerations, MutationRate, CrossOverRate

```
1  Generate Initial Population P_0
2  gen = 0
3  while gen < MaxGenerations do
4      CalculateFitness(P_gen)
5      Parents_gen = SelectionFunction(P_gen)
6      Children_gen = CrossOver(Parents_gen, CrossOverRate)
7      Mutate(Children_gen, MutationRate)
8      P_gen+1 = SelectNextGen(P_gen, Children_gen)
9      gen = gen + 1
10 end
```

2.2.6 Evolutionary Algorithms

Evolutionary Algorithms (EA) are a set of guided search heuristics inspired by Darwinian theories of natural selection [92]. Each generation, a series of individuals, consisting of an encoded representation of their genetic structure, partakes in a probabilistic game of survival of the fittest. A distribution of individuals, weighted in favour of their fitness, are selected for reproduction. Reproduction takes place through the exchanging of two or more individual's genetic structures, called crossover, to produce two or more offspring. During the reproductive phase, offspring my have some of their genetic structure altered through a mutation process. The offspring of the current generation are then used as the parent population of the next generation. This process repeats until the rate of change in the average fitness is below some threshold or the computational budget has been spent. The fitness function, sometimes called an optimization function, is the driving force behind a EA. It is defined by a modeller beforehand and represents a method of mathematically scoring individual solutions. In the case of ABM, it may be scoring an individual on its closeness to a particular output [30] or by ranking agents by their ability to acquire/maintain a particular resource [78]. For more information on individual encoding, selection, crossover and mutation operators, see Mitchell [93] (Sections 5.2-5.5). For a brief outline of the algorithmic structure of a EA, see Algorithm 3.

In Agent-based Modelling, EAs typically serve two purposes: to introduce adaptive-behaviour and Parameter Tuning. With regards to Parameter Tuning, EAs are suitable for a number of reasons. Firstly, they are particularly adept at extracting parameter subsets that produce desirable behaviour with Cioffi-Revilla et al. [94] noting that EAs are undeniably useful in understanding the underlying characteristics of both simple

and complex social processes. Secondly, specialization techniques such as niching and speciation allow for further exploration of the parameter space and, consequently, the discovery of additional local maxima/minima candidate solutions [92]. Lastly, EAs are well suited to parallelization with a number of distributed models, like the island model, fitting naturally within the internal representation of an ABM.

In the context of introducing adaptive-behaviour into an ABM, EAs provide a mechanism for simulating generational change. Oloo and Wallentin [95], using a EA, were able to simulate the flight paths of pigeons with each new generation of agents becoming more adept at emulating the pigeon's flight paths. Macy and Skvoretz [28] observed the emergence of cooperative behaviour in a modified Prisoner's Dilemma scenario. Each agent's behaviour was presented as a 15 bit chromosome and, following agent interaction, the fitter agent transfers some of their genetic material, with a 1% mutation rate, to the less fit agent. This crossover method meant that genetic transfer could only occur at a local scale. This resulted in the emergence of heterogeneous groups with cooperative behaviour starting at a local level and then spreading globally throughout the environment.

Although uncommon, Genetic Programming (GP) has also been used in some ABM frameworks. Instead of representing the properties/characteristics of an agent, genetic structures in GP solutions represent a definition for some agent decision making process. Manson [58] applied GP to land-use and land-cover change research of the Southern Yucata´n Peninsular Region. In his ABM, Manson applied GP to model an agent's ability to access a given grid cell's suitability for agricultural activities. The model performed acceptably (Kappa Index of Agreement: 0.482 & Relative Operating Characteristic: 0.905) when compared to real-world data.

2.2.7 Cultural Algorithms

Similarly to EAs, Cultural Algorithms (CA) fall into the category of Evolutionary Computation. Created by Reynolds [96], CAs can be viewed as an extension to a traditional EA whereby the individuals transfer knowledge through a shared belief space rather than by reproduction. As seen in Algorithm 4, CAs share the same general algorithmic structure of EAs but have replaced the Mutation function with a belief space and a communication protocol. In terms of optimization, the belief space represents the range of values that an individual's genetic structure is bound by. In terms of cultural evolution,

Algorithm 4: Algorithmic Structure of a Cultural Algorithm

Input: MaxGenerations, NormKnowledge, DomKnowledge

```
1  Generate Initial Population P_0
2  Initialize belief space b(NormKnowledge, DomKnowledge)
3  gen = 0
4  CalculateFitness(P_0)
5  while gen < MaxGenerations do
6      Parents_gen = SelectionFunction(P_gen)
7      Children_gen = CrossOver(Parents_gen)
8      InfluenceFunction(Children_gen, b)
9      P_gen+1 = SelectNextGen(P_gen, Children_gen)
10     gen = gen + 1
11     CalculateFitness(P_gen)
12     AcceptanceFunction(P_gen , b)
13 end
```

the belief space represents the beliefs of a community or society. Information exchange between the population and the belief space is controlled by the communication protocol (See Figure 2.6). The communication protocol controls how the population affects the belief space, called the acceptance function, and, conversely, how the belief space affects the population, called the influence function. While the implementation of the acceptance and influence functions are largely problem dependant, Chung and Reynolds [97] provide examples such as specialization and generalization for acceptance functions and boundary mutation and heuristic crossover for influence functions to name a few.

In the context of Agent-based Modelling, CAs can be used for both parameter tuning and to introduce adaptive-behaviour. CAs can be implemented in a variety of different ways. Traditional CAs used genetic programming while newer CA implementations have transitioned to separating the population into two classes, the elites and the commoners, which can be manipulated differently. The elites traditionally represent the best candidate solutions while the commoners represent the remainder of the population. The elites are responsible for altering the belief space while the commoners' offspring are influenced and altered according to the belief space. The elites' offspring may undergo some kind of mutation to further explore the solution space [98]. Yang et al. [99] use this concept for the development of their CA parameter tuning algorithm called the Cultural Algorithm Quantum-Particle Swarm Optimizer (CAQPSO). Commoners evolved according to a Particle Swarm Optimization (PSO) algorithm while the elites undergo a mutation process that allows for incremental exploration of the parameter space. Omran [98] describes a similar algorithm, called the intellects-masses optimizer (IMO), whereby

elites and commoners serve similar purposes but undergo a unified and simple update procedure. Omran argues that the IMO is better than the CAQPSO due to its smaller parameter space, making it easier to parameter tune, and the ease at which it can be implemented, in code, when compared to the CAQPSO.

In Agent-based Modelling, CAs are particularly attractive in that they provide us with a framework in which we can view cultural adaptation over time. Kobti et al. [100] demonstrate this through the introduction of a belief space into a kinship network of trading villagers in the drought ridden Central Mesa Verde region. The belief space served as a cultural space whereby groups could generalize about an individuals tendency to trade and where individuals could remember their own trading experiences with other individuals. They found the introduction of the CA increased the ABM systems resilience with respect to drought conditions while simultaneously increasing the system's reliance on hub nodes (villagers with many connections).

The use of networks for information exchange have been further investigated by Reynolds and Ali [101]. They found that the introduction of a social fabric, a social network, to influence an individual's behaviour is dependent on the type of network topology used. In their case, a ring topology lead to a gradual convergence around the optimal point on the cone map while the square topology resulted in an exploratory phase which, once the optimal point had been found, resulted in a rapid convergence around the optimal spot. Reynolds and Ali separated the belief space into five knowledge sources. These sources are defined in another paper, by Reynolds et al. [5], as follows:

- **Normative**: A set of desirable variable ranges that serve may serve as a guideline in which individual adjustments can be made.
- **Domain**: Knowledge specific to the domain in which the CA is being applied to.
- **Situational**: A set of specific values in which specific events must occur.
- **Topographical**: Knowledge about the environment and search space occupied by the individuals.
- **Historic**: Records of past, or important, events and actions.

A CA can incorporate any number of these knowledge sources. Interestingly, the addition of these knowledge sources results in an emergent phenomena called "Knowledge

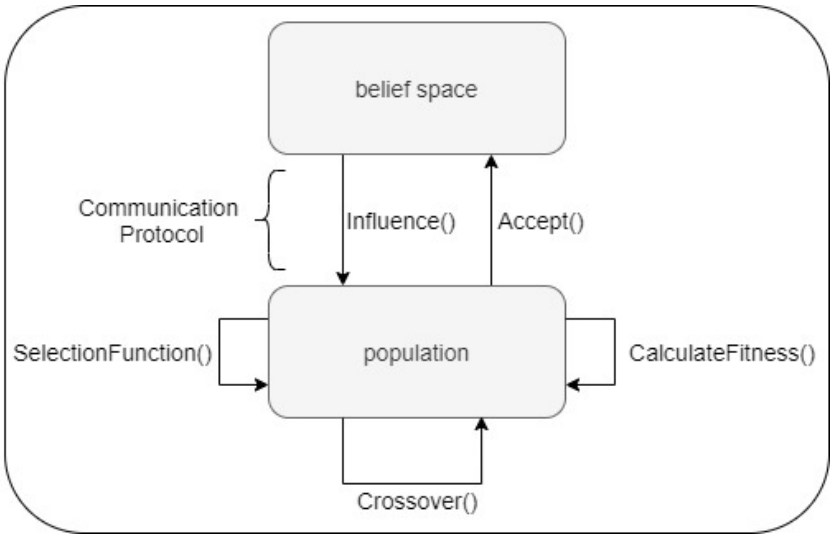


FIGURE 2.6: A component view of a CA. (Adapted from Reynolds et al.[5])

Swarming" [102]. The phenomena is described as an emulation of a branch-and-bound procedure whereby the candidate solutions first explore the solution space and then converge to a local optima.

2.3 A Review of the State of the Art

In this Section we will review the current state of Agent-based Modelling in the Social Sciences (Anthropology, Sociology and Archaeology for example) and several accompanying fields such as Epidemiology and Disaster Management. We achieve this by reviewing 50 real-world ABM applications published from 2017 to 2021[2] and categorizing them by the following metrics:

1. Research Field.
2. The ML / EC technique used (if any).
3. Purpose of ML / EC technique (if any).
4. Software package used.
5. Measures used to ensure repoducibility.

[2]A list of the papers used in this review and their respective categorizations can be viewed at the following link:`https://docs.google.com/spreadsheets/d/1VvGlMFn9jK_BSu5lgNmfZdPDWyBuXK_L3UXEzGRuONs/edit?usp=sharing`

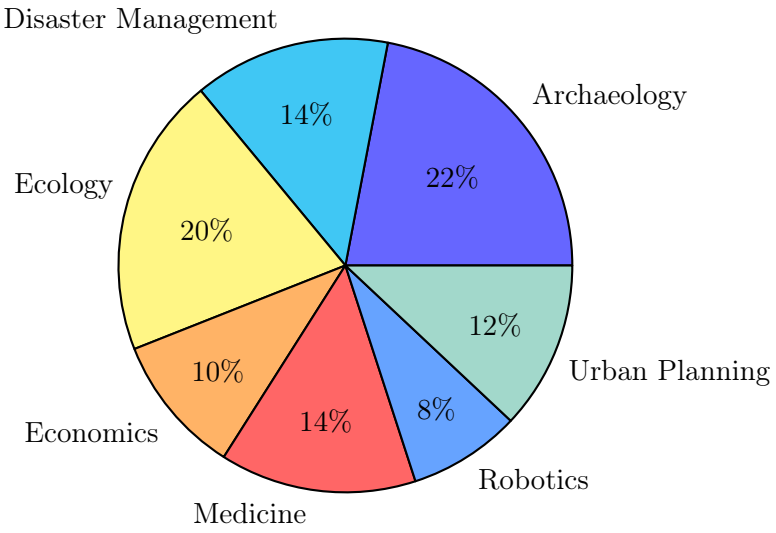


FIGURE 2.7: Distribution of primary fields of research for papers reviewed.

While the collected data is by no means exhaustive, we believe it is sufficient to make certain claims about current research trends. Our findings are presented below.

2.3.1 Agent-Based Modelling in Practice

It does not take long to discover that ABM are used in a variety of research fields. Long gone are the days of abstract sandbox environments like Sugarscape [46] with recent ABM design shifting towards construction through qualitative and quantitative data. A snapshot of these research fields can be seen in Figure 2.7 where not only do we see ABM being broadly applied across research fields like Ecology, Economics and Medicine, we also see ABM emerging in more niche fields such as Computational Archaeology, Disaster Management and Urban Planning. It is worth pointing out that Figure 2.7 is not meant to give a representative indication of the current distribution of ABM research but, rather make clear the categories of papers reviewed for this discussion. For transparency, Ecological and Economic ABM are under-represented and Archaeological ABM are over-represented. This is due to the focus of the dissertation and should be taken into consideration when reading this review. It is also worth mentioning that most of the categories listed in Figure 2.7 share a great deal of overlap. For example, ABM that fall under the Disaster Management category undoubtedly have to consider and simulate both the ecological and economic consequences of a disaster event. Given this, we define each category as follows:

1. **Archaeology:** ABM designed to study the underlying dynamics, such as the emergence and evolution, of people and civilizations in ancient history.

2. **Ecology:** ABM with a particular focus on living organisms and their relationship with the environment. ABM in this category often include humans as the primary agent, although the focus of the research is typically on the impact said agents have on their environments.

3. **Economics:** This refers to ABM that are primarily concerned with studying the underlying dynamics of human influence on financial markets and supply chain networks.

4. **Robotics:** ABM concerned with accessing the viability of using robots and other autonomous entities to achieve a wide variety of goals. The development and utilization of these models are often considered prerequisite steps in applying the proposed autonomous entity in real-world scenarios.

5. **Medicine:** ABM that evaluate the effectiveness of policy or treatment strategies from a medicinal context. These ABM are often used in the sub-field of Epidemiology to evaluate the effectiveness of various policies in preventing or reducing the spread of disease.

6. **Urban Planning:** ABM designed to understand both the ecological, economic and sociopolitical impact of Urban expansion / development. These models often assess the viability of proposed expansion plans or are used to identify the consequences of previous policy decisions.

7. **Disaster Management:** ABM in this category are identifiable by their focus on exploring the social phenomena that emerge under various types of disaster events. They may also access the economic impact of said events.

Of the fields that are present, Medicine is the most interesting. The primary reason for this is the emergence of COVID-19 which has also brought about the creation of several ABMs that have been used to understand the spread of the disease as well as the economic impact it has had on numerous countries. Despite the tragic loss of life, COVID-19 has proven that ABM are capable of helping us understand complex social phenomena in real-time and real-world scenarios.

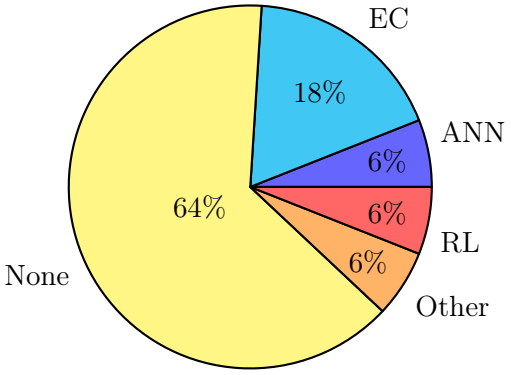


FIGURE 2.8: Distribution of Machine Learning Algorithms used in papers reviewed.

Looking at the papers as a whole, new ABM seemingly fall into two categories: ABM with abstract environments in which social phenomena are explored in a general sense and ABM with highly detailed environments in which applying conceptual models is used to understand the dynamics of said environment. These highly detailed environments incorporate real-world data directly into the model, often through GIS layers, in an effort to increase model fidelity. The verdict on which of these approaches is "better" is arguably a moot point but, what is clear is that construction of ABM both in the context of modelling environments and, more importantly, the modelling agent behaviour has shifted from analogical modelling to empirical modelling. Although in opposition with current trends, Drost and Vander Linden [32] argue that "Toy-Box" environments are indeed useful for testing whether certain models from one field of research can be applied to another.

2.3.2 Machine Learning in Agent-Based Modelling

Figure 2.8 shows the distribution of ML techniques used in the reviewed ABM. Of the 18 ABM that did use ML techniques, EC techniques were used most often. Furthermore, 2 ABM used ML for surrogate modelling, 2 used ML for parameter tuning and 14 used ML as adaptive mechanisms. While these results are encouraging, there is a distinct lack of adaptive-mechanisms present across the entire set of reviewed ABM. This, for the most part, is perfectly adequate for a simple ABM. However, the ABM reviewed are specifically concerned with examining the emergent phenomena produced by the dynamic micro-interactions of complex-adaptive systems. A task so extraordinarily unachievable that all ABM resort to abstraction at some point. This can be seen in Bogle and Cioffi-Revilla's [34] model where group collective action is abstract. This is similarly done by

Perret et al. [53] and in the case of Wren et al. [103], they limited the number of unique types of terrestrial resources available to the foragers (Their model only considered plant resources on terrestrial cells).

Simplification typically serves as a mechanism by which specific processes or emergent behaviours can be observed in isolation. However, to ignore the adaptability of humans is a fundamental failure to acknowledge that humans are continuously adapting to their environments both in a personal and evolutionary sense [26]. We are by no means suggesting that failure to include adaptive mechanisms invalidates any particular ABM but rather that we believe there should be a concerted effort to include adaptive-mechanisms within ABM modelling complex social phenomena. The benefit of adaptive mechanisms is evident in Chliaoutakis and Chalkiadakis' [40] paper on dynamic trade network formation. An extension of previous work [25], Chliaoutakis and Chalkiadakis used utility optimization in lieu of a traditional rule-based approach to agent decision making. This gave agents the capability to learn about their environment which was crucial step in determining which patches of land to farm. Additionally, their model allowed for the formation of dynamic hierarchical structures based on an agent's ability to gather resources more efficiently than others. The flexibility of this model is rarely exhibited in a traditional rule-based ABM and in order to find a rule-based model with similar flexibility, it often requires that numerous decision making strategies are created. A process that both adds complexity to the model and takes additional time to implement. Adaptive mechanisms don't have to be introduced at the decision-making level. As exhibited by Perret et al. [53] and Cucart-Mora et al. [104], generational adaption can also be introduced. In both cases, the introduction of evolutionary mechanics allowed both models to simulate generational agent adaption. In the case of Perret et al. [53], it allowed them to study the formation of hierarchy and in the case of Cucart-Mora et al. [104], it allowed them to model the extinction of the Neanderthals while retaining some of their genetic structure found in some humans to this day. The ability of adaptive ABM to produce emergent behaviour as well as overcome the rigid structure of traditional ABM, exploring realms unanticipated [71] is an undeniable strength and one that should be examined further.

The reason for the lack of adaptive ABM is unclear. In fact, there may be a number of reasons. Firstly, the use of ABM in the Social Sciences is relatively new. This is especially true for ABM in Archaeology and, as such, it may be the case that the field

has not matured to a point where modelling adaptive agents is the norm. This can be seen in a number of financial ABM, a more mature field of research, which regularly employ adaptive mechanisms (See Schulenberg and Ross [105], Zhang et al. [106] and Bernard [107]). However, this was only the case once the models had progressed from analogical models, similar to Sugarscape [46], to models built using empirical data [108] (stock market data for example). This can also be seen in the field of Robotics where EC is often used to evolve different and desirable robot behaviour [109]. We believe a similar phenomena can be observed in current ABM where modellers are starting to utilize real-world data to not only guide their design process but further increase the fidelity of their model as well. It may simply be the case that adaptive ABM are the next stage in the developmental process of ABM as a form of computational Social Science.

Another two, although speculative, reasons for the lack of adaptive ABM relate to the general unfamiliarity of researchers with ABM as a method of computer modelling and the notion that the current state of ABM is emblematic of the popularity of the Keep-It-Simple-Stupid (KISS) design methodology. With regards to the former, Romanowska [1] and Janssen et al. [110] note that Agent-Based Modelling is seldom taught as a method of computational modelling in a standard Archaeology curriculum. This, in combination with the added complexity that adaptive-mechanisms, especially ML techniques, incur on the modelling process may mean that the average Social Scientist is ill-equipped to design adaptive ABM. With regards to the latter, the KISS methodology, as proposed by Axelrod [111] in the late 1990s, has had an undeniable influence on the design of ABM. The KISS methodology advocates that models be kept as simple as possible with complexity being added when absolutely necessary. This design philosophy is indeed attractive when the unpredictability of ABM needs to be considered. However, Edmonds and Moss [112], the originators of the Keep-It-Descriptive-Stupid (KIDS) approach, note that it is often the case that modellers, even when faced with complex phenomena, strive to keep their model as simple as possible. This statement can quite easily be applied to that of adaptive ABM when modelling complex social processes. Adaptive-mechanisms add complexity to a model that not only increase its development time but go against the KISS principles upon which the model has been built. Taking into account that even simple stochastic rule-based ABM can produce quite interesting emergent behaviour, it is quite easy to rationalize that adaptive-mechanisms are unnecessarily complex despite being a fundamentally human trait.

This begs the question, when and why should we use adaptive-mechanisms when modelling complex social processes? There is certainly an argument to be made that adaptive-mechanisms may not be particularly useful when exploratory or "Toy Box" models are being developed. In fact, it may even be a hindrance. However, when a model is intended to explain the emergence of complex phenomena and inform or guide the formation of formal hypotheses or policies particularly when the drivers of the system represent adaptive organisms (humans, primates and so forth), adaptive-mechanisms should be seriously considered. The benefit of these mechanisms is undeniable. Even agents that are "hard-coded" with multiple rule-based strategies are more advantageous than rule-based agents with a singular strategy (See Bianchi et al. [113]). This is because adaptive ABM are better suited at producing emergent behaviour as well as overcoming the rigid structure of traditional ABM, exploring realms unanticipated when the model was created [71]. This statement is even more appropriate when ML techniques are used. As evident by their success in other fields, Genetic or Cultural Algorithms (GA and CA) and Reinforcement Learning (RL) are suitable techniques for introducing generational and individual adaptation respectively (See Andreoni and Miller [114] for a model that utilized a GA to simulate bidding error phenomena, typically observed in real world auctions, that were unexplainable by traditional Nash Equilibrium bidding models. See Ostrowski et al. [115] for a model that utilized a CA to adapt a near optimal pricing strategy for car buyers in a more complex market environment. Their model was capable of developing a more-optimal strategy than a traditional rule-based model. See Xue et al. [116] for an example of RL used in an Iterated Prisoner's Dilemma problem using different social network structures in which cooperative behaviour, an atypical phenomena, emerges.).

Lastly, we believe it is also worth talking about the lack of ABM using ML as parameter tuning mechanisms. We believe there are two reasons for this: Firstly, that the tuning of an ABM is often given very little real estate (or none at all) in a lot of ABM literature. This is not to say that most researchers don't consider it an important part of the model development process, but rather that describing what the model does and the results it produces may be considered more important when papers have page or word count limitations. Secondly we also believe that as ABM move away from analogical models to empirical models, the need for using complex parameter tuning mechanisms decreases. It is not uncommon for an ABM to have several of its input parameters already "tuned" when the model has been constructed using significant amounts of available data. Taking

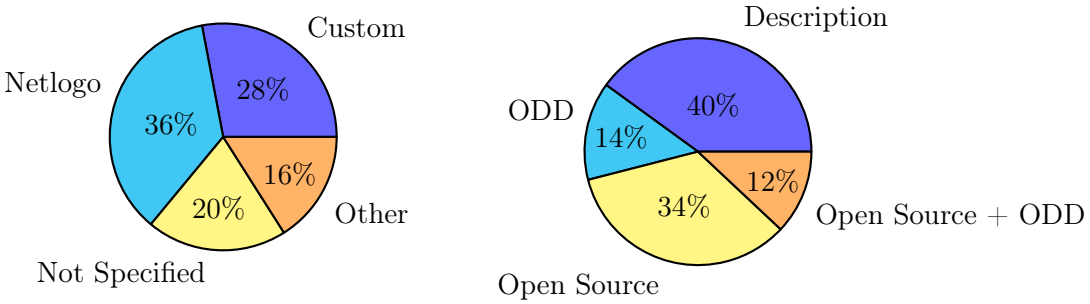


FIGURE 2.9: Distributions of Software Packages (Left) and Reproducibility measures (Right) utilized in reviewed papers.

this into consideration, it is understandable why one might not want to use comparatively complex parameter tuning techniques when there isn't a need for it in the first place.

2.3.3 ABM Software Packages and Reproducibility

Figure 2.9 highlights the current state of model transparency in current ABM design. While certainly more prevalent than it was a decade ago, the concerted effort to adopt ODD [63–65] and ODD+D [66] as the universally accepted description protocol is still ongoing. Whether or not an ABM uses ODD or ODD+D as a description protocol is research field specific and although our review doesn't capture the descriptive quality of the non-ODD models, it is clear that as models increase in complexity, researchers should adopt some standard description protocol in an attempt to reduce the black-box nature of current ABM research. Further work in this regard has been done by Romanowska et al. [117], Davies et al. [118] and Crabtree et al. [119] who wrote a three part series for non-specialists on how to design ABM, how to combine ABM with GIS and how to present ABM to the public respectively.

Figure 2.9 also shares some additional insights with us about what software packages are being used to implement ABM. Netlogo and custom solutions are by far the most popular with niche packages such as Anylogic[3] and GAMA[4] used when the researchers have a specific goal in mind. Custom solutions are often built in Python which is interesting given that Mesa[5] exists.

[3]Anylogic is a simulation modelling software package available at: `https://www.anylogic.com/`

[4]GAMA is a modeling and simulation development environment for building spatially explicit agent-based simulations available at: `https://gama-platform.org/`

[5]Mesa is an Apache2 licensed agent-based modeling (or ABM) framework in Python available at: `https://mesa.readthedocs.io/en/latest/`

Another interesting facet of these results to consider is the implication software homogenization has on the necessity of standardized descriptive reproducibility measures. As seen in Figure 2.9, most of the ABM reviewed were open sourced. One could quite easily argue that open sourcing an ABM removes or reduces the need for a unified description protocol. It is indeed true that having access to the same source code as the model's authors provides a level of clarity to their design decisions that one may not be able to replicate in written text. Further considering this notion, we would argue that this is only the case when the software used to develop ABM is homogenized. In a world where everyone uses Netlogo to develop ABM, the need for ODD is reduced as there exists a unified *modus operandi*. Conversely, Figure 2.9 also shows that a significant number of ABM were custom solutions which, in the context of this discussion, are heterogeneous. We argue that unless the usage of ABM software packages are homogenized, there exists a need for a unified description protocol, ODD or otherwise, to act as an intermediary ensuring that ABM are not only reproducible but, reproducible across ABM software solutions.

2.4 Discussion and Conclusions

In this chapter, we presented an extensive overview of ABM literature in the Social Sciences. Four limitations with the state-of-the-art were identified (See Table 2.1) with the lack of adaptive mechanisms present in most ABMs identified as an issue with no satisfactory solution. The reason for this phenomena is not entirely clear but, history points to the relative infancy of ABM literature in the Social Sciences with other, more mature, fields (Economics most notably) having followed a similar trajectory from simple analogical models to data-oriented models with adaptive mechanisms. Furthermore, models are also deliberately kept simple to minimize unpredictability. This, in combination with the emergent behaviour even simple models are capable of producing, has seemingly resulted in adaptive-mechanisms appearing as unnecessary complexities.

We argue that the state-of-the-art is insufficient given the subject matter. Using ABM in scenario-based experimentation is arguably the most useful to social scientists where the evolution of collectives may be evaluated under various conditions (derived from data or otherwise) to better inform policy decisions and support or oppose theories about the evolution of said collectives. In the social sciences, these collectives are human.

Problem	Solution
ABM Suffer at Scale	Distributed or Parallel Programming
ABM are Black Boxes	ODD Description Protocol
ABM are Unpredictable	KISS or KIDS Design Methodology
ABM Often Lack Adaptive Mechanisms	**No Sufficient Answer**

TABLE 2.1: A summary of ABM limitations described in Section 2.1.5.

Technique	Adaptive Mechanisms	Parameter Tuning	Complexity Reduction
ANNs	Yes*	No	Yes
RL	Yes	No	No
EAs	Yes	Yes	No
CAs	Yes	Yes	No

TABLE 2.2: A summary of Section 2.2 which introduces various ML Techniques and their uses in the development of ABM. **Note:** ANNs can be used as adaptive mechanisms but, a suitable network architecture would need to be used (like a Deep Q-Nework).

Adaptability is a fundamentally human trait and failure to include adaptive-mechanisms in ABMs with human-like agents greatly reduces the applicability of the models' results as they fail to capture the nuance of human behaviour.

ML Algorithms were identified as suitable mechanisms for creating adaptive-agents. Table 2.2 summarizes our findings with biologically and psychology inspired techniques presenting the most promise. In particular, we identify RL as a suitable candidate for introducing individual (agent) adaptation while EAs and CAs are suitable for generational adaptation. The primary advantage these ML techniques offer over simple, rule-based, methods is that they promote agent heterogeneity while facilitating information and strategy exchange between agents such that not only are agents capable of finding an optimal strategy for their unique circumstances, they can also exchange information with other agents such that they will also adopt globally optimal strategies.

When to use ML as adaptive-mechanisms is an interesting question. It could be argued that using ML techniques are not particularly useful when exploratory or "Toy Box" models are being developed or studied. These ML techniques may be complex, hard to implement and provide no additional benefit over rule-based models given the simplicity of the model itself. It is our belief that even in those scenarios, a simple adaptive mechanism (such as an EA) would be more beneficial than not having one at all. Similarly, as model complexity increases, so does the need for more complex adaptive-mechanisms

(such as mixing a RL algorithm with an EA to facilitate both individual and generational adaptation).

In conclusion, we have identified the lack of adaptive mechanisms in ABM studying complex social phenomena as a limitation of the state-of-art. This is likely due to the relative infancy of ABM in the Social Sciences and a general aversion to adding complexities to model definitions. Machine Learning as adaptive-mechanisms were identified as a possible solution to this limitation with biology and psychology inspired techniques such as Reinforcement Learning, Evolutionary Algorithms and Cultural Algorithms showing the most promise. It is our belief that demonstrating the benefits of adaptive mechanisms in suitably complex environments (the recreation of an Ancient Society for example) is rife with research opportunity.

Chapter 3

Methodology

In this Chapter, we seek to illustrate how we developed an ABM to compare the efficacy of ML adaptive-mechanisms to traditional, rule-based, approaches. To achieve this, we identified a 4-step development process described as follows:

First, a suitable modelling framework was needed. *Netlogo* is the most popular but, it was deemed a poor fit due its performance limitations at scale. We instead developed our own framework called *ECAgent* that uses the Entity-Component-System (ECS) architectural design pattern to facilitate the design of complex, large-scale, ABM (Section 3.1).

Second, we needed to find a process for designing adaptive-agents and quantifying their adaptive capacity. Section 3.2 highlights these efforts and illustrates how we aim to design and measure the adaptive capacity of our adaptive-agents using information exchange networks.

Third, a suitably complex contextual backdrop was needed to truly discern the benefits ML adaptive-mechanisms have over traditional, rule-based, approaches. We chose Ancient Egypt as this backdrop due to the complex social processes associated with the formation of the state during the Predynastic period. A comprehensive summary of these processes is included in Section 3.3.

Lastly, using *ECAgent* as the development framework, Predynastic Egypt as the contextual backdrop and information exchange (using RL and two EAs) to create our adaptive-agents, we designed *NeoCOOP* (Neolithic Cooperation Model), an ABM that enables the

study of the complex social processes of ancient societies with agents exhibiting varying degrees of adaptive capacity. Section 3.4 provides a detailed description of *NeoCOOP*.

Note: All software developed for this thesis is open sourced and available at the following urls:

ECAgent: `https://github.com/BrandonGower-Winter/ABMECS`

Simple Predator Prey: `https://github.com/BrandonGower-Winter/ECAgentTutorials/tree/master/SimplePredatorPrey`

Ants Foraging Simulator: `https://github.com/BrandonGower-Winter/ECAgentTutorials/tree/master/ForagingAntSimulator/`

NeoCOOP: `https://github.com/BrandonGower-Winter/NeoCOOP`

3.1 ECAgent - An ECS framework for developing ABM

In this section, we aim to highlight the compatibility the Entity-Component-System architectural design pattern has with the development, deployment and scaling of ABM. Object-oriented programming (OOP) design patterns have long dominated the ABM landscape and our goal is demonstrate that alternative, possibly better, design patterns exist. To do this we develop *ECAgent*, a Python-based ECS framework for developing ABM applications.

Section 3.1.1 summarizes the motivations for *ECAgent*, Section 3.1.2 provides an overview of *ECAgent*, Section 3.1.3 demonstrates the framework's capabilities by recreating a Simple Predator-Prey model developed by Tatara et al. [7]. Section 3.1.4 concludes with a discussion of the current state of *ECAgent* and what improvements can be made in future research endeavours.

3.1.1 Motivation

ABM are typically designed with an OOP mindset. This is largely due to analogical compatibility. Viewing agents as objects just makes sense, even more so when adding behaviour to these agents (the central driving force of ABM simulations) literally means

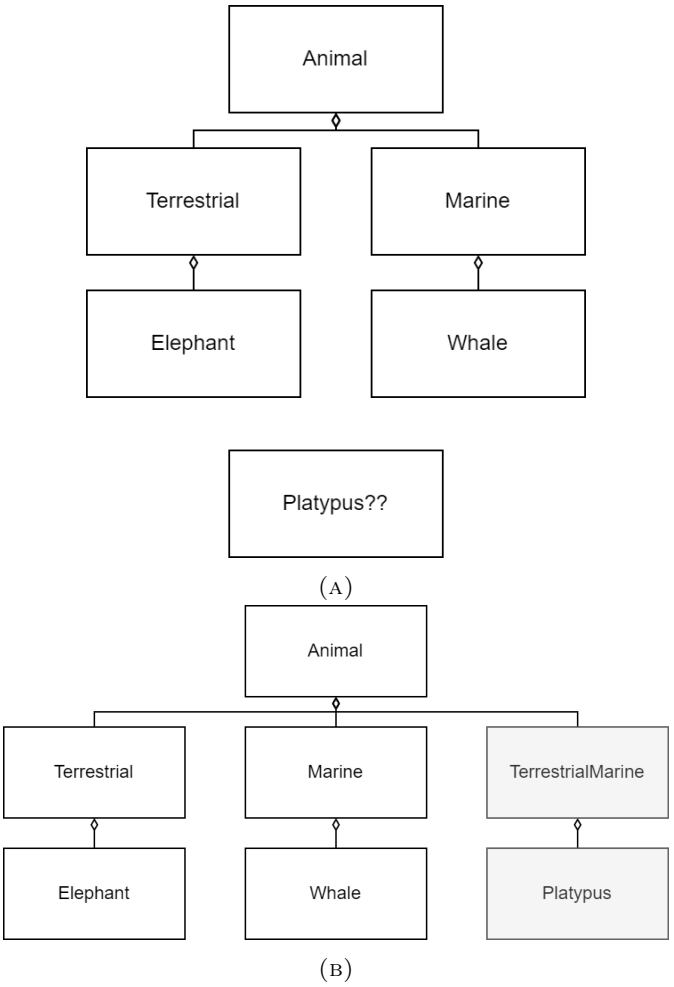


FIGURE 3.1: A figure showing the difficulty of maintaining inheritance trees for certain types of software. (a) Demonstrates that the *Platypus* class can't be created without inheriting from both *Terrestrial* and *Marine*. (b) Shows the inelegant solution of creating a third class called *TerrestrialMarine* which the *Platypus* class can inherit from.

adding code to an "Agent" class. Coupled with the fact that ABM are mostly developed for fields outside the realm of traditional Computer Science, it is not surprising that other design paradigms have not been investigated.

OOP is by no means perfect, in fact there are several deficits that arise when OOP systems are pushed to their limits. They are:

1. The difficulty of maintaining inheritance hierarchies [6].
2. Rigid Object Categorization [6].
3. Poor Memory Utilization [120]

Point 1 refers to the fact that as projects scale, so does their inheritance tree. As demonstrated by Figure 3.1a, it can be seen that both the *Terrestrial* and *Marine* classes inherit from a base *Animal* class. *Elephant* and *Whale* inherit from the *Terrestrial* and *Marine* classes respectively. What if we wanted to add a *Platypus* class? This is where the weaknesses of OOP start to rear their head. Most programming languages do not support multiple inheritance[1] so a third *TerrestrialMarine* class must be created which the *Platypus* class must then inherit from (See Figure 3.1b). What if we now wanted to add *Airborne* creatures, well the aforementioned tree maintenance issue is compounded as we now need to consider that some creatures may be *TerrestrialAirborne* or *MarineAirborne*, there are also creatures (such as Gannets) which possess terrestrial, marine and airborne characteristics (i.e. *TerrestrialMarineAirborne*)

With regards to point 2, OOP systems do not allow objects to change type. Yes objects may be typecast up an inheritance hierarchy but this change is superficial given that runtime polymorphism still treats the typecast object as its original type (An *Elephant* typecast to an *Animal* is still an *Elephant*).

Point 3 is only an issue for applications that instantiate lots of objects. Objects use memory (more than simple primitives), so when thousands of similar typed objects are being created, you incur memory overhead penalties. This is very applicable to ABM research where cells in a grid world are often considered separate entities and therefore separate objects. Additionally, as ABM become more complex and data-oriented, their memory usage will rise and, at some resource limit, memory efficiency considerations (i.e. complexity reduction) will need to be made.

This is where the Entity-Component-System (ECS) architectural design pattern comes into play. Not attributed to any one individual, ECS arose from Games Development and is still widely used today[2]. Video Games and ABM share a lot of similarities. In fact, one could view an ABM as simply a video game where the agents are the players. This sentiment is shared by other researchers such as Szczepanska et al. [121] where they argue that Video Games and ABM can be viewed as highly related methodologies that seek to achieve different things (Games for leisure, ABM for research). Video games use ECS because they are data-intensive and require that entities (game objects) be dynamic. Modern data-oriented ABM are also data-intensive and requite that entities (agents) be

[1] It is also worth mentioning that even if multiple inheritance is allowed, it is considered bad practice.
[2] The two most popular game development engines *Unity3D* and *Unreal Engine* use variations of ECS.

dynamic. At the very least, this commonality warrants that ECS be considered as a design pattern for developing ABM.

As seen in Figure 3.2, a typical ECS framework consists of three main elements [122]:

- **Entities:** An entity is just a unique identifier with a container of components. It does not have any logic as its only purpose is to keep track of which components said entity currently possesses.
- **Components:** A component is a plain-old-data (POD) class that encapsulates data associated with a particular part of a program (e.g. A *ResourceComponent* may be associated with an entity that can gather and consume resources). Components traditionally do not implement any logic either and are dynamic meaning that they may be removed or added to an entity during the execution cycle of the program.
- **Systems:** Systems are the real "meat" of the design pattern. They run in fixed order and execute program logic using components (A *ResourceSystem* might act upon *ResourceComponent*s). In order for an agent to be 'seen' by a system, it must just have the appropriate components.

In ECS, composition is favoured over inheritance. That is to say that if you want to add functionality to an entity, you simply give it the appropriate components as opposed to the OOP method of inheriting from some base class that encapsulates the functionality. This completely removes the need to manage complex inheritance hierarchies given that there are no inheritance hierarchies to begin with. Similarly, if you want add or remove behaviour from an entity, you simply add or remove the appropriate component(s) given that the systems act upon the components directly, not the entities. This allows for extremely flexible and scalable systems.

With regards to memory efficiency, mature ECS frameworks may forgo objects entirely in favour of storing component data in large contiguous arrays. This greatly increases memory efficiency as all object related bloat is removed and the likelihood of incurring cache misses are reduced. It is worth pointing out that these object-less ECS frameworks are incredibly difficult to implement and if careful consideration is not made with regards to the accessibility of the system, it may be completely unusable to non-specialists.

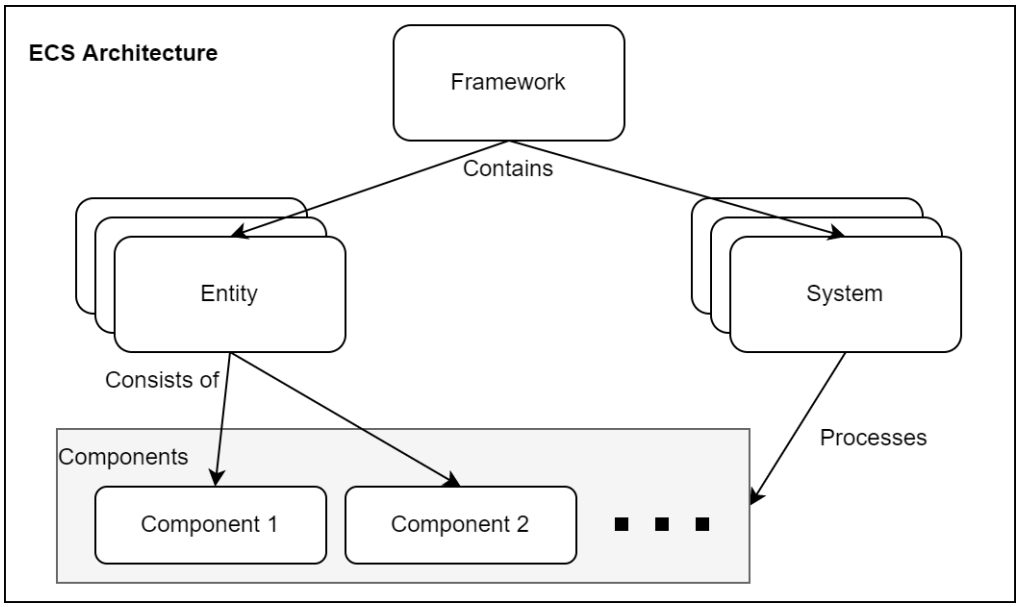


FIGURE 3.2: The structure of a typical ECS framework. Adapted from Hatledal et al. [6].

3.1.2 Framework

ECAgent is a Python-based framework for creating ABMs. Python was chosen because it is easy to learn, has a plethora of packages that can be easily integrated into any Python program, and when performance is of concern, can directly interface with shared libraries written in C for maximum performance. Figure 3.3 shows a high-level overview of *ECAgent*. For the most part, the general guiding principle of *ECAgent* is that Agents in ABMs and Entities in ECS are, from an abstract perspective, functionally equivalent.

3.1.2.1 Model

The *Model* class is just a wrapper for all of *ECAgent*. It contains an *Environment* for the *Agent*s to occupy, a *SystemsManager* for managing the execution order of *System*s, a *Logger* for storing data logs of a model's run and a pseudorandom number generator for if users was to introduce reproducible stochasticity to their models.

This *Model* class is intended to be inherited from with users adding *Agent*s and *System*s to the *Environment* and *SystemsManager* (respectively) as necessary.

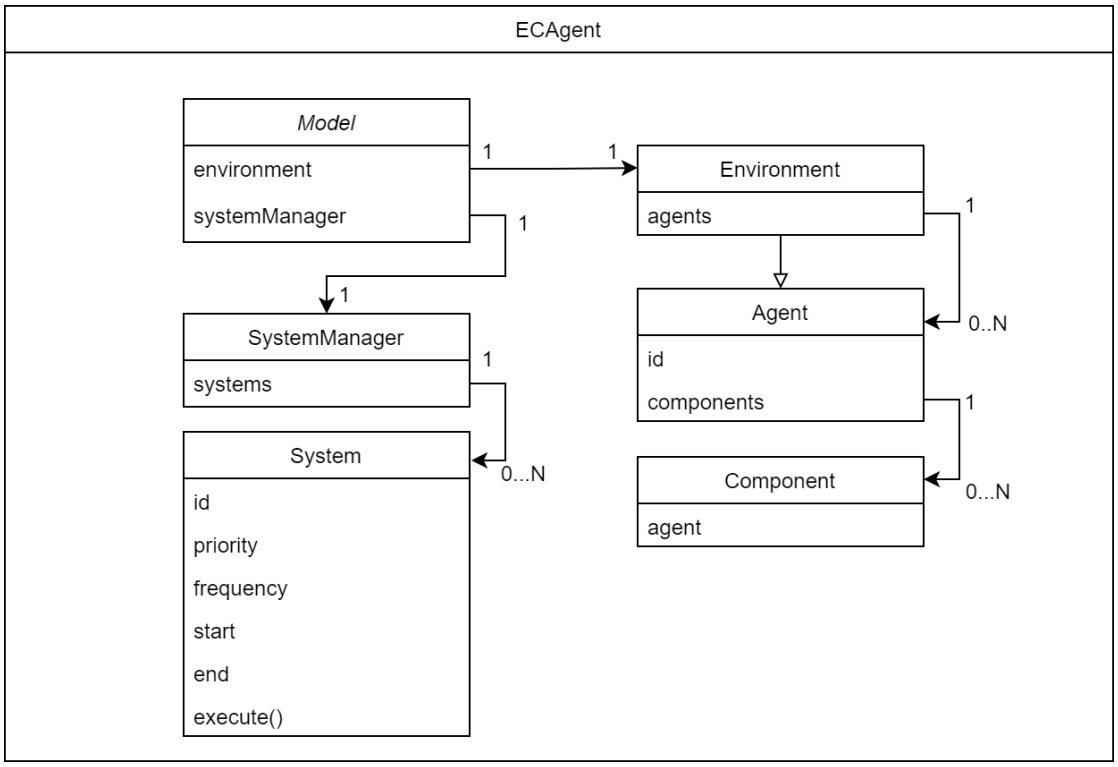


FIGURE 3.3: High-level overview of *ECAgent*.

3.1.2.2 Agents (Entities)

*Agent*s in *ECAgent* follow the same design principles as Entities in typical ECS frameworks. Their only function is to be uniquely identifiable *Component* containers. As such, the *Agent* class simply contains a unique identifier (*id*) and a list of *Component*s attached to the Agent. If a user wants to add a component to an agent, they can simply call the *addComponent* method and, conversely, if they want to remove a component from an agent, they can call the *removeComponent* method.

The container used for the components is a dictionary. It uses the *Type* of the *Component* as the key and the actual component as the value. This means that it is not possible to add two components of the same type to an agent (i.e. An agent can't have two *ResourceComponent*s). This design decision is largely motivated by the fact that we could not think of many circumstances where an agent would need more than one of a given component. Secondly, this implementation facilitates runtime debugging by throwing an error if an agent is given two of the same components.

3.1.2.3 Environment

As noted in Section 2.1.3, ABM environments are not clearly defined. They can be void of characteristics or incredibly detailed recreations of real-world locations. Discretized multi-dimensional lattices or graph structures are common (The former called a *LineWorld, GridWorld or CubeWorld* depending on the dimensions in *ECAgent*). ECS frameworks do not have typically have environments (Entities are simply stored in a container managed by the framework itself) so there is no clear solution to this problem.

In *ECAgent*, an *Environment* is considered an *Agent*. Specifically, it is a special type of agent that is responsible for keeping track of all agents occupying said environment. Every *Model* keeps a reference to at least one *Environment*. Users can then add an *Agent* to the environment by calling the *addAgent* method. Similarly, an agent can be removed from the environment using the *removeAgent* method.

The motivation for making the environment an *Agent* is twofold. First, the environment is often considered an entity in many ABM applications with some modellers going so far as to consider each cell on a grid world as a separate entity. By making the environment a specialized type of *Agent*, it is capable of storing *Component*s and having *System*s act upon it (Using the same *addComponent* and *removeComponent* methods used by *Agent*s). Secondly, *ECAgent* provides a special type of component called a *cell component* for the aforementioned discrete multi-dimensional lattice environments. Unlike OOP methods, these cells are not objects but rather *Numpy ndarrays* stored in a *Pandas* dataframe. Each row in the dataframe represents a different cell component and each column stores the values of said component for each cell in the environment. Cell components are added using the *addCellComponent* method exclusive to the *LineWorld, GridWorld* and *CubeWorld* environment types. Furthermore, Figure 3.4 demonstrates the distinction between standard *Components* and *cell components*. By leveraging a pure data-oriented approach offered by *Numpy* and *Pandas*, modellers can make use of code vectorization and be more memory efficient.

Lastly, the *Environment* class also stores agents using a dictionary where the agent's *id* is the key and the agent object is the value. This was done for the same reason as noted above, it makes runtime debugging easier as it is impossible to add an agent to the environment twice.

Component
property 1
property 2
...
property N

Cell Components				
	Cell 1	Cell 2	...	Cell N
Property 1				
Property 2				
...				
Property N				

FIGURE 3.4: A figure showing the difference between regular *Component*s (left) and *cell components* (right). A *Component* is a POD object that stores some number of properties whereas *cell components* are stored contiguously as rows in a *Pandas* dataframe for a discrete lattice environment of some arbitrary size.

3.1.2.4 Systems

As noted above, systems are the "meat" of any ECS architecture as they contain program logic and execute said logic on the one or more types of components. In *ECAgent*, a *System* is defined by several properties:

- **id**: A unique identifier for the system.
- **priority**: An integer value that determines the order of execution of the model's systems. A higher value means earlier execution in an iteration. This value defaults to 0.
- **frequency**: An integer value that determines how often a system should run for. A value of 5 means that the system will execute every 5 iterations.
- **start**: An integer that determines at which iteration the system will start executing from. This value defaults to 0.
- **end**: An integer that determines at which iteration the system should stop executing. This value defaults to the value of Python's *sys.maxsize*.

All *System*s are maintained by the *SystemManager*. The *SystemManager* is stored as a local variable of the *Model* and systems can be added or removed from the model by calling the *SystemManager*'s *addSystem* and *removeSystem* methods respectively.

When users implement their own *System*s, they must inherit from the base *System* class and overload the *execute* method. All program logic for a single iteration must be written or called from within this method which the *SystemManager* will automatically call when the *executeSystems* method is called by the user in their main loop.

Like *Agents*, a *System*'s identifier is not type based as with *Components*. This makes it possible for a model to hold two of the same systems (possibly with two different input parameters) as long as they have different *id*s. The motivation for this what that we could see a world in which the same system could be used to apply different logic to components depending on the timestep of the model run.

Lastly, their is no rigid execution structure imposed by the base *SystemManager*. This means that systems can be added or removed during a simulation run as well as have all of their aforementioned properties changed and the *SystemManager* will automatically update the systems' order of execution. This can be incredibly useful if the logic of your model needs to change dynamically based on its current state.

3.1.2.5 Other Features

There are several additional features that are offered by *ECAgent*:

- *Collectors*: These are a set of systems that automatically collect data during a simulation run.
- *ILoggable*: An interface class that exposes Pythons loggers to allow users to easily create an efficient logging system to track the status of their model.
- *Decoders*: A set of classes that allow you to construct models entirely from data-interchange files such as JSON or XML.

The *Collector*s are just specialized *System*s that store data in a local property called *records*. By inheriting from *Collector* and overload the *collect* method, a user can create a custom data collector. *ECAgent* also offers the *AgentCollector* and *FileCollector* which allow for the automatic collection of agent properties and writing of data to files respectively.

The *ILoggable* interface gives each system access to its own Python logger[3]. This can be configured to write data directly to files or to even send email notifications when large simulations are complete. This feature is almost entirely meant for larger simulations which take hours or days to run.

*Decoder*s are still in their infancy but they are intended to be an entirely text-based method for creating and executing *Model*s. In their current state, a *Decoder* will read in either a JSON or XML file (*JSONDecoder* and *XMLDecoder* respectively) and construct the model in accordance with the parameters specified in said JSON or XML file. This allows users to create highly reusable code given that if they want a different model (or model with different parameters) to be created, they simply just need to point the decoder to a different JSON or XML file. For larger simulations, one can envision sending a data-interchange file to a decoder (via some remote connection or file transfer) which would then automatically build and execute a model storing the results for processing.

3.1.3 Case Study: A Simple Predator-Prey Model

In order to demonstrate the capabilities of *ECAgent*, we decided to implement a simple predator-prey model as described by Tatara et al. [7]. They used this model to demonstrate the capabilities of *Repast Simphony*[4] so, given the common goal, it was deemed appropriate.

In this model, there are three types of entities: *Sheep*, *Wolves* and *Grass*. The *Sheep* eat the *Grass*, the *Wolves* eat the *Sheep* and *Grass* regrows after a random amount of time. The model is intended to show very basic population dynamics namely, the carrying capacity of an ecosystem with predation.

As shown by Figure 3.5, this model is very easy to translate to an ECS architecture and implement in *ECAgent*[5]. Both the *Wolf* and *Sheep* agents use energy. Given this, we can create an *Energy* component like so:

[3] Documentation for the Python logger can be found here: `https://docs.python.org/3/library/logging.html`

[4] *Repast Simphony* is a family of advanced, free, and open source agent-based modeling and simulation platforms available at: `https://repast.github.io/`

[5] The source code for *SimplePredatorPrey* implemented in *ECAgent* is available at the following url: `https://github.com/BrandonGower-Winter/ECAgentTutorials/tree/master/SimplePredatorPrey`

```
class EnergyComponent(Component):
    def __init__(self, agent: Agent, model: Model, energy: float):
        super().__init__(agent, model)
        self.energy = energy
```

This component only stores a float variable which denotes how much energy agents with this component have.

Given that the *EnergyComponent* is the only component we will need to create the mobile (*Sheep* and *Wolf*) entities, we can now create the agents as follows:

```
class Sheep(Agent):

    gain = 1.0
    reproduce_rate = 0.01
    sheep_counter = 0

    def __init__(self, model: Model, energy: float = None):
        super().__init__('s{}'.format(Sheep.sheep_counter), model)

        self.addComponent(
            EnergyComponent(
                self, model, energy if energy is not None
                else model.random.random() * 2 * Sheep.gain
        ))

        Sheep.sheep_counter += 1

class Wolf(Agent):

    gain = 1.0
    reproduce_rate = 0.01
    wolf_counter = 0
```

```
def __init__(self, model: Model, energy: float = None):
    super().__init__('w{}'.format(Wolf.wolf_counter), model)

    self.addComponent(
        EnergyComponent(
            self, model, energy if energy is not None
            else model.random.random() * 2 * Wolf.gain
    ))

    Wolf.wolf_counter += 1
```

As shown above, to create the agents we create custom classes that inherit from the base *Agent* class and, upon initialization, give both agent types an *EnergyComponent*. The initial value of the energy is either specified by the optional *energy* parameter in *__init__* or it is randomly initialized.

Both the *Sheep* and *Wolf* have three class properties:

- *gain*: The amount of energy gained after consuming a resource.
- *reproduce_rate*: The rate at which the agent type reproduces (Spawns additional agents)
- *counter*: An integer counter that ensures that each agent created has a unique identifier.

Sheep agents' *id*s start with 's' while *Wolf* agents' *id*s start with 'w'. This just allows us to easily identify the type of agent we are dealing with.

From a ECS design perspective, our agent implementations are complete. We have ensured that each agent is uniquely identifiable and will have the appropriate *Component*s attached to it upon initialization. However, they do not have any program logic so they do not actually do anything. To fix this, we can start adding in our *System*s of which we have identified four:

- *MovementSystem*: The system responsible for moving the sheep and wolves across the environment.

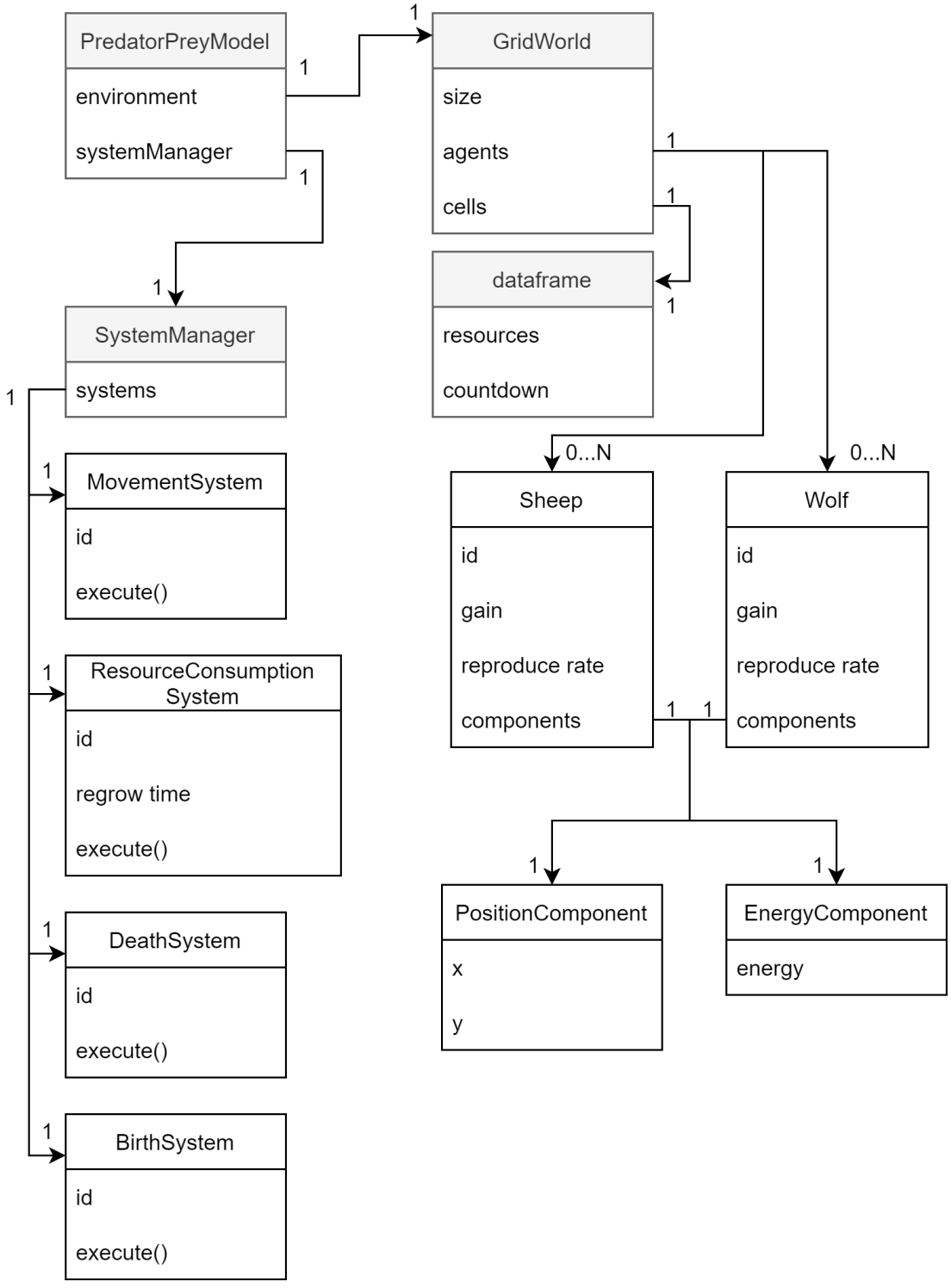


FIGURE 3.5: UML Diagram of the Simple Predator-Prey Model implemented in *ECAgent*. Classes highlighted in grey are included with *ECAgent*.

- *ResourceConsumptionSystem*: The system responsible for managing the consumption of new resources. This includes sheep eating grass, wolves eating sheep and grass regrowing.
- *DeathSystem*: The system responsible for removing sheep and wolves with no energy remaining.
- *BirthSystem*: The system responsible for stochastically adding new sheep and wolves to the simulation.

We implement the *MovementSystem* as follows:

```
class MovementSystem(System):

    def __init__(self, id: str, model: Model):
        super().__init__(id, model)

    def execute(self):
        upper_bound = self.model.environment.width -1
        for agent in self.model.environment.getAgents():

            # Move within Moore Neighbourhood
            newX = max(0, min(upper_bound, agent[PositionComponent].x
            + int(round(2* self.model.random.random() - 1))))

            newY = max(0, min(upper_bound, agent[PositionComponent].y
            + int(round(2* self.model.random.random() - 1))))

            # Spend Energy
            agent[EnergyComponent].energy -= 1

            agent[PositionComponent].x = newX
            agent[PositionComponent].y = newY
```

We create the *MovementSystem* by inheriting the *System* base class and overloading the *execute* method. In this method, we iterate over every agent using *Environment.getAgents()*.

For each agent, we randomly assign it a new position within its Moore Neighbourhood. We use *model.random* to ensure that the model's psuedorandom number generator is used.

To actually move the agents about the environment, we make use of a special component. The *PositionComponent* is automatically given to every agent added to our model's environment. It stores the x and y coordinates of our agent which we can modify directly. We do this by getting the *PositionComponent* using the index *[ComponentType]* operator. This is functionally equivalent to calling *Agent.getComponent(ComponentType)* and is only used as syntactic sugar. After the agent has moved, we reduce its energy (By accessing the *EnergyComponent*) by 1.

The next *System* to create is the *ResourceConsumptionSystem* which is the most complex part of the model given that it strays from traditional OOP design principles. It is implemented as follows:

```
class ResourceConsumptionSystem(System):

    def __init__(self, id: str, model: Model, regrow_time: int):
        super().__init__(id, model)

        self.regrow_time = regrow_time

        def resource_generator(pos, cells):
            return 1 if model.random.random() < 0.5 else 0

        # Generate the initial resources
        model.environment.addCellComponent('resources',
                                           resource_generator)

        def countdown_generator(pos, cells):
            return int(model.random.random() * regrow_time)

        # Generate the initial resources
        model.environment.addCellComponent('countdown', countdown_generator)
```

```
def execute(self):

    # Get resources data
    cells = self.model.environment.cells
    resource_cells = cells['resources'].to_numpy()
    countdown_cells = cells['countdown'].to_numpy()

    eaten_sheep = []
    targets_at_pos = {}
    environment = self.model.environment


    # Process Sheep and Wolves first
    for agent in environment.getAgents():

        posID = discreteGridPosToID(
                        agent[PositionComponent].x,
                        agent[PositionComponent].y,
                        self.model.environment.width)

        # Is wolf or is sheep
        if agent.id.startswith('w'):

            # Get all agents at position
            if posID not in targets_at_pos:
                targets_at_pos[posID] = environment.getAgentsAt(
                                        agent[PositionComponent].x,
                                        agent[PositionComponent].y)

            for target in targets_at_pos[posID]:
                # If sheep
                if target.id.startswith('s') and
                target.id not in eaten_sheep:
```

```
                # Mark Sheep for death
                eaten_sheep.append(target.id)
                agent[EnergyComponent].energy += Wolf.gain
                break

    elif agent.id not in eaten_sheep:
        # Check is grass is Alive
        if resource_cells[posID] > 0:
            # Consume and Gain Energy
            agent[EnergyComponent].energy += Sheep.gain
            resource_cells[posID] = 0

# Remove eaten sheep
for sheep in eaten_sheep:
    environment.removeAgent(sheep)

# Regrow Grass
countdown_cells[resource_cells < 1] -= 1
mask = countdown_cells < 1
resource_cells[mask] = 1

countdown_cells = numpy.where(mask, numpy.asarray(
[
    int(self.model.random.random() * self.regrow_time)
    for i in range(len(countdown_cells))
]), countdown_cells)

self.model.environment.cells.update(
{'resources': resource_cells, 'countdown': countdown_cells}
)
```

First, we inherit from the *System* base class and overload the *execute* method. During

initialization, we set the *regrow_rate* input parameter which specifies the maximum number of iterations a grass entity will be inactive for. We then add two *cell components* to the model using the *Environment.addCellComponent* method. We pass the name of the cell components (*resources* and *countdown* respectively) and two functors which generate the initial value of each cell. These 'generator' functors take the position of the cell and the *cells* dataframe as input. For the *resources* cell component, the generator randomly returns a 0 or 1. For the *countdown* cell component, the generator returns a random integer $\in [0, regrow\_time]$. Both of these components (*resources* and *countdown*) represent the grass entities occupying the environment. These two components are stored in a *Pandas* dataframe as contiguous *Numpy* arrays. We can access these arrays by calling *Environment.cells[ComponentName]* as seen in the *execute* method.

In the *execute* method, we loop over every agent and determine its *posID*. Given that the *cell components* are stored contiguously (i.e. In a 1D array), we need to convert the agent's position from a 2D value, into a 1D value. We do this using the *discretePosToID* method included with *ECAgent* and store it as the *posID*. If the agent is a *Wolf* (i.e. *id* starts with a 'w'), we get a list of all agents at its current position using the *Environment.getAgentsAt* method and loop over them. If any one of them is a *Sheep*, the *Wolf* consumes it and gains energy equal to the value of *Wold.gain*. A *Wolf* will only eat one *Sheep* at a time.

If the agent is a *Sheep*, it will look to see if its current cell has any resources (i.e. the grass cell at postion *posID* has a *resource* value of 1.0). If the cell has resources, it consumes them and gains energy equal to *Sheep.gain*. The resources at that current cell are then set to 0.0.

After both agent types have eaten, all dead *Sheep* are removed from the environment using *Environment.removeAgent*. The cells are then updated using *Numpy* masks and code vectorization. This code is can be hard to understand if a user is not familiar with *Numpy* but what is happening is that the *countdown* cell component for all cells who currently have no resources (i.e. a *resource* component of 0.0) is decremented by 1.0. We then check to see if any of these resource-less cells have reached a *countdown* value of 0.0 and if so, they are given a new random *countdown* value $\in [0, regrow\_time]$ and have their *resource* component set to 1.0. Because of *Numpy*'s code vectorization, these operations are applied to all cells simultaneously which is much faster than traditional

OOP methods which typically evaluate each cell independently. Lastly, the new *resources* and *countdown* values are commited to the environment using *Environment.cells.update*.

The *DeathSystem* is self explanatory. It is responsible for removing agents from the environment when their energy is depleted. It is implemented as follows:

```
class DeathSystem(System):

    def __init__(self, id: str, model: Model):
        super().__init__(id, model)

    def execute(self):
        toRem = []

        for agent in self.model.environment.getAgents():
            if agent[EnergyComponent].energy <= 0:
                toRem.append(agent.id)

        for a in toRem:
            self.model.environment.removeAgent(a)
```

Once again, we inherit from *System* and overload the *execute* method. We loop over all agents and if their *energy* (which we get from by accessing the *EnergyComponent*) is less than or equal to 0, we remove the agent using *Environment.removeAgent*. The method takes the agent's *id* as input.

Similarly, the *BirthSystem* is just reponsible for stochastically adding new *Sheep* and *Wolf* agents to the environment. It is implemented as follows:

```
class BirthSystem(System):
    def __init__(self, id: str, model: Model):
        super().__init__(id, model)

    def execute(self):
```

```
        for agent in self.model.environment.getAgents():
            if agent.id.startswith('w') and
            self.model.random.random() < Wolf.reproduce_rate:

                agent[EnergyComponent].energy /= 2.0

                # Birth Wolf
                self.model.environment.addAgent(
                    Wolf(
                        self.model,
                        energy=agent[EnergyComponent].energy
                    ),
                    xPos = agent[PositionComponent].x,
                    yPos = agent[PositionComponent].y
                )

            elif self.model.random.random() < Sheep.reproduce_rate:
                # Birth Sheep
                agent[EnergyComponent].energy /= 2.0

                # Birth Wolf
                self.model.environment.addAgent(
                    Sheep(
                        self.model,
                        energy=agent[EnergyComponent].energy
                    ),
                    xPos=agent[PositionComponent].x,
                    yPos=agent[PositionComponent].y
                )
```

Yet again, we inherit from *System* and overload the *execute* method. We loop over all agents and check to see what type of agent they are by looking at the first character of their *id*. Both agents reproduce in the same way, a random number $\in [0, 1]$ is generated and if it is less than the agent type's *reproduce_rate*, a new agent (of the same type) is

spawned. Half the *energy* of the parent is given to the child agent. We add new agents to the environment using *Environment.addAgent*. Lastly, the position of the child agent is set to that of the parent's position.

Given that we are interested in monitoring both the *Sheep* and *Wolf* populations. It is probably worth developing a mechanism to record these values. We do that by making use of the *Collector* class as follows.

```
class DataCollector(Collector):

    def __init__(self, id: str, model):
        super().__init__(id, model)

        self.records = {'sheep': [], 'wolves': []}

    def collect(self):
        # Count Sheep
        self.records['sheep'].append(
            len([1 for a in self.model.environment.getAgents()
            if a.id.startswith('s')]))
        # Count Wolves
        self.records['wolves'].append(
            len([1 for a in self.model.environment.getAgents()
            if a.id.startswith('w')]))
```

*Collector*s are a special type of system in *ECAgent* meant to record model or agent properties. Here we've just implemented a simple *Collector* called *DataCollector*. It inherits from *Collector* (not *System*) and overloads the *collect* method (not the *execute* method). During initialization, we setup a dictionary that will store the population of the sheep and wolves. When the *collect* method is called, we count the number of sheep and wolves currently in the environment using simple list comprehension.

With all of *System*s created, we can create our predator-prey model as follows:

```
class PredatorPreyModel(Model):

    def __init__(self, size: int, init_sheep: int, init_wolf: int,
                 regrow_rate: int, sheep_gain: float, wolf_gain: float,
                 sheep_reproduce: float, wolf_reproduce: float):

        super().__init__()
        # Create Grid World
        self.environment = GridWorld(size, size, self)

        # Add Systems
        self.systemManager.addSystem(MovementSystem('move', self))

        self.systemManager.addSystem(ResourceConsumptionSystem('food',
        self, regrow_rate))

        self.systemManager.addSystem(BirthSystem('birth', self))

        self.systemManager.addSystem(DeathSystem('death', self))

        self.systemManager.addSystem(DataCollector('collector', self))

        # Parameterize Agents
        Wolf.gain = wolf_gain
        Sheep.gain = sheep_gain

        Wolf.reproduce_rate = wolf_reproduce
        Sheep.reproduce_rate = sheep_reproduce

        # Create Agents at random locations

        for _ in range(init_sheep):
```

```
            self.environment.addAgent(
                Sheep(self),
                xPos = self.random.randint(0, size - 1),
                yPos = self.random.randint(0, size - 1)
            )

        for _ in range(init_wolf):
            self.environment.addAgent(
                Wolf(self),
                xPos = self.random.randint(0, size - 1),
                yPos = self.random.randint(0, size - 1)
            )
```

We create the *PredatorPreyModel* model by inheriting from *Model*. Upon initialization, the model accepts eight input parameters:

1. *Size*: The size of the grid world. This value dictates both the width and height of the grid world (i.e. The grid world is square).
2. *Initial Sheep*: The number of *Sheep* agents to initialize.
3. *Initial Wolves*: The number of *Wolf* agents to initialize.
4. *Regrowth rate*: The max number of iterations it will take for a grass cell to regrow.
5. *Sheep Gain*: The amount of energy a *Sheep* gains when it consumes a grass cell.
6. *Wolf Gain*: The amount of energy a *Wolf* gains when it consumes a *Sheep*.
7. *Sheep Reproduction Rate*: The rate at which *Sheep* agents spawn additional *Sheep* agents.
8. *Wolf Reproduction Rate*: The rate at which *Wolf* agents spawn additional *Wolf* agents.

These are the same parameters specified by Tatara et al. [7] and are intended to be user configurable. The first thing we do is create the *GridWorld* that the agents will occupy. The *Model* class adds a *VoidWorld* (dimensionless) environment by default. By creating

Parameter	Value
Size	50
Grass Regrowth Rate	30 (iterations)
Initial Sheep	100
Initial Wolves	50
Sheep Gain	4
Sheep Reproduce	4%
Wolf Gain	25
Wolf Reproduce	6%

TABLE 3.1: Input Parameters of the Simple Predator Prey Model. These values are exactly the same as those presented in Tatara et al. [7].

the *GridWorld*, we ensure that any agents added to the environment will automatically get a *PositionComponent* attached to them. We can also now create cell components which are not available in *VoidWorld*s. The size of the *GridWorld* is set to be the value of the *size* parameter.

Each *System* is added using the *SystemManager.addSystem* method and given a unique string *id*. We do not specify the *priority, frequency, start* or *end* properties is this case. This is because we want our *System*s to execute every iteration, from iteration zero until the end of the simulation run. By not specifying a priority, the *SystemManager* treats the execution order as 'first-come first serve' meaning that *MovementSystem* will always execute first and the *DataCollector* will always execute last.

We then specify some of the other user configurable properties *gain* and *reproduce* for both the *Sheep* and *Wolf* agent types and create our agents (and add them to environment) using the *Environment.addAgent* method. Each agent's position is randomly assigned to somewhere on the grid world.

3.1.3.1 Validation

In order to validate that the model was producing expected results, we ran the model using input parameters listed in Table 3.1 with a seed of 345968. The model ran for 1000 iterations and the results are presented in Figures 3.6 and 3.7.

Looking at figure 3.6, we can see that the model is indeed behaving as expected, there are clear oscillations in the *Sheep* and *Wolf* populations. They are also clearly related with *Sheep* population loss occurring during *Wolf* population gains. Conversely, *Wolf*

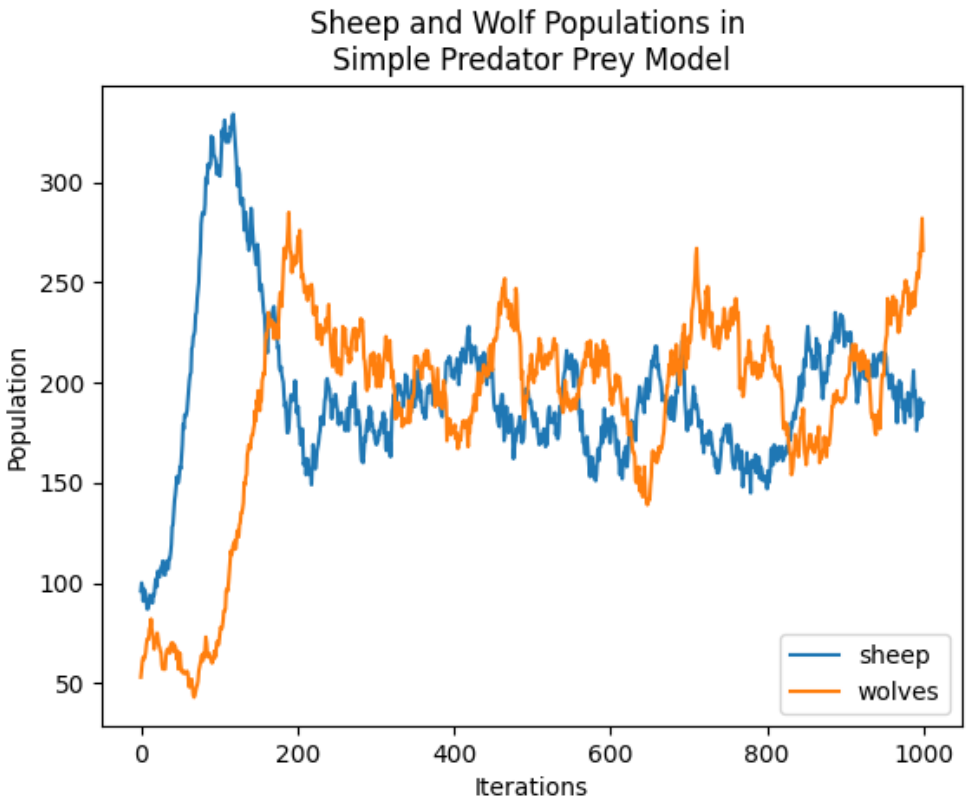
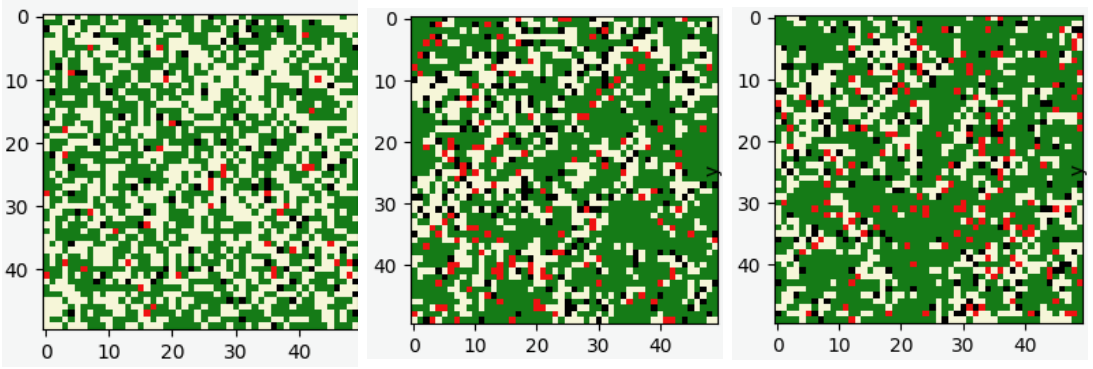


FIGURE 3.6: *Wolf* and *Sheep* populations simulated over 1000 iterations.



(A) (B) (C)

FIGURE 3.7: Figures of the Predator-Prey model at $t = 0$ (a), $t = 400$ (b) and $t = 600$ (c). *Grass* cells are green if they have resources and light yellow if they are empty. *Sheep* are represented as black pixels and *Wolves* are red.

population loss (as a result of a low *Sheep* population) causes gains in the *Sheep* population. This if further confirmed by looking at Figure 3.7 where we can visually see changes in the *Sheep* and *Wolf* populations as the simulation progresses[6].

3.1.4 Discussion

In this section, we sought to demonstrate the applicability of the ECS design pattern for creating ABM. The primary motivation for this was that traditional OOP design patterns were limiting when it came to making truly data-oriented applications. ABM are becoming increasingly data-oriented and, if adaptive-mechanisms (ML or otherwise which are data-oriented by their very nature) are truly to be adopted by the mainstream, an alternative design paradigm (ECS) is needed. We achieved our goals by developing *ECAgent*, a Python-based ECS framework for developing ABM and using it to create a simple predator-prey model based on work by Tatara et al. [7]. Overall, we consider the entire process a success and discuss the advantages, limitations and future prospects of *ECAgent* below.

First and foremost, it was quite surprising to see just how analogically compatible ECS design principles were with ABM development. We do not claim that they are more analogically compatible than OOP design principles but, viewing entities and components as the biological or cultural makeup of the agent and the systems as the processes or actions the agents executes is, in our opinion, both effective and appropriate. Additionally, the predator-prey model is both scalable and can easily be adapted without worrying about creating conflicts in the code base. If a user wanted to add a new agent type, that could easily be done and replacing or reorganizing one or more of the systems is trivial. We attribute this directly to the ECS design pattern.

One benefit that is not obvious in the predator-prey model but is relevant to *ECAgent* as whole is how easy it is to incorporate any Python library into a custom model. *Numpy* and *Pandas* are already used by *ECAgent* but if a user wanted to use a ML library, such as *Pytorch*, it would as trivial as ensuring the package is installed (using *PIP* for example) and that the appropriate modules have been imported. This benefit is understated but, it arguably *ECAgent*'s greatest strength. This is also true interfacing with C code (custom or library-based). Standard *Python* is known to perform quite poorly but, using *Cython*

[6] A video of this entire simulation run is available at: `https://youtu.be/aoGbJh6hxxk`

or otherwise, a modeller could relatively easily interface with memory efficient and, more importantly, fast code written in C.

Despite these promising results, creating the predator-prey model did illuminate some of *ECAgent*'s limitations. The foremost issue in this regard is the infancy of codebase. There are a lot of quality of life (QoL) features that *ECAgent* needs before it would ever see mainstream adoption. Verbose documentation, a suite of introductory tutorials and general ABM features (such as agent shuffling) are but a few of these requirements. In fact, the lack of a dedicated visual interface would immediately make some modellers not consider using *ECAgent*. It is also worth mentioning that data-oriented programming is a huge paradigm shift from OOP and could be very difficult to grasp for non-specialists. This issue is most evident in the updating of the *cell components*. Given that operations are performed on all the components at the same time, there are times where it can be difficult to figure out how to do that most effectively.

Future prospects for *ECAgent* include adding a robust visual interface, adopting a more data-oriented approach to entity design and supporting vectorization on regular components. Given that the current implementation of entities and components is object-based, effort should be made to uncover the best method for incorporating these data-oriented features while maintaining user friendliness. QoL features such as entity tags, class components (components for an entire class not just a single entity) and general performance improvements (such as moving the core framework to *Cython*) would also be beneficial. Nevertheless, we believe *ECAgent* is already capable of facilitating the design of real-world, complex, ABM.

3.2 Designing Adaptive-Agents using Information Exchange

In Agent-based Modelling, adaptability refers to the ability of agents to not only modify the actions they take but, also the strategies they use to determine which actions to take [4]. Implementing adaptive-agents is a complex task. This is, in part, because there is no formal approach to creating them. Similarly, determining if an agent is adaptive and, if so, quantifying its adaptive capacity is equally ambiguous.

In this section, we aim to address these issues by defining what adaptability is (Section 3.2.1), how it can be measured (Section 3.2.2) and how adaptability can be viewed as

simply the capacity to exchange information (Section 3.2.3). We then formally describe how to create adaptive-agents using information exchange (Section 3.2.4) and demonstrate the effectiveness of this approach by developing a simple adaptive ant foraging model (Section 3.2.5). This section then concludes with a discussion (Section 3.2.6).

3.2.1 What is Adaptability?

Adaptability (or adaptive capacity) is a term that comes directly from the field of Ecology and the study of the resilience of Social-Ecological Systems (SES). The future trajectories of SESs are defined by three related attributes [123, 124] :

- **Resilience:** The capacity of a system to absorb disturbance and reorganize while undergoing change such that its identity is retained.
- **Adaptability:** The capacity of entities in a system to influence resilience.
- **Transformability:** The capacity to transform the stability landscape of a system such that a new system emerges when the old system is untenable.

For context, a SES is made up of a set of states. The extent to which this system remains stable (unchanged in identity) is called the stability landscape. The stability landscape is defined by a set of control variables which embed a system's basin of attraction (the states it tends towards). If a control variable's value breaches a threshold (each control variable has threshold levels), the system will become unstable and collapse. Resilience is then the capacity of a system to remain stable in the face of disturbance. Adaptability is the capacity of the entities within the system to influence (increase or decrease) the resilience of the system while transformability refers to the capacity to transform an unstable system into a stable one (that is distinct from its predecessor). Regime shifts or system transformations may be active or forced. That is to say that the transformation of a one system to another may be deliberately (active) or unintentionally (forced) induced by the entities of a system.

In ABM literature, resilience, adaptability and transformability are seemingly grouped under the moniker of adaptability[7]. This is understandable given that in ABM literature,

[7] A notable exception to this statement are ABM that are used to study the resilience of SESs.

adaptability broadly refers to the ability of agents to modify their strategies and the actions they take. It may not always be clear when the prerequisites for determining transformability (when a control variable breaches a threshold) have been met. Thus, simply referring to it as the adaptability of the model or entity is appropriate.

For the rest of this section, we adopt the same nomenclature found in ABM literature grouping resilience, adaptability and transformability as simply adaptability. Specifically we adopt this notion of adaptability to entities[8] at all scales.

In terms of scale, adaptability does not need to be measured on a population of entities as a whole (general resilience). In fact, the adaptability of a particular subset of the model can be measured. Moreover, the adaptability of a subset of the model in relation to one or more specific system disturbances can be measured . This is known as specified resilience [123]. This is extremely useful to ABM given that they are often scale agnostic and used to investigate the emergent phenomena that arise as a result of specific scenarios.

3.2.2 Measuring Adaptability

As noted in Section 3.2.1, adaptability is a broadly defined term in ABM literature. This has made it difficult for ABM literature to quantify the adaptive capacity of a model, entity or system. However, answers can once again be found in resilience literature where it has been noted that the multidimensional nature of resilience is not innately quantifiable but its sub-components might be [33]. These four components, which can be applied to the ABM definition of adaptability, are:

- **Recovery:** Time taken for a state variable to reach pre-disturbance event values.
- **Resistance:** The change in a state variable after a disturbance event.
- **Persistence:** Existence of an identifiable system through time. The system is identified by its ability to maintain state variables within predefined ranges.
- **Variability / Invariability:** Change of a state variable over time. Often used as a proxy for persistence.

[8] This is discussed later but, we use the word entity specifically. An agent is an entity but an entity does not need to be an agent.

To measure one of these components, specific state variables need to be monitored. For example, measuring population loss and the time taken to recoup those losses after a disturbance event would be an example of resistance and recovery respectively. Measuring the evolution of resource sharing beliefs in cooperative agents would be an example of measuring persistence and variability. Furthermore, these components are scale agnostic meaning that they can be applied directly to the agents themselves (what is the adaptive capacity of this agent) or to collective representations of them (what is the adaptive capacity of the entire population of agents).

Understanding how these components relate to the adaptability of agents is also important. An entity with more adaptive capacity will be able to resist the effects of a disturbance event and, in the case were it is affected by said event, recover faster. Relating persistence to adaptability is not as straight forward. In fact, persistence is not a measure of adaptability but rather a measure of the mechanisms used by the entities to achieve their adaptive capacity. If an entity maintains one state variable and changes another to adapt to a disturbance event, both the persistence of the first and the variability of the second variables could explain the aforementioned entity's adaptive capacity assuming the necessary experimentation is done before drawing any conclusions.

Lastly, we believe it is worth discussing the heuristic model of Adaptive Cycling which modellers may be able to use to explain the adaptive process of entities in their models. As described by Folke et al. [123], the adaptive cycle consists of four phases that typically follow the following trajectory:

1. *r-phase*: Resources are abundant and resilience is high.
2. *K-phase*: Resources are locked-up, their is little novelty and resilience is low.
3. Ω-*phase*: There is a collapse and the chaotic emergent dynamics cause the undoing of relationships and systems.
4. α-*phase*: A reorganization phase where novelty may prevail.

While it is possible for any phase to transition to another, the *r*-*k* dynamics typically reflect a predictable and "slow" foreloop with the Ω-α dynamics reflecting a chaotic and "fast" backloop that influences the dynamics of the next foreloop. ABM should not seek to explicitly model this behaviour, it should be an emergent property that the modellers

use to explain how their model, entities or systems transitioned from one identity to another.

3.2.3 Adaptation and Information Exchange

Despite the lack of a formal approach for creating adaptive-agents, one concept that is strikingly common when reading about adaptive-behaviour is *information exchange*. Specifically, the exchange of information, biological or cultural, between one or more entities (See Chli and De Wilde [125], Fitzhugh et al. [126] and Giuliani and Castelletti [127] for specific examples). Utilizing information exchange, we seek to fill a gap in the state of the art by outlining an approach for creating adaptive-agents.

For clarity, we view entities and agents separately. We consider an entity as simply a container of state and an agent as a decision making automata. That is to say that entities simply maintain information while agents also maintain information but are capable of using that information to make decisions. An agent is an entity but an entity is not necessarily an agent.

We posit that in order for an entity to facilitate adaptive-behaviour, it must be able to both maintain and exchange state. Furthermore, if that entity is an agent, it must explicitly utilize its state when making decisions such that when its state changes, so might its decisions. An agent that can maintain, exchange and utilize state when making decisions is considered an adaptive-agent[9].

Given this, we can now describe how to create adaptive-agents. A lot of research already covers designing agents that utilize and maintain state. Conceptual and cognitive models such as BDI [74], CLARION [128], CONSUMAT [129] and PECS [2] already exist in abundance giving modellers several options of varying complexity to choose from. We will instead focus on the state exchanging aspect of introducing adaptability and assume that a sufficient cognitive model for the agents has been chosen or developed.

To facilitate information (state) exchange between entities, we adopt a network-based approach. In this network (that represents the entire ABM), entities are the nodes (vertices) and their ability to communicate (exchange information) is defined by the

[9]This is explicitly referring to the Agent-based Modelling definition of adaptability. While information exchange can facilitate the implementation of prediction, goal-orientation and learning mechanisms, they are not required in order to create an 'adaptive-agent' as it known in Agent-based Modelling.

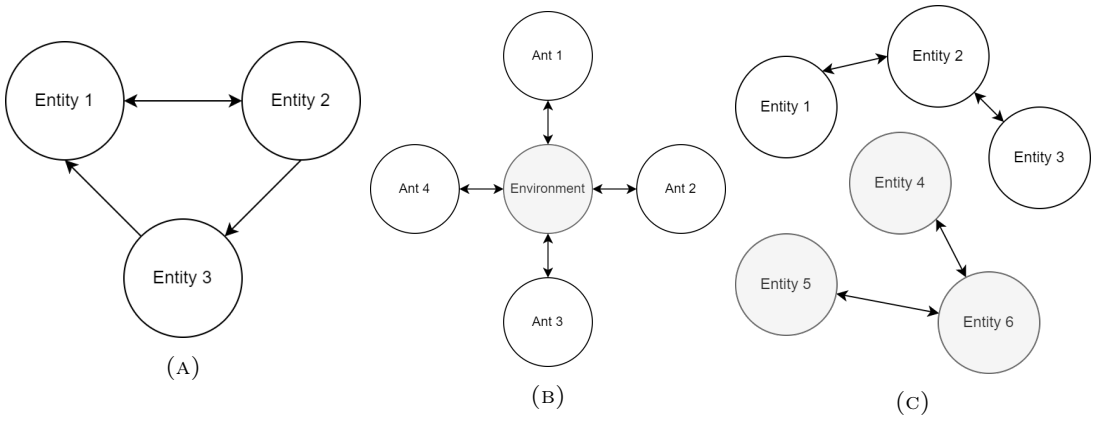


FIGURE 3.8: Various types of information exchange networks. (a) Showcases an example of direct exchange. Connections are directed meaning that Entity 1 and Entity 2 can exchange information while Entity 3 can only receive information from Entity 2 and send information to Entity 1. (b) Showcases an example of indirect exchange. Ants do not directly exchange information amongst themselves. They instead communicate indirectly by exchanging information with the environment entity. (c) Showcases exclusive exchange whereby Entities 1, 2 and 3 are able to exchange information amongst themselves but unable to exchange information with Entities 4, 5 and 6 which form their own sub-network.

directed connections between them. If an entity (the source) is connected to another entity (the target), the source entity can send information to the target entity. Given that information exchange may be an imperfect or even transformative event, information that travels from the source to the target is governed by a *communicative* function. These *communicative* functions are intended to represent both the biological and cultural processes that facilitate information exchange from one entity to another.

In essence, our proposed design process is to view the ABM as a network of interconnected entities. With this approach, facilitating adaptive-behaviour is just a process of ensuring your agents are connected to this network. The nuance to this admittingly simple approach is the flexibility is provides. Information exchange may be temporal (e.g. information exchanged may only reflect in the target entity's state on the next iteration) or instantaneous (e.g. signalling for collective action). The *communicative* functions may support the direct transfer of information or they may manipulate it. Additionally this methodology facilitates adaptation arising as the result of direct, indirect and exclusive information exchange (See Figure 3.8).

In terms of conceptual accessibility, this methodology is appropriate for both biological and sociological ABM alike. For example, a simple reproductive network can be created mapping the exchange of genetic information from source (parent) entities to target

(child) entities. The *communicative* function maps this exchange and, if found desirable by the modeller, may include mutation as a transformative process. From a sociological perspective, social hierarchies, social circles, collective action, and many other group functions are innately network-based in that collectives are, by definition, a group of connected entities.

It is worth discussing state and what state means. We adopt the perspective that state is a set of properties that represent the current 'state' of an entity, system or model. State properties may be tangible (resources), intangible (hunger), ephemeral (social status) or even permanent (death). Given this, we view entities as state templates. This means that entities themselves are just maintainers of state and different entities may capture different subsets of the model's global state. With this definition, collective representation is possible (e.g. a settlement entity could exchange information with other settlements and relay this information to its denizens) but, the *communicative* function that governs that edge may need to transform the source entities' state representation such that it matches the target entities' state representation.

One aspect of the model that may not be clear from the above conceptualization is that it is possible to determine if an agent is capable of adaptation. If an agent does not receive any information from any additional entities (i.e. There are no entities that connect to it), adaptation will not occur. We believe this will prove beneficial in facilitating the creation of adaptive-agents. Furthermore, similar analysis could also be applied dynamically as the edges of the network change from one timestep to another opening additional avenues for quantifying the adaptability of an ABM.

3.2.4 Formal Definition

Using the concepts discussed in Section 3.2.3, we formally define the process of creating adaptive-agents using information exchange below:

Consider an ABM M at some arbitrary timestep t. We can define this ABM M_t by a 4-tuple (S_t, E_t, A_t, ϕ_t) where $S_t \in S$ is the global state of the model at timestep t, $E_t \in E$ and $A_t \in A$ are the sets of entities and agents occupying the model at timestep t respectively and $\phi_t \in \phi$ is the set of processes (or systems) that executed at timestep t.

S_t is defined as the values of the properties captured by the model's global state at timestep t. That is:

$$S_t = \{p_0, p_1, ..., p_n\} \quad \forall p \in P_t \tag{3.1}$$

where $P_t \in P$ is set of all model M's properties at timestep t and p_n is the value of the n^{th} property.

An entity $e \in E_t$ is a state container. That means that it captures a subset of the global state: $e \subseteq S_t$. Entities can be grouped by type k such that E^K is that set of all entities of that type.

An agent $a \in A_t$ is defined as a 2-tuple (e, π) where $e \in E_t$ is the state information captured by the agent and $\pi(e)$ is the policy the agent follows. We adopt the Reinforcement Learning definition of policy whereby a policy π maps an agent's state (e in our case) to a probability distribution of actions [10]. Similar to entities, agents can be grouped by type k such that A^k is the set of all agents of type k.

While the formal definition of an ABM is our own, it is generally in agreement with other formal definitions of ABM (See page 38 of Lemos [130] for example). They key difference between our definition of an ABM and others is that we view all processes that change state as systems not actions. In our definition, the process of deciding and executing actions are systems and systems, in general, may facilitate state change independent of the actions of the agents. For example, generating climate conditions in a localized model studying the effects of environmental stress may be implemented or viewed as a system that acts independently of the agents' actions.

With these prerequisites, we now define the information exchange network at timestep t as a directed multigraph (or multidigraph) $G_t \in G$ (Equation 3.2):

$$G_t := (E_t, C_t, L_t) \tag{3.2}$$

where E_t is the set of entities (vertices), $C_t \subseteq \phi_t$ is the set of communicative functions that govern the exchange of information between one or more entities and L_t (Equation

3.3) is a mapping of directed edges between two sets of vertices X and Y given some communicative function $c \in C_t$:

$$L_t : (X, c) \rightarrow Y \tag{3.3}$$

Both X (the source entities) and Y (the target entities) are $\subseteq E_t$. X and Y do not need to be mutually exclusive meaning that information propagated by entities in X using communicative function c may also be received by entities in X if $e \in X$ and $e \in Y$.

A communicative function c is defined as a mapping of one or more source entities' properties to one or more target entities' properties (Equation 3.4):

$$c : (X, Y, \theta) \rightarrow Z \tag{3.4}$$

where X and Y are the sets of source and target entities respectively. θ is the communicative functions internal properties which may be captured by S_t but they do not have to be. $Z \subseteq E_{t+i}$ is the new state of the target entities Y such that updating the state of the entities in Y is written as (Equation 3.5):

$$Y_{t+i} = c(X_t, Y_t, \theta) \tag{3.5}$$

With this definition, state updates of target entities may be instantaneous ($i = 0$) or they may be temporal ($i > 0$)[10]. Additionally, we apply the restriction that communicative functions cannot map to a constant z_c. This is because a communicative function that does not consider the state of the source entities when determining the new state of the target entities does not facilitate adaptation.

Using the above definitions, we can determine if an agent a is adaptive at timestep t if, and only if, it can receive information from another entity. This entity cannot be itself.

[10]If $i = 0$, the definition of S_t changes slightly to mean the set of all possible states at timestep t where $S_t \subseteq S$. A new value $s \in S_t$ must then be defined to replace the original definition of S_t

We can write that formally as a boolean function (Equation 3.6):

$$\alpha_t(a, E'_t) = \left\{ \begin{array}{ll} 1, & \textit{if } \deg^-(a, E'_t) > 0 \\ 0, & \textit{else} \end{array} \right\} \textit{ where } E'_t = \{e \,|\, e \in E_t \text{ and } e \neq a\} \tag{3.6}$$

Where α is the boolean function that returns true if our agent is adaptive and false if not. $\deg^-(e, E)$ calculates the indegree of an entity with respect to the set of entities E and E'_t is the set of all entities in the model M at timestep t other than agent a.

We may also think of α in terms of the cardinality of the multiset produced by the addition[11] of sets Q where Q is the sets of entities produced by L_t for ordered pair (p, y) for every entry of the Cartesian product of the power set $\mathcal{P}(E_t)$ and set of communicative functions C_t (Equations 3.7 and 3.8):

$$Q(a, E_t, C_t) = \sum_{(p,y) \in W} L_t(p, y) \quad \text{where } W = (\mathcal{P}(E_t) - \{a\}) \times C_t \tag{3.7}$$

$$\alpha_t(a, E_t, C_t) = \left\{ \begin{array}{ll} 1, & \textit{if } |\{q \,|\, q \in Q(a, E_t, C_t) \textit{ and } q = a\}| > 0 \\ 0, & \textit{else} \end{array} \right\} \tag{3.8}$$

Lastly, an entity e_0 is capable of sending information (directly or indirectly) to another entity e_n if there exists a sequence of entities T_j that starts at e_0 and ends at e_n (Equation 3.9):

$$T_j(e_0, e_n) =< e_0,\ e_1,\ ...,\ e_n > \text{ where } \exists\, e_{i+1} \in L(x, c) \,|\, i \in [0, n),\ x \subseteq E_t,\ e_i \in x,\ c \in C_t \tag{3.9}$$

Given that there may be several sequences that start at e_0 and end at e_n, we denote $T(e_0, e_n)$ as a function that returns the set of all these sequences. Determining if one entity can send information to another entity is then defined as Equation 3.10.

[11] Specifically multiset addition such that given $A = \{1, 2, 3\}$ and $B = \{3, 4, 5\}$, $A+B = \{1, 2, 3, 3, 4, 5\}$. This would be the same as writing $\sum_{x \in C} x$ where $C = \{A, B\}$

$$\sigma(e_0, e_n) = \left\{ \begin{array}{ll} 1, & if\ |T(e_0, e_n)| > 0 \\ 0, & else \end{array} \right\} \tag{3.10}$$

where σ returns true if entity e_0 can send information to entity e_n and false if not. If $\sigma(e_0, e_n) \wedge \sigma(e_n, e_0) = 1$, we say that e_0 and e_n are capable of exchanging information (directly or indirectly) with each other.

Determining if information exchanges from e_0 to e_n are direct ($\sigma^+(e_0, e_n) = 1$) or indirect ($\sigma^-(e_0, e_n) = 1$) can be determined by Equations 3.11 and 3.12:

$$\sigma^+(e_0, e_n) = \left\{ \begin{array}{ll} 1, & if\ \exists\, T_j(e_0, e_n) \in T(e_0, e_n) \mid |T_j(e_0, e_n)| = 2 \\ 0, & else \end{array} \right\} \tag{3.11}$$

$$\sigma^-(e_0, e_n) = \left\{ \begin{array}{ll} 1, & if\ \exists\, T_j(e_0, e_n) \in T(e_0, e_n) \mid |T_j(e_0, e_n)| > 2 \\ 0, & else \end{array} \right\} \tag{3.12}$$

For direct exchange, we look for a sequence of entities of exactly length 2 ($T_j(e_0, e_n) = \{e_0, e_n\}$ specifically). For indirect exchange we look for a sequence of entities of length >2 signalling that any information that entity e_n receives from e_0, must've passed through one or more entities $\notin \{e_0, e_n\}$.

3.2.5 Case Study: Stigmergic Adaptation of Foraging Ants

Now that we've formally described how to create adaptive-agents using information exchange, we will demonstrate it *in silico* using an ABM simulating the artificial life (ALIFE) of foraging ants[12] [13]. The motivation for choosing to simulate Ants mostly pertains to their high degree of adaptive capacity and the unique method by which they exchange information to facilitate said adaptation.

Ants communicate indirectly via a process called stigmergy. They lay down pheromones as messages which other ants use to facilitate their decision making [131]. In its simplest form, an ant may lay down one type of pheromone as it leaves the nest in search for food

[12]This model was implemented in ECAgent and is available at the following link: `https://github.com/BrandonGower-Winter/ECAgentTutorials/tree/master/ForagingAntSimulator`

[13]The design for this model was partially inspired by the following YouTube video: `https://www.youtube.com/watch?v=X-iSQQgOd1A&t=458s&ab_channel=SebastianLague`

(A) Phase 1. (B) Phase 2.

FIGURE 3.9: The different ant hill topographies used in the model.

and another once it has found food and returns to the nest. Other ants will pick up on this "food" pheromone and follow it to collect their own food to take back to the nest using the "home" pheromone. This simple process is so effective at allowing ants to adapt to both unknown and dynamic environments that it inspired a meta-heuristic optimization algorithm called Ant Colony Optimization (ACO) which has found success in solving a variety of static and dynamic optimization problems [132]. Furthermore, stigmergy has been used to create adaptive-agents in the past [133] and in ABM simulating the dynamics of evacuation procedures in disaster management scenarios [134] (where it is called signalling).

In our model, ants will be initialized at position (24,24) of 50x50 grid-world environment that looks like an ant nest. There will be resources dotted around the environment and the task of the ants will be to collect and return as many of those resources as possible within 1000 timesteps. The environment has two phases which it will cycle between every env_switch timesteps (See Figure 3.9). If our ants are adaptive, they should still be able to collect resources despite the change in the environment's topography.

To create adaptive-agents, we must meet the three criteria stipulated in Section 3.2.2: Our entities must be able to maintain state and be capable of exchanging said state with one or more entities. Additionally, our agents must be able to utilize their state when making decisions. In our model, there are two types of entities: Ants and environmental

cells. The Ants are also capable of making decisions so they are also agents. The state maintained by each environmental cell is categorized as follows:

1. *position*: A 2-tuple describing the coordinates of the cell (e.g. (10, 10)).
2. *resources*: The number of resources available on the cell.
3. *F pheromone*: The intensity of the food pheromone on the cell.
4. *H pheromone*: The intensity of the home pheromone on the cell.

The state maintained by the Ants are:

1. *position*: A 2-tuple describing which cell the ant is currently on.
2. *direction*: A 2-tuple describing which direction the ant is facing.
3. *holding resource*: A boolean describing if the ant is holding a food resource.
4. *perceived pheromones*: A vector containing the pheromone values the ant perceives. These values are determined by the direction the ant is facing.

Every iteration, each ant must make a decision about which neighbouring cell to travel to. This partly stochastic process is is based on the state of the agent and the direction it is facing. Similarly to the approach used by Jones [135], each ant will query up to three grid cells in the direction it is facing and choose the cell which has the highest pheromone concentration. The type of pheromone the ant perceives depends on the state on the ant. If the ant is carrying resources, it will use the H pheromone to determine which cell to move to and, conversely, the ant will perceive the F pheromone if it is not currently holding any resources. Lastly, a wander factor $\in [0, 1.0]$ can be defined to determine the probability that the ant will take a random action instead (See Figure 3.10 to see what an ant's perception cone looks like).

The last thing we need to do is create an information exchange network, describe its connectivity and define the set of communicative functions that govern how information travels from entity to another. At a given timestep t, the entities (vertices) in our model are the 2500 environment cells and the n (user defined) ants. Ants send information to an environment cell (at the same location as the ant) by depositing pheromones.

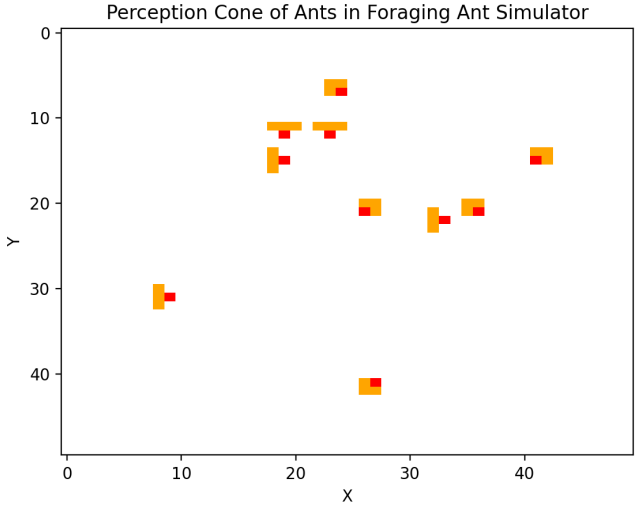


FIGURE 3.10: A Figure demonstrating the perception cone of the Ant agents. The red pixels are ant agents and the orange pixels are the cells they query to determine which path to follow.

Environment cells send information to the Ants via the ant's perception function and, if a user chooses, an environment cell may send information to itself decaying the amount of pheromones on a given cell by the user defined *decay_rate* parameter. Figure 3.11 demonstrates the connectivity of the model's information exchange network[14] and we list the set of communicative functions as follows:

1. *deposit*: A function that deposits pheromones on environment cells at the location of Ant agents in accordance with the *deposit_rate* parameter. The type of pheromone deposited depends on the state of the Ant with an ant depositing H pheromones when looking for resources and F pheromones when carrying resources.

2. *perceive*: A function that sends the environmental cell pheromone intensities to an Ant agent. The selected cells depend on the Ant's perception cone. The type of pheromone perceived depends on the state of the Ant with an ant perceiving F pheromone when looking for resources and H pheromone when carrying resources.

3. *decay*: A function that, every iteration, decays the amount of pheromone (both types) at a given environmental cell in accordance with the *decay_rate* parameter.

[14]The Ant icon in Figure 3.11 was provided by Freepik using the standard Flaticon license: `https://www.flaticon.com/free-icons/ant`

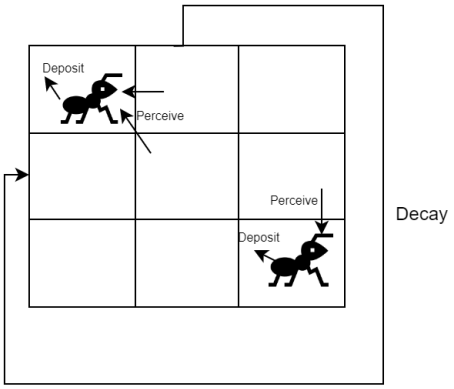


FIGURE 3.11: The information exchange network of the Foraging Ant Model. Ants deposit pheromones onto grid cells they visit. They perceive the pheromones on grid cells in the direction they're facing and, if enabled, the environment will decay the amount of pheromones on each of its grid cells in accordance with the *decay_factor*.

Using the formal description provided in Section 3.2.4 and Figure 3.11 as reference, we can confirm that our Ants agents are adaptive ($\alpha(a, E_t, C_t) = 1$) because they are capable of receiving information from environment cells. Additionally, our ants are directly connected to an environment cell $\sigma^+(a, e_{cell}) = 1$ (and vice versa) if they share the same position. Ants are also connected indirectly $\sigma^-(a_1, a_2) = 1$ as they are forced to use an environment cell if they wish to exchange information. Lastly, an alternative view of model's information exchange network similar to Figure 3.8b can be created where the indirect exchange between the ants can be more clearly observed.

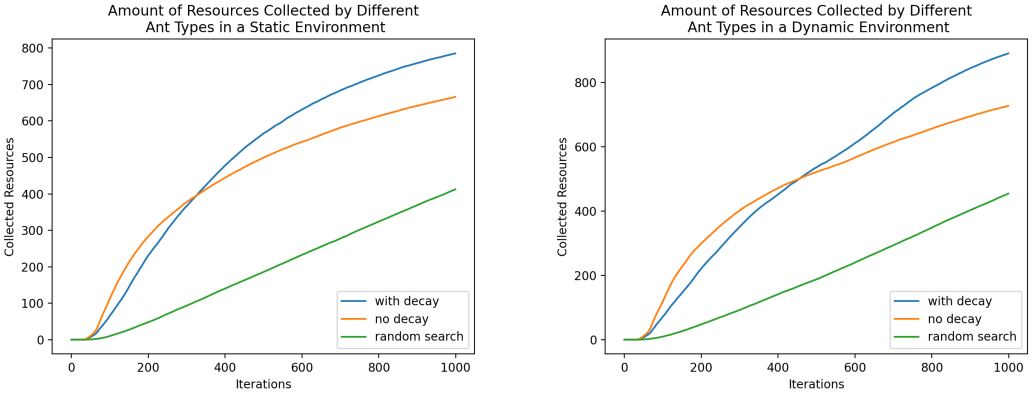
3.2.5.1 Validation

To validate the model, we performed scenario-based experimentation whereby we compared two-types of information exchanging ant-types to purely stochastic ants. The first two ant-types are called 'with decay' and 'no decay' because of the inclusion and exclusion of the decay communicative function respectively. The purely stochastic ant-type is called 'random search' because that is essentially the behaviour of the ants when they are not allowed to deposit or perceive pheromones.

The first scenario instantiates ants (using one of the network-types described above) into a static environment (Figure 3.9a specifically) and measured the amount of resources collected by each agent-type. This is repeated in scenario two except at timestep 500, the environment will change to Figure 3.9b. For each scenario, 50 simulations were run for each ant-type for a total of 300 simulation runs. A pseudorandom number generator

Parameter	Random Search	No Decay	With Decay
Iterations (t)		1000	
Number of Ants (n)		300	
Random Action	100%	5%	5%
Pheromone Deposit	0.0	0.25	0.25
Pheromone Decay	0.0	1.0	0.9

TABLE 3.2: Parameters used in Foraging Ant Simulations.



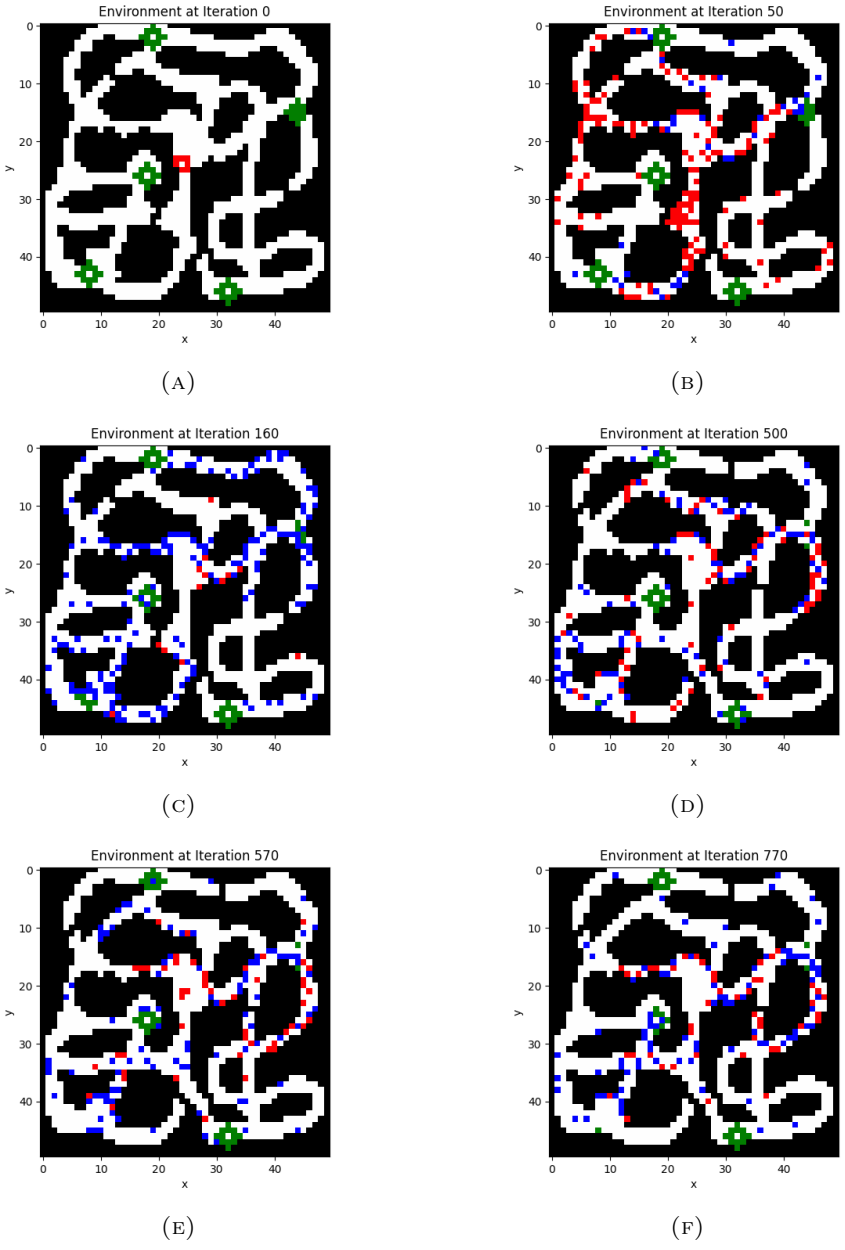
(A) (B)

FIGURE 3.12: Number of collected resources for each Ant-type across both the static (a) and dynamic (b) scenarios.

was used to ensure reproducibility. Table 3.2 includes a list of initialization parameters used by the model. The motivations for choosing these parameters values were mostly arbitrary or fixed by design. For example, the random search ants must, by design, always take random actions. The exception to this is the Pheromone Decay property for the 'with decay' ants where a one-factor-at-a-time (OFAT) sensitivity analysis of the parameter motivated us to choose a value of 0.9.

Looking at Figures 3.12, we see that the 'random search' ants collected the fewest resources while the 'with decay' ants collected the most. Interestingly, the 'with decay' agents were able to collect even more resources in the dynamic environment (scenario 2). This is attributed the more complex information exchange network that allowed pheromones along the blocked tunnels to dissipate allowing the ants to find a new path back to the nest. These aforementioned results are expected as the inclusion of pheromone decay facilitates increased information exchange thus increasing the opportunity for the agents to exhibit adaptive behaviour. Figures 3.13 demonstrate the typical run of the 'with decay' ants[15] including a looping emergent phenomena whereby ants get

[15]A video of this run can be viewed at the following url: `https://youtu.be/9GRKxAp2ghg`

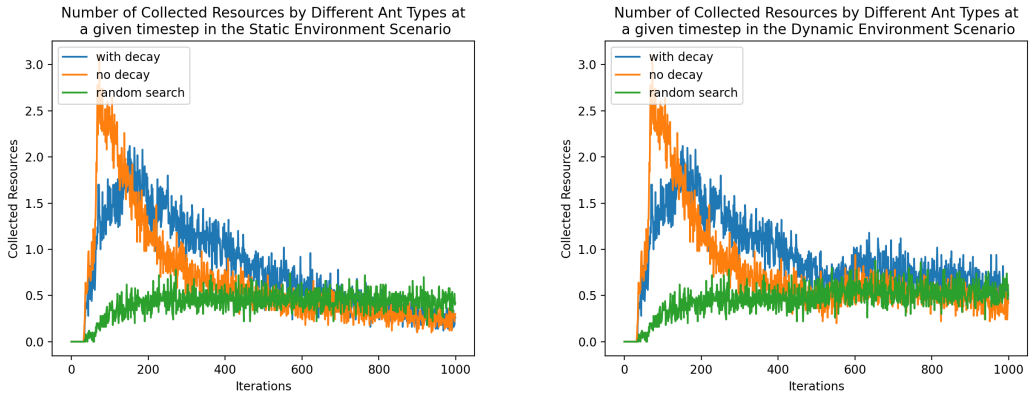


(A) (B) (C) (D) (E) (F)

FIGURE 3.13: Figures showcasing the 'with decay' Ants at various stages in a typical model run. Red pixels are ants looking for resources, blue pixels are ants carrying resources and green pixels are cells that contain resources.

stuck following each others paths in a circular manner. This is a documented behaviour in real-world ants called an ant milling.

Figures 3.14 plot the rate of resource collection for each ant type. This can be used to study the resistance and recovery of the ant-types. Both the adaptive ant types are disturbed by the environment phase change equally (from approximately timestep 500-650) but, the 'with decay' agent is able to recover past its pre-disturbance rate of resource collection to a greater degree than the 'no decay' ants. To demonstrate why



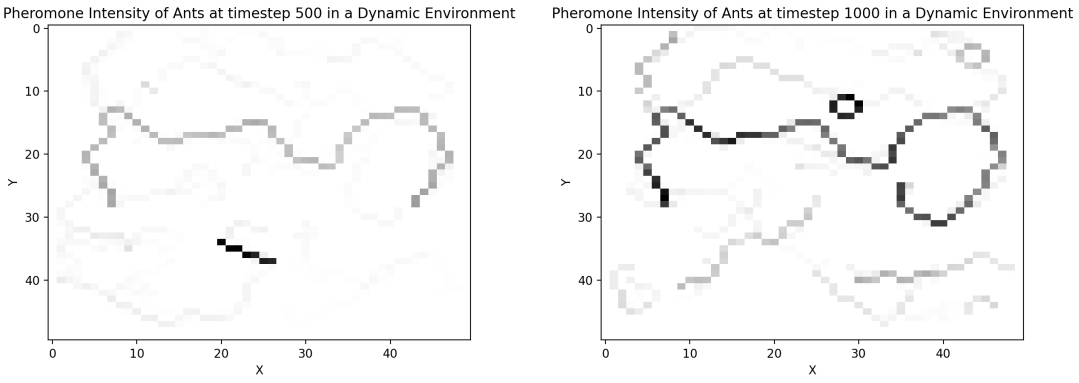
(A) (B)

FIGURE 3.14: Rate of resource collection for each Ant-type across both the static (a) and dynamic (b) scenarios.

this occurs, we plot the persistence of the pheromone intensity of the 'with decay' ants at timestep 500 (before the phase switch) and at timestep 1000 (Figure 3.15). We can see that by timestep 1000, the 'with decay' ants reinforce the primary path that spans the width of the environment and make new paths that were not possible in the phase 1 environment. The emergence of these new paths follows the adaptive-cycle discussed in Section 3.2.2. An $a - K$ phase can be observed from iterations 1-499 whereby the ants freely and effectively collect resources and then get trapped into using the pheromone paths demonstrated in Figure 3.13c. Once the disturbance event occurs in the form of an environment change (Figure 3.13d), an $\Omega - \alpha$ phase occurs whereby some of the paths the ants used to collect resources become blocked and cause the ants to scatter (Figure 3.13e). This phase is brief (typically between 50-150 timesteps) and leads into a new $a - K$ phase where the ants find a new path back to the nest (Figure 3.13f showcases this whereby a group of ants have now started collecting resources from the cavern to the left of their nest. Some ants have also started collecting resources from the bottom right resource patch).

3.2.6 Discussion

In this section we sought to fill a gap in the state of the art by using information exchange networks to facilitate the creation of adaptive-agents. In resilience research, adaptability refers to the ability of an entity or system to increase or decrease its resilience. In Agent-based Modelling, adaptability refers to an agent's ability to not only change its decisions but also the strategy it uses to make decisions. Adopting the ABM definition



(A) (B)

FIGURE 3.15: The pheromone intensity of the 'with decay' ant type at timestep 500 (a) and timestep 1000 (b).

of adaptability, we examined how adaptability could be measured by monitoring the resistance, recovery, persistence and variability of one or more agent or model properties.

We then proposed an approach for defining and implementing adaptive agents using an information exchange network. With entities defined as state containers, and agents as decision making automata, we formally defined the process of creating a directed multigraph governed by communicative functions. Using this approach, determining if agents are adaptive or connected (directly or indirectly) were formulated as graph traversal problems with adaptive-agents being defined as agents capable of receiving information (an indegree > 0) from entities other than themselves.

We then demonstrated the applicability of this approach by implementing a simple foraging ant simulator. We empirically demonstrated that agents capable of information exchange are capable of exhibiting greater adaptive capacity in both static and dynamic environments.

One point of contention that we foresee regards the simplicity of the ant foraging model. It could easily be argued that the results observed were both predictable and uninformative. It is glaringly obvious that agents that exhibit even slightly more advanced decision making processes would perform better than purely stochastic ones. That is, for the most part, completely true but it is not the point we are trying to demonstrate. For one, the ants' decision making process is the same for all agent-types. The only aspect that changes is their capacity to exchange information. Secondly, these experiments are intended to be an analogical representation of the current state of ABM research. We

have argued that as researchers move from simple toy-box models to complex data-driven models, the need for adaptive-agents will become more apparent. In our analogy, the static environment represents the toy-box models and the dynamic environment represents these newer complex models. Even in the triviality of the static environment scenario, the benefits of adaptive-agents are clear. The 'no decay' and 'with decay' ants are able to collect more resources because they exchange information indirectly through stigmergy. The benefit of adaptive-agents in the dynamic environment is even more apparent with the 'with decay' ants able to adapt to their environment more effectively than the 'no decay' and 'random search' ants due to the increased complexity in which information is exchanged. If the benefits of using adaptive-agents is clear in even the simplest of circumstances, then surely that must also be the case as model complexity increases.

One caveat to the aforementioned discussion is the diminishing returns of adaptive complexity in increasingly simple environments. What we mean by this is that perpetually adding adaptive capacity to the agents via new methods of information exchange is likely limited by the complexity of the model. In the ants model, this could be observed in the static (simpler) environment where the difference in the number of collected resources between the 'with decay' and 'no decay' ants was not as significant as it was in the dynamic (more complex) environment. This means that adding an unnecessary amount of adaptive mechanisms to an agent is not beneficial and, in some cases, may even be detrimental to the overall model as these mechanisms not only take time to implement, but may make synthesizing the results produced by the model more challenging.

One aspect of our information exchange modelling approach that we could not demonstrate with the ants foraging model is the plethora of graph-based heuristics that can be used to understand agent behaviour. As described by Anderson and Dragićević [136], some of these graph heuristics include:

1. **Degree:** The number of connections a vertex has to other vertices. In a directed graph, this can be further divided into *indegree* and *outdegree* metrics which describe the number of incoming and outgoing connections respectively.

2. **Average node degree:** The number of connections a vertex has averaged across all vertices in the network. This can also be divided into measure of the *average indegree* and *average outdegree*.

3. **Degree distribution ($P(c)$):** The fraction of vertices in the network which have a degree $= c$. This can also be specified per agent or entity type (A^K or E^k) if looking over the entire network is inappropriate.

4. **Clustering coefficient:** Measure of how likely vertices connected to vertex v are also connected to each other.

5. **Average clustering coefficient:** The average clustering coefficient of all the vertices in the network. Again, this can also be specified per agent or entity type (A^K or E^k) if looking over the entire network is inappropriate.

6. **Path length:** The shortest sequence of nodes that connect two vertices in the network together.

7. **Average path length:** The average path length between all pairs of vertices in the entire (or a subset) network.

Additionally, our approach opens up the field of information theory as another potential avenue by which adaptive-behaviour can be quantified and understood. Measuring the magnitude and change in entropy of entities as they change state has potential but, was deemed out of scope. Future work should certainly investigate the matter further.

Lastly, we believe it is worth discussing how this entire section relates to the rest of the thesis. Given our overall goal of evaluating ML techniques as adaptive-mechanisms, we identified that an approach for implementing and integrating the algorithms into a model was needed. Information exchange networks were chosen because a wide variety of the algorithms identified in Section 2.2.1 are primarily concerned with the exchange of information. In EAs, genetic information is transferred from parent entities to children entities. In RL, agents observe reward and value (which are both types of information) when taking actions. Information exchange networks are capable of facilitating the design of adaptive-agents using a variety of techniques but, they are even more appropriate when using ML techniques. Similarly, terms such as entities and systems are taken directly from SES literature but, they are also conveniently the terms used in ECS further demonstrating the natural compatibility the design pattern has with ABM development.

Naqada Chronology [137]	Hassan Chronology [138]	Absolute estimate cal. (BC)
Badarian	Late Neolithic/Early Predynastic	4400-3800
Naqada IA–IIB	Middle Predynastic	3800/3750–3450
Naqada IIC–D	Late Predynastic	3450–3325
Naqada IIIA–IIIB	Terminal Predynastic	3325–3085
Naqada IIIC–D	First Dynasty	3085–2867

TABLE 3.3: A summary of the absolute chronology of the Egyptian Predynastic with alternative chronological terms. (Based on Stevenson [8]).

3.3 The Curios Case of Predynastic Egypt

In this section, we aim to synthesize several papers about the complex social processes that underpinned the emergence of the Ancient Egyptian State during the Predynastic period. There is strong iconography associated with Ancient Egypt, however, we know very little about how it came to be. This presents an opportunity in which a sufficiently complex ABM may be able to validate, invalidate or assist in constructing theories about the formation of the Egyptian state. Section 3.3.1 contains a brief history of the Egyptian state as highlighted by Brewer and Teeter [11]. Section 3.3.2 contains five theories that pertain to the underlying causes and systems that underpinned the emergence of Predynastic Egypt and Section 3.3.3 coalesces these theories. Table 3.3 has been included to provide context to the timeline we are reviewing.

3.3.1 Background

In this Section we outline the chronology of the formation of the Ancient Egyptian state during the Predynastic period as described by Brewer and Teeter [11]. Sections 3.3.1.1, 3.3.1.2 and 3.3.1.3 cover the late Neolithic, the Predynastic and Early Dynastic periods respectively.

3.3.1.1 Neolithic Period 6000 - 4600 BC

The Neolithic Period (New Stone Age) marked a shift from the hunter-gatherer lifestyle to one of cultivation and animal husbandry. Agriculture, which broadly describes these two processes, formed the economic base that gave rise to all early civilizations.

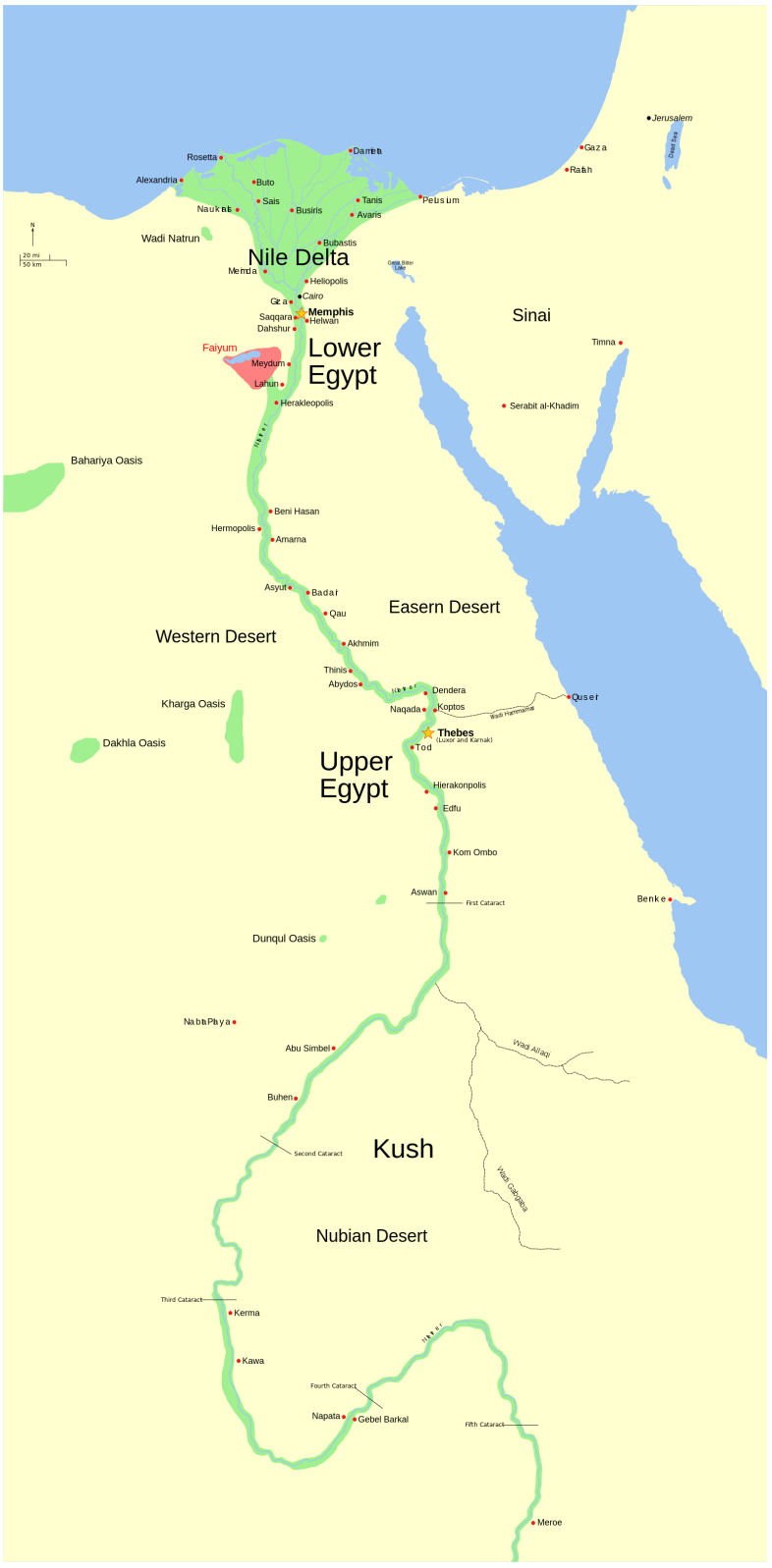


FIGURE 3.16: A map of Egypt courtesy of Wikimedia Commons user H.Seldon. Licensed CC BY-SA 3.0, see: `https://commons.wikimedia.org/wiki/File:Faiyum_oasis.svg`.

In Egypt, indisputable evidence of domesticated plants and animals have been found. Excavations provided evidence of two distinct cultural adaptations and lifestyles (Nile Valley, Desert). In the Nile Valley, bones of Goat, Pig, Sheep and Cattle have been recovered from Merimde. Domestic animals and grain silos, containing wheat and two types of barley, were found in Fayum. Evidence of animal domestication was also found in the form of cave paintings in the western desert and, more compellingly, from excavations in the Nabta region. The bone fragments and plant matter uncovered in the western desert were identical to that of the wild variants found in Egypt. This leads some to believe that this is evidence of a more opportunistic and mobile dessert lifestyle. It is thought, in general, that late Palaeolithic and Neolithic groups maintained a hunter-gather lifestyle supplemented by these new agricultural processes. A sedentary lifestyle would not have been possible as the cattle and crops would not have survived the ever increasing arid conditions.

Life along the Nile would not have changed as fish and fowl were both plentiful and predictable. However, as the areas to the east and west of the Nile became more arid. There was an increased movement to agriculture and attempts to get as much from the land as possible. This was not without its disadvantages. Fewer food sources meant that the likelihood of a famine increased. With the establishment of agriculture, villages started appearing and growing in size. Local leaders emerged and Egypt moved into the Predynastic period.

3.3.1.2 Predynastic Period 4650 - 3150 BC

The Predynastic period marks the cultural transformation of the Egyptian people from Hunter Gatherers to Agricultural villages. While the timeline of this period is still unclear, there is a clear understanding that both the North and the South had undergone cultural evolution at different stages. The different stages of Predynastic Egypt have no specific start and end date. They rather serve as a guideline as to what might have occurred. The different stages (specifically in the North) may not even represent the same cultures, but rather a series of localized cultures.

During the Fayumn and Meridem stages, there is evidence of a mixed hunter-gatherer/ farmer lifestyle which later (Towards the Naqada I/ Omari A period) developed into a lifestyle with a greater reliance on cultivated plants and animals. Upper Egypt also seems

to have developed a distinctive style of pottery while the Delta remained, largely, localized with no distinctive style of pottery dominating. By Naqada II there is evidence of a increasingly complex and socially stratified society emerging. Evidence of the emergence of a social elite, fortifications and conflict can be found in artistic depictions.

By Naqada III, Egyptian centers of political powers had emerged in both Upper and Lower Egypt (Hierakonpolis, Naqada, Maadi and Buto). It was during this time that a definable national identity was formed with Upper Egyptian characteristics finding their way into the Eastern Delta. By the end of the Predynastic period, most of the country exhibited the same cultural characteristics, attesting to the gradual dominance of the North by the South. During this period, there is evidence of local-chieftains differentiating themselves from their subjects.

3.3.1.3 Early Dynastic Period 3050 - 2686 BC

The actual mechanism that resulted in the unification of the North and South is unknown. Traditional views suggest a conqueror king forcibly united the North and South. With little evidence backing this theory, a different picture is emerging whereby a number of larger Upper-Egyptian states, that extended their power northwards, assimilated with Northern tribes, forcefully or through alliances. Little is known about the influence that outsiders had on this unification process. Once a king had been established, the kingdom was separated into states overseen by a governor. Their was a culture of meritocracy and birthright such that the humble could equal the powerful through skill and eloquence.

3.3.2 Theories

In this Section we review several theories related to the formation of Ancient Egypt during the Predynastic period. Section 3.3.2.1 covers Anđelković's [12] theories of political organization through the lens of social and natural factors. Section 3.3.2.2 discusses Allen's [36] theory of state formation as a result of surplus economies. Section 3.3.2.3 highlights Kemp's [139] state formation through the lens of a self-engendering complexity generation as a result of human cognition. Section 3.3.2.4 reviews Flannery's [140] work on state formation through the lens of opportunistic individuals. Lastly, Section 3.3.2.5 summarizes Stevenson's [8] phasic descriptions of state formation in Ancient Egypt.

3.3.2.1 Political Organization of Egypt in the Predynastic Period

Anđelković's [12] proposes a theory into the emergence of a unified state in Egypt over many generations and highlights the numerous social and environmental factors that played a role in such a phenomena. They are detailed as follows:

The Passive Natural factors: The Nile River is one of driving natural factors in the emergence of the Egyptian state. The immediate areas along the river receive a yearly inundation of fertilization and solar radiation. The extremely arid regions to the east and west of the river also create a 'tube effect' whereby the areas along the Nile river are comparably more suitable for living than the arid regions further away from the river. This also increases social compacting forcing communities to interact with each other.

The river provided natural fertilization and flowed constantly which prevented salinization and the development of a swamp basin. This provided ideal conditions for agriculture to flourish as well as allow for an extremely diverse ecosystem to thrive. The river also provided access to water, wind and sunlight. The sunlight was ideal for farming while the water and wind provided ideal circumstances for water-based transportation. There are also gold bearing regions in Upper Egypt which were mined during the Predynastic period. It is suggested that the "flesh of the gods" was an important strategic currency used by the power centers.

Active Social factors The social setting was dominated by the idea of sacred power blended with the concentration of military, economic and political power. By examining the archaeological record, there is a suggestion that there was a strong belief in the subjugation of enemies. It is likely the social elite were the centralizing force of these systems, using there power to unify the religious, military, political and economic systems around one ruler.

Anđelković's proposed timeline of political organization can be summarized as follows:

1. **Pre-nomes (Pre-Naqada I)**: These refer to independent local villages defined by political autonomy.

2. **Proto-nomes (Naqada 1A - 1B)**: Proto-nomes refer to a number of villages unified under one chiefdom. This results in an increase in economic, military and

political power of a number of Proto-nomes. By the 'tube effect' and natural selection, the close proximity of villages to each other and the clearly superior military, economic, and political power of Proto-nomes would have forced autonomous villages to unify into Proto-nomes in order to compete with their neighbours.

3. **Nome Pre-states (Naqada 1C - 2B)**: This refers to the further unification of a number of Proto-nomes. They can be identified by the emergence of a social elite and the comparatively lower number of Nome Pre-states compared to the number of Proto-nomes. These social structures indicate the emergence of state to come whereby a number of the Nome Pre-states will be further unified through conquest or absorption by more-predatory neighbours.

4. **The Upper Egyptian Proto-state (Naqada 2C - 2D1)**: This, sometimes called the "Upper Egyptian Commonwealth", refers to the unification of the Nome Pre-states into one identifiable polity. Expansion towards the North would most likely have started with Nome Proto-states absorbing Northern Proto-nomes. This period is not thought to have lasted long as the overwhelming power of the South over the North could only result in its unification (through absorption, warfare and cultural domination). It should be noted that the South is expected to contain a few Nome Pre-states which would be more resistant to unification.

5. **All Egyptian early state (Naqada 2D2 - 3C1)**: This stage refers to a unified North and South named Dynasty 0. Possible revolts from local elites are possible at this time.

6. **Egyptian Empire (Naqada 3C1 - 3C2)**: At this stage their should be evidence of a stable unified polity.

Anđelković's theory describes a power struggle where rulers jostle for domination over the political, economic, religious and military systems. The victor of this power struggle would have come from Upper Egypt. This is, at least in some part, due to the smaller livable region in the South which would have forced villages to adapt, by adopting agrarian practices, earlier than the North which, as a result, would have given the South a technological advantage.

3.3.2.2 Agriculture and the Origins of the State in Ancient Egypt

In his paper, Allen [36] describes the rapid adoption of farming in Predynastic Egypt and the unique conditions of the Egyptian landscape which allowed for the relatively fast emergence of a unified kingdom.

The paper described crowding of the population around the Nile river. This was largely due to arid landscapes that had become more prominent forcing the population closer to the Nile. This limited the chance of flight (moving) amongst the population in times of hardship as individuals would have to migrate along the river where numerous other villages would be present. Due to Egypt's relatively low population density, two surpluses arose. An economic surplus in the form of grain, cattle, and other resources and a labour surplus which allowed for the construction of the complex structures such as temples. This was because the communities could be fed by a small amount of farmers while the others worked on construction endeavors. Allen notes that food produced by farming could be stored while the food produced by foragers could not. This meant taxation could only be applied to farming villages. Additionally, farming was seasonal and foraging was a year long ordeal. This means that farmers could participate in other activities such as construction while foragers could not.

While the farming provided an advantage over foraging, it was not the sole reason for the emergence of a unified state. Allen notes that Egypt, due to its low population density, was an economic frontier. This meant that individuals had ample land to farm for themselves. This decreases the likelihood of social and financial inequality. Allen argues that, due to the sheer volume of land available, individuals could very easily migrate to other tracts of open land along the Nile should they be unhappy with the taxation or working conditions of their current situation. This limited the ability of the social elite to extract a surplus from their workers as unfavourable working conditions would result in the loss of workers.

There were two different dominant cultures in the North and South (Maadi and Naqada). The South or Upper Egypt was significantly more overpopulated than that of North or Lower Egypt. The Maadi culture was also more egalitarian than the Naqada culture where tomb analysis has shown clear evidence of social inequality. This overpopulation in Upper Egypt is what Allen argues is the driving factor in the domination of the Maadi culture by the Naqada culture. Overpopulation meant the emergence of inequality. The

emergence of inequality meant the propagation of the Naqada culture northwards. A culture built upon agriculture was more beneficial than that of one built upon foraging. The Naqada culture slowly dominated the Maadi culture as a result.

This theory might accurately describe the propagation of the Naqada culture over Egypt but it does not describe the emergence of a social elite. As previously mentioned. The availability of land meant that a social elite could not form due to the mobility (ability to move to another location) of the region. Allen argues that the social elite would have been aware of this and would have sought to restrict the movement of individuals and thus decreasing regional mobility. Due to the surplus of labour, opportunistic individuals could mobilize the surplus labour into a military force which could then be used to limit the mobility of individuals. The geography of Egypt would have made this all the easier as the arid regions surrounding the Nile were less habitable. This meant that it was significantly easier to limit population mobility as families could not live far from the Nile. The use of a military force meant that land-owners could force taxation becoming increasingly wealthy. This may have also introduced a dynamic of land-owners offering lower tax rates to encourage individuals to work for them.

Allen's proposed series of events are as follows:

1. The desertification of the areas around the Nile force families and villages to move towards agriculture practices as a more reliable method of survival.
2. The emergence of agriculture creates a surplus of labour and food
3. Landowners start taxing individuals who farm on their land accruing more wealth and increasing their social status.
4. Unfavourable working/living conditions cause people in the more densely populated Upper Egypt to move northwards where the land is more plentiful
5. At this time the landowners start competing with each other by varying tax rates and incentives.
6. Landowners mobilize a military force to restrict the movement of individuals. This is aided by the narrow window of livable land along the Nile.
7. At this time, those who have moved northwards bring along their agrarian culture.

8. The new, farming reliant, families and villages in the North become significantly more successful than the people, originally from the North, who forage. This advantage causes the adoption of the Southern culture as a more beneficial way of living.

9. During this time, it is presumed that a number of opportunistic individuals successfully entice a large number of people to work under them. These individuals started accruing an enormous amount of wealth and labour thus allowing them to extend their rule over an ever expanding area gaining additional labour while doing it.

10. The larger 'states' then fight over complete control through enticing/forcing individuals from other states to work for them.

11. This eventually leads to a polity so large and powerful that it conquers (not necessarily by military force) the entire region resulting in a unified Egypt.

3.3.2.3 Ancient Egypt: Anatomy of a Civilization

In his paper [139], Kemp describes a model for the formation of Early Egypt. Kemp calls to question the processes and events that sought to disrupt their hunter-gatherer lifestyle which, as a result, gave rise to the development of dominant leadership. Kemp attributes this phenomena, observed throughout history, to the nature of human cognition and its ability to generate complexity. Kemp describes how a set of ideas or behaviours that form a baseline, subdivide indefinitely. This process is both self-reinforcing and facilitates its own acceleration.

However, Kemp notes that this process propagates throughout societies unevenly. In the context of Predynastic Egypt, the northern Nile communities followed a similar trajectory to those in the Mediterranean, however, this process emerged significantly later on. The uneven emergence of this process is also evident on a smaller scale with Upper Egypt's transition to an agrarian society earlier than Lower Egypt.

Kemp likens this process to a game of 'Monopoly' whereby a number of individuals start with roughly equal potential in an environment with virtually unlimited potential. The game proceeds by chance whereby a combination of environmental and locational factors as well as individual decision making favours of one or more individuals. The process

starts slow. Much like an egalitarian society where the advantage oscillates between a number of individuals. At some point an individual gains such an advantage that it becomes self-reinforcing. This self-reinforcing process leads to the emergence of an individual who is unstoppable and the only outcome is total monopolization. By moving closer to reality, individuals represent a number of generations and the time-scale widens. Kemp applies this process to Predynastic Egypt by illustrating the three stages of state emergence:

1. **Egalitarian Communities:** These communities were small in size and relied primarily on hunting, fishing and foraging. Although, Some communities may have limited access to crops or cattle.

2. **Agricultural Communities:** These communities were the result of the merger of a number of egalitarian communities. They relied heavily on agriculture for food and may have even traded with other communities.

3. **Proto-States:** These communities were dense, fortified, had highly sophisticated irrigation systems, traded with other communities and had distinct territorial bounds. The emergence of a social elite meant that a number of proto-states may have been under the rule of a single individual and thus taxes were likely to have existed.

3.3.2.4 Process and agency in early state formation

Flannery's [140] paper is not specifically about Egypt but rather early state formation in general. The essay is mainly concerned with demonstrating that process and agency are complementary in the formation of 'Pristine States'. Flannery defines 'Pristine States' as huge, politically centralized and socially stratified.

Process: Flannery proposes the following process for state formation (based on Carneiro's [141] theory of chiefdom creation):

1. Defeat neighbouring villages by force

2. Incorporate them into your political unit

3. Take prisoners and work them as slaves

4. Use close supporters to administer conquered territory
5. Require subjects to pay tribute
6. Require them to supply a military force in times of war

Flannery argues that the above process, combined with chiefly cycling, is capable of giving rise to states. Chiefly cycling is the recurrent process of emergence, expansion and fragmentation of chiefdoms. Flannery also argues that states could form in the presence of competition from other chiefdoms. Wright [142] proposes the following process for state formation.

1. As a result of long-term conflict, areas on the verge of statehood may have dispersed populations that form buffer zones between chiefdoms.
2. Suddenly a rapid coalescing of the population occurs at one of the political centers.
3. Since a number of multi-layered political hierarchies are being merged, a fourth hierarchy emerges as an administrator over all the other layers.

Upon arriving at a stagnant growth curve. Chiefs will have one of three decisions to make. Either increase taxes, intensify production or territorial expansion. Flannery argues that this is an accurate model for state formation and defines the 'near-state' chiefdoms as having 2 or 3 hierarchical levels with management of areas further than a day away being given to highly regarded individuals or family.

Agency: Additionally, Flannery outlines the requirements for an agent to cause state formation. The requirements are as follows:

1. Be an alpha male born with an aggressive authoritarian personality.
2. Be of elite parentage but not in the main line of succession. Just close enough enough to covet chieftainship.
3. Gain upward mobility as a military commander.
4. Usurp the position of chieftain by any means necessary.
5. Seek a competitive advantage over neighbouring chiefdoms.

6. Using that advantage, expand your territories.
7. Where the environment permits. Covet your land by investing in it by building irrigation systems, raising the population of subjects and keeping them content.
8. Where the environment does not permit, raid enemies and neighbours.
9. Solidify your position by power sharing, even if it is little more than a gesture.

The last rule is often misunderstood. It refers to the delegation of power as no single ruler can manage all the affairs of an entire state. Who the delegates might be and where they come from is dependent on the civilization.

Lastly, Flannery links both process and agency by way of an analogy. Just as both mutation and natural selection are related in biological evolution so too are process and agency in state formation. In fact, processes are often the result of long-term behavioural patterns by a collection of individuals with agency. Mutation needs natural selection to persist across generations. Just as the agency of chiefly individuals needs the preconditions of social inequality and chiefly competition to begin the process of becoming a king.

3.3.2.5 The Egyptian Predynastic and State Formation

In her essay [8], Stevenson synthesizes recent Predynastic Egypt literature and presents the formation of state as five phases detailed below. Stevenson puts forth that the emergence of the first dynasty was the culmination of a number of independently progressing entities. During this process, various entities would be torn down and reconstructed over time. Stevenson argues in favour of the view that inequality gave rose to the formation of state. She notes that their are in fact multiple possibilities for inequality. Food, exotic goods, and knowledge to name a few. Interestingly, this also includes the idea of 'wealth-in-people' which describes the success of leaders not only by their ability to secure resources and labour but also by their ability to gather and compose different sets of knowledge.

Neolithic Egypt: Stevenson challenges the idea that Neolithic groups became sedentary and suggests, using archaeological evidence, that the process was not as linear as once predicted. Groups would combine hunting and farming during different seasonal periods.

Farming appears to have been a method for surviving harsher conditions during certain periods of the year. The discovery of burial grounds in the delta suggest that groups grew attachments to specific areas of land and certain paths of travel. The mobile lifestyle of the early communities precludes the formation of any one form of social hierarchy. Stevenson suggests that both social differentiation and inequality were driving factors in the flux of mobility but does not state to what degree.

Naqada IA–IIB: There is substantially more evidence of sedentary living circumstances during this time period. This period marked the emergence of distinct material and visual cultures between Upper and Lower Egypt. There is also evidence of social stratification during this time. There is also evidence that structures, such as breweries, formed along trade routes.

Naqada IIC–D: This period represented a technological revolution. New technology allowed for new agricultural techniques as well as goods production. There was also a shift in culture as ritual objects from the previous period all but disappeared. The abandonment of some structures may suggest that conflict occurred during this time. During this period foreign trade is said to have been established as well as social migration from one locale to another. There is also evidence to suggest that Upper Egyptians wanted to control Northern land in order to facilitate better trade with local and foreign communities. It is also during this time that exotic goods started to be associated with a social elite. Stevenson describes this period as one of great prosperity.

Naqada IIIA–B/C: According to Stevenson, this period represented the age of monopolization and composition of both material and knowledge sources. Long standing cultural practices started to become associated with political power. This most likely required a monopoly over the use of force and violence. There is also evidence that the social elite used extremely rare and exotic goods to show their power. During this time the North Eastern Delta seems to have been densely populated

3.3.3 Putting it all together

In Section 3.3.2, we highlighted a number of theories related to state formation and, more specifically, state formation in Predynastic Egypt. Fortunately, the theories presented

share a great deal of overlap. Using Anđelković's [12] distinction of natural and social factors, the commonalities amongst the theories are presented as follows:

3.3.3.1 Natural Factors:

There are two distinct, but not mutually exclusive, natural factors present in a number of the aforementioned theories. They are desertification [12, 36] and the presence of the Nile river [12, 36]. It is easy to understand how desertification could throw Neolithic Egypt into turmoil. Hunting and foraging was no longer a predictable process for resource acquisition. This forced groups or villages to adapt and adopt farming as their primary source of food. This is not unbelievable as Stevenson [8] notes, their was already evidence of small-scale agricultural practices and although Kemp [139] argues that agricultural practices would have appeared as a consequence of a human's natural ability to generate complexity, desertification could have only accelerated that process. This new found reliance on farming meant that groups or villages could no longer be nomadic. This is where the importance of Nile river becomes evident. The soil near the Nile floodplain remained fertile all year round and was thus prime real estate for agricultural practices. Allen [36] argues that this would have drawn people towards the Nile thus increasing the population density and social tension. Competition over fertile land would have existed as groups jostled for territory. With Egypt thrown into further chaos, the social factors would come to fruition.

It should be noted that the Nile Delta is likely to have slowed these processes for Northern groups or villages. This is because the Nile Delta was more resilient to the effects of desertification and consequently, the lack of environmental pressure meant that Northern villages need not change their way of life.

3.3.3.2 Social Factors:

The turmoil caused by the Natural factors meant that Predynastic Egypt was open to the exploits of opportunistic individuals. These individuals were likely to exhibit specific behaviours and characteristics. As Flannery [140] mentions, these individuals were likely authoritarians with an aggressive personality. These despots likely used the turmoil to expand and secure their power. This is further supported by Allen's [36] theory where the

landlords (the despots) would have mobilized a military force to restrict movement out of their kingdoms. It is through these despots and their need to control the economic, military, political and religious systems that a power struggle, as described by Kemp [139], emerges. It is also likely to have been a non-linear process with chiefdoms or proto-states rising and collapsing over time until one kingdom was able to monopolize both the material and knowledge sources of the region [8]. It is also worth noting that the domination of the North by the South was likely a complex process that arose from both conquest and cultural transmission. This process is likely to have occurred during the end of the Predynastic period when the power centres of South had already been established.

3.3.3.3 What should an ABM of Predynastic Egypt look like?

ABMs that seek to explain the emergence of the Ancient Egyptian state are few and far between. This is most likely attributed to the delayed acceptance of agent-based modelling in the field of Archaeology. Lehner [143] designed a conceptual model of Ancient Egypt through the lens complex adaptive systems. The model is thorough with Lehner describing everything from household, village, nome and state structure to the taxation of landlords and the managing of land portfolios to account for the Nile flood waves. While Lehner's model is purely conceptual, it is insightful and can serve as inspiration for the implementation of an ABM. Symons and Raine [144] do exactly this and take aspects of Lehner's model to investigate the spread of information and population aggregation in an agrarian society. The model takes place on an abstracted Egyptian landscape. The agents are households and are grouped into villages. They are capable of transmitting information to other households and can migrate across the landscape. Villages claim plots of land and put the households to work. If a household feels the need for a better life, it can leave the village and move to another village it has knowledge of. While Symons and Raine's model is rather simple, it was capable of describing the benefits of partial information in preventing overcrowding and the aggregation of households along the Nile river. Nitschke et al. [145], influenced by Symons and Raine's ABM, sought to clarify some of the assumptions and characteristics in Kemp's [139] conceptual model. Similarly, agents were households on an abstracted Egyptian landscape and harvested grain from fields. Uniquely, agents were given ambition and competency values that were altered every [10,15] years to represent a change in management. The ambition and

competency were used to determine whether households could increase their membership, in the presence of surplus grain, and, in the case of competency, affect the yield of a patch of land. The results of the ABM suggested that, regardless of the variation in competency and/or ambition amongst households, a wealth disparity still emerges.

Taking into consideration the aforementioned related work and rest of the theory discussed in this section, we now attempt to identify the model functionality and agent behaviour that is pivotal to modelling the complex social processes of state formation in Predynastic Egypt. We will do this by blending the components of an ABM identified in Section 2.1 and the ECS design pattern discussed in Section 3.1.

Environment: All related work implemented a spatially-explicit environment. This makes sense given how the landscape of Egypt is heavily referenced in literature with desertification and the presence of the Nile floodplain thought to have been key natural factors that facilitated the formation of the state. However, related work abstracts these concepts significantly and often focus on specific regions as opposed to Egypt as whole. Symons and Raine [144], only use a $5km \times 5km$ modelling area and completely forego modelling both sides of the Nile as well as greatly abstracting Nile flood dynamics. Nitschke et al. [145] follow a similar approach.

While the simplified approach is understandable, it is limiting in that modelling cultural evolution across the entire Egyptian landscape is all but impossible. There is also something to be said by the simplicity in which farming yields are determined. A model that considered using a larger landscape in combination with GIS data, a vegetation model supplemented by archaeological data has great potential to recreate the dynamics of the Ancient Egyptian landscape to a degree not currently possible by the state-of-the-art.

Entities and Agents: As noted by Lehner [143] the primary decision making unit is the household. This term 'household' is vague but typically represents a familial unit. Lehner notes that the 'household' may be abstracted further to represent all individuals living under a single roof. This includes workers, slaves and extended family.

Collective representation of the households is also important. Across all the theories reviewed, it is clear that the ability for an agent to subjugate, coerce and cooperate with other agents is key. That is to say that in order to model the state formation of Predynastic Egypt, the agents should be able to organize themselves within their collective.

Allowing for the emergence of social stratification as well as various social organizational configurations (egalitarian and authoritarian for example) should seriously be considered. What the 'collective' actually represents is also interesting. Tribes, bands, villages, settlements, proto-nomes and nomes are all terms used in the literature to describe collectives at different stages in their development but, simply calling them villages [144] or settlements [25] seems appropriate regardless of scale. Specific nomenclature such as nome or tribe could be used if the modeller was interested in modelling the dynamics of those collectives specifically. In the cases where the size and structure of a settlement are variable, specific terms can be assigned to the collectives in a simulation posthumously as a method of describing the emergence and evolution of said collectives.

Systems: We have identified five systems that we believe are fundamental to modelling the formation of the Egyptian state. First, natural systems as noted in the environment discussion such as increased desertification and Nile flood dynamics. These systems should specifically make it increasingly difficult for inhabitants to live outside of the Nile floodplain. Secondly, different resource types and methods of acquiring them should be considered. Farmers relied on a completely different set of resources and acquisition mechanisms when compared to hunter-gatherers. It has also been noted the social elite used the quantity and variety of their resource coffers to differentiate themselves from their subjects [8, 12]. Coupled with resource acquisition is the system of resource transfer. This may refer to the transfer of resources between agents within a settlement, between settlements or even with outside interests. The mechanisms by which this transfer occurs should also be considered. It may be the case that an egalitarian society would use resource pooling and equal distribution whereas a socially stratified society may have elites employing commoners and taxing them.

Also, a population migration system is required. Allen [36] specifically states population migration and control over the mobility of the population were key to the formation of Ancient Egypt. Stevenson [8] notes that population migration northwards could have been the result of Upper Egyptians wanting to secure better trade routes. Whether it be due to the harshness of the land, high taxation rates, or the possibility of a better future, household agents should be able to attempt to relocate.

Lastly, processes of cultural differentiation and evolution should be considered. Predynastic Egypt is full of dichotomies. Foraging and farming, authoritarianism and egalitarianism, the desolateness of the desert and the near unlimited potential of the Nile. A model of Predynastic Egypt needs to facilitate these extremes and the various states in-between. This should be done at the agent level whereby agents may differentiate themselves in accordance with their personal experience and beliefs. The model should allow for an egalitarian hunter-gatherer and a despotic agrarian landlord to exist within the same simulation. The existence of these agent archetypes should be an emergent phenomena. That is to say that while the agents may all start out as hunter-gatherers, their lived experience should allow them to undergo cultural adaptation such that their past beliefs are replaced by new ones more suited to the current state of their environment. This should be occurring at the household and settlement levels simultaneously. Agents should also be able to convince other agents that their cultural beliefs are better (either through objective success [36], coercion [12] or force [140]) such that, over time, some cultural beliefs dominate others.

Using the above discussion, we categorize the list of minimally acceptable features needed to implement an ABM of Predynastic Egypt as:

1. A spatially-explicit environment of Egypt. An exact recreation of the environment is not necessary but, the modeller(s) should take specific steps to include Egypt's unique topography at the time whereby areas near the Nile were fertile and areas outside the floodplain were becoming increasingly arid.

2. One agent in the model represents a single family household. Further abstractions may be made to include slaves, workers or extended family as part of the 'household unit'.

3. Households should be able to group themselves into collectives such as villages or settlements.

4. Agents should be able to organize themselves in a social hierarchy. The complexity of the hierarchy and metrics for determining how it forms may vary but, it should at the very least facilitate the partitioning of the agents into social classes such that both Egalitarian and Authoritarian organization schemes are possible.

5. Household agents should have the ability to choose how they want to gather resources. While there no limit to how many resource types or resource acquisition methods can be added to the model, the agents must, at the very least, be able to choose between abstract farming and foraging actions such that nomadic and agrarian households can exist. A model may also allow the agents to change their resource acquisition strategies over time.

6. Agents should be capable of sharing or trading resources among themselves. Agents should exchange resources for other goods or services based on their standing in a social hierarchy. For example, a social elite agent may allow a few commoner agents to farm for them and take a cut of the resources farmed as tax. An egalitarian collective may pool their resources together and distribute them equally.

7. Lastly, the agent population should be able to undergo cultural evolution such that their resource trading beliefs, resource acquisition strategies and social hierarchies may change over time in accordance with the cultural beliefs associated with the most prosperous collectives and other processes such as cognitive innovation.

3.4 NeoCOOP - An ABM for Simulating Complex Social Phenomena in Ancient Societies

In this section, we highlight the design and implementation of *NeoCOOP* (Neolithic Cooperation Model), an ABM capable of simulating the Paleolithic-Neolithic transitionary period that saw humanity move from largely egalitarian hunter gatherer groups to agrarian super polities typically ruled by a social elite. Although *NeoCOOP* has already been used to study the emergence of social stratification in Neolithic-inspired households [41, 42], this section serves as the complete description of *NeoCOOP* and highlights how it will be used to answer the research questions presented in the Thesis (See Section 1.2). The model was created using *ECAgent* (See Section 3.1), the adaptive-agent types were designed using the information exchange process described in Section 3.2.3 and all contextual design decisions such as the scale at which to model the agents come from our review of Predynastic Egypt in Section 3.3.

Section 3.4.1 describes the environment of *NeoCOOP*, Section 3.4.2 details the design of the various agent-types supported by *NeoCOOP* and Section 3.4.3 presents *NeoCOOP*'s

systems. For completeness, an ODD+D compliant description of *NeoCOOP* has been included in Appendix A.1 and the source code is available at `https://github.com/BrandonGower-Winter/NeoCOOP`.

3.4.1 Environment

In *NeoCOOP*, agents are placed onto a grid-world consisting of $width \times height$ cells. All cells are considered square and uniform in size such that the area of any given cell is $cell\ dimension^2$ where *cell dimension* (meters) is a model input parameter.

Depending on the type of environment (*simple* or *complex*), a single cell comprises the following properties:

1. **position**: A 2-tuple that stores the x and y-coordinates (integers) of the cell. This property is immutable.
2. **resources**: The amount of resources (kg) at a given cell. This floating point value is bound between 0 and the *carrying capacity* input parameter.
3. **is settlement**: An integer value that stores whether a settlement is located on the cell. A value $= -1$ indicates that the cell is not occupied by a settlement. A value ≥ 0 indicates the id of the settlement occupying the cell.
4. **is owned**: An integer value that stores whether the cell is owned by a farming household. A value $= -1$ indicates that the cell is not owned. A value ≥ 0 indicates the id of the household that owns the cell.
5. **height**: A floating point value indicating the height (m) of the cell above sea level.
6. **slope**: A floating point value indicating the slope (°) of the cell.
7. **is water**: A boolean value that stores whether the cell forms part of a body of water.
8. **soil moisture**: A floating point value indicating the soil moisture (mm) of the cell. This value is bound between 0 and the *capacitance* of the cell.
9. **sand content**: A floating point value indicating the sand content (%) of the cell. This value is bound between 0 and 100.

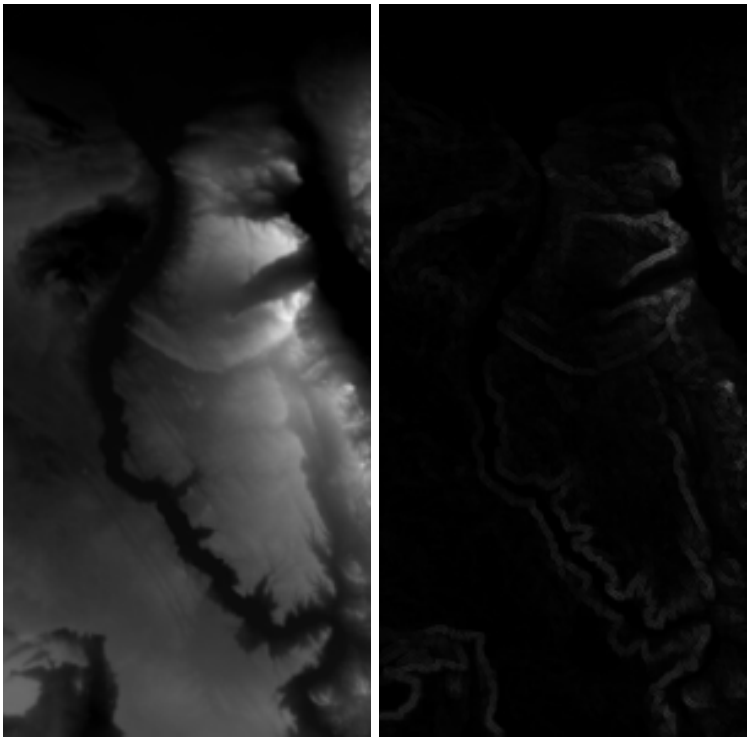
If the *simple* environment is used, cell properties 5-9 are excluded. The purpose of the *simple* environment is to ensure that testing and iterating on the model is fast given that the Vegetation Model (discussed later) comes at an additional computational cost which can make testing and debugging the code difficult. Additionally, the *simple* environment can be used to study the behaviour of the agents in toy-box environments which some modellers may find useful.

At the beginning of a simulation run, the initial values of a given cell's height, slope, is water and sand content properties are based on the GIS data-maps that are fed to the model as grayscale images (See Figure 3.17). The dimensions of the images are the same as the environment such that one pixel corresponds to exactly one environment cell at the same pixel coordinates. Calculating the value of the cell using the pixel value of the respective image $\in [0, 1]$ is done by linearly interpolating between min and max model input properties. For example, the height of some arbitrary pixel is calculated as follows:

$$height_c = height_{min} + IMG_{h,c}(height_{max} - height_{min}) \tag{3.13}$$

where $height_c$ is the height of cell c. $height_{min}$ and $height_{max}$ are the minimum and maximum possible height values respectively and $IMG_{h,c}$ is the pixel value $\in [0, 1]$ of the height image at cell c. This logic is also applied to the slope and sand content properties using their respective minimum and maximum values. For *is water*, *true* is returned if the pixel value of the water image is greater than 0.

Considerations were made to compress the data-maps into one or two images utilizing separate colour channels to more efficiently store the data but, the idea was quickly abandoned due to the added complexity it introduced when images needed to be quickly modified. The inclusion of GIS data allows us to model the any topographical location for which the required data maps (height, slope, is water and sand content) are available. Figures 3.17 and 3.18 showcase how this is applied to the Predynastic Egyptian landscape more clearly.



(A) Heightmap (B) Slopemap

FIGURE 3.17: Example GIS data-maps.

(A) isWater (B) Sand Content

FIGURE 3.18: Example GIS data-maps (continued).

3.4.2 Agent-Types

Given that we developed several types of agents with varying degrees of adaptive capacity, we will first describe what they all have in common and then discuss their differences in each each of their respective sections. *NeoCOOP* supports four agent types called: *Traditional*, *Rule-based Adaptive*, *Utility* and *Information Exchanging*.

In *NeoCOOP*, the primary decision making agent the household. This primary motivation for this is that in Predynastic Egypt, and other ancient societies, a household was typically ruled by a patriarchal figure who oversaw the management of the entire household [143]. Across all agent-types, a household h is defined by the following properties:

1. **id:** A household's unique identifier. This value is an integer.
2. **Resources:** A float that stores the amount of resources (kg) the household has in its personal storage.
3. **Load:** A float that stores the amount of resources (kg) a household has donated to other households over a given period of time.
4. **Storage History**: A first-in-first-out list storing the history of agent resource acquisitions. This property is not enabled by default and is only used when the input parameter *storage decay* > 0.
5. **Occupants:** A list of occupant entities that are contained in the household agent.
6. **Hunger:** A float $\in [0, 1]$ that describes how hungry the agent is. A value of 0.0 means the agent is starving and a value 1.0 means the agent is completely satiated.
7. **Satisfaction:** A float value $\in [0, 1]$ representing the agent's average hunger over the past *years per move* iterations. *Years per move* is a user-defined property discussed later.
8. **Attachment (α):** A float value $\in [0, 1]$ that describes how 'attached' an agent is to its current settlement. If the value is > 0.5, the agent is less likely to move even in the face of environmental stress. If the value is < 0.5, the agent is more likely to move at the slightest indication of hardship.

9. **Peer Transfer:** A float value $\in [0, 1]$ that indicates how likely the agent is to donate its resources to its peers. A value close to 1.0 indicates that the agent is altruistic and will help its peers regardless of the circumstances. A value close to 0.0 indicates that the agent is selfish and is less likely to help its peers.

10. **Subordinate Transfer:** A float value $\in [0, 1]$ that indicates how likely the agent is to donate its resources to its subordinates. A value close to 1.0 indicates that the agent is altruistic and will help its subordinates regardless of the circumstances. A value close to 0.0 indicates that the agent is selfish and is less likely to help its subordinates.

11. **Authority Transfer:** A float value $\in [0, 1]$ that indicates how likely the agent is to donate its resources to its authority agents. A value close to 1.0 indicates that the agent is altruistic and will help its authorities regardless of the circumstances. A value close to 0.0 indicates that the agent is selfish and is less likely to help its authorities.

Additionally, the *Traditional* and *Rule-based Adaptive* agents are defined by an additional property:

1. **Farm Threshold:** A float value $\in [0, 1]$ that indicates how likely the agent is to farm. A value close to 1.0 indicates that the agent will always choose to farm whereas a value close to 0.0 indicates that the agent is more likely to forage.

The *Utility* agents are defined by the following additional properties:

1. **Farm Utility:** An unbounded float that represents the utility the agent associates with the farm resource acquisition action.

2. **Forage Utility:** An unbounded float that represents the utility the agent associates with the forage resource acquisition action.

3. **Stubbornness (η):** A float value $\in [0, 1]$ that describes the degree to which an agent will accept new information. A value close to 1.0 indicates that the agent is naive and will accept new information regardless of its own opinions. A value close to 0.0 indicates that the agent is stubborn and less likely to accept new information

over its own opinions. From an implementation perspective, *stubbornness* is the learning rate of the agent for all Reinforcement Learning processes.

Lastly, the *Information Exchanging* agents extend the *Utility* agent's definition to include the following additional property:

1. **Conformity (σ):** A float value $\in [0, 1]$ that describes the degree to which an agent will accept new information from its neighbours. This property is functionally identical to *stubbornness* but, it is associated with the Evolutionary Algorithm processes which involve collective learning as opposed to individual learning.

A household contains one or more *occupants*. They are not decision making entities and are purely used to track the population and resource acquisition capabilities of a household. *Occupants* are two-tuples defined by a unique identifier and their age (both integers). Optionally, *NeoCOOP* allows a modeller to separate households into two sub-populations: *able workers* and *unable occupants*. By specifying two input parameters *age of maturity* and *age of senility*, a household's *able workers* can be determined as follows:

$$able\_workers = \{o \,|\, o \in h \text{ and age\_of\_maturity} < o.age < \text{ age\_of\_senility}\} \quad (3.14)$$

Where o is a occupant and h is a household agent. The set of *able workers* is used to determine how many cells the households can forage or farm (discussed in more detail later).

Unlike most ABMs that facilitate agent-to-agent cooperation, *NeoCOOP* allows agents to make decisions based on their social status and the social status of the agents they are interacting with. In *NeoCOOP*, social status is defined as the sum of an agent's available *resources* and its *load*. To facilitate social stratification, we use the self-organization scheme described by Chliaoutakis and Chalkiadakis [25] whereby a relationship type can be determined for every agent pair by comparing their social statuses. We define each of the relationship types as follows:

$$is\_acq(h_1, h_2) = h_1.settlement == h_2.settlement \quad (3.15)$$

$$is\_peer(h_1, h_2) = \frac{|h_2.status - h_1.status|}{max(h_1.status, h_2.status)} < L \tag{3.16}$$

$$is\_auth(h_1, h_2) = \frac{(h_2.status - h_1.status)}{max(h_1.status, h_2.status)} > L \tag{3.17}$$

$$is\_sub(h_1, h_2) = is\_auth(h_2, h_1) \tag{3.18}$$

Where is_acq, is_peer, is_auth, is_sub describe whether household h_2 has an acquaintance, peer, authority or subordinate relationship with household h_1 respectively. $h_n.status$ is a household's social status. L is the *load difference* $\in [0, 1]$ input parameter which describes how much more social status an agent requires to be considered an authority over another agent. Note that in order for a peer, authority or subordinate relationship to be formed, the two households must be from the same settlement (i.e. $is\_acq = true$).

3.4.2.1 Traditional Agents

The *traditional* agent-type is simplest of all the agents developed in this work as they are meant to represent early rule-based agents found in other model's such as Epstein's Sugarscape [46] and Axelrod's cultural dissemination model [146]. More specifically, *traditional* agent's are built using the threshold design illustrated in Chapter 2 (See Algorithm 1).

In *NeoCOOP*, agent's are responsible for making three types of decisions classified as *Resource Acquisition*, *Resource Transfer* and *Migration*. The systems that use these decisions are defined in Sections 3.4.3.4, 3.4.3.5 and 3.4.3.7 respectively but how each decision is made differs depending on agent-type. Unless specifically stated, agents default to the forthcoming decision making strategies. From a software design perspective, one can view the *traditional* agent-type as a base class and other agent-types simply overload the *traditional* agent's behaviour.

Resource Acquisition: The first decision an agent makes each iteration is how it will acquire resources. In *NeoCOOP*, agents can either FARM or FORAGE. As noted in Section 3.3, these two actions represent high-level abstractions of the two primary methods of resource acquisition during the Predynastic period. On the one hand, foraging represents actions such as gathering, hunting and fishing while farming represents both

Algorithm 5: Pseudocode for *Traditional* Agent's resource acquisition decision making system.

```
def get_num_to_farm(farm_threshold : float, num_choices : int):
    num_to_farm = 0
    forall i in range(num_choices) do
        if rand() < farm_threshold then
            num_to_farm += 1
    end
    return num_to_farm
```

crop cultivation and animal husbandry. The actual mechanisms that govern how these actions work are discussed in Section 3.4.3.4 but, farming is designed to provide more resources at the cost of immobility while foraging offers the inverse (less resources and greater mobility).

As shown in Algorithm 5, traditional agents make their decisions by sampling a random value $\in [0.0, 1.0]$ from a uniform distribution. If that value is less than its *farm_threshold*, they will choose to farm. This process is repeated for the number of times the agent needs to make that decision. For example, if the agent needs to make three choices, one result might see the agent choosing to FARM twice and FORAGE once. The key property to determine the likelihood of choosing the FARM action is the agent's *farm_threshold*.

For the *traditional* agents, the *farm_threshold* for a household is determined using Equation 3.19.

$$farm\_threshold = \frac{forage\_grad * t + forage\_offset}{forage\_duration} + rand(-margin, margin) \tag{3.19}$$

where t is the simulation timestep, $forage\_grad$, $forage\_offset$ and $forage\_duration$ are user-defined parameters that describe the rate at which the agents adopt farming. Lastly, $margin$ is a user-defined float $\in [0.0, 1.0]$ that is used to introduce agent heterogeneity as agents will adopt farming asynchronously.

This decision making process is clearly simple but, it is still quite versatile. It does assume that farming must be adopted and that this process is linear but, one can view Equation 3.19 as a line of best fit with the $margin$ property representing the error bars. These values could be directly derived from the archaeological data should it exist.

Algorithm 6: Pseudocode for Agent's resource transfer decision making system. **Note:** This pseudocode assumes that both the *recipient* and *donor* are acquaintances.

```
1  def request_resources(recipient : Household, donor : Household, resources_required
     : float):
2      if is_Auth(donor, recipient) then
3          if rand() <= donor.sub_transfer then
4              return grant_resources(donor, resources_required)
5      else if is_Peer(donor, recipient) then
6          else if rand() <= donor.peer_transfer then
7              return grant_resources(donor, resources_required)
8      else if rand() <= donor.auth_transfer then
9          return grant_resources(donor, resources_required)
10     return 0.0
```

Resource Transfer: As demonstrated in Algorithm 6, the agents' resource transfer decision making system is based on the two interacting agent's relationship to each other. When a donor agent receives a request, it will determine its relationship type to the potential recipient and generate a random value $\in [0.0, 1.0]$. If that random value is less than or equal to the donor agent's respective transfer property (*sub_transfer*, *peer_transfer* and *auth_transfer* for Authority, Peer and Subordinate relationships respectively), the request will be granted.

Migration: For population migration, the agent's follow Equation 3.20.

$$move(h) = 2\alpha_h * satisfaction(h) < random() \tag{3.20}$$

where h is the household agent, *alpha* is the attachment of household h and $satisfaction(h) \in [0.0, 1.0]$ (discussed more in Section 3.4.3.6) is a measure of the degree to which the agents resource needs have been met. As noted earlier, attachment is a float value $\in [0, 1]$ that describes how 'attached' an agent is to its current settlement. We multiply the attachment value by 2.0 such that if a household has an attachment value > 0.5, it is willing to endure some degree of resource hardship before deciding to move. An agent with an attachment value < 0.5 exhibits inverse behaviour, more likely to flee or migrate at even the slightest sign of hardship. From a design perspective, satisfaction represents the agent's objective circumstances while attachment represents its subjective or emotional status with regards to its current location.

3.4.2.2 Rule-Based Adaptive Agents

Given the simplicity of the *traditional* agents and that their design philosophy arises from pre-2000s ABMs, we were motivated to create a rule-based agent that more closely represents what you would find at the time of writing this thesis. In this work, we call this agent-type the *rule-based adaptive* (*rb-adaptive*) agent. Its design is inspired by Hailegiorgis et al. [147]'s OMOLAND-CA model which uses protective motivation theory [148] (PMT) to introduce subjective adaptive-capacity to their agents.

The *rb-adaptive* agents are almost identical to the *traditional* agents except that their *farm_threshold* and household properties (attachment, peer, subordinate and authority transfer specifically) may change over time.

The method for updating a household's farm threshold is defined by four separate steps. The first step is prediction. At every iteration, *rb-adaptive* agents will predict the severity of the current living conditions. In the OMOLAND_CA model, this is done by looking at rainfall onset values. In *NeoCOOP* we just use the agent's hunger as it is clear indicator of the severity of the agent's living circumstances (See Equation 3.21).

$$severity(h) = 1.0 - h.hunger \tag{3.21}$$

Where h is the agent household. The next step is risk assessment. At this step, agents will use the severity value and Equation 3.22 to calculate the risk associated with its current circumstances.

$$risk\_appraisal(h) = 0.6 * severity(h) + 0.4 * rand() \tag{3.22}$$

Using their assessed risk, the agents then calculate their adaptation intention using Equation 3.25 which will asses the agent's capability to adapt at a given timestep.

$$r(h) = risk\_appraisal(h) * risk\_elasticity \tag{3.23}$$

$$p(h) = adaptation\_appraisal(h) * (1 - cognitive\_bias) \tag{3.24}$$

$$adapt\_intention(h) = r(h) - p(h) \tag{3.25}$$

Where $adaptation\_appraisal(h)$ is the agent's ability to adapt which is discussed more in Section 3.4.3.9. $risk\_elasticity$ and $cognitive\_bias$ are user configurable parameters.

Lastly, the agent will perform a learning step that updates their *farm_threshold*, *attachment* and *resource transfer properties* if their $adapt\_intention(h) > adapt\_threshold$ where $adapt\_threshold$ is another user configurable parameter. The update to the agent's properties are done using Equation 3.26.

$$G(p)_{h,t+1} = G(p)_{h,t} + \gamma * (\frac{1}{|S'|} \sum_{h' \in S'} G(p)_{h',t} - G(p)_{h,t}) \tag{3.26}$$

Where, p is the agent property ($farm\_threshold$ is one of these properties), t is the timestep, G is a function that gets or sets property p for household h, γ is the *learning rate* configurable parameter and S' is the set of all households that share a settlement with household h and will adapt at timestep t.

Given the complexity of the *rb-adaptive* agent, it will undoubtedly exhibit greater adaptive capacity than the *traditional* agent-type. The caveat is that *rb-adaptive* agents consist of significantly more tunable parameters. A lot of these properties can simply be derived from data but, that data may not be available and thus will need to derived using other means.

3.4.2.3 Utility Agents

The third type of agent is called the *Utility* agent. It is the first agent-type to use ML as an adaptive-mechanism. The central concept that underpins the design of the *utility* agent is Utility theory. More specifically, *utility* agents seek to maximize utility such that they are constantly taking actions that result in the greatest rewards. Inspired by Chliaoutakis and Chalkiadakis [25], *utility* agents use RL to discover whether foraging or farming is better.

As shown in Algorithm 7, Each *utility* agent has two local properties called *forage_utlity* and *farm_utlity* which indicate the utility said agent associates with each action respectively. These utility values are state-agnostic meaning that their utility values are not directly associated with where the agent is located in the environment. This is a limitation but, the alternative solution where agents maintain both a general utility

Algorithm 7: Pseudocode for *Utility* Agent's resource acquisition decision making system.

```
def get_num_to_farm(h : Household, num_choices : int):
    num_to_farm = 0
    forall i in range(num_choices) do
        if rand() > h.satisfaction then
            num_to_farm += 1 if rand() < 0.5 else 0
        else
            num_to_farm += 1 if h.farm_utility > h.forage_utility else 0
    end
    return num_to_farm
```

value for each action and utility values for each cell in the environment was considered out of scope as it would make comparing behaviour produced by *utility* agents and the *traditional* agents quite difficult[16].

For each choice, a random value $\in [0.0, 1.0]$ is generated and compared to the agent's satisfaction. If the generated value is greater, the agent will randomly decide to FORAGE or FARM. If the generated value is less than the satisfaction, the agent will take the action that it perceives as having the maximum utility. In the context of RL, Algorithm 7 is the exploration-exploitation heuristic used by the agents. If the agents are satisfied, they are less like to take exploratory actions. Alternatively, if the agents are less satisfied, they are more likely to take exploratory actions in the hopes of finding a better strategy than they currently have. It is also worth noting that *utility* agents do not guarantee that farming will emerge as the dominant strategy. It also allows farming to emerge in several ways. For example, a sudden drop in the yield produced by foraging would result is low satisfaction across the board and mass action space exploration which may result in the rapid adoption of farming. Alternatively, if satisfaction remains relatively high throughout a simulation run, there would be fewer exploratory actions which may result is slower adoption of farming if at all.

When the *utility* agents update their utility values, they use Equation 3.27.

$$U_{h,t+1}(a) = U_{h,t}(a) + \eta_h(R_{h,t}(a) - U_{h,t}(a)) \tag{3.27}$$

[16] More specifically, the *utility* and *traditional* agents are meant to represent the simplest implementations of the two different design paradigms being compared in this thesis (ML and rule-based agents).

where $U_{h,t}(a)$ is the utility value household h associates with action a (FORAGE or FARM) at timestep t. η_h is the *stubbornness* of the agent and $R_{h,t}(a)$ is the reward the agent received for performing the action a at timestep a.

3.4.2.4 Information Exchanging Agents

The final agent-type is the *Information Exchange* agent (*IE*-agent). As its name suggests, the agent is constructed using the IE process described in Section 3.2.2. One of the main limitations of the *utility* agent is that it cannot share its beliefs with other households. This means that other household agents cannot learn from successful agents. As shown in Figure 3.19, *IE* agents explicitly deal with this issue by adding *communicative* functions that facilitate both biological (households to offspring household exchange) and cultural (households to settlement, settlement to settlement and settlement to households exchange) adaptation. These *communicative* functions take the form of two Evolutionary Algorithms (EAs): A Genetic Algorithm [149] (GA) and a Cultural Algorithm [96] (CA).

Using the resource acquisition strategy described in Algorithm 7 and the two EAs, the *IE* agents are not only capable of learning from their environment, but they are also capable of exchanging what they've learned with other households. Given that EAs are fitness-based algorithms, we determine the fitness of a household agent h using Equation 3.28.

$$f(h) = h.status \tag{3.28}$$

Where, as noted earlier, social *status* is the sum of the agent's *resources* and its *load*. With this fitness function, we do not bias any particular type of resource sharing behaviour. For example, if the fitness function did not take the agent's generosity into account, then selfish behaviour would be unfairly promoted. Additionally, if the agent has enough resources such that it can fulfill donation requests, it is arguably as fit as an individual with the same amount of resources who rejects all donation requests[17].

[17]This assumes that altruistic and selfish behavior are equal. Under different environmental conditions that may not be the case but, by design, agents can evolve towards the most situationally fit solution.

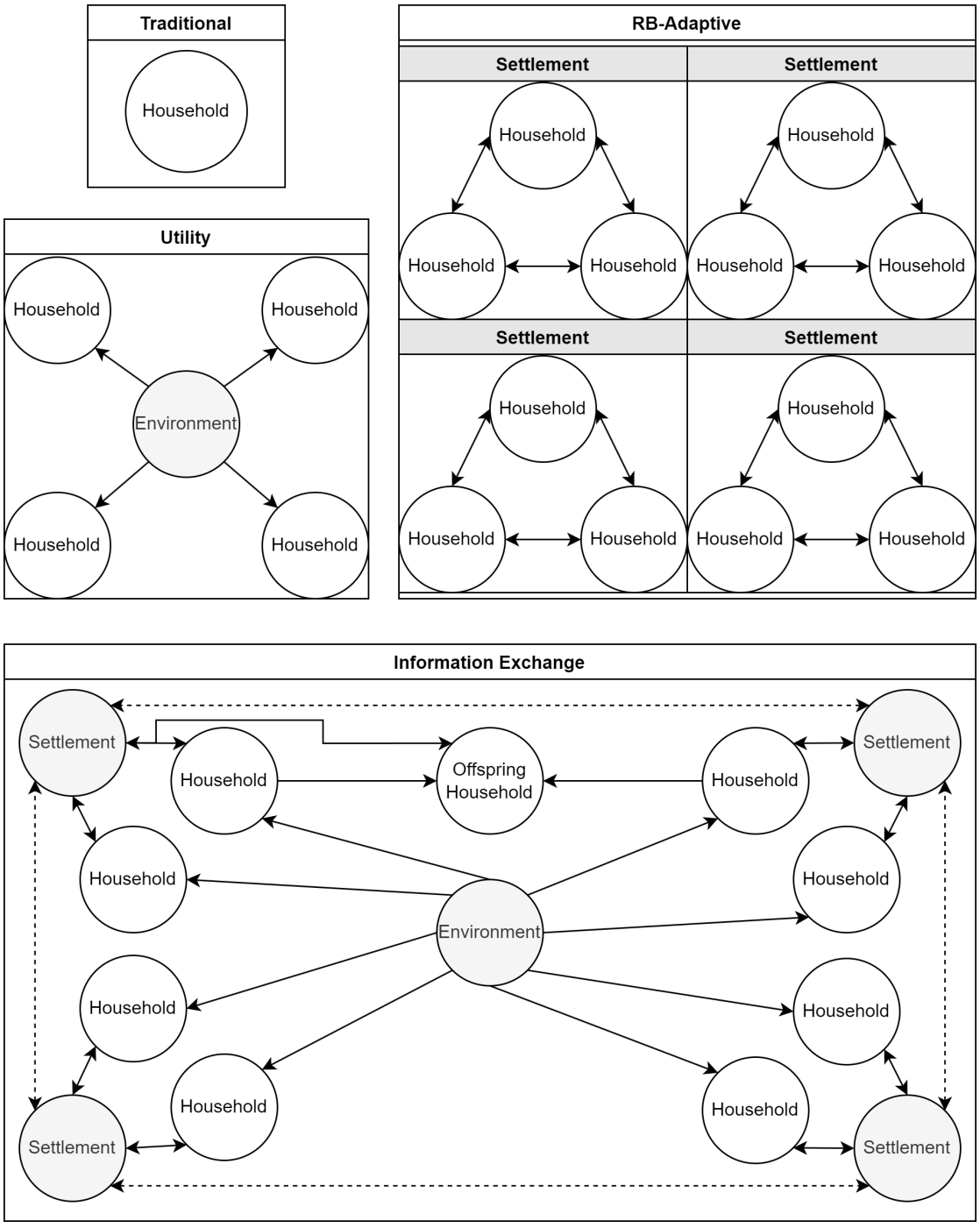


FIGURE 3.19: An information exchange perspective of the agent-types at some arbitrary timestep. Note: All *settlement* entities in the *Information Exchange* agent network should be linked. The diagonal connections were omitted for visual clarity.

Both the GA and CA utilize the *IE* agent's genotype which consists of the following genes (agent properties): Farm Utility, Forage Utility, Stubbornness, Conformity, Peer Transfer, Subordinate Transfer and Authority Transfer and Attachment. The algorithms also make use of a concept called *influence*. *Influence* is used to determine best performing settlements and describes the probability that two settlements will interact with each other. This is done using XTENT [40] (Equation 3.29):

$$I(s_1, s_2) = (s_2.status)^{\beta} - mD(s_1, s_2) \tag{3.29}$$

Where, s_1 and s_2 are settlements, $I(s_1, s_2)$ is the influence of s_2 on s_1, $s_2.status$ is the social status of s_2 (the sum total social status of all agents in s_2), $D(s_1, s_1)$ is distance from s_1 to s_2. β and m are coefficients describing the required social status of one settlement to influence another.

Calculating the *influence* of every settlement on a given settlement, gives a probability distribution (Equation 3.30).

$$P(s_1, s_2) = \frac{I(s_1, s_2)}{\sum_{k \in K} I(s_1, s_k)} \tag{3.30}$$

Where $P(s_1, s_2)$ is the probability of settlement s_2 influencing settlement s_1 and K is the set of neighbouring settlements that have a positive *influence* value $I(s_1, s_k)$ on s_1.

Genetic Algorithm: The GA executes whenever the *split_household* function is called (Algorithm 15). The child agent produced is a combination of two parents with the first parent being the household that called the *split_household* function and the second parent determined by *roulette wheel* selection. This selection uses the social status of other agents within the same settlement of the first parent and from other settlements that have enough influence ($I(s_1, s_2) > 0$). The offspring agent is produced using *Uniform crossover* and *Random mutation*. The only exception to this is are the Farm and Forage Utility values which use Gaussian mutation.

Cultural Algorithm: The CA uses *Knowledge Sources* [102] to diversify how agents are influenced. These are:

- **Normative:** Influence on agent genes from its settlement.
- **Spatial:** Influence on agent genes from another settlement.
- **Domain:** Equivalent to GA mutation function, where domain influence mutates one of the agent's genes (See Section 3.4.3.8).

Every *influence_frequency* iterations, agents are influenced in accordance with the *influence_rate*. If an agent is selected for influencing, a roulette wheel is spun to determine from which knowledge source influence will come from. Influence from the Domain knowledge source occurs at a rate defined by the *mutation_rate* parameter. Influence from the Normative and Spatial knowledge sources occur with varying probability defined by equations 3.31 and 3.32.

$$N(s_h, s_i) = max(\frac{s_h.status}{s_i.status}, 1.0) \tag{3.31}$$

$$S(s_h, s_i) = 1.0 - N(s_h, s_i) \tag{3.32}$$

Where, N and S are the probability of choosing the Normative and Spatial knowledge sources respectively, s_h is the settlement of the agent being influenced, s_i is the settlement that would influence agent h. If the spatial knowledge source is selected. s_i is determined by performing roulette wheel selection on all neighbouring settlements with a positive *influence* on settlement s_h. Roulette wheel weights are determined by the values returned by Equation 3.30.

Each settlement's beliefs are represented by *Belief Spaces* B_s. Belief Spaces have the same structure as the agent genotype with each property calculated using a weighted average of the corresponding property of all agents within that settlement. The weight an agent contributes to the belief space is determined using its social status relative to the social status of the other agents in the same settlement. If an agent is influenced by the *normative* knowledge source, the belief space that influences it is the belief space of the settlement the agent belongs to B_{s_h}. If the agent is influenced by the *spatial* knowledge source, the belief space that will influence the agent is the belief space of the

settlement selected during roulette wheel selection (B_{s_i}). Agent properties are influenced as follows (Equation 3.33):

$$G_{h,t+1}(p) = G_{h,t}(p) + \sigma_h(B_{s,t}(p) - G_{h,t}(p)) * \Phi(h, B_{s,t}) \tag{3.33}$$

Where, p is the agent gene property, t is the timestep, G is the agent's genotype, σ_h is the *conformity* of the agent, B is the selected belief space (B_{s_h} or B_{s_i}) and Φ is the Homophily term which returns a value $\in [0, 1]$ describing how similar the agent's genes are to the belief space that is influencing it. Homophily is a sociological principle that describes the tendency for individuals that are similar, either biologically or culturally, to gather together. The value of *Phi* is 1.0 for interacting entities that have exactly the same genes, and close to 0.0 for entities whose gene values are further apart. This approach is similar to interaction probability in Axelrod's cultural dissemination model [146]. In the model, Φ limits the degree to which an agent is influenced if the belief space influencing it contains drastically different gene values. Formally, Φ is one minus the average absolute difference between the agent and influencing belief space's genes (Equation 3.34).

$$\Phi(h, b) = 1.0 - \frac{1}{|G_{h,t}|} \sum_{p \in G_{h,t}} |h.p - b.p| \tag{3.34}$$

When comparing the four agent-types introduced, the *IE* agent is most similar to the *rb-adaptive* agent in terms of potential adaptive capacity but with far fewer configurable parameters. This, in theory [18], should make them easier to parameter tune.

The four agent-types discussed include both simple and complex implementations of the two design paradigms (ML and rule-based) that we are interested in studying. By exposing these agents to a sufficiently complex and adverse environment, we should attain a more-complete understanding of the benefits, if any, the ML agents (*utility* and *IE*) have over the rule-based (*traditional* and *rb-adaptive*) agents in terms of their adaptive capacity.

[18] This assumes that the algorithms are less sensitive to variations in input-space. Early exploration with different agent designs showed that some complex systems were noticeably more resistant to variations in their input space despite have several additional parameters that could affect them.

Algorithm 8: Pseudocode for the Global Environment System.

```
def global_environment_system():
    min_rainfall, max_rainfall = interpolator(start_rainfall, end_rainfall,
     rain_params)
    min_temperature, max_temperature = interpolator(start_temperature,
     end_temperature, temperature_params)
    min_flood, max_flood = interpolator(start_flood, end_flood, flood_params)
    rainfall = rand(min_rainfall, max_rainfall)
    temperature = rand(min_temperature, max_temperature)
    flood = rand(min_flood, max_flood)
    return rainfall, temperature, solar, flood
```

3.4.3 NeoCOOP Systems

In this section we will discuss the implementation and execution of *NeoCOOP*'s systems. Figure 3.20 depicts the list and execution cycle of the model and Sections 3.4.3.1 - 3.4.3.9 discuss each of the systems listed in detail.

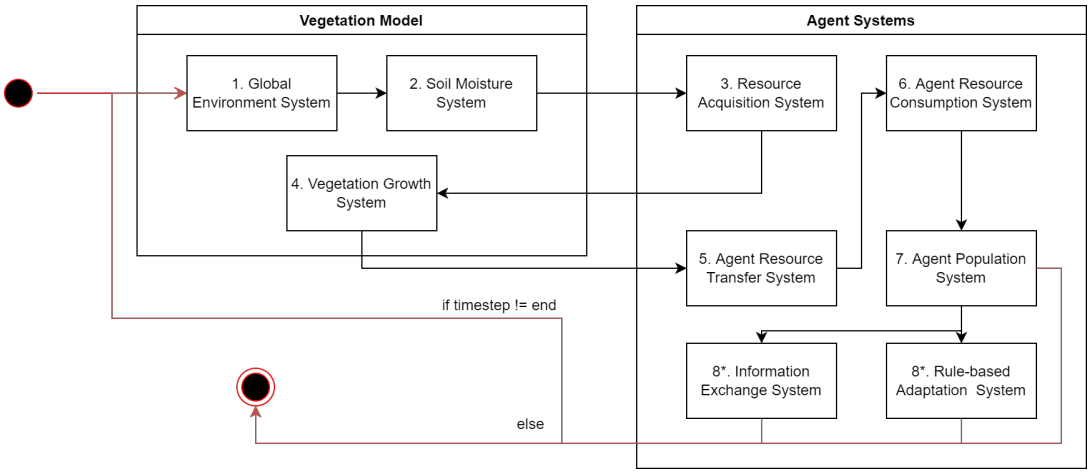


FIGURE 3.20: A figure depicting the execution order of *NeoCOOP*'s systems. The number that prefixes the system's name indicates its position in the execution queue. The *Information Exchange System* and *Rule-based Adaptation* systems are present when the type of agent being investigated are the *IE* and *rb-adaptive* agents respectively.

3.4.3.1 Global Environment System

NeoCOOP utilizes a Vegetation Model. Adapted from Xu et al. [150], the Vegetation Model comprises three systems which act sequentially to calculate an estimate of the total growth penalties of each environmental cell using the Carnegie-Ames-Stanford-Approach (CASA) approach. The first of these components is the Global Environment System (See Algorithm 8). As its name suggests, it is responsible for generating values

Algorithm 9: Pseudocode for the Soil Moisture System.

```
def soil_moisture_system():
    PET = thornthwaite(temperature)
    forall cell : Cell in environment do
        if cell.is_water then
            continue
        else if is_flooded(cell) then
            cell.moisture = wfc(cell.sand_content)
        else
            if PET > rainfall_buffer[i] then
                rdr = RDR(cell.moisture + rainfall, cell.sand_content,
                 cell.clay_content)
                cell.moisture = max(0.0, cell.moisture - (PET - rainfall) * rdr
            else
                wfc = WFC(cell.sand_content, cell.clay_content)
                cell.moisture = min(cell.moisture + (rainfall - PET), wfc)
    end
    return
```

for the global environment properties every iteration. These three properties are rainfall (mm), temperature (°C) and Nile flood height (m). The exact method for generating these values is discussed in Chapter 4 but every iteration, the Global Environment System will generate yearly average rainfall, temperature and flood height values. The system does not do much by itself but the values it generates are used several of the systems yet to be described.

3.4.3.2 Soil Moisture System

The Soil Moisture System (See Algorithm 9) is the second Vegetation Model system and it is responsible for updating the amount of soil moisture at a given cell. Every iteration, the Soil Moisture System will iterate over every cell in the environment. The potential evaporation (PET) is calculated using the *Thornthwaite* equation [151] and compared to the rainfall value generated by the Global Environment System. If PET is greater, the soil moisture of a cell is reduced by calculating the difference in PET and rainfall multiplied by the *relative dry ratio* (RDR) [152] defined by Equation 3.35:

$$RDR(m, c_{sand}, c_{clay}) = \frac{1 + A(c_{sand}, c_{clay})}{1 + A(c_{sand}, c_{clay}) * m^{B(c_{sand}, c_{clay})}} \tag{3.35}$$

where m is soil moisture, c_{sand} is the sand content (%), c_{clay} is the clay content (%) and A and B (See Equations 3.36 and 3.37) are curves that describe the texture related drying functions [153]:

$$A(c_{sand}, c_{clay}) = exp(-4.396 - 0.0715 * c_{clay} - 0.000488 * (c_{sand})^2 - 0.00004258 * (c_{sand})^2 * c_{clay}) * 100.0 \tag{3.36}$$

$$B(c_{sand}, c_{clay}) = -3.140 - 0.00222 * (c_{clay} * *2) - 0.00003484 * (c_{sand})^2 * c_{clay} \tag{3.37}$$

If the monthly rainfall is greater than the PET, their difference is added to the cell's stored moisture. This value is capped at the soil's saturated water content value (wfc) which is calculated using a Equation 3.38 [153].

$$WFC(c_{sand}, c_{clay}) = 0.332 - 7.251 \times 10^{-4}(c_{sand}) + 0.1276 log_{10}(c_{clay}) \tag{3.38}$$

Lastly, if the cell is a water cell, this entire process is skipped. If the cell is flooded, the soil moisture is just set to its maximum value (wfc) for the iteration. The motivation for this decision is that if the cell is flooded, it would provide sufficient resources and nutrients to the plants growing there for that iteration. It is admittedly a simplistic approach but, more complex flood dynamics were considered out-of-scope for this project due to the intense workload required to add such a feature, and the minimal return it would provide to an already complex model. The exact method for determining if a cell is flooded is included in Appendix C.3.

3.4.3.3 Vegetation Growth System

The last process of the Vegetation Model is the Vegetation Growth System (See Algorithm 10). In a standard Vegetation Model, net primary productivity (NPP) is defined as the amount of photosynthetically available radiation multiplied by the growth inhibiting environmental factors (water and temperature specifically) represented as a penalty score $\in [0, 1]$. We do not actually calculate NPP in this work, instead we only calculate

Algorithm 10: Pseudocode for the Vegetation Growth System.

```
def vegetation_growth_system():
    forall cell : Cell in environment do
        penalty = Equation 3.39
        penalty *= Equation 3.40
        penalty *= Equation 3.41
        penalty *= cell.slope
        cell.resources = penalty
    end
    return
```

the penalty factor and derive available resources from that value. The motivations for this decision are twofold. First, converting NPP into a unit that makes sense to consume (kg for example) is not straightforward and requires data for several additional input parameters that are plant species specific (which doesn't integrate nicely with our generalized 'resources'). Given the abundance of sunlight in Egypt [12], one could reasonably argue that the greatest factors affecting the amount of resources available are the penalty factors. Thus, we just assume that their is sufficient solar radiation and use the penalty factor to indicate the quantity of resources available. The second reason for this decision is simplicity. Given that the penalty factor is a value $\in [0, 1]$, it is easy to determine if the environment produced sufficient resources. A value close to 1.0 indicates near perfect living conditions while a value close to 0.0 indicates unlivable conditions.

Every iteration, the Vegetation Growth System calculate the growth inhibiting penalties for each environment cell. The first of these penalties is the water penalty [152] (Equation 3.39)

$$W(c_{moisture}) = min(0.5 + 0.5 \times c_{moisture}, 1.0) \tag{3.39}$$

where W is the water penalty and $c_{moisture}$ is the soil moisture of the environment cell. The temperature penalty takes the form of two penalty functions (Equations 3.40 and 3.40).

$$T1(x) = 0.8 + 0.02 \times x_{opt} - 0.0005 \times (x_{opt})^2 \tag{3.40}$$

$$T2(x) = \frac{1.1814}{(1 + exp(0.2 \times (x_{opt} - 10 - x)))/(1 + exp(0.3 \times (-x_{opt} - 10 + x)))} \tag{3.41}$$

Where x is the average monthly temperature, x_{opt} is average monthly temperature where the normalized vegetation density index (NVDI) is maximized. Field et al. [154] explain these equations in detail but, in short, Equation 3.40 describes temperature penalties for environments with extreme (hot and cold) average temperatures (close to $0°C$ or $40°C$) and Equation 3.41 describes growth penalties for environments with large seasonal temperature swings. The next penalty type is the slope penalty which just queries the cell's slope which is a value taken directly form the slopemap depicted in Figure 3.17.

Once the growth penalties have been calculated, their values are stored in the environments resources cells (See Algorithm 10, line 7).

3.4.3.4 Resource Acquisition System

The Resource Acquisition System (See Algorithm 11) is responsible for managing the agent's FARM and FORAGE actions. Every iteration, the Resource Acquisition System will iterate over every household and determine how many times they want to farm by calling *get_num_to_farm* (Algorithm 5 for *traditional* and *rb-adaptive* agents and Algorithm 7 for *utility* and *IE* agents).

If a household would farm and it does not own any land, it will try to claim some by looking for any unoccupied environmental cells neighbouring its settlements. The exact amount is determined by how many cells the agent wants to farm at a given iteration. For every FARM action, an agent will randomly select one of its cells to farm at.

This process is repeated for every FORAGE action except that the agent will look for cells that contain the most resources in its vicinity and take resources directly from them. After all actions have been taken, the agent will add its newly acquired resources to its resource coffers. Lastly, if the agent is a *Utility* or *IE* agent, its utility values will be updated in accordance with Equation 3.27[19].

The amount of resources returned by these actions are determined by multiplying the amount of *resources* available at the cell (the penalty factor discussed in Section 3.4.3.3) multiplied by the *forage production rate* and *farm production rate* input parameters for FORAGE and FARM actions respectively.

[19]The reward R for each action is the average of the total resources returned by said action. This ensures that agent do not overvalue an action just because they executed more than the other.

Algorithm 11: Pseudocode for the Resource Acquisition System.

```
def resource_acquisition_system():
    forall h : Household in Households do
        num_choices = ceil(h.able_workers() / farms_per_patch)
        num_to_farm = get_num_to_farm(h, num_choices)
        total_farm = 0.0
        total_forage = 0.0
        # Check if household needs farmland
        if len(h.owned_land) < num_to_farm then
            acquire_farmland(h, num_to_farm)
        owned_land = [x for x in h.owned_land]
        for i in range(num_to_farm) do
            to_farm = random_select(owned_land)
            owned_land.remove(to_farm)
            total_farm += farm(to_farm)
        end
        # In practice, this is only done if the agent is going forage because
        # it is computationally expensive.
        candidates = get_all_near_vegetation_cells(h)
        sort(candidates)
        for i in range(num_choices - num_to_farm) do
            total_forage += forage(candidates.pop())
        end
        h.resources += total_farm + total_forage
        # Update Utility Values
        if type(h) == Utility or type(h) == IEAgent then
            total_forage = total_forage / (num_choices - num_to_farm)
            total_farm = total_farm / num_to_farm
            # See Equation 3.27
            h.forage_utility += h.stubbornness * (total_forage - h.forage_utility)
            h.farm_utility += h.stubbornness * (total_farm - h.farm_utility)
    end
    return
```

3.4.3.5 Resource Transfer System

Once resource acquisition is complete, agents determine if they have enough resources to satisfy their needs for the iteration (See Algorithm 12). This is done by the Resource Transfer System. If an agent does not have enough resources, the agent asks its authority agents if they would be willing to give some of their excess resources to it as a donation. For each authority asked, a random value $\in [0, 1]$ is generated and compared to the authority agent's *subordinate_transfer* property. If the generated value is less than the *subordinate_transfer* property, the authority agent is willing to grant donations for that iteration. Whenever a donation is granted, the authority agent has its *load* property

Algorithm 12: Pseudocode for the Resource Transfer System.

```
def transfer_resources(recipient : Household, donor : Household,
 resources_required : float):
    resources_granted = request_resources(recipient, donor, resources_required)
    recipient.resources += resources_granted
    donor.load += resources_granted
    return resources_granted
def ask_for_resources(h : Household, required_resources : float):
    forall auth : Household in get_authorities(h) do
        required_resources -= transfer_resources(h, auth, resources_required)
        if required_resources == 0 then
            return
    end
    forall peer : Household in get_peers(h) do
        required_resources -= transfer_resources(h, peer, resources_required)
        if required_resources == 0 then
            return
    end
    forall sub : Household in get_subordinates(h) do
        required_resources -= transfer_resources(h, sub, resources_required)
        if required_resources == 0 then
            return
    end
    return
def resource_transfer_system():
    forall h : Household in Households do
        surplus = h.resources - h.required_resources()
        if surplus < 0 then
            ask_for_resources(h, surplus)
    end
    return
```

increased by the resources donated. If an agent has asked all of its authority agents for resources and it will still go hungry, it then repeats this process for its peer relationships with the donating agent using its *peer_transfer* property to determine if the donation succeeds. If that is still not sufficient, the agent will then ask all of its subordinates for resources.

One limitation we identified with equivalent models when developing *NeoCOOP* pertains to their tendency to assume that resource sharing beliefs were uniform regardless of the two interacting agents. Inspired by Chliaoutakis and Chalkiadakis [25]'s dynamic organizational schemes, the *peer_transfer*, *subordinate_transfer* and *authority_transfer* properties allow us to simulate different agent-types that exhibit varying degrees of altruistic and selfish behaviour. For example, one might envision a truly altruistic agent

Algorithm 13: Pseudocode for Resource Consumption System.

```
def required_resources(h : Household):
    res_req = 0.0
    forall o : Occupant in h do
        if o.age >= age_of_maturity then
            res_req += consumption_rate
        else
            res_req += child_factor * consumption_rate
    end
    return res_req
def resource_consumption_system():
    forall h : Household in Households do
        h.hunger = min(h.resources / required_resources(h), 1.0)
        h.resources -= max(0.0, h.resources - required_resources(h))
        h.satisfaction_buffer.append(hunger) # Used to calculate satisfaction later
    end
    return
```

to be willing to share its excess resources regardless of their relationship with the donee (i.e. the agent has peer, subordinate and authority transfer values close to 1.0) whereas a nepotistic agent may be willing to share excess resources with its peers and authorities but not its subordinates (i.e. the agent has peer and authority transfer values close to 1.0 but a subordinate transfer close to 0.0).

3.4.3.6 Resource Consumption System

The Resource Consumption System (See Algorithm 13) is arguably *NeoCOOP*'s simplest system. Simply put, the system will loop over every agent in the environment, consume the agent's resources and calculate their *hunger*. The agent's *hunger* is determined by calculating the ratio of resources the agent compared to how much it needs to consume (known as *required resources*). The value of required resources is determined by calculating the resource needs of the household. An optional parameter, called *child_factor*, can be used to decrease the resource requirement needs of occupants who have not reached the *age_of_maturity*. Once the household's *hunger* has been calculated, the agents will take the resources from their resource pool. *Hunger* and *resources* are capped at 1.0 and 0.0 because one cannot be more than 100% full and have less than 0 resources respectively. We also store the agent's hunger in a buffer which will be used to calculate its *satisfaction* in Algorithm 14.

Algorithm 14: Pseudocode for the Population Management System.

```
def population_management_system():
    forall h : Household in Households do
        # Reallocation Check
        if timestep != 0 and mod(timestep + 1, yrs_per_move) == 0 then
            h.satisfaction = avg(h.satisfaction_buffer)
            h.satisfaction_buffer.empty()
            if move(h) then
                relocate_household(h)
        # Population Growth
        for i in range(h.able_workers() do
            if random() < birth_rate then
                h.add_occupant()
        end
        toRemove = [ ]
        # Population Loss
        for i in range(h.occupants.size() do
            if random() * hunger < death_rate then
                toRemove.append(h.occupants[i])
            else
                h.occupants[i].age += 1
        end
        # Remove Dead Occupants
        forall o : Occupant in toRemove do
            h.occupants.remove(o)
        end
        if h.occupants.size() > carrying_capacity then
            split_household(h)
        if h.able_workers() == 0 then
            remove_household(h)
    end
    return
```

3.4.3.7 Population Management System

The Population Management System (See Algorithm 14) is responsible for managing the population growth, loss and migration. For population growth, the system will loop over every household in the environment and, for every occupant in said household, generate a random number. If that number is less than the *birth_rate* input parameter, a new occupant will be created and added to the household. For population loss, the same process occurs but the random number is multiplied by the agent's *hunger* and compared to the *death_rate* input parameter. If the calculated value is less than the *death_rate*, the occupant is removed from the household (dies). If the occupant is not

removed from the household due to this process, it ages up by 1. If a household reaches 0 *able_workers*, it will be considered abandoned and be removed from the simulation.

One point of interest with regards to this process is the fact that population growth is not affected by household *hunger* whereas population loss is. Careful consideration was given to this design decision and the motivations were as follows: Whether birth rate is affected by resource scarceness is an open question. In fact, it is more likely the case that the time required to get resources plays a greater role on birth rate than the actual resources themselves [155]. This sentiment is also echoed in research pertaining to the Neolithic transition where birth rates seemingly skyrocketed at the advent of agriculture [156]. Conversely, death rates are undoubtedly linked to resource scarceness or famines which motivates its inclusion in determining the loss of household occupants. Additionally, both parameters are entirely configurable meaning that the birth rate can be increased of decreased by the modeller should the results conflict with archaeological data.

For population migration, the Population Migration System will, every yrs_per_move iterations, query each household by generating a random number and comparing it to the agent's *satisfaction* (See Equation 3.20). If the function returns *true*, the agent will move to another settlement with the most resources. If no settlements exist that will allow the household to meets its resource requirements[20], it will instead form its own settlement at a random location in the environment.

Lastly, we will discuss the logic for the *split_household* (See Algorithm 15) function as it differs slightly depending on the type of agent being used. First, the function is called when a household reached carrying capacity. Then, for all agent types, a new household (of the same type) is created. This household is called the offspring household and it will be given half the parent's *occupants*, *resources* and *owned_land*[21]. The offspring household is not given the parent's *load* as their creation signifies the arrival of a new social figure who has to earn their non-monetary related social status.

Other properties such as *hunger* and *satisfaction* are simply copied across to the offspring household. For the *traditional*, *rb-adaptive* and *utility* agent-types, properties that make

[20]The agent will explicitly look at nearby settlements average resource level and, if that value is greater than its *required resources*, it will move to that settlement.

[21]This is a smart split function such that the distribution of able workers and child occupants is approximately equal across the two new households.

Algorithm 15: Pseudocode for the *split_household* function. Note: This code assumes that the correct agent-type is being created when *new Household()* is called.

```
def split_household(parent : Household):
    offspring_household = new Household()
    offspring_household.settlement = parent.settlement
    offspring_household.resources = parent.resources * 0.5
    parent.resources -= offspring_household.resources
    offspring_household.hunger = parent.hunger
    offspring_household.satisfaction = parent.satisfaction
    split_land(parent, offspring_household)
    split_occupants(parent, offspring_household)
    # Evolutionary Algorithm needs to operated on IEAgents
    if type(parent) == IEAgent then
        other_parent = select_other_parent(parent)
        forall p : Gene in offspring_household do
            offspring_household.p = random_select([parent.p, other_parent.p])
            if random() < mutation_rate then
                offspring_household.p = mutate_gene(offspring_household.p)
        end
    else
        forall p : Gene in offspring_household do
            offspring_household.p = parent.p
        end
    add_agent_to_environment(offspring_household)
    return
```

their genotype are also just copied across (that is, no GA is applied to them). If the agent is an *IE* agent, another parent will be selected via roulette wheel and uniform crossover will be applied to select the offspring households gene values. The other parent household will typically be from the same settlement as the original parent but, it is possible for households, from other settlements, with extremely high social status to be selected. Additionally, each gene has a chance, defined by the mutation_rate, to mutate. For all genes (excluding *Farm Utility* and *Forage Utility*) random mutation is used while Gaussian mutation is used for the *Farm Utility* and *Forage Utility* genes[22].

3.4.3.8 Information Exchange System

The Information Exchange System is the first agent specific system. When using the *IE*-agent type, the Information Exchange System will execute every *influence_frequency*. The motivation for this is that early on in development, we found that executing this

[22]This is because the minimum and maximum value for all other gene properties are known beforehand so random mutation can be used.

Algorithm 16: Pseudocode for the Information Exchange System. Note: This code assumes that a belief space has already been created for each settlement. It also represents the unoptimized version of the code because it improved readability.

```
1  def information_exchange_system():
2      influenced_belief_space =
3      normative_prob =
4      # Determine Spatial Belief Spaces for each settlement
5      forall set : Settlement in settlements do
6          set.update_belief_space()
7          other_settlements = [s for s in settlements if s.id != set.id]
8          status_array = [s.get_social_status() for s in settlements if s.id != set.id]
9          distance_array = [set.get_distance(s) for s in settlements if s.id != set.id]
10         xtent_array = xtent_distribution(status_array, distance_array, β, m)
11         if max(xtent_array) > 0.0 then
12             index = roulette_wheel(other_settlements, xtent_array)
13             influenced_belief_space[set] = other_settlement[index]
14             normative_prob[set] = set.get_social_status() / status_array[index]
15         else
16             normative_prob[set] = 1.0
17     end
18     # Influence each household probabilistically
19     forall h : Household in Households do
20         if random() < influence_rate then
21             # Domain Influence
22             if random() < mutation_rate then
23                 p = select_random_gene(h)
24                 h.p = mutate_gene(p)
25             else
26                 # Normative Knowledge Source
27                 if random() < normative_prob[h.settlement] then
28                     bs = h.settlement.belief_space
29                 # Spatial Knowledge Source
30                 else
31                     bs = influenced_belief_space[h.settlement].belief_space
32                 forall p : Gene in h do
33                     h.p = influence_gene(p, bs) (Equation 3.33)
34                 end
35     end
36     return
```

system every iteration caused rapid homogenization of the belief spaces and poorer performing agents. The reason for this phenomena is due to optimizing nature of the CA. EAs in general are used as optimization processes. In most optimization processes, the fitness of a potential candidate solution can instantly be evaluated. This is not the case in *NeoCOOP* where newer, potentially more successful, households are not given a enough time to show that they are better than other households. In fact, it was not uncommon

for a newly created or mutated household to instantly have their beliefs influenced in the original CA implementation. This is seemingly a unique problem with using optimization techniques (like ML algorithms) in ABM. This is evident in Angourakis et al [157]'s Food-for-all model where they also delayed the "optimization" process by several iterations so that an agent's beliefs could be adequately evaluated.

Belief Spaces are central to how adaptation occurs in the *IE* agents. In CA literature, a belief space is a representation of a process' best solutions. Over time the belief space is updated to include newly found solutions that are equal or better performing than previous solutions. The belief space is also what is used to influence the solution population. In *NeoCOOP*, we considered using a single global belief space but the idea was quickly abandoned because it would oversimplify the role local geography and climate may have on the agent populations beliefs. In Predynastic Egypt, agricultural practices are theorized to have been adopted asynchronously in part due to local geography (See Section 3.3.3). A global belief space could not facilitate that kind of behaviour.

Due to these limitations, we opted to create belief spaces for every settlement and instead treat adaptation over the course of a simulation as a localized optimization process with each settlement adapting to their local environmental conditions. Formally we determine a belief space's gene property as defined by Equation 3.42.

$$B_{s,t}(p) = \sum_{h \in s} \frac{h.status}{S_s} G_{h,t}(p) \tag{3.42}$$

Where $B_{s,t}$ is the belief space of settlement s at timestep t. $h.status$ is the social status of h and S_s is the sum total social status of all households in settlement s. $G_{h,t}(p)$ the value of gene property p for household h at timestep t.

While this solution allowed for local adaptation, it did not account for the fact the settlements could affect the cultural beliefs of other settlements. This feature is essential to the model. As noted in Section 3.3.3, the southern Naqada culture is theorized to have overrun the northern Maadi/Buto culture over time. To address this limitation, we adopted *Knowledge Sources* [102] (See Section 3.4.2.4) which, in traditional CA implementations, serve as isolated belief spaces which get updated and influence solutions independently. We adapt this approach by specifying three knowledge sources that settlements may be

influenced by. The first is their own belief space, this is called the Normative knowledge source, we also stochastically select another settlement's belief space as the Spatial knowledge source. We then finally add the Domain knowledge source which is just the mutation function from the GA. With these three knowledge sources, agents may be influenced by their own settlement (normative), other settlements (spatial) and other abstract means[23] (domain).

From an implementation perspective (See Algorithm 16), the Information Exchange System will first determine the spatial knowledge sources for each settlement s. This is done by first building a probability distribution using the XTENT formula (Equation 3.29) and Equation 3.30. Once the distribution has been created, roulette wheel selection is used to select which settlement's belief space will be used as the spatial knowledge source for settlement s. Additionally, the ratio of social status for s and the spatial knowledge source are calculated. We call this the *normative probability* as it describes the likelihood that s will influence itself. With this implementation, a settlement with little-to-no wealth or social status is more likely to be influenced by another settlement with greater social status [40]. This is a direct result of the XTENT formula and is what allows settlements to influence each other over time.

Once all spatial knowledge sources have been determined, the Information Exchange System will iterate over all households and determine if they should be influenced in accordance with the *influence_rate*. If they are influenced, a knowledge source it selected randomly using the *mutation_rate* and the *normative probability*. If the random value is less than the *mutation_rate*, the domain knowledge source is chosen. If the domain knowledge source is not selected, another random number is generated and if less than the *normative probability*, the normative knowledge source (the household's settlement's belief space) is selected. If the normative knowledge source is not chosen, the spatial knowledge source is chosen. This knowledge source is the previously roulette wheel selected settlement's belief space (See Algorithm 16, line 13). Lastly, the agent is influenced by the selected belief space using the Equation 3.33.

Algorithm 17: Pseudocode for the Rule-based Adaptation System.

```
def rb_adaptation_system():
    intention_mask = [0.0 for h in households]
    # Determine adaptation intention each household
    for i in range(len(households)) do
        # Calculate Risk Assessment
        severity = 1.0 - hunger
        risk_assessment = 0.6 * severity + 0.4 * rand()
        # Calculate Adaptation Appraisal
        w_age = 1.0 - 0.12 / (0.12+(min(max_age,households[i].average_age()) / max_age)^3)
        w_hh_size = 1.0 - 0.12 / (0.12+(able_workers(households[i]) / carrying_capacity)^3)
        adaptation_efficacy = 0.55 * w_age + 0.45 * w_hh_size + (0.2 - 0.3 * rand())
        w_wealth = 1.0 / (1.0+exp(-3.0*(households[i].resources / required_resources(households[i]) -0.5)))
        self_efficacy = 0.3 * w_wealth + 0.6 * households[i].percentage_to_farm + (0.1 - 0.2 * rand())
        adaptation_appraisal = clamp(0.5 * (adaptation_efficacy + self_efficacy))
        # Calculate Adaptation Intention
        r = risk_elasticity * risk_appraisal
        p = adaptation_appraisal * (1 - cognitive_bias)
        intention_mask[i] = p - r
    end
    # Update Household properties
    for i in range(len(households)) do
        if intention_mask[i] > adapt_threshold then
            forall p : Property in h do
                h.p = Equation 3.26
            end
    end
    return
```

3.4.3.9 Rule-based Adaptation System

The second optional system is the Rule-based Adaptation System (See Algorithm 17). A lot of the pseudocode presented is an exact replica of the code used in the OMOLAND-CA's model (with the exception of the update function which needed to be translated to our model). Similar to the Information Exchange System, the Rule-based Adaptation System will execute every $adapt\_frequency$ iterations. For each household, the system will calculate the household's intention to adapt just as described in Section 3.4.2.2. Once

[23] The domain knowledge source represents abstract sources of influence such as household innovation, change in leadership and foreign (outside of the modelling area) interaction.

that is complete, the system will update each of the adaptive households (households who have chosen to adapt) in accordance with Equation 3.26.

3.4.4 Limitations of NeoCOOP

NeoCOOP is, to the best of our knowledge, the most complete realization of any ABM modelling the rise of the Ancient Egyptian State during the Predynastic period. However, there are several limitations that we wish to discuss. The first of these issues is the necessity to abstract certain processes to achieve parity with the time-scale of the model. The agricultural revolution of the Predynastic took thousands of years while vegetation models and forager behaviour dynamics operate on a time-scale of days or months. It was not computationally feasible for the model to operate on such a scale so both forager and vegetation model dynamics had to be abstracted to operate on a yearly scale. Future versions of *NeoCOOP* can look at making these processes time-scale agnostic.

The second limitation is the general lack of empirical data available to tune the model with. We do not have satellite images, exact temperature or flood data from that time period so all resource availability dynamics are best guess estimates based on data from present times. Despite this, Section 4.3 shows that the implemented vegetation model still produces acceptable results.

Third, all models must at some point resort to simplification, *NeoCOOP* is no exception. These implemented processes are often more detailed than related models' equivalent processes but they are still extremely simple. For one, climate data is applied over the entire modelling space (as opposed to regional climate data), agents acquire one type of resource that generically represents all forage and farming food stuffs (at different yields depending on the action used). This directly ties into the assumptions with agent decision making. For one, the *traditional* agents assume that adopting farming is guaranteed in all circumstances. The *utility* agents assume that maximizing utility at all costs is the best course of action. It also assumes that this particular type of decision making is appropriate when modelling Predynastic Egyptians.

Lastly, we do want to note that despite these limitations, we still believe the results our model will produce are both useful and valid. As noted before, *NeoCOOP* is already more "realistic" than most other archaeological ABM and, with parameter tuning, has

the potential to provide insight into the emergence of the Ancient Egypt State that would not be attainable from the archaeological record alone.

3.5 Summary

In this Chapter we sought to highlight the methodological approach we applied when designing the Neolithic Cooperation Model (*NeoCOOP*), an ABM capable of modelling the emergence of the Ancient Egyptian State during the Predynastic period.

We first described *ECAgent*, an entity-component-system (ECS) based modelling framework we designed for implementing ABM in *Python*. We highlighted the natural compatibility ECS has with ABM development and demonstrated, by implementing a simple predator-prey model, the ease at which models could be implemented with *ECAgent*. Despite the lack of features, *ECAgent* shows great potential as an ABM development framework especially when considering that using Python gives developers access to some of the most robust software packages available (*Numpy*, *Pandas* and *Pytorch / Keras* are just some examples).

We then provided an extensive overview of what adaptability or adaptive capacity means in the context of agent-based modelling. We found that ABM literature seemingly conflates the definitions of adaptability, resilience and transformability as just "adaptability". Additionally, we noted that no formal process exists for creating truly adaptive agents. Describing adaptive agents as entities who maintain state information, are capable of updating their state when they receive new information and have a decision making process that changes when their state changes, we highlighted how creating such an agent could be done by ensuring that the agent was connected in an information exchange network. We demonstrated the effectiveness of this approach by implementing a simple foraging ant simulator and illustrated how more complex information exchange networks could create agents with greater adaptive capacity.

This chapter also synthesized literature regarding the formation of Ancient Egypt. The main findings suggest that the emergence of the ancient state during the Predynastic period was due to a mixture of both natural and social factors with the social factors resulting from the natural factors. These factors included the presence of Nile floodplain and desertification of the surrounding areas. We used these findings to construct a set

of requirements some hypothetical ABM would need to implement in order to effectively simulate the Predynastic period.

Using these requirements, we constructed *NeoCOOP*, and iteration-based ABM implemented using *ECAgent*. Incorporating a vegetation model, four agent-types and several additional systems, *NeoCOOP* was designed to answer the research questions we stated in Chapter 1. The central focus of the ABM are the four agent-types. The *traditional* agents that are simple, rule-based, agents akin to those found in early ABMs. They are not adaptive but serve as a benchmark for to compare the other agents to. Taking the rule-based paradigm further, the *rb-adaptive* agents, based on Hailegiorgis et al. [147]'s OMOLAND-CA model, are a more accurate representation of modern, adaptive rule-based agents. We then shifted our designs to ML-based agents and designed the *utility* agent which just uses a simple RL algorithm to aid in creating resource acquisition strategies. We then extended the *utility* agent to create the *IE* agents who are capable of exchanging information which each other using two Evolutionary Algorithms.

In the next chapter, we will detail how we will use *NeoCOOP* to study the adaptability of the four agent-types.

Chapter 4

Experiments and Results

In this Chapter, we to aim to highlight the experiments used to evaluate the following research questions stated in Chapter 1:

1. Do agents using machine learning techniques as adaptive mechanisms, exhibit greater adaptive capacity (recovery and resistance of population and resource levels) [33] than traditional, rule-based, agents when placed into sufficiently adversarial environments (A model of Egypt during the Predynastic period)?

2. Are our adaptive-agents capable of producing new emergent behaviour (such as polity cycling [34] or strategy specialization [35]) that the traditional, rule-based, agents could not?

3. Using both the traditional and adaptive-agents, what insights do they provide with regards to the Predynastic agricultural revolution as it relates to the presence of the Nile floodplain [12] and desertification [36]?

To achieve this, we conduct scenario experimentation using the *NeoCOOP* ABM introduced in Chapter 3. By altering the amount of yearly rainfall, temperature and Nile flood height at each timestep in the model, we will be able to virtually recreate the Predynastic Egyptian landscape which each of the four agent-types will have to inhabit. We then monitor how the agents adapt to these changing environmental conditions and, by measuring the population and resource levels of the agents, we can get an indication of which agent-types are most adaptive.

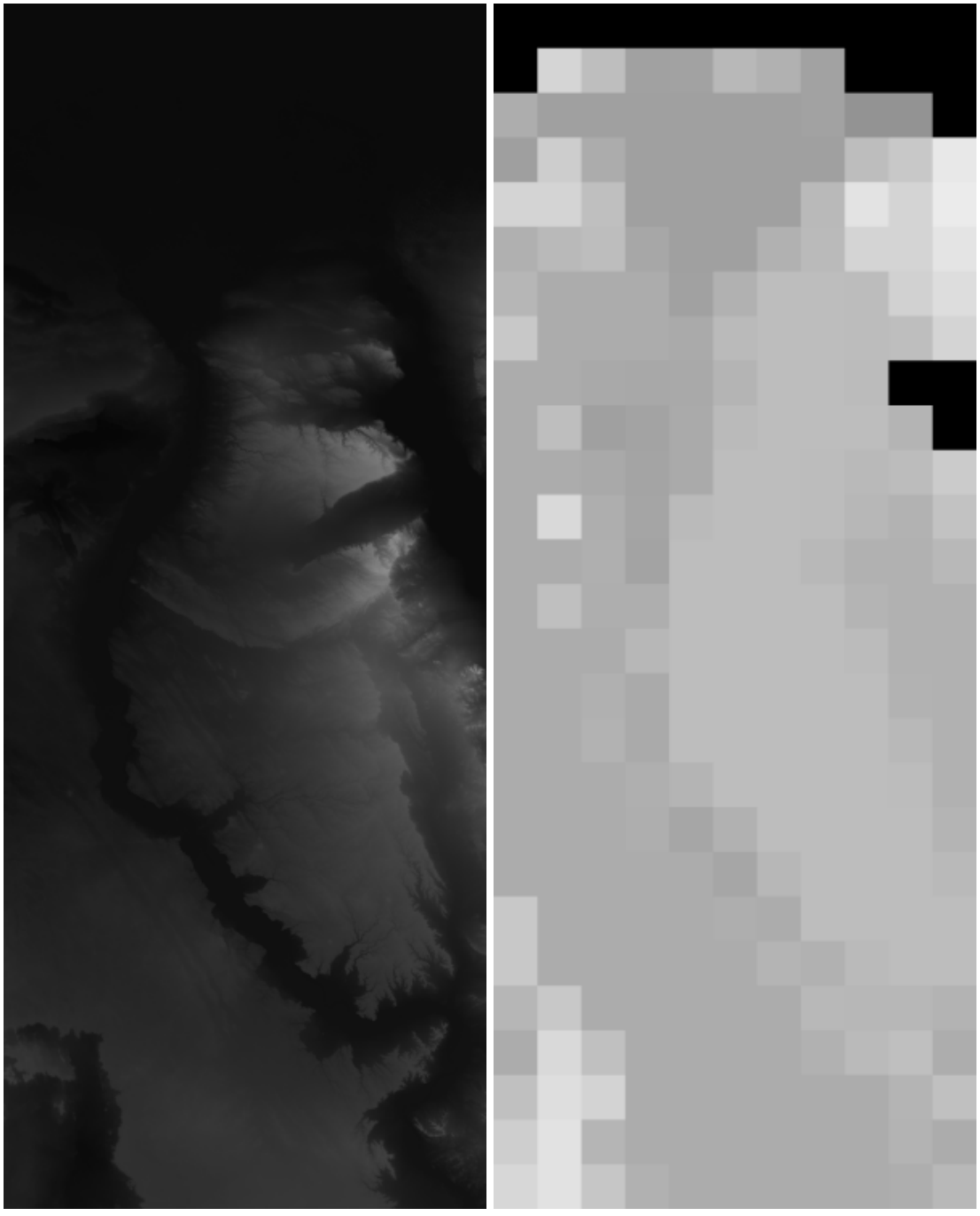
The Chapter proceeds as follows: Section 4.1 summarizes the GIS and Archaeological data used in the model. Section 4.2 describes how we generated approximate climate data for the Egyptian Predynastic, Section 4.3 details how we validated model behaviour. Section 4.4 highlights the parameter tuning process for the agents as well as the final input parameters used in our experiments. Section 4.5 contains the results and statistical evaluation of our experiments. Lastly, Section 4.6 analyzes the results of our experiments and Section 4.7 concludes the Chapter with a summary.

4.1 Data Acquisition

Given our goal to model Predynastic Egypt as accurately as possible, several outside data sources were utilized. The difficulty of this process pertained to the unavailability of several of the input parameters from an Archaeological context (The GIS layers most notably). To combat this, we extended our data acquisition sources to other Neolithic communities and, when insufficient still, we utilized modern analogs.

The first of the input data parameters are the GIS layers. Specifically the height and soil content maps. This data does not exist in an Archaeological context and as such, we utilized contemporary data instead. The heightmap data was sourced from the ALOS Global Digital Surface Model [158] and the soil-clay content maps were sourced from the GLDAS project [159]. Data from both sources were pre-processed into a NeoCOOP friendly format, the details of which may be viewed in Appendix C. These input images can also be seen in Figure 4.1.

The next step of the data acquisition process was to determine the scale of the simulations. The population of Egypt at the time is said to have been approximately 350 thousand [36]. Using Lehner's [143] description of the household unit, we may assume a average household size of around 5 - 10 people given that it may or may not have included extended family. This would require us to instantiate between 35 - 70 thousand agents. Similarly, a 1ha resolution (1 pixel = 1ha) datamap would require a modelling area of 3000×7500 pixels for a total of 22500000 environmental cells. Running a simulation of that size is not feasible simply due to the time it would take to complete a single run. Because we are interested in the entire modelling area (Both Upper and Lower Egypt), we opted to reduce the environment's resolution and number of agents by a constant



(A) (B)

FIGURE 4.1: The processed height (a) and sand content (b) datamaps used in our experiments. The sand / clay content data was not available in a higher resolution so it is noticeably less accurate than the heightmap.

factor. Determining the factor by which to reduce the scale of the simulation was not a straightforward process. We considered using an arbitrary value, but we felt it would detract the overall value of the model and left room for valid criticism. We elected to to use Allen's [36] estimation of the population carrying capacity of both foraging and agricultural practices in Ancient Egypt. Allen states that nomadic hunter-gatherer lifestyle would have only sustained about 30 persons per square kilometer. This ratio is particularly useful because it describes both the spatial and population factors. These values are similarly echoed by Gallagher [160] who studied Neolithic communities globally.

Using a square kilometer (100ha) resolution and a reduction in population by a factor of 30, we only need to instantiate 1610 households which the model can easily handle. Allen and Gallagher also approximate that farming could sustain populations up to approximately 4 times that of a purely foraging one. This further provided us with data on the production quantities of both the foraging and farming actions respectively. Additionally, Gallagher notes that hunter-gatherer groups consisted of, on average, 30 individuals. Given this, we can assign an initial average settlement density of 3 households across 573 initial settlements.

These abstractions of the agent population are not ideal, but it is not without precedent. Simplifications of the problem domain are commonplace in ABM research and by ensuring that all the resource and distance related properties are scaled correctly, the model should still be able to produce valid results.

4.2 Climate Data Generation

Given that accurate climate data of the Predynastic period does not exist, we constructed *interpolator* functions which would generate the data instead. The interpolators (See Equation 4.1) work by defining minimum and maximum ranges and a mixing parameter. The mixing parameter $x \in [0.0, 1.0]$ is derived from various mathematical functions and when used with the min and max ranges, produces two values l_{min} and l_{max}. A random number is then generated between $[l_{min}, l_{max}]$ and set as the generated value for the global environment property for a single iteration. For rainfall and temperature, min and max ranges were provided by the World Meteorological Organization (WMO). For Nile flood height, we use values stated by Bell [161].

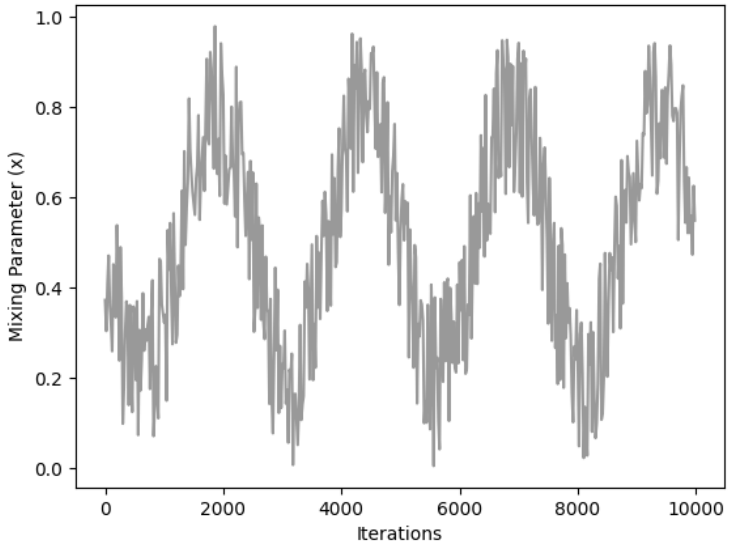


FIGURE 4.2: An example of how the mixing parameter x is generated over the course of a simulation run for a single vegetation model property (temperature, rainfall, flood height). In this example, $f = 2500$.

$$lerp(a, b, x) = a + x(b - a) \text{ where } a \leq b \tag{4.1}$$

As for the interpolators themselves, several were investigated during the early development of *NeoCOOP* (See Appendix D). Interesting results were obtained from these experiments but they are not the primary focus of this work. What these experiments did reveal though was that while a simple linear interpolation from a start range to an end range of values was sufficient at reducing the global rainfall over the course of a simulation run, it would not be able to accurately reflect the flood dynamics of the Nile river. Hence, a sinusoid (See Equation 4.2) was used instead.

$$s(t) = 0.5 + 0.5sin(\frac{2\pi \times t}{f}) \tag{4.2}$$

Where t is the timestep with and f is period of the sinusoid. The motivation for using a sinusoid relates to the cyclical nature in which environments experience periods of stress and abundance. Egypt is no different with Hassan [162] illustrating that this was certainly the case during the formation of Ancient Egypt which would have undergone several dry and wet seasons. The pseudocode for this process can be seen in Algorithm 18 and example of the output generated by the process can be seen in Figure 4.2.

Algorithm 18: Pseudocode detailing how climate data (rainfall, temperature and flood height) was generated for the simulations investigated in this work. Here $func$ refers to type of functor that takes in the timestep t as input and returns a value $\in [0.0, 1.0]$. The sinusoid function defined in Equation 4.2 is one such function.

```
1 def interpolator(t : int, min_range : (float, float), max_range : (float, float), func):
2     x = func(t)
3     lower = lerp(min_range[0], min_range[1], x)
4     upper = lerp(max_range[0], max_range[1], x)
5     return random(lower, upper)
```

4.3 Validation

ABM are notorious for being unpredictable and hard to debug. To combat this, both *ECAgent* and *NeoCOOP* were unit tested to ensure the validity of the results produced in this work. Code coverage reports were generated to ensure testing completeness and can be seen in Appendix B.

Additionally, reviews of the outputs of several hundred simulation runs were performed over the course of *NeoCOOP*'s development. No formal review methodology was used but the process consisted of looking at event logs, generated graphs and animations to ensure correct model behaviour. Similarly, the model was stress tested and optimized several times during its development.

4.4 Parameter Tuning and Experiment Setup

The last step before running our experiments was the parameter tuning of the agent-types. Parameter tuning is a interesting topic in social simulation research. On the one hand, it is an essential step in the development of ABM but on the other, the metrics used to find the optimal parameter set(s) can dramatically affect the results produced by the tuned model. For example, choosing to parameter tune a set of resource sharing agents based on their ability to accrue surplus resources would inherently bias them towards selfish behaviour whereas tuning them to minimize community hunger may bias them towards altruistic behaviour. Alternatively, sensitivity analysis could be used to better understand the model's behaviour but this process is complicated for ABMs and scales poorly as the number of parameters under consideration increases [163].

Name	Value	Comment
	Traditional	
forage gradient	0.68	
forage offset	0.0	
forage duration	2500	Same as N (See Table 4.2)
forage margin	0.23	
	RBAdaptive	
risk elasticity	0.4	From OMOLANDCA [147]
cognitive bias	0.3	From OMOLANDCA [147]
adaptation intention threshold	0.05	
learning rate	0.2	
	Utility	
learning rate range	[0.4, 0.6]	
	IE	
learning rate range	[0.3, 0.7]	
conformity range	[0.3, 0.7]	
influence rate	0.83	
influence frequency	25	iterations
b	2.0	Adjusted from Chliaoutakis
m	0.005	and Chalkiadakis [40]

TABLE 4.1: Initialization parameters of each agent-type.

We chose to parameter tune our model using *Optuna*, a Python-based optimization framework[1]. Using *Optuna*, we performed multi-objective optimization on each agent type. The metrics optimized were the population and resources levels of the entire population at the end of a simulation run given that we had previously identified these metrics as indicators of generalized success within the agent population. The fact that both total surplus resources and population levels were optimized simultaneously also prevents the agents from biasing their behaviour in one direction that would dramatically affect the other.

Each agent-type was optimized independently over 50 separate simulation runs (for 150 total). The exception to this is the *RBAdaptive* agent where we first tested the model using the same values used in the OMOLANDCA model (See Section 3.4.2.2) and then adjusted the values after the fact to account for the differences between the two models. The final results of this process can be seen in Table 4.1.

With the parameter tuning complete, we setup our experiments as follows: For each agent-type (*Traditional, Utility, RBAdaptive and IE)*, a simulation of length $N = 2500$ is run (The length of the Predynastic period with buffer zones from the Neolithic and

[1]Optuna is available at: `https://optuna.org/`

Name	Value	Comment
N	2500	iterations
Initial Agents	1610	See Section 4.1
Initial Settlements	537	See Section 4.1
Birth Rate	0.1%	Derived from Allen [36]
Death Rate	0.01%	Derived from Allen [36]
Forage Production Rate	1.0	Abstract food unit
Farm Production Rate	4.0	See Section 4.1
Agent Consumption Rate	1.0	Same as Forage Production Rate
Load Difference (L)	0.6	Chliaoutakis and Chalkiadakis [25]

TABLE 4.2: Initialization parameters for experiments evaluated in this work.

First Dynastic periods). Each simulation is initialized with the properties discussed in Sections 4.1 and 4.2. 537 settlements were assigned to the environment at different locations and 1610 Agents (of whichever type of agent is being investigated) were then randomly assigned to these settlements.

At initialization, agents are randomly assigned values $\in [0.0, 1.0]$ for each of their peer and subordinate resource transfer and attachment properties. Authority transfer was fixed to 1.0 to ensure that authority figures would always be granted resource transfer requests when asking the subordinates. The motivation being that an authority figure would have the means to coerce a subordinate agent to give up their surplus resources. For the *Utility* and *IE* agents, conformity and learning rates were randomly assigned values in accordance with the ranges stipulated in Table 4.1. Forage and Farm utility were set to 1.0 and 0.0 respectively. This is because the agents start out as hunter-gatherers and need discover the utility value associated with farming organically over the course of a simulation run.

Given the stochastic nature of the model, 50 uniquely seeded simulations were run for each agent-type (totalling 200 simulation runs across all agent types). Additionally, the same *seed* values are used across these agent-types such that given some seed value, the same initial settlement distributions and climate conditions will be pseudo-randomly generated regardless of agent-type, ensuring that each agent-type is fairly evaluated. A summary of these initialization parameters have been included in Table 4.2[2].

[2]Several parameters have been omitted from Table 4.2 due to their irrelevancy in this work. Given the macro scale of the simulation area, micro properties such as household carrying capacity and individual maturity were disabled.

4.5 Results

Kruskal: 4.59×10^{-36}

	Traditional	Utility	IE	RBAdaptive
Traditional	–	5.38×10^{-27}	1.70×10^{-25}	1.28×10^{-4}
Utility	5.38×10^{-27}	–	1.00	1.51×10^{-10}
IE	1.70×10^{-25}	1.00	–	1.25×10^{-9}
RBAdaptive	1.28×10^{-4}	1.51×10^{-10}	1.25×10^{-9}	–

TABLE 4.3: Summary of the Posthoc Dunn Test ($p = 0.05$) performed on the Total Household Population for each Agent Type. Significant differences have been highlighted in grey.

Kruskal: 3.52×10^{-40}

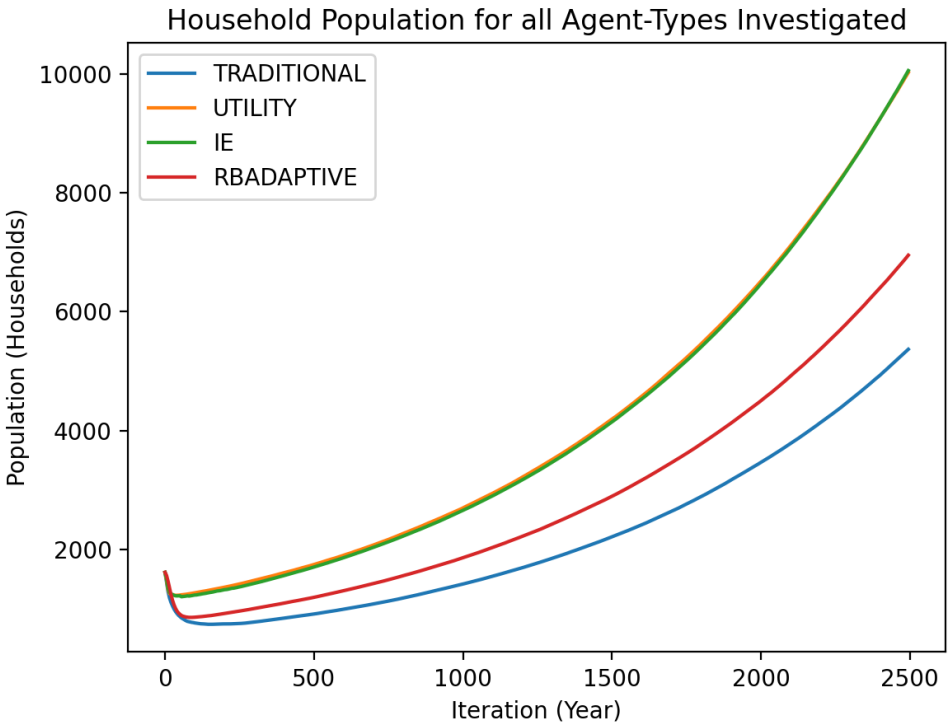
	Traditional	Utility	IE	RBAdaptive
Traditional	–	9.38×10^{-5}	3.26×10^{-17}	1.36×10^{-37}
Utility	9.39×10^{-5}	–	9.17×10^{-5}	3.57×10^{-17}
IE	3.26×10^{-17}	9.17×10^{-5}	–	9.84×10^{-5}
RBAdaptive	1.36×10^{-37}	3.57×10^{-17}	9.84×10^{-5}	–

TABLE 4.4: Summary of the Posthoc Dunn Test ($p = 0.05$) performed on the Total Surplus Resources for each Agent Type. Significant differences have been highlighted in grey.

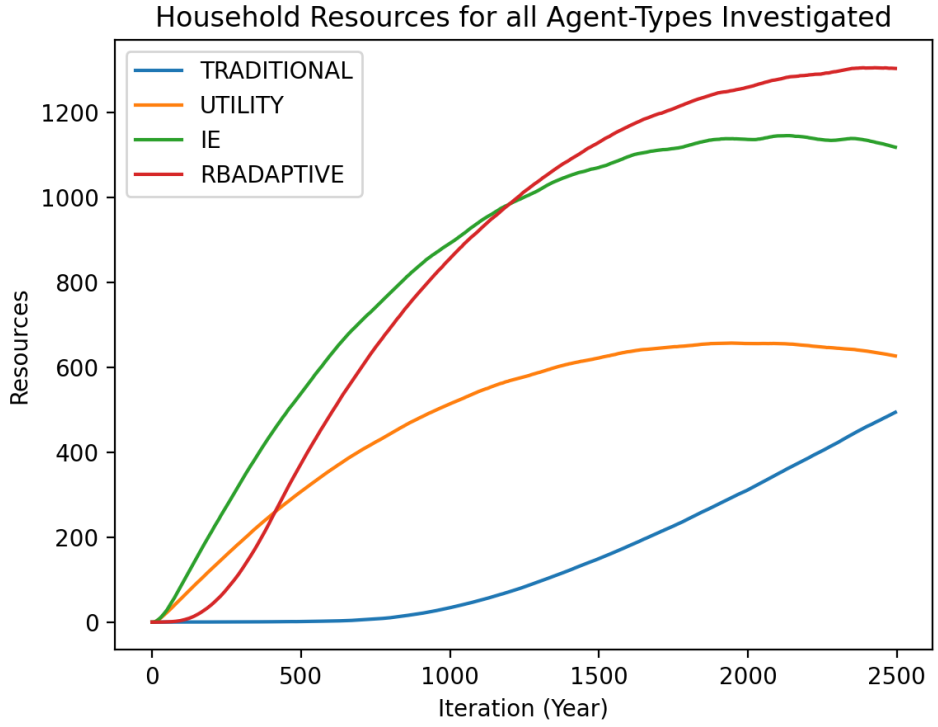
To compare the adaptive capacity of the agent-types, their population levels and surplus resources were compared. As seen in Figure 4.3a, a visual distinction between the ML agents (*IE* and *Utility*) and the rule-based agents (*Traditional* and *RBAdaptive*) can be seen. Conversely, Figure 4.3b demonstrates a clear visual distinction between the agents with information sharing capabilities (*IE* and *RBAdaptive*) and those without (*Traditional* and *Utility*).

To confirm this, Kruskal-Wallis tests ($p = 0.05$) were performed on both the population and surplus resources of each agent type. Both tests revealed significant differences between the agent-types and a posthoc Dunn test ($p = 0.05$) with Bonferroni correction was performed to identify which differences between the agent pairs were significant. The results of these tests are reported in Tables 4.3 and 4.4.

The results of the Dunn test revealed that for total Household population, the *IE* and *Utility* agents did not have significantly different populations but, they produced populations significantly greater than the population of the *RBAdaptive* and *Traditional* agents. Additionally, the *RBAdaptive* had significantly greater population levels than the *Traditional* agents.



(A)



(B)

FIGURE 4.3: Plots of the average total agent population (a) and surplus resources of the entire agent population (b) for all agent-types investigated.

For total surplus resources, the *RBAdaptive* agents gathered significantly more surplus resources than all other agent-types. The *IE* agents gathered significantly more surplus resources than the *Utility* and *Traditional* agents and the *Utility* more surplus resources than the *Traditional* agents.

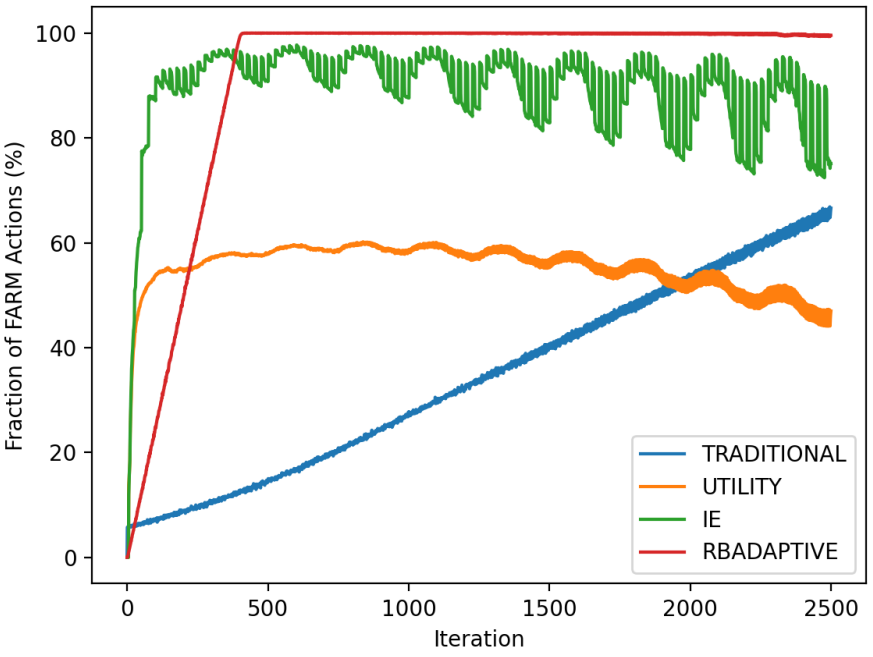
Agent-Type	Population	Resources	Final
IE	1	2	1
Utility	1	3	2
RBAdaptive	3	1	2
Traditional	4	4	4

TABLE 4.5: Final adaptability rankings of the agents based on their ability to maintain and increase both population and surplus resource levels across a simulation run. For example, a rank of 1 means that the agent ranked first in that metric.

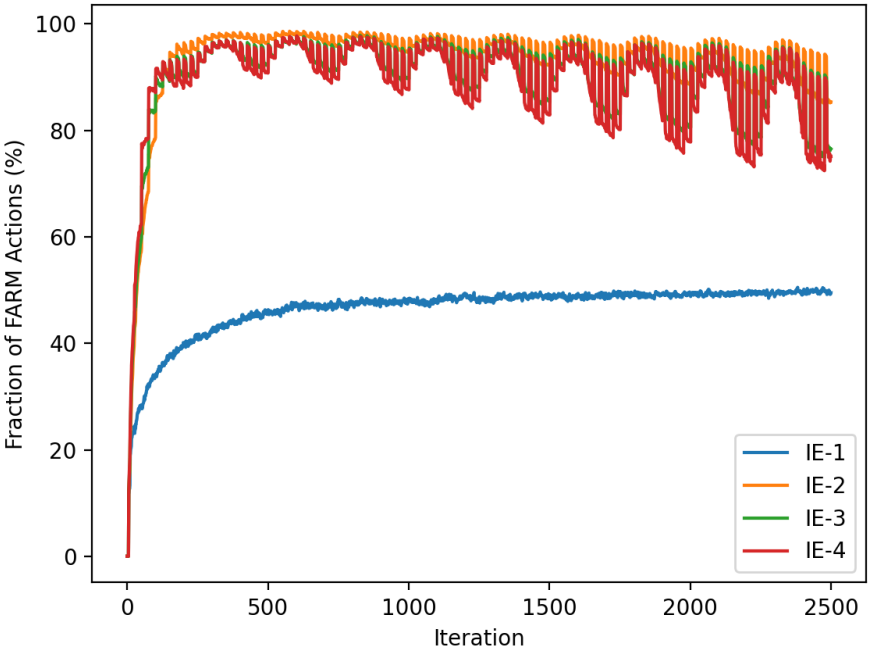
Finally, we ranked the agents based on their performance across both metrics investigated. The results of this ranking are reported in Table 4.5. Unsurprisingly, the *Traditional* agents performed the worst, ranking last across both metrics. The *Utility* and *RBAdaptive* agents were jointly ranked in second due to both of them scoring a first and a third place. Lastly, the *IE* agents were ranked first and, by the metrics investigated in this work (total population and resource levels), are considered to have the greatest adaptive capacity out of the agent-types investigated.

4.6 Analysis

In this Section, we seek to explain the underlying dynamics that produced the results reported in Section 4.5. We will do this by analyzing the emergent behaviours of the agent-types. In particular, Section 4.6.1 analyzes the rate at which agricultural practices are adopted across the agent-types, Section 4.6.2 reviews settlement density and population migration dynamics and Section 4.6.3 details the importance information throughput played on the adaptive-capacity of the ML agent-types. Lastly, 4.6.4 synthesizes these findings and highlights how they relate to the formation of Ancient Egypt during the Predynastic period.



(A)



(B)

FIGURE 4.4: Percentage of FARM actions performed by the original agent-types (a) and the supplementary experiments run for the *IE* agent-type (b). Note: Figure (b) is labelled such that *IE-N* indicates that the *IE* agent-type was used with a *farm_production_rate* of N.

4.6.1 Rate of Emergent Agricultural Practices

An examination of the complementary data generated by the simulations revealed the importance the rate at which the agent-types adopt farming as the primary resource acquisition strategy has on their final population level. Figure 4.4a demonstrates this whereby it can be seen that ML agents (*Utility* and *IE)*) rapidly adopt agricultural practices while the rule-based agents take longer. This coincides with the magnitude of their final population levels whereby our results suggest that rapid adoption of farming positively affects the final population levels exhibited by a group of agents. Additionally, Figure 4.4a reveals that the *Utility* agents, while rapid in their initial adoption of agricultural activities, fail to make farming their dominant resource acquisition strategy. Conversely, farming does emerge as the dominant strategy in the other agent-types with the *RBAdaptive* agents completely adopting farming while the *IE* agents oscillating above 75% adoption. Interestingly, the results obtained by the *Utility* agents suggest that this adoption need not be dominating. In fact, the results indicate that the rate of adoption is only important for agents who are at an increased risk of dying due to increased environmental stress.

Kruskal: 9.88×10^{-19}

	IE-1	IE-2	IE-3	IE-4
IE-1	–	0.0124	2.54×10^{-10}	4.62×10^{-17}
IE-2	0.0124	–	2.64×10^{-3}	2.0×10^{-7}
IE-3	2.54×10^{-10}	2.64×10^{-3}	–	0.27
IE-4	4.62×10^{-17}	2.0×10^{-7}	0.27	–

TABLE 4.6: Summary of the Posthoc Dunn Test ($p = 0.05$) performed on the total Household population for each *farm production rate* supplementary experiment. Significant differences have been highlighted in grey.

Given that Utility maximization is the central driver of the ML agents, we first assumed that the rapid adoption of farming was due to the resource production rate of farming when compared to foraging. To further investigate this claim, supplementary experiments were run using the *IE* agent whereby we altered the rate of production produced by the FARM action. The exact values used were $farm\_production\_rate = [1.0, 2.0, 3.0, 4.0]$ where 1.0 indicates that the FARM and FORAGE actions return the same amount of resources on average and 4.0 is the value used in the original experiments. Figure 4.4b reveals that even when there is a slight surplus in agricultural production, it will rapidly emerge as the dominant strategy. Additionally, Figure 4.5 shows that the rate

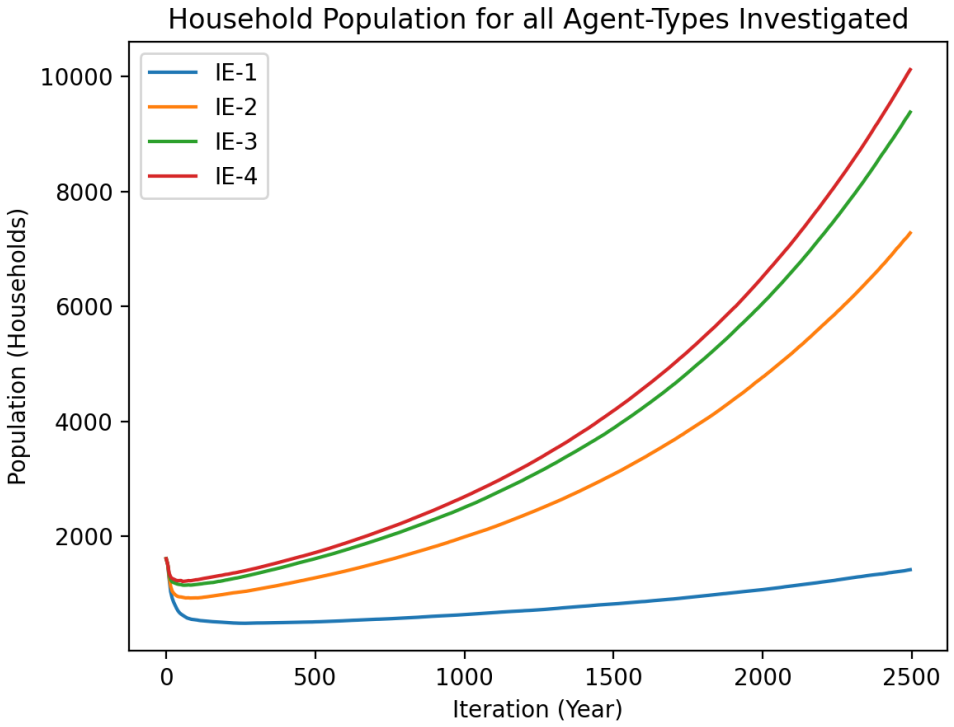


FIGURE 4.5: Household population levels for farming production rate supplementary experiments. Note: The Figure is labelled such that *IE-N* indicates that the *IE* agent-type was used with a *farm_production_rate* of N.

of agricultural adoption significantly (See Table 4.6) affects final Household population levels, albeit with diminishing returns. Interestingly, when the production rate of foraging and farming are the same, agricultural communities will still emerge. This supports Gallagher's results were she showed that the production rate of farming may not be the only reason why agricultural communities emerged throughout Neolithic communities across the globe [160].

The ML agents failure to fully adopt farming is the reason they didn't accrue the most amount of resources across the simulation runs. The *RBAdaptive* agents may have taken longer to transition from foraging to farming, but they consistently maintained a close to 100% adoption rate once it was reached. This, coupled with the period (approximately 2000 iterations) over which they maintained this behaviour meant that they could accrue more surplus resources. Oscillations are also observed in the agricultural adoption of the ML agents. This emergent behaviour coincides with periods of environmental stress. During periods of stress the ML agents will explore the non-dominant method of resource

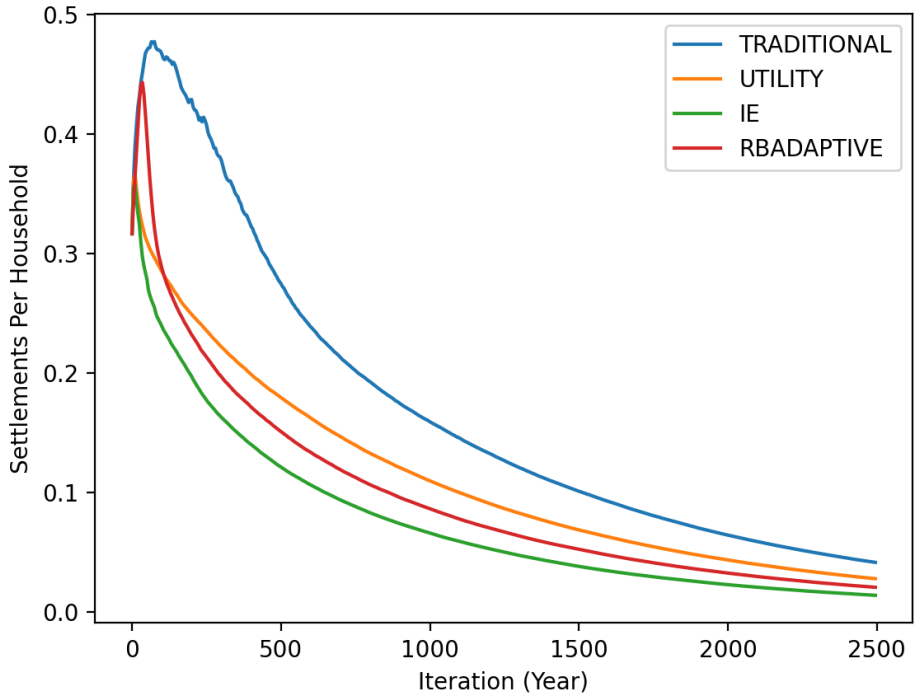
acquisition (foraging in the case) because of the how their exploration-exploitation heuristic works (See Section 3.4.2). The magnitude of these oscillations are also increasing. This is discussed more in Section 4.6.2 but the reason for this behaviour is related to settlement density and the agent-type's significantly greater populations. This is further supported by Figure 4.4b where the lower Household population scenario's (*IE-1, IE-2 and IE-3*) show similar oscillations at reduced magnitudes.

The *Utility* agent-type's failure to adopt farming as the primary resource acquisition strategy is interesting. For one, it suggests that the adoption of agricultural practices by Predynastic Egyptians cannot be solely explained by environmental stress. This is discussed more in Section 4.6.4 but there are regions (the Delta specifically) which did not suffer from the increasingly arid conditions surrounding the Nile Valley. The *Utility* agents in these regions simply do not adopt farming because they have no need to. This behaviour is also an indication that cultural influence from other settlements or households is needed to convince these foraging agents that farming is better. This is demonstrated by the *IE* agents which adopt farming to a much greater degree (See Figure 4.4a). The only difference between the two ML agents is the *IE* agents' ability to exchange information among themselves. Our results suggest that this cultural information exchange is key to explaining the widespread adoption of agricultural practices across all of Predynastic Egypt.

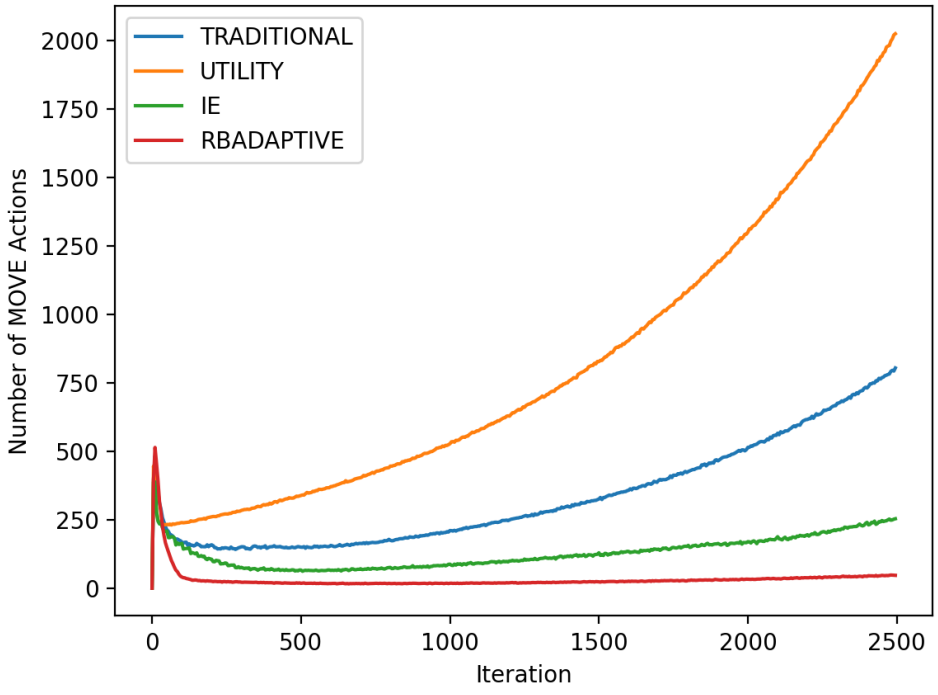
Lastly, it is appropriate to indicate that the rate of adoption observed in the ML agents is so fast ($\sim$ 50 iterations) that it is may not necessarily be realistic. This is discussed further in Sections 4.6.4, but this phenomena is mostly attributed to the simplicity of farming and the fact that no learning phase needs to occur to reap maximum rewards.

4.6.2 Settlement Density and Population Migration

Unsurprisingly, our results indicate that an increase in agricultural practices negatively correlates with population migration. As shown in Figure 4.6b, the *Utility* agents are the most mobile with one-fifth of the entire population moving on average towards the end of a typical simulation run. They are also the agent-type that adopted agricultural practices the least (See Section 4.6.1). Furthermore, an increase in the amount of move actions can be observed in the *IE* agents towards the end of the simulation. This corresponds with the decrease in agricultural adoption noted in Section 4.6.1. Lastly, the *RBAdaptive*



(A)



(B)

FIGURE 4.6: Plots of the average number of settlements per household (a) and the average number Household move actions (b) for each agent-type.

agents exhibit close to zero population migration which corresponds to their near 100% agricultural adoption rate.

Further analysis of the type of move actions performed revealed that most move actions performed by the agents are from one settlement to another (as opposed to moving and creating a new settlement). This is shown in Figure 4.6a where we it can be seen that settlement density, the number of settlements per Household agent, decreases for all agent-types investigated. This means that despite increased movement, the agents prefer to move between already established settlement sites. Furthermore, the large quantity of movement actions exhibited by the *Utility* and *Traditional* agents results in persistent settlements with transient identities. That is, the location of the settlements remain fixed but the Households that make up the settlement frequently change.

Explaining the migratory behaviour of the agents required further investigation. In *NeoCOOP*, agents move when they are dissatisfied. This dissatisfaction is an indication that an agent's resource needs are not being met. As shown in Section 4.5, none of the simulations exhibited resource scarcity. This motivated us to investigate the distribution of these resources which we measure using the Gini Index. As shown in Figure 4.7, the *Utility* agents show extreme inequality while the other agent-types only have moderately high inequalities. The emergence of high socioeconomic inequality is expected behaviour (See Section 4.6.4), and explains the migratory behaviour of the agents.

As detailed in Section 3.4.3.7, agents will migrate to settlements where the average household resource levels are about equal or above their consumption needs. If no settlements meet these requirements, the agents will create a new settlement and move there. In the *Utility* simulation runs, the magnitude of the emergent inequality arises due to the distinct partitioning of the agents into those who farm and those who forage. The sedentary farming agents accrue a surplus of resources which attracts the foraging agents to them. The foraging agents then move between these farming settlements. This creates an effect whereby large settlements emerge housing wealthy farmers and 'visiting' foragers. While this phenomena is certainly interesting, its applicability to Predynastic Egypt is discussed in Section 4.6.4. At the very least, the *Utility* agents do highlight the impact the agent's migration function has on its outputs and raise questions about what emergent behaviour could arise if the function or its parameters were different.

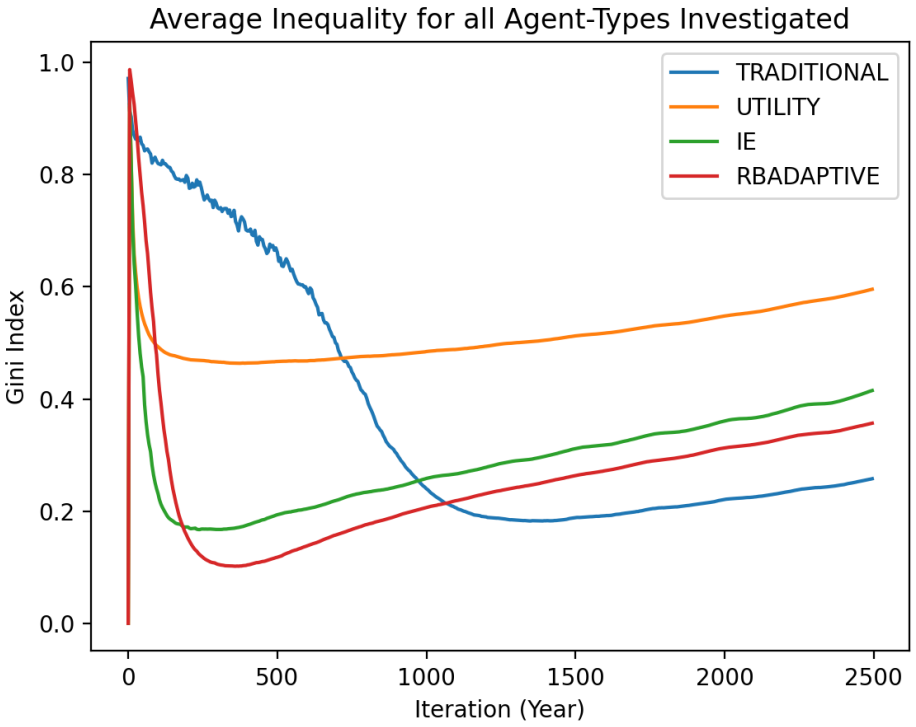


FIGURE 4.7: Average Gini-Index of each agent-type investigated.

4.6.3 The Importance of Information Throughput

As shown in Section 4.6.1, the production rate of farming played a significant role in determining the final population level of a simulation run. However, it failed to explain the rapid adoption of farming by the *IE* agents. This Section explores this behaviour through the lens of cultural and biological information exchange. In particular, we look at the effect information throughput has on the model's final output.

In *NeoCOOP*, the *IE* agents adapt their cultural properties based on the information they learn by themselves and receive from their peers (See Sections 3.4.2.4 and 3.4.3.8 for information on this learning and exchange process). The properties that moderate the rate at which agents receive this information are the *stubbornness* and *conformity* agent parameters. In a ML context, these values are the agents' learning rates for individual and social learning processes. It stands to reason that throttling the rate of information exchange may affect the model's final results.

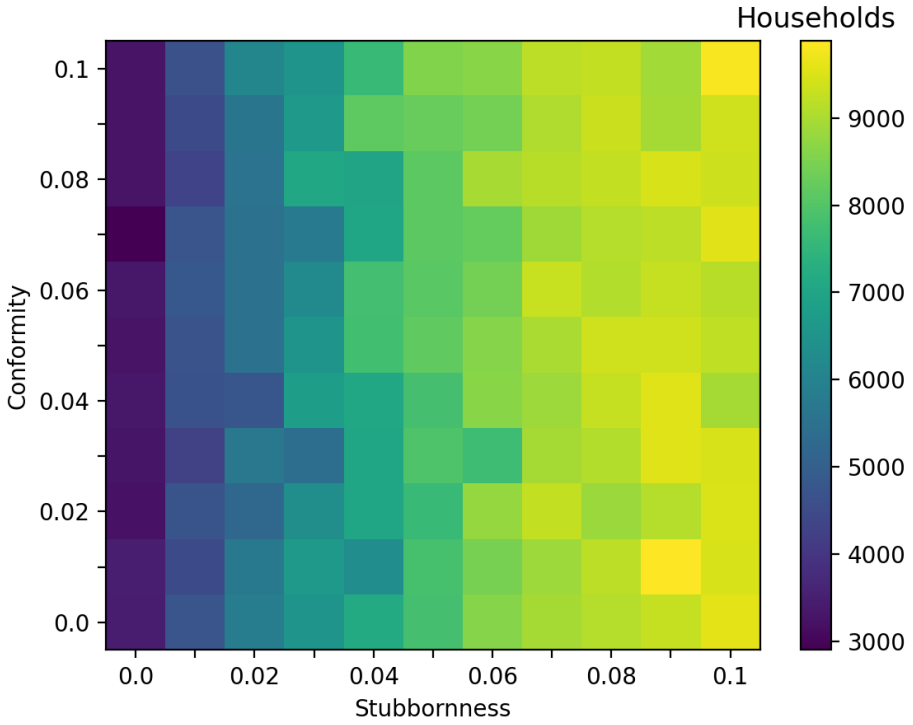
We investigate this claim by conducting additional experiments whereby we alter the agents' *stubbornness* and *conformity* values to the following 11 discrete values:

$\in [0.0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]$ for a total of 121 parameter combinations[3]. *Stubbornness* and *conformity* values selected per scenario are fixed and homogenized across the agent population. That is to say that we disabled the evolutionary algorithms from acting on these two agent properties such that they remain constant, and equal, for all agents for the duration of a simulation run.

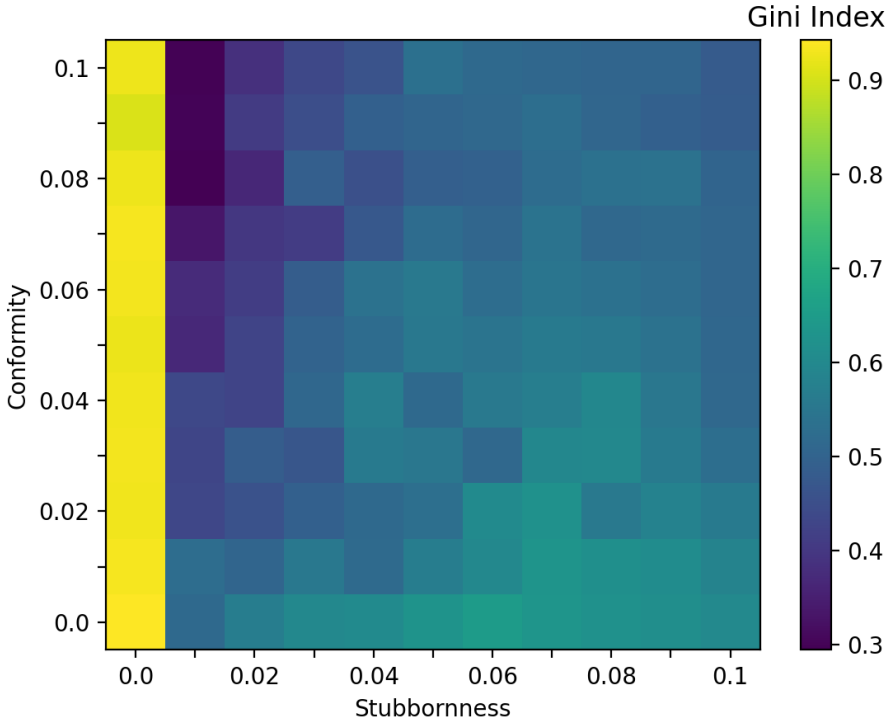
Figures 4.8a and 4.8b show the total Household population and Gini Index for the supplementary experiments investigated. The results indicate that *stubbornness* (individual learning rate) is the key parameter in determining final Household population levels and that these two values (individual learning rate and final population level) are positively correlated for the range of values investigated. This result is not surprising. The ability for an agent to quickly identify and adapt to external stressors is clearly going to be more effective at ensuring the agent's survival than an agent that must rely on receiving the information from its peers before adapting (high conformity low individual learning) or an agent that does not learn from its peers and is incapable of adapting themselves (low conformity low individual learning rate). In terms of understanding how these parameters affect the model, we know from Section 4.6.4 that the faster the rate of agricultural adoption, the greater the final Household population. These results further confirm this behaviour by demonstrating that limiting the individual learning rate of the agents, their rate of agricultural adoption is slowed and their final population levels are lower.

Results indicate that, with the exception of the low individual learning rate scenarios ($stubbornness = 0.0$), *conformity* negatively correlates with inequality and that individual learning rate positively correlates with inequality for the range of values investigated. This is can be seen in Figure 4.8b where the top scenarios (high conformity) have a lower Gini Indices between 0.3 and 0.5 while the bottom scenarios (low conformity) have Gini Indices between 0.5 and 0.75. This suggests that while high individual learning rates are important for determining population growth, the ability to effectively share this learned information is what promotes equality (relative to the other scenarios). Lastly, the extreme inequality found at low individual learning rate scenarios is directly attributed to the inability for the agents to learn how to farm by themselves (because the *stubbornness* value is 0.0). Thus, only mutations of the *forage* and *farm* utility values (by either the GA or CA) will produce agents which farm. These lucky few agents who do

[3]The rest of the model's parameters are the same as those presented in Section 4.4.



(A)



(B)

FIGURE 4.8: Final Household population (a) and Gini Index (b) for supplementary learning rate experiments.

farm will acquire excess resources which causes the inequality value to rise dramatically to the > 0.9 value observed in Figure 4.8b.

Contextually, these results demonstrate how culturally distinct societies may emerge based on their ability to share knowledge. In Predynastic Egypt, distinct northern and southern cultures emerged [11]. To some researchers, northern communities are said to have been more egalitarian than their southern counterparts [164]. Our results suggest that, if this was the case, it may mean that the northern cultures had low individual adaptive capacity but high collective adaptive capacity (high conformity low learning rates in our experiments). Conversely, the northern communities would have exhibited greater individual adaptive capacity with lower collective adaptive capacity (low conformity high learning rates in our experiments). Interestingly, lack of knowledge sharing between southern Egyptian communities may have been intentional but, investigating this claim is beyond the scope of this research.

It is worth noting that the *stubbornness* and *conformity* properties were never intended to represent external learning limitation factors, but these experiments certainly leveraged them as such. These experiments have identified that the acquisition and dissemination of knowledge likely played a significant role in forming of cultural identifies in Predynastic Egypt. With that in mind, our supplementary experiments were limited as our model does not contain any explicit information limiting systems[4] and the forced homogenization of the agents' *stubbornness* and *conformity* values prevented us from understanding the dynamics that would emerge from a heterogeneous population, facets that can be explored in future work.

4.6.4 Results in the context of Predynastic Egypt

In this Section, we contextualize the results obtained in both our main and supplementary experiments to Ancient Egypt during the Predynastic period. This is done by looking at each of the Sections before this (From Section 4.6.1 to Section 4.6.3) and illustrating how, if at all, the model's results relate to relevant literature.

[4]That is, natural or social systems that limit the exchange of information. For example, an impassable river would a natural information exchange limiter while the deliberate hoarding of knowledge by elites would be a social limiter.

In general, the model's results are mixed with all agent-types simultaneously exhibiting both plausible and improbable behaviour. This is most clearly shown in Section 4.6.1 where all agent-types adopt farming as their primary resource acquisition strategy. However, the rate at which this adoption occurs is not realistic. A primary reliance on domesticated animals and plants is likely to have taken hundreds of years [11]. The ML agents did it in less than 50. Conversely, *RBAdaptive* agents took a more plausible 400 years to fully adopt farming but, the agents did it uniformly across the population demonstrating a distinct lack of cultural individuality.

Additionally, our results revealed the importance surplus resource production has on the adoption of farming across the entire population. Neolithic communities are likely to have invested significant time to get the production rate of farming to provide such a surplus, further reinforcing the adoption of agricultural practices. It is noted by several authors that farming produced a significant surplus great enough to enable the construction of mega-structures such as temples and pyramids [36]. Our experiments suggest that agricultural practices would have have emerged without such a surplus initially present but, that it is only through the discovery of technological mechanisms of extracting said surplus, that agricultural adoption would become the dominant resource acquisition strategy across the entire region's population.

Section 4.6.2 looked explicitly at the migratory patterns of the agents. It was revealed that as agricultural practices became dominant, so too did a preference towards a sedentary lifestyle. This well documented across many ancient societies (including Predynastic Egypt) [165] as farming required investing in infrastructure and equipment that would be difficult or impossible to move.

The ML agents produced an emergent behaviour whereby foraging households emerged and regularly moved between these larger settlements. This behaviour is attributed to the heuristics used by agents when determining where they should relocate to. Allen [36] describes the migration of Upper Egyptians to Lower Egypt due to unsatisfactory living conditions. Our model, while likely unrealistic in its final results, does reveal that unfavourable living conditions certainly did arise and population migration does occur under these circumstances. The choice of when and where to move is complex and all of its facets were not captured by the model.

Lastly, Section 4.6.3 reviewed the effect information throughput had on the emergent properties of the model. Results indicate that the ability to acquire and share information has a significant effect on the dynamics of the model. When throughput is high, a greater population that is relatively more equal emerges. When information sharing is low, greater inequality emerges. Stevenson [8] describes knowledge as a resource to be monopolized. Our results support this claim, indicating that the monopolization (high information throughput of only a few individuals) and exploitation of knowledge may explain[5] many of the dynamics observed in Ancient Egypt.

Our model was unequipped to tackle this endeavour directly but, it would be interesting for future work to evaluate what dynamics emerge in populations with variable information throughput as well as mechanisms by which the agents might deliberately withhold or share specific information to maximize their social standing.

4.7 Summary

In this Chapter, we outlined the process used to setup and run the experiments we designed to investigate this work's primary research questions. We first discussed how we acquired the Archaeological and GIS data used in this work and how we used this data to correctly scale our model.

We then described how the absence of accurate climate data from the Neolithic period forced used to generate the data ourselves. We used interpolators to do this which allowed us to simulate general climate change as well as seasonal Nile floods.

We discussed how our model and agent's were parameter tuned. We first used sensitivity analysis to identify extrema in the model's input parameters and then used *Optuna* to paramater tune each of agent-types within these extrema.

For our experiments, we ran 50 simulations per agent-type initializing each simulation to run for 2500 iterations with 1610 agents and 537 settlements, recording the total household population and total surplus resources (our adaptability metrics).

Visually distinct results were obtained and Kruskal-Wallis and Posthoc Dunn tests ($p = 0.05$) were performed to test for significance. The tests revealed that the *Traditional*

[5] That is not to say that no other causal factors exist, just that monopolization of the knowledge domain was one of them.

agents performed the worst, the *Utility* and *RBAdaptive* agents performed similarly and that the *IE* agents ranked highest and are considered the most adaptive agent-type investigated.

Further analysis of our results revealed that while the model's outputs were interesting, they were not necessarily realistic in a Predynastic Egyptian context. Despite this, several emergent phenomena were observed such as emergent inequality and population migratory patterns. Furthermore, our model identified agricultural productivity and information (knowledge) monopolization as key facets of the formation of Ancient Egypt during the Predynastic period, signaling opportunity for future research endeavours.

Chapter 5

Discussion

In this Chapter, we synthesize our findings from Chapters 3 and 4 and present a critical evaluation of the research questions presented in this work. Overall, the findings of this work clearly show that adaptive-agents are more suited to modelling the dynamics of complex environments than their rule-based counterparts. More specifically, our results demonstrate that ML algorithms are particularly well suited as these adaptive mechanisms given that they not only allowed our agents to maintain high population and resource levels, they facilitated the emergence of additional emergent phenomena such as resource acquisition strategy specialization. It is our hope that the findings presented in this work pushes the state of the art such that future research endeavours seek to use truly adaptive-agents in their complex Archaeological ABM.

The rest of the Chapter is divided into sections that correspond to said research questions. Section 5.1 compares and contrasts the Machine Learning agents to the rule-based agents, Section 5.2 details the emergent phenomena produced by *NeoCOOP* across all agent-types, Section 5.3 discusses the *NeoCOOP* model and how it can be further used to study the emergence of Ancient Egypt during the Predynastic Period and Section 5.4 reviews the cost of complexity in Agent-based Modelling and how it applies to development of adaptive-agents. The Chapter concludes with a summary (Section 5.5).

5.1 Machine Learning vs. Rule-based Agents

We sought out to investigate whether agents using machine learning techniques as adaptive mechanisms, would exhibit greater adaptive capacity than traditional, rule-based, agents when placed into a sufficiently adversarial environment. Our results suggest that ML agents are indeed more adaptive than their rule-based counterparts. The *IE* agent type ranked first and the *Utility* agent ranking similarly to the more complex *RBAdaptive* agent. Additionally, our suspicions about rule-based agents were appropriate. They were less adaptive and produced fewer emergent behaviours overall. However, ML agents are not a panacea and further work will need to be done to address several issues identified in this work.

The foremost issue we encountered was the unrealistic emergent behaviour produced by the ML agents. This took the form of rapid agricultural adoption and cyclic population migration patterns. Conversely, the *RBAdaptive* agent-type, did not exhibit this type of behaviour. As noted by Zhang et al. [166], this is quite a common issue when integrating ML techniques with ABM. This issue primarily occurs because of the nature in which ML techniques learn or evolve. Most ML techniques used in ABM are traditionally used to optimize some objective function. For example, ANNs minimize loss, RL algorithms maximize reward and EAs can be used to either maximize or minimize a fitness function(s). This means that, regardless of whether it is realistic or not, an ML agent will perform actions that optimize its objective functions as much as possible. The nature of the objective function will dictate the outputs produced by the model. In our model. our agents discovered that farming was beneficial and adopted it rapidly because it was the most optimal thing to do (it maximized their social status). They achieved this by abusing the simple nature of the model's farming mechanisms which is, from our perspective, both a blessing and a curse. For one, it supports Recknagel's [71] statement that ML agents are capable of producing emergent behaviour as well as overcoming the rigid structure of traditional ABM, exploring realms unanticipated which our rule-based agents just simply could not do. On the other hand, the fact that ML agents will abuse potential loopholes and the simplicities of a model naturally dictate that more effort must be made during the development phase to ensure it does not happen.

The second issue pertains to the untranslatable nature of the ML algorithms' parameters when compared to rule-based algorithms whose parameters may have been derived from

external data sources. For example, we called the individual learning rate parameter of the ML agents *stubbornness* because that is the type of behaviour it caused the agents to exhibit. However, what does a *stubbornness* of 0.9 actually mean? Additionally, how one translates the agent's *stubbornness* to actual human behaviour is unclear. Conversely, rule-based mechanisms have the advantage of potentially being constructed from sociological and psychological studies or the subsequent conceptual models constructed from said studies. For example, the *risk elasticity* parameter of the *RBAdaptive* agent comes directly from protective motivation theory and so translating its parameter value between the model and reality is easier.

This ML parameter set to reality translation dilemma is further elucidated by the fact that Zhang et al. [166] describe raising learning rate parameters in ABM despite us having to lower the learning rates substantially in Section 4.6.3 in order to get the *IE* agents to exhibit different and interesting behaviour. Parameter values do not necessarily translate from one model to another and the effect said parameters have on the model are also unlikely to be similar, even if the models are similar in domain. Additionally, parameter tuning mechanisms may not necessarily be the best way to parameterize ML agents. Again, parameter tuning requires that a modeller define objective functions that define the quality of a given parameter set. Human behaviour may not be capturable by simple objective functions because human behaviour is inherently unoptimal [167].

Lastly, we would like to discuss the method by which the ML agents were created. In Section 3.2 we detailed how adaptive agents could be developed using the concept of information exchange. Overall, we found this process to be promising. The RL and EA algorithms implemented in this work fit naturally within the model and this is part due to using information exchange to understand how these algorithms would affect the agents. One aspect that we did not consider was information throughput which, as shown in Section 4.6.3, significantly impacts the adaptive capacity of the agents. While it may seem obvious that information throughput affects the dynamics of a model, it may do so in obscure ways. For example, restricting information throughput may limit adaptive capacity like it did so in our work, but it may increase adaptive capacity in others. The same logic could be applied to increasing information throughput. While we could not explore this concept much in this work, it would be interesting to do so in future.

5.2 Emergent Phenomena

Our second research objective sought to determine if our adaptive (ML) agents were capable of producing new emergent behaviour (such as polity cycling [34] or strategy specialization [35]) that the traditional, rule-based, agents could not. In total, three identifiable emergent phenomena (increased settlement density as a result of agricultural adoption and emergent social inequality) were observed for all agent-types and two additional phenomena (cyclic population migration and mixed resource acquisition preferences) for the ML agents. Zero unique emergent phenomena were observed for the *RBAdaptive* agents while the *Traditional* agents exhibited the same cyclic population migration phenomena as the ML agents. These results suggest that ML agents are indeed capable of producing new emergent behaviour when compared to similarly complex rule-based agents.

While these results are promising, Section 5.1 revealed that these phenomena may be the result of unrealistic agent behaviour. In our case, cyclic population migration was once such phenomena. That is not to say that the migration pressure was unrealistic, but rather that the action of moving cyclically from one settlement to another was the result of failing to capture necessary dynamics that would allow the agents to make more informed migratory decisions as opposed to moving between the same few settlements that could not meet their resource requirements. It is our belief that had the agents' decision making processes accounted for these dynamics, their migratory patterns would have changed substantially.

A limitation of this work relates to the complexity of the agents. In order to make the agents comparable, we could not leverage ML techniques to their fullest extent. The *Utility* agent-type was intended to be equivalent in complexity to the *Traditional* agent-type and the *IE* agent-type equivalent to the *RBAdaptive* agent-type. Had we not followed this convention, *NeoCOOP* would likely have been two separate, incomparable, models. Research published over the duration this project [41–43] has demonstrated that focused development and exploration of the ML agents capabilities can produce meaningful results. It is our belief that this trend will continue and that future efforts, using *NeoCOOP*, will demonstrate the compatibility, potential and benefits ML adaptive-mechanisms have on the study and simulation of artificial ancient societies.

5.3 The Formation of the Ancient Egyptian State

Our final research objective sought to gather insights into the Predynastic agricultural revolution as it related to the presence of the Nile floodplain [12] and desertification [36] of regions beyond the Nile Valley. Overall, our results were able to confirm several claims about the living conditions of Egypt during Predynastic period. For one, the Egyptian landscape was indeed unlivable outside of the valley. This did restrict the population and force movement to remain along the Nile and, without the Nile, Ancient Egypt is unlikely to have existed as we know it today. Given this, we can confidently state that the active natural factors (the presence of the Nile and desertification of the surrounding areas) had a significant impact on the formation of the Ancient State. In particular, Allen [36] and Anđelković's [12] theories on the dynamics that arise from these natural factors are attractive explanations.

To simply state that the Nile had an impact on the formation of Ancient Egypt is perhaps obvious and uninformative. There is not much debate that this was the case but the social dynamics that arose from these natural factors are. Various theories reviewed in Section 3.3.2 place a specific emphasis on specific elements of this process. The power of the individual [140] or the social and natural contexts that supersede any one individual's ability to affect this process as a whole [12, 36] being the two most common. In truth, each theory is likely to be accurate to some degree. It is undoubtedly true that opportunistic individuals could have exploited Egypt's unique circumstances to their advantage resulting in its grand unification at the start the Dynastic Period but, it is also undoubtedly true that the preceding social and natural factors could have played a role in this process regardless of any one individuals actions. Our results do not refute or prove any these theories but, they do provide insights that could refine or produce new theories about the formation of Ancient Egypt.

In particular, the production rate of agricultural activities and the ability acquire and exchange knowledge were identified as the potential candidates for future work. Our results showed that the surplus produced by agricultural activities was pivotal in determining the rate of its adoption and that even without a surplus, farming could still merge as a non-dominating resource acquisition strategy. It would then follow that determining when, where and how this surplus was discovered is of great importance to understanding the dynamics that underpinned the formation of the Ancient Egyptian state. There is

evidence to suggest that farming was introduced by traders from the Near East [36] but, that does not explain its adoption nor the cultures that emerged from its adoption.

In terms of the acquisition and sharing of knowledge (information), our results indicate that the individual's capacity to acquire and adapt to knowledge, positively correlates with the population's ability maintain steady population growth. Similarly, our results indicate that the capacity for individuals to share information is negatively correlated with the emergence of inequality. That is, societies that share information more freely are likely to be, relatively, more equal. There is much work still be done in this regard because our experiments did investigate mixed populations. That is, populations where individual and group knowledge acquisition is varied. In particular, it would be interesting to observe the emergent dynamics that result from groups of agents that are capable of monopolizing the knowledge domain [8] such that they are the only ones who may reap the benefits of new found discoveries (increased agricultural production for example).

5.4 Agent-based Modelling and Simulation Complexity

ABMs are fundamentally developed in one of two ways: They are either constructed top-down, general-to-specific, using the KISS approach or they are constructed bottom-up, specific-to-general, using the KIDS approach (See Section 2.1.6) with *NeoCOOP* developed using the latter. The motivation for this decision was primarily due to the complexity of the domain being simulated (Ancient Egypt during the Predynastic Period). In particular, the several natural and social factors identified in Section 3.3.3 necessitated that we start with a complex model if we had any hope of accurately representing the Egyptian Predynastic Period. We do not believe that it was incorrect to use the KIDS approach, but the decision came with a set of undeniable disadvantages, particularly for archaeological ABM, that we wish to discuss further.

The experiments conducted in Section 3.2.5 suggested that there would be diminishing returns to using adaptive (more complex) agents in increasingly simple environments. The results of our main experiments (See Sections 4.5 and 4.6) confirmed this. Despite the complex nature of the environment, the non-adaptive *Traditional* agent was able to mimic, to a lesser degree, many of the results obtained by the adaptive-agents. This begs the question, is there some way to measure the complexity of the modelling environment

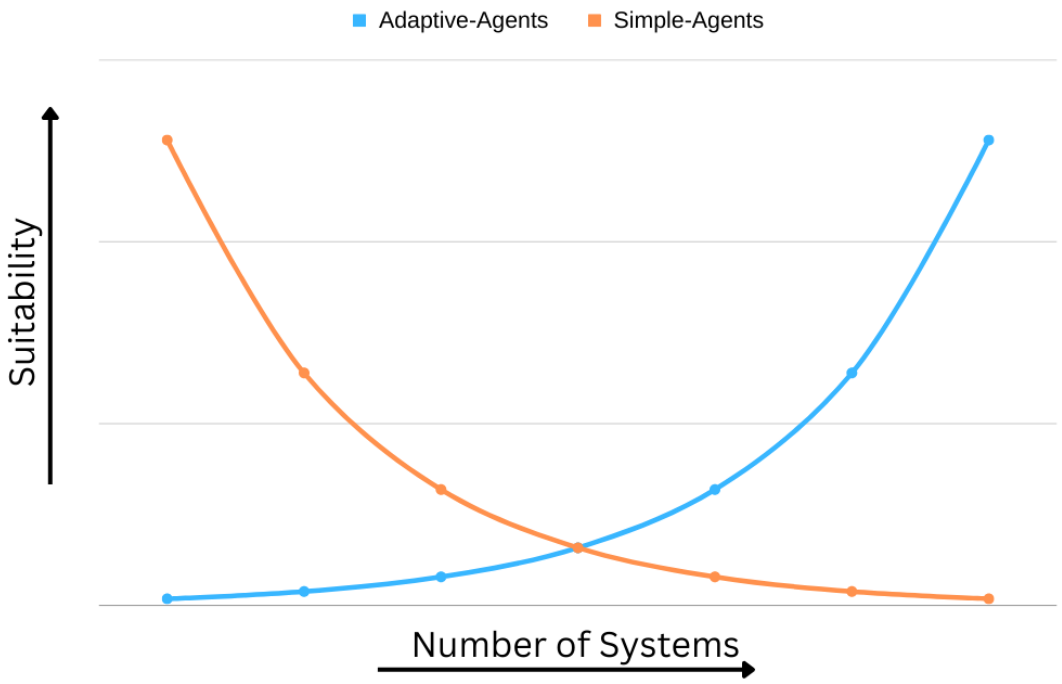


FIGURE 5.1: The suitability trade-off. As the number of systems in your ABM increases, the use of adaptive-agents becomes more desirable. This is due to the increasingly complex rule-based agents that would need to be constructed to account for the interconnectedness of the model's systems.

to determine whether using adaptive-mechanisms is worthwhile? If so, how? Considering the limitations of this work, we argue that determining if adaptive-agents would be useful (or required) when modelling a particular social system can, at best, be reduced to an educated estimation. More specifically, we argue that as more (sub)systems are added to a model, the benefits provided by adaptive-agents rises exponentially. This is due to the increasingly complex rule-based agents that would need to be constructed to account for the interconnectedness of said systems. At some point, this becomes an untenable process outweighed by the scalability provided by adaptive-agents (See Figure 5.1).

Consider Axelrod's cultural dissemination model [146]. It contains one system and, as a result, has no need for adaptive-agents. In fact, using adaptive-agents in the model does not make sense and would likely be detrimental. On the other hand, consider *NeoCOOP*. It has eight (seven if you combine the vegetation model's systems) distinct subsystems that form the model's social ecosystem. It is unsurprising then that the simple *Traditional* agent could not fully capture the models dynamics but, the more

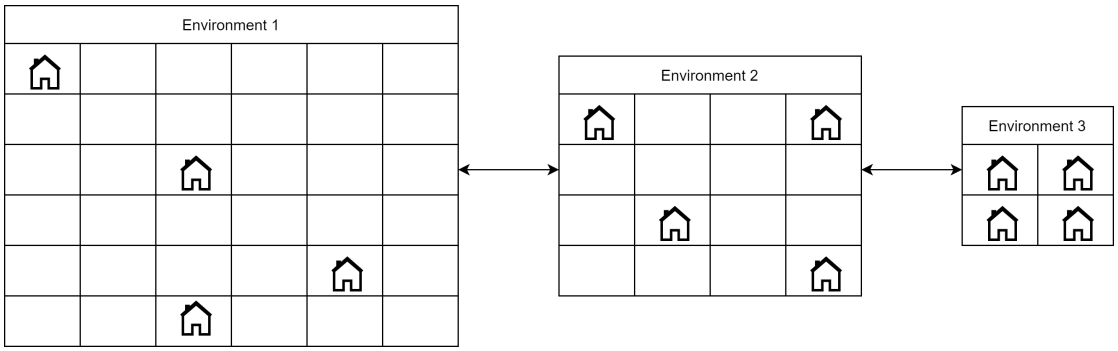


FIGURE 5.2: An example of the proposed "Islands" Model. Here with have three environments of decreasing size (increased resource stress). The environments are represented as a graph such that an agent in Environment 3 would first need to migrate to Environment 2 before migrating to Environment 1.

complex adaptive-agents could. Although, this came at the cost of interpretability as more work needed to be done in order to understand the model's dynamics.

Future work with *NeoCOOP* should seek to simplify the model while maintaining the benefits brought by the adaptive-agents. One such area pertains to the amount of data required to get the simulation working. Archaeological data has the distinct disadvantage of being naturally fragmented and, as a result, it was difficult to get the necessary data for the model. As detailed in Section 4.1, we had to resort modern analogs to fill in the gaps and the model's results certainly suffered because of it. Implementing complex systems often requires additional data. Data which you simply may not have access to.

Future versions *NeoCOOP* will need to directly address this issue by limiting its reliance on external sources of data. It may be worthwhile to remove all of the Vegetation Model's systems completely and replace them with an alternative. One such avenue would be adopting a more stylized environment design whereby the environment is divided into separate grid-worlds of various sizes and environmental conditions. Using the example of Egypt, two grid-worlds can be created, one large environment with an abundance of resources representing the delta and one smaller environment with fewer resources representing the rest of the valley. Using this "Islands" approach (See Figure 5.2), the Households can evolve separately within these environments and, every so often, agents may migrate from one "island" to the other. The size of the "island" inherently facilities resource stress (the smaller the environment, the fewer resources it is capable of producing) and, by connecting "islands" together in a graph representation, migration dynamics may even be studied as well. This approach would completely remove the need for GIS data and meets all of the constraints identified in Section 3.3.3. Subject

to further investigation, this approach may facilitate the emergence of distinct cultural identities in a simpler, easier to understand, model.

5.5 Summary

In this Chapter, we synthesized our findings from Chapters 3 and 4 and presented a critical evaluation of the research questions presented in this work. First, we compared the ML agents to their rule-based counterparts. Our results clearly showed that the ML agents had greater adaptive capacity but, they are not without their limitations. In particular, we identified that the objective optimization present in most ML algorithms can result in the agents exploiting model simplicities which may result in unrealistic model behaviour. Furthermore, we noted that the, sometimes arbitrary, nature in which ML algorithm parameters are related to their real-world analogs can make translating model results to reality difficult or even impossible.

We then highlighted the emergent phenomena produced by the agent-types. In addition to increased settlement density and emergent inequality, the ML agents produced two additional emergent phenomena in the form of cyclic population migration and resource acquisition specialization. We claimed that this was evidence that ML agents have the potential to produce emergent behaviour beyond that of their rule-based counterparts.

We discussed the implications our results have on understanding the formation of Ancient Egypt during the Predynastic period. We noted that while our results do not support or refute any theories related to the formation of the Ancient Egyptian state, our results clearly demonstrate that the presence of the Nile and the desertification of the surrounding areas had a significant impact on the dynamics of that time period. Furthermore, we identified that information throughput and agricultural production rate as potentially significant factors in further understanding the dynamics of the Egyptian Predynastic.

The Chapter concluded with a discussion on the complexity of social simulations. We showed that as systems are added to a model, the need for adaptive-agents increases exponentially. We argued that this is attributed to the non-adaptive agents' inability to capture the dynamics of the model's increasingly interconnected systems.

Chapter 6

Conclusions and Future Work

In this work we sought to investigate whether Machine Learning algorithms could be used as adaptive mechanisms for Agent-based Models simulating the complex social phenomena of ancient societies. We aimed to do this by comparing ML agents (*Utility* and *IE*), developed using Reinforcement Learning and two Evolutionary Algorithms, to rule-based agents (*Traditional* and *RBAdaptive*) typically found in contemporary literature.

In order to effectively evaluate these agent-types, a suitably complex environment was needed. Ancient Egypt during the Predynastic Period was identified as this environment. The motivation being that several natural and social factors (desertification [36] and emergent social stratification [140] for example) are theorized to have played a role in the ancient state's formation. If the ML agents had greater adaptive capacity than the rule-based agents, it would most certainly be identifiable in this context through their ability to gather more surplus resources and maintain higher population levels [24, 38, 39].

In conducting scenario experimentation, we found that our ML agents performed better or on par with even the complex rule-based agents given that they were able to maintain higher population levels and comparable resource levels. When averaged across both metrics, the *IE* agent-type was the most adaptive, the *Utility* and *RBAdaptive* agents jointly ranked second and the *Traditional* agent ranked last.

Our results show that when placed into a sufficiently complex environment, the use of adaptive, learning, agents is more desirable than that of their rule-based counterparts. More specifically, our results show that when applying Agent-based Modelling to the study of ancient societies, adaptive-agents should be used because they are able to resist,

recover and modify themselves to a variety of environmental perturbations and initial parameter distributions in addition to their ability to produce emergent phenomena (such as strategy specialization) that their rule-based counterparts cannot.

6.1 Future Work

Future research efforts will directly tackle the limitations of this work. Namely, effort will be made improve the realism of the emergent phenomena of the ML agents. This will be done by critically evaluating the objective functions they use to make decisions. Additionally, *NeoCOOP* will be amended with the changes discussed in Section 5.4. Work will also be done to support our findings by applying *NeoCOOP* to simulating the formation of different ancient societies. If *ML* agents are truly more adaptive than rule-based agents, our results should hold up in similarly complex modelling environments.

Secondly, work will done on *ECAgent* (the modelling framework introduced in Section 3.1) to ensure it is more feature complete as well as an attractive prospect to those searching for a framework to develop ABM with.

Lastly, we will investigate the concepts of agricultural production rate and knowledge monopolization further. Specifically, the effects these two concepts have on the dynamics of the model's results will be studied as we attempt to further understand the complex social phenomena that underpinned the formation of Ancient Egypt during the Predynastic Period.

Appendix A

ODD+D Description

A.1 Overview

A.1.1 Purpose

A.1.1.1 What is the purpose of the study?

The general purpose of *NeoCOOP* is to be a model by which we can simulate the Paleolithic-Neolithic transitionary period that saw humanity move from largely egalitarian hunter gatherer groups to agrarian super polities typically ruled by a social elite. In this work, we use *NeoCOOP* to study the formation of Ancient Egypt during the Predynastic Egypt.

A.1.1.2 For whom is the model designed?

This model is designed primarily for Computational Archaeologists. The model will also be of interest to Computational Social Scientists modelling complex social phenomena and the Artificial Life (ALIFE) community in general.

A.1.2 Entities, State Variables and Scales

A.1.2.1 What kinds of entities are in the model?

There are four kinds of entities within NeoCOOP:

1. **Households**: They are the primary decision making entity in *NeoCOOP* (the agents). They represent a collection of occupants ruled by a patriarchal figure.
2. **Occupants**: Occupants are contained within households. They are not decision making entities and are only present to determine household resource gathering and consumption levels.
3. **Settlements**: Settlements represent a collection of households. They are also not decision making entities but are used by several of the model's systems for simulating / restricting agent adaptation.

The model represents its environment as a $n \times m$ grid-world. Each cell in the grid can technically be thought of as an entity but, their primary purpose is to store local geographical information.

A.1.2.2 By what attributes(i.e. state variables and parameters) are these entities characterized?

See Sections 3.4.1 and 3.4.2.

A.1.2.3 What are the exogenous factors/drivers of the model?

Climate Change. Specifically an increasing or decreasing likelihood of drought over time.

A.1.2.4 If applicable, how is space included in the model?

The environment is a grid-world made up of equally sized cells. *NeoCOOP* supports using GIS layers to add environmental information to the environment. This includes but, is not limited to, height, slope, water and sand content data inputted into the model

as images where each pixel in the image represents the value of a given attribute at that pixel coordinate in the grid-world.

A.1.2.5 What are the temporal and spacial resolutions and extents of the model?

One iteration in *NeoCOOP* represents a single year.

The size of each grid cell is configurable. In this study, each cell is $1km^2$ in size and the total size of the grid-world is determined by the *height* and *width* parameters.

A.1.3 Process Overview and Scheduling

A.1.3.1 What entity does what and in what order?

The order of execution can be seen in Figure 3.20.

A.2 Design Concepts

A.2.1 Theoretical and Empirical Background

A.2.1.1 Which general theories concepts, theories or hypotheses are underlying the model's design or at the level(s) of the submodel(s) (apart from the decision model)? What is the link to complexity and purpose of the model?

The model heavily relies on the vegetation model description by Xu et al. [150]. We simplified their process for *NeoCOOP* by only looking at growth penalties associated with determining NPP.

The Agent-based component of the model is also loosely based on the work of Chliaoutakis and Chalkiadakis [25, 40] and the model makes use of their self-organization scheme for simulating emergent social hierarchies.

Our Agent's resource trading preferences are probability-based. This is an extension to the typically simple cooperative - defective approach. Each agent has a probability p

associated with its resource trading preferences and every time a resource trading request needs to be decided on, a random number is generated $\in [0, 1]$ and the number is less than p, the resource transfer request is granted.

A.2.1.2 On what assumption is/are agents' decision model(s) based?

The model assumes that resource trading preferences can be simulated as a stochastic process. Additionally, it assumes that all households were 'ruled' by a single individual and that personal storage is the preferred method of resource storing (as opposed to collective resource pooling or some other hybrid approach).

A.2.1.3 Why is a/are certain decision model(s) chosen?

Most of the model's input parameters are based on published works or publicly available data. Table B.1 provides said references and where no references are made, NeoCOOP is tuned using *Optuna*.

Optuna performs multi-objective optimization maximizing total population and resources levels in the last iteration. These measures are used because a greater population level is indicative of a more successful parameter configuration and we also consider total resources because having a higher population level with more resources is a greater indicator of success than an equally large population with no resources.

A.2.1.4 If the model/ a submodel is based on empirical data, where does that data come from?

See Table B.1.

A.2.1.5 At which level of aggregation were the data available?

It varies from source to source. Table B.1 clarifies how the data was derived.

A.2.2 Individual Decision Making

A.2.2.1 What are the subjects and objects of decision-making? On which level of aggregation is decision-making modelled? Are multiple levels of decision making included?

The decision making units are Households represented as one of the four agent-types discussed in Section 3.4.2.

Every iteration, agents choose to FARM or FORAGE actions equal to the number of able workers (An able worker is an agent who is older than or equal the *agent_of_maturity* property. Agents are restricted to choosing one action or the other, in fact, agents may choose to have some of their occupants farm, and the rest will forage.

A.2.2.2 What is the basic rationality behind agents' decision-making? Do agents pursue an explicit objective or have other success criteria?

This varies depending the agent-type. The *Traditional* agents simply follow a linear adoption curve. The *RBAdaptive* agents use PMT and the ML agents (*Utility* and *IE*) explicitly perform Utility maximization, implicitly trying to minimize their hunger and maximizing their social status.

A.2.2.3 How do agents make their decisions?

See Sections 3.4.2, 3.4.3.4, 3.4.3.5, and 3.4.3.7.

A.2.2.4 Do the agents adapt their behaviour to changing endogenous and exogenous state variables? And if yes, how?

Yes, agents will seek to explore alternative resource acquisition strategies when their current strategy does not work. Similarly, agents will move from one settlement to another when their overall satisfaction is low.

A.2.2.5 Do social norms or cultural values play a role in the decision making process?

Yes, if multiple agents decide to leave a settlement in the same iteration, they will all move to the same location.

A.2.2.6 Do spacial aspects play a role in the decision making process?

Yes, agents can only farm / forage within a specified max acquisition distance. Similarly, the distance an agent can travel when moving from one settlement to another can be controlled by the *vision_square* property.

A.2.2.7 Do temporal aspects play a role in the decision making process?

Yes, agents only decide whether or not to move every *yrs_per_move* iterations.

A.2.2.8 To which extent and how is uncertainty included in the agents' decision rules?

Not Applicable.

A.2.3 Learning

A.2.3.1 Is individual learning included in the decision process? How do agents' change their rules over time as consequence of their experience?

Yes, the ML agents follow a standard reinforcement learning approach to determine which action (farm or forage) to take. The *RBAdaptive* agent uses PMT to adapt to its environments.

A.2.3.2 Is collective learning implemented in the model?

Yes, in the form of generational adaptation. The ML Agents use a genetic algorithm and a cultural algorithm to exchange information regarding their beliefs and *RBAdaptive* agents use collective learning in the rule-based adaption system (See Section 3.4.3.9).

A.2.4 Individual Sensing

A.2.4.1 What endogenous and exogenous state variables are individuals assumed to sense and consider in their decisions? Is their sensing process erroneous?

It is not erroneous and the agents don't explicitly detect any of the environment's properties. Agents are aware of their hunger, satisfaction and the ML agents are aware of their perceived utility of the forage and farm actions.

A.2.4.2 What state variables of which other individuals can an individual perceive? Is the sensing process erroneous?

Agents are aware of the average resource levels of the neighbouring settlements. They are indirectly aware of the general beliefs held by their settlement (captured in their belief space).

A.2.4.3 What is the spatial scale of sensing?

When moving, agents are able to sense the vegetation density and cells they own within *max acquisition distance cells* around them.

A.2.4.4 Are the mechanisms by which agents obtain information modelled explicitly, or are individuals simply assumed to know these variables?

It is assumed.

A.2.4.5 Are costs for cognition and costs for gathering information included in the model?

Not explicitly.

A.2.5 Individual Prediction

Agents do not make any explicit predictions.

A.2.6 Interaction

A.2.6.1 Are interactions among agents and entities assumed as direct or indirect?

Cultural Influence occurs indirectly while resource transfer is direct.

A.2.6.2 On what do the interactions depend?

Resource transfer requires that agents belong to the same settlement. Cultural Influence depends on the social status of the two 'interacting' agents.

A.2.6.3 If the interactions involve communication, how are such communications represented?

Not Applicable.

A.2.6.4 If a coordination network exists, how does it affect the agent behaviour? Is the structure of the network imposed or emergent?

Not Applicable.

A.2.7 Collectives

A.2.7.1 Do the individuals form or belong to aggregations that affect and are affected by the individuals? Are these aggregations imposed by the modeller or do they emerge during the simulation?

Yes. As mentioned above agents are may form settlements. The simulation does not enforce settlements (except at initialization). Agents may form new settlements, leave old ones or even move to other settlements every *yrs_per_move* iterations. An agent always needs to belong to a settlement (so that the adaptation systems can work) but it is entirely possible that a simulation run may result in every agent forming their own settlement. This is equivalent to having no settlements since each household will adapt individually.

A.2.7.2 How are collectives represented?

A collection of one or more households makes a settlement.

A.2.8 Heterogeneity

A.2.8.1 Are the agents heterogeneous? If yes, which state variables and/or processes differ between the agents?

Yes. All variables listed in Section 3.4.2.

A.2.8.2 Are the agents heterogeneous in their decision-making? If yes, which decision models or decision objects differ between the agents?

Yes. All agent decisions are heterogeneous. This includes resource acquisition, resource transfer and relocation.

A.2.9 Stochasticity

A.2.9.1 What processes (including initialisation) are modelled by assuming they are random or partly random?

All stochastic processes are pseudorandom. This is to ensure model reproducibility. A list of stochastic processes during model execution are listed below:

1. Global Environment values (rainfall, temperature and solar radiation).
2. ML Agent explorative action selection (e-greedy).
3. Rule-based agent resource acquisition action selection.
4. Agent farm / forage resource gathering patch selection.
5. Household birth.
6. Household death.
7. Resource Transfer Requests.
8. House Split Parent Selection (Genetic Algorithm)
9. Household Split Mutation (Genetic Algorithm).
10. Household Influence knowledge source selection (Cultural Algorithm)
11. Household Move decision.

A list of stochastic processes at initialization are listed below:

1. Settlement placement.
2. ML Agent initial gene creation.
3. Agent settlement placement.

A.2.10 Observation

A.2.10.1 What data are collected from the ABM for testing, understanding and analysing it, and how and when are they collected?

Snapshots of the model are collected every (user-defined) iterations. These snapshots capture all of the necessary aspects for, essentially, recreating the simulation run from the ground up.

All environment data is collected in csv files. All agent and settlement data is collected in JSON files. A log file of the simulation is also recorded which records the result of every stochastic process the model simulates.

A.2.10.2 What key results, outputs or characteristics of the model are emerging from the individuals? (Emergence)

This is heavily reliant on the input parameters of the model but the key emergent properties are discussed in Section 4.6 and Gower-Winter and Nitschke [41–43].

A.3 Details

A.3.1 Implementation Details

A.3.1.1 How has the model been implemented?

The model was implemented in Python 3 using the *ECAgent* framework.

A.3.1.2 Is the model accessible, and if so where?

Yes, it is publicly available at: `https://github.com/BrandonGower-Winter/NeoCOOP`.

A.3.2 Initialization

A.3.2.1 What is the initial state of the model world, i.e. at time t = 0 of a simulation run?

Agents have been randomly allocated to settlements. Settlements have been randomly placed. All agents have zero resources and zero load. The rest of the initial conditions are based on the decoder file used to create the simulation.

A.3.2.2 Is the initialisation always the same, or is it allowed to vary among simulations?

It varies depending on the seed used. If the same seed is used, the initialization (and model execution) will be exactly the same.

A.3.2.3 Are the initial values chosen arbitrarily or based on data?

Randomly. If agent homogeneity is enforced, the model will not randomly assign agents to each settlement (all settlements will be given the same number of starting agents) but the settlement locations will still be random.

A.3.3 Input Data

A.3.3.1 Does the model use input from external sources such as data files or other models to represent processes that change over time?

Yes. *NeoCOOP* uses what we call *decoder* files. They share the same structure as table B.1 (in JSON format) and are given to the model at initialization. The model also takes in a heightmap, sandcontant map and slopemap (usually derived from the heightmap) and optional floodmap.

A.3.4 Submodels

A.3.4.1 What, in detail, are the submodels that represent the processes listed in 'Process overview and scheduling'?

See Section 3.4.3. Additionally, a visualization of the model's output has been shown in Figure A.1.

A.3.4.2 What are the model parameters, their dimensions and reference values?

See Table B.1.

A.3.4.3 How were the submodels designed or chosen, and how were they parameterised and then tested?

See Section 4.4. Final input parameters for our model can be seen in Table B.1. A report of the optimization process is outlined in Appendix B.

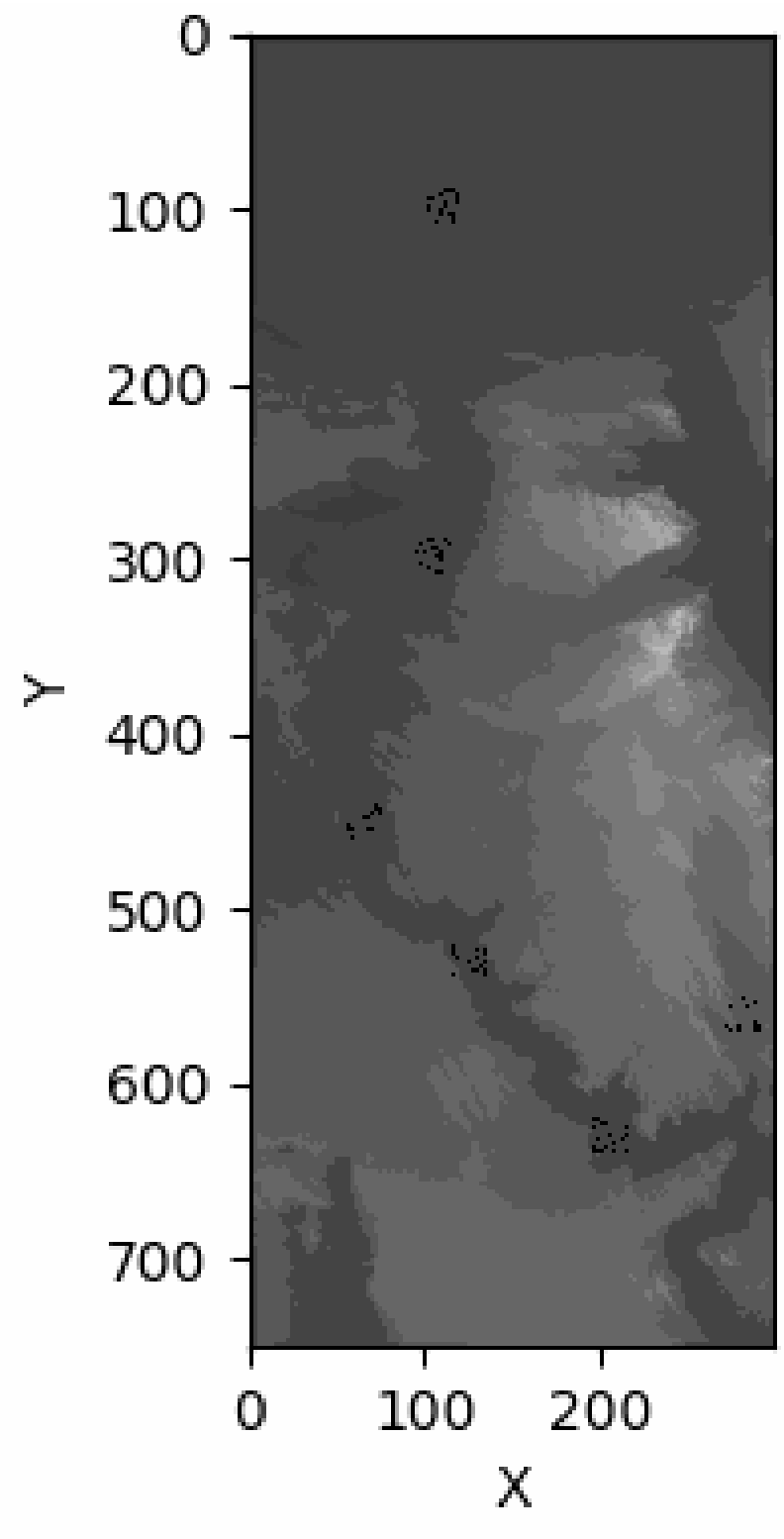


Figure A.1: A visualization of the final results produced by a typical simulation run. Black pixels indicate settlements or farmland.

Appendix B

Parameter Tuning and Model Analysis

B.1 Code Coverage and Validation

Unit testing was performed on both the *ECAgent* framework and *NeoCOOP* model to validate code implementations. We used the package *pytest* to conduct the unit testing and *pytest-cov* was used to ensure that code coverage of the unit tests was sufficient. Below is the raw output of the code coverage report for both *ECAgent* and *NeoCOOP*.

ECAgent:

```
Name                        Stmts   Miss  Cover
---------------------------------------------
ECAgent/Collectors.py          51      0   100%
ECAgent/Core.py               152      9    94%
ECAgent/Decode.py              59      1    98%
ECAgent/Environments.py       147      0   100%
ECAgent/Visualization.py      126     94    25%
ECAgent/__init__.py             0      0   100%
---------------------------------------------
TOTAL                         535    104    81%
```

Note that the poor code coverage of *ECAgent.Visualization* is due to the UI-based nature of the package.

NeoCOOP:

```
Name                          Stmts   Miss  Cover
-----------------------------------------------
NeoCOOP/Agents.py               223     34    84%
NeoCOOP/VegetationModel.py      146      7    95%
NeoCOOP/NeoCOOP.py               32      0   100%
NeoCOOP/Visualization.py         17      0   100%
-----------------------------------------------
TOTAL                           418     41    90%
```

Note that the several individual scripts were summarized under the model's major components and the results have been adjusted to reflect only code that could be unit tested.

B.2 Optuna

NeoCOOP is parameter tuned using multi-objective optimization in *Optuna*. This process seeks to find a parameter set that maximizes the model's final population and resources levels. The motivation for this is because a higher population level is indicative of a more resilient agent and because resources are essential to their survival. Agents who are able to maintain a higher degree of surplus resources are more successful than agents that have less. Note that we do not care about the distribution of the resources per agent and only consider mean surplus resources across all agents. This is to ensure that we do not favour one type of organizational scheme over another (Authoritarian vs. Egalitarian)

The optimization process was as follows: For each agent-type, 50 simulations were run (excluding the *RBAdaptive* agent-type which uses from the OMOLAND [147] model) and the final pareto optimal parameter combinations were recorded. This can all be replicated using the *OptunaRunner.py* script included with the source code). The final outputs from this process are included below:

Traditional:

```
Pareto Optimal Solutions:
{'forage_grad': 0.27371, 'forage_offset': 0.0685242}
{'forage_grad': 0.944704, 'forage_offset': 0.0999721}
{'forage_grad': 0.299912, 'forage_offset': 0.388338}
{'forage_grad': 0.905317, 'forage_offset': 0.441458}
{'forage_grad': 0.808227, 'forage_offset': 0.107181}
{'forage_grad': 0.88712, 'forage_offset': 0.273298}

Final Solution:
{'forage_grad': 0.6864983333333333, 'forage_offset': 0.2297952166666667}
```

Utility:

```
Pareto Optimal Solutions:
{'learning_rate_lower': 0.099245, 'learning_rate_upper': 0.463606}
{'learning_rate_lower': 0.521943, 'learning_rate_upper': 0.528274}
{'learning_rate_lower': 0.285754, 'learning_rate_upper': 0.471596}
{'learning_rate_lower': 0.72279, 'learning_rate_upper': 0.784563}
{'learning_rate_lower': 0.473312, 'learning_rate_upper': 0.55982}
{'learning_rate_lower': 0.67954, 'learning_rate_upper': 0.688166}
{'learning_rate_lower': 0.123563, 'learning_rate_upper': 0.578573}

Final Solution:
{'learning_rate_lower': 0.41516385714285714,
'learning_rate_upper': 0.5820854285714285}
```

IE:

```
Pareto Optimal Solutions:
{'influence_frequency': 17, 'influence_rate': 0.96187,
 'mutation_rate': 0.13745, 'learning_rate_lower': 0.410563,
 'learning_rate_upper': 0.644966}
{'influence_frequency': 46, 'influence_rate': 0.561638,
```

'mutation_rate': 0.111579, 'learning_rate_lower': 0.00935602,
'learning_rate_upper': 0.605442}
{'influence_frequency': 14, 'influence_rate': 0.977973,
'mutation_rate': 0.199105, 'learning_rate_lower': 0.49406,
'learning_rate_upper': 0.846665}

Average Results:

{'influence_frequency': 25.666666666666668, 'influence_rate': 0.833827,
'mutation_rate': 0.149378, 'learning_rate_lower': 0.3046596733333333,
'learning_rate_upper': 0.6990243333333334}

B.3 Model Parameters

See Tables 4.1 and B.1.

NeoCOOP		
Property	Value	Reference
Model Parameters		
Iterations	2500	
map_height	750	
map_width	300	
offset_x	0	
offset_y	0	
max_height	1400m	
min_height	0m	
cell_dimensions	1000m	This makes the area of a single cell = $1km^2$
Global Environment System		
Priority	10	
start_rainfall	[32,42]mm	Midpoint of Semi-Arid yearly rainfall [168]
end_rainfall	[0,10]mm	This range represents a range of drought rain conditions. Based on [169].
start_temperature	[26,32] $°C$	Arbitrary range to minimize Temperature penalties.
end_temperature	[26,32] $°C$	Same as above.
start_flood	[12.0, 21.0] m	Derived from [162] and pre-processed heightmap
end_solar	[0.0, 12.0] m	Same as above.
soil_depth	1000mm	
interpolator_range	2500	Note: Must be equal to the number of iterations.
temperature_interpolator	linear	
rainfall_interpolator	cosine	Set the frequency equal to 250 derived from [162].
flood_interpolator	cosine	Same as above.
Soil Moisture System		
Priority	9	These values are arbitrary and represent a generic semi-arid climate.
L	9.5 hrs	
N	30 days	
I	86.5 $°C$	
Vegetation Growth System		
Priority	7	
Agent Resource Acquisition System		
Priority	8	
farmers_per_patch	1	
max_acquisition_distance	20	
farming_production_rate	4.0	See Section 4.1.
foraging_production_rate	1.0	See Section 4.1.
storage_yrs	-1	This means quantity-based storage decay is disabled.
Agent Resource Transfer System		
Priority	6	
load_decay	0.0	This means it is disabled.
Agent Resource Consumption System		
Priority	5	
Agent Population System		
Priority	4	
birth_rate	0.1%	Derived from [36]
death_rate	0.01%	Derived from [36]
yrs_per_move	5 iterations	. [25]
init_settlements	537	See Section 4.1
cell_capacity	100	
Agents		
number	100	
age_of_maturity	0	This means it is disabled.
consumption_rate	1.0	
child_factor	0.0	This means it is disabled.
init_occupants	1	
vision_square	2500	
move_lookback	3	
load_difference	0.6	[25]

TABLE B.1: A list of *NeoCOOP*'s properties. Note that specialized agent-type properties are described in Section 4.4.

Appendix C

GIS Data Preprocessing

In this Chapter, we describe how the GIS data-maps used in our experiments were created. Section C.1 describes how the heightmap was generated. Section C.2 describes how the slopmap was generated using the heightmap. Section C.3 describes how the floodmap was generated and Section C.4 describes how the soil texture maps were created.

Note: Several scripts will be mention in this Chapter. All of them are available for download in the *NeoCOOP* code repository (See Chapter 3).

C.1 Height Map

The heightmap data was sourced from the ALOS Global Digital Surface Model [158]. The process for converting the raw *AWD30* data to a single PNG was as follows:

1. Download the *AWD30* DSM TIFF files from the *AWD30* website available at: `https://www.eorc.jaxa.jp/ALOS/en/dataset/aw3d30/aw3d30_e.htm`
2. Use the ALOS provided *aw3d2srtm* shell script to convert the DSM files SRTM Data files aliased with the hgt and hgt.aux extensions.
3. Convert the hgt files into PNGs using the *HGT_TO_PNG.py* script.
4. Stitch the PNGs together using the *PNG_stitcher.py* script.
5. Manually downscale the image to the appropriate image size (750 by 300 pixels in our case).

Algorithm 19: Pseudocode for generating slopemap data using heightmap data. Here *cellsize* refers to the distance between orthogonal cells calculated using the scale of the heightmap.

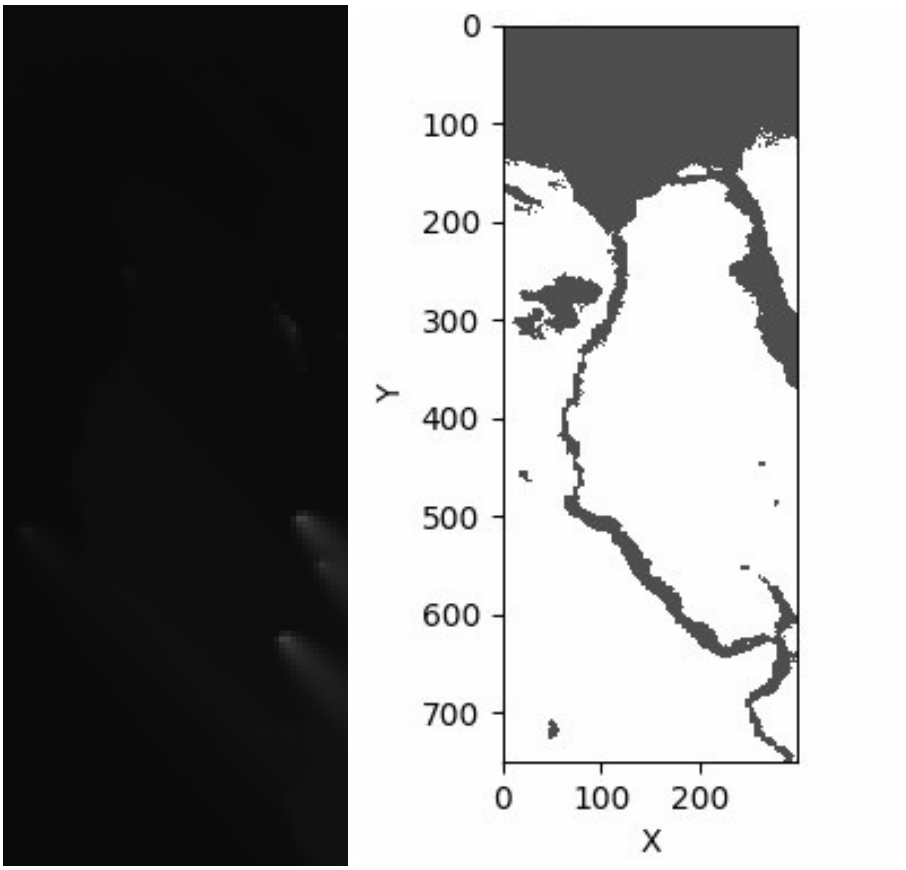
```
1  def get_slope_data(heightmap : array of size (Y,X)):
2      slopemap = array of size (Y,X) for y in Y do
3          for x in X do
4              maxslope = 0.0 for i in range(-1, 2) do
5                  xPos = x + i if 0 < xPos < X then
6                      for j in range(-1, 2) do
7                          yPos = y + j if 0 < yPos < Y then
8                              newslope = degrees(arctan(abs(heightmap[xPos][yPos] -
                                heightmap[x][y]) / cellsize)
9                              maxslope = max(maxslope, newslope)
10                     end
11             end
12             slopemap[y][x] = maxslope
13         end
14     end
15     return slopemap
```

C.2 Slope Map

The slopemap data requires that the heightmap data (Section C.1) has been generated already. The process for generating the slopemap is as simple as using the *heightmap_slopemap_converter.py* script. The process used by the script is described in Algorithm 19.

C.3 Flood Map

The floodmap data requires that the heightmap data (Section C.1) has been generated already. The process for generating the floodmap only requires that the using the *flood_mapper.py* script be used. The process used by the script can be thought of as a flood-fill algorithm that recursively calculates the minimum required height a given cell needs in order to be considered "flooded" on a given iteration. This process expands outwards from cells that have been marked as "water" cells. The result of this process can be seen in Figure C.1a. Given the extreme changes in height on the terrain, the final result can be hard to understand. The generated image also needs to stay at the same scale as the heightmap. Water sources tend to be at lower points in localized terrain areas, thus most of the pixels appear to be visual indiscernible shades of black. Figure



(A) (B)

FIGURE C.1: Figures showcasing the generated floodmap (a) and the result of using the floodmap to generate environment resources (b). Figure (a) is supposed to be mostly black (See Section C.3). In Figure (b), darker pixels indicate resource abundance and lighter cells indicate resource scarcity.

C.1b shows the accuracy of this approach by simulating the expected resource density. As can be seen, most of the Nile Valley and Delta are correctly resource dense and the surrounding areas resource scarce. It should be noted that this process is not perfect. Figure C.1b does have artefacts and a separate "is water" mask must be used to ensure that sea water does not contribute to resource abundance.

C.4 Soil Texture Maps

The soil-clay content maps were sourced from the GLDAS project [159]. The process for doing so was as follows:

1. Download the GLDAS Soil Land Surface data as netcdf files from: `https://ldas.gsfc.nasa.gov/gldas/soils`.

2. Convert the netcdf files to geotiffs using the *GLDAS_TO_GEOTIFF.py* script.

3. Manually convert the Sand and Clay GEOTIFFS into PNGS.

4. Manually downscale the image to the appropriate image size (750 by 300 pixels in our case).

Appendix D

Supplementary Experiments

In this Chapter, we summarize the preliminary experiments we ran to further understand the dynamics of *NeoCOOP* early on in its development. Section D.1 describes how we designed the experiments and Section D.2 reports our findings.

D.1 Experiment Design

One of the main challenges we encountered during development pertained to the mechanism(s) we were going use to simulate global environment data. As noted in Section 4.2, data from the Predynastic period does not exist and an alternative would need to be found. The approach we took was partly inspired by work by Molin et al. [24] and the Food-for-All model [157]. These authors generated their "climate data" using functions and thresholds respectively. We decided to blend these two approaches together as described in Section 4.2. The final part of this process required us to choose an appropriate interpolator function s. Four such functions were identified and described as follows: Linear (Equation D.1), Sinusoid (Equation D.3), Dampended Sinusoid (Equation D.2) and Linear Dampened Sinusoid (Equation D.4) where t is the timestep, T is the denominator for the linear function, f is the frequency for the sinusoids, k is an exponential constant and m is the gradient.

$$s_1(t) = \frac{t}{T} \quad \text{(D.1)} \qquad s_2(t) = 0.5 + 0.5sin(\frac{2\pi \times t}{f}) \quad \text{(D.3)}$$

$$s_3(t) = s_2(t) \times e^{-kt} \quad \text{(D.2)} \qquad s_4(t) = s_3(t) + (m(1-t)(1-s_3(t))) \quad \text{(D.4)}$$

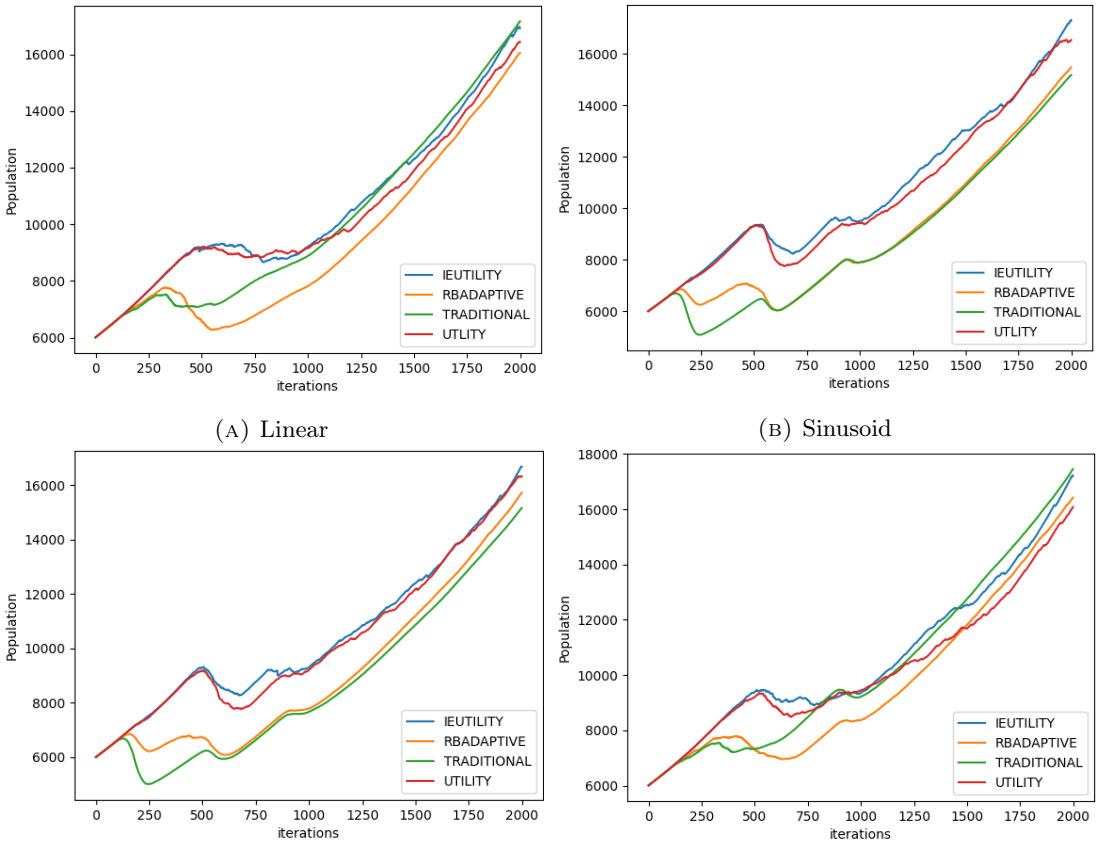
The motivation for choosing these functions pertained to the cyclic nature of dry and wet seasons [162] simulated by the sinusoids and the generally increasing arid conditions simulated by the linear components of Equations D.1 and D.4. We conducted scenario experimentation under the same conditions of our main experiments (50 repetitions for each agent for each scenario resulting in 800 simulation runs).

Note: The model's parameters had not been finalized (we were still using Households with individuals and stylized data-maps as opposed to the satellite GIS data used in our final experiments for example). Additionally, the simulation length was only 2000 iterations instead of the 2500 iterations we settled for in our main experiments. Similarly, the agents had not been parameter tuned completely so we could not make concrete claims about the adaptive capacity of the agent-types. The main purpose of these experiments was to ascertain which interpolator would be worth using and to identify if there were glaring issues with the agent implementations.

D.2 Results and Discussion

Figure D.1 plots the final Individual population level for the scenarios explored. For the most part, all scenarios investigated were capable of generating sufficient environmental stress that would result in population loss and adaptation. Surprisingly, the ML agents were able to maintain similar population levels regardless of the scenario. We initially believed this may have been luck due to our domain knowledge of the algorithms used as their adaptive mechanisms but, after conducting our main experiments, we now know that this is because the ML agents are able to optimize themselves to a variety of scenarios across a range of parameter values. Conversely, the *RBAdaptive* agent was the poorest performer across all scenarios which is what prompted us to scale the input parameters of the agent-type from its original *OMOLAND* model implementation.

Interestingly, the rule-based agent-types seemingly caught the ML agents in their final population levels towards the end of the simulation runs. Further investigation revealed that this was due to the rule-based agents exploiting the resource protection offered by Nile Delta. In fact, some experiments had 95% of the rule-based agents situated in the Delta. This result was almost entirely the result of using stylized data-maps as, when we used the satellite GIS data, this behaviour all but disappeared.



(A) Linear

(B) Sinusoid

(C) Dampened Sinusoid

(D) Linear Dampened Sinusoid

FIGURE D.1: Final Individual population levels for the exploratory experiments.

In the end, we opted to use the sinusoid for our final experiments. The motivations were three-fold. First, sinusoids were used by Molin et al. [24] so there was already a precedence set. Similarly, our results showed that all functions could induce population-based stress so there was no reason to against the status-quo. Lastly, Hassan [162] described the Nile floods as seasonal (behaviour captured by the sinusoid interpolator). Yes there was likely to have been a gradual change towards more arid conditions which the linear function could capture but, it is not as if Egypt was a tropical paradise before the Neolithic. During the time period we were interested in modelling, Egypt was always semi or extremely arid.

Bibliography

[1] Iza Romanowska. So you think you can model? a guide to building and evaluating archaeological simulation models of dispersals. *Human biology*, 87(3):169–192, 2015.

[2] Bernd Schmidt. *The modelling of human behaviour: The PECS reference models*. SCS-Europe BVBA Delft, 2000.

[3] Christoph Urban and Bernd Schmidt. Pecs–agent-based modelling of human behaviour. In *Emotional and Intelligent–The Tangled Knot of Social Cognition, AAAI Fall Symposium Series, North Falmouth, MA. www. or. unipassau. de/5/publik/urban/CUrban01. pdf*, 2001.

[4] William Rand. Machine learning meets agent-based modeling: when not to go to a bar. In *Conference on Social Agents: Results and Prospects*, 2006.

[5] Robert G Reynolds, Mostafa Ali, and Thaer Jayyousi. Mining the social fabric of archaic urban centers with cultural algorithms. *Computer*, 41(1):64–72, 2008.

[6] Lars I Hatledal, Yingguang Chu, Arne Styve, and Houxiang Zhang. Vico: An entity-component-system based co-simulation framework. *Simulation Modelling Practice and Theory*, 108:102243, 2021.

[7] Eric Tatara, M North, T Howe, N Collier, Jerry Vos, et al. An introduction to repast simphony modeling using a simple predator-prey example. In *Proceedings of the Agent 2006 Conference on Social Agents: Results and Prospects*, pages 83–93. Argonne National Laboratory and The University of Chicago, 2006.

[8] Alice Stevenson. The egyptian predynastic and state formation. *Journal of Archaeological Research*, 24(4):421–468, 2016.

[9] William G Kennedy. Modelling human behaviour in agent-based models. In *Agent-based models of geographical systems*, pages 167–179. Springer, 2012.

[10] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[11] Douglas James Brewer and Emily Teeter. *Egypt and the Egyptians*. Cambridge University Press, 1999.

[12] Branislav Anđelković. Political organization of egypt in the predynastic period. *Before the Pyramids: The Origins of Egyptian Civilization, ed. Emily Teeter*, pages 25–32, 2011.

[13] Charles M Macal. Everything you need to know about agent-based modelling and simulation. *Journal of Simulation*, 10(2):144–156, 2016.

[14] Mark Pogson, Rod Smallwood, Eva Qwarnstrom, and Mike Holcombe. Formal agent-based modelling of intracellular chemical interactions. *Biosystems*, 85(1): 37–45, 2006.

[15] Enrique Frias-Martinez, Graham Williamson, and Vanessa Frias-Martinez. An agent-based model of epidemic spread using human mobility and social network information. In *2011 IEEE third international conference on privacy, security, risk and trust and 2011 IEEE third international conference on social computing*, pages 57–64. IEEE, 2011.

[16] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the national academy of sciences*, 99(suppl 3): 7280–7287, 2002.

[17] Gary Bogle and Claudio Cioffi-Revilla. Zambeziland: A canonical theory and agent-based model of polity cycling in the zambezi plateau, southern africa. In *Simulating prehistoric and ancient worlds*, pages 359–375. Springer, 2016.

[18] Mark W Lake. Magical computer simulation of mesolithic foraging. *Dynamics in human and primate societies: agent-based modelling of social and spatial processes*, pages 107–143, 2000.

[19] Shawn Graham and Scott Weingart. The equifinality of archaeological networks: an agent-based exploratory lab approach. *Journal of Archaeological Method and Theory*, 22(1):248–274, 2015.

[20] Wendy H Cegielski and J Daniel Rogers. Rethinking the role of agent-based modeling in archaeology. *Journal of Anthropological Archaeology*, 41:283–298, 2016.

[21] Helen Couclelis. Modeling frameworks, paradigms, and approaches. *Geographic information systems and environmental modeling*, pages 36–50, 2002.

[22] Hazel R Parry and Mike Bithell. Large scale agent-based modelling: A review and guidelines for model scaling. In *Agent-based models of geographical systems*, pages 271–308. Springer, 2012.

[23] Erik Cuevas. An agent-based model to evaluate the covid-19 transmission risks in facilities. *Computers in biology and medicine*, 121:103827, 2020.

[24] Lara Dal Molin, Jasmeen Kanwal, and Christopher Stone. Resource availability and the evolution of cooperation in a 3d agent-based simulation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 93–101, 2021.

[25] Angelos Chliaoutakis and Georgios Chalkiadakis. Agent-based modeling of ancient societies and their organization structure. *Autonomous Agents and Multi-Agent Systems*, 30(6):1072–1116, 2016.

[26] David J Buller. *Adapting minds: Evolutionary psychology and the persistent quest for human nature*. MIT press, 2006.

[27] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[28] Michael W Macy and John Skvoretz. The evolution of trust and cooperation between strangers: A computational model. *American Sociological Review*, pages 638–660, 1998.

[29] Michel Jules Dreyfus-León. Individual-based modelling of fishermen search behaviour with neural networks and reinforcement learning. *Ecological Modelling*, 120(2-3):287–297, 1999.

[30] Forrest Stonedahl and Uri Wilensky. Finding forms of flocking: Evolutionary search in abm parameter-spaces. In *International workshop on multi-agent systems and agent-based simulation*, pages 61–75. Springer, 2010.

[31] Fang Yuan Xu, Xue Wang, Loi Lei Lai, and Chun Sing Lai. Agent-based modeling and neural network for residential customer demand response. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1312–1316. IEEE, 2013.

[32] Cornelis J Drost and Marc Vander Linden. Toy story: Homophily, transmission and the use of simple simulation models for assessing variability in the archaeological record. *Journal of Archaeological Method and Theory*, 25(4):1087–1108, 2018.

[33] Lukas Egli, Hanna Weise, Viktoriia Radchuk, Ralf Seppelt, and Volker Grimm. Exploring resilience with agent-based models: state of the art, knowledge gaps and recommendations for coping with multidimensionality. *Ecological Complexity*, 40: 100718, 2019.

[34] Gary Bogle. *Polity Cycling in Great Zimbabwe via Agent-Based Modeling: The Effects of Timing and Magnitude of External Factors*. PhD thesis, George Mason University, 2019.

[35] Lynne M Rouse and Lloyd Weeks. Specialization and social inequality in bronze age se arabia: analyzing the development of production strategies and economic networks using agent-based modeling. *Journal of Archaeological Science*, 38(7): 1583–1590, 2011.

[36] Robert C Allen. Agriculture and the origins of the state in ancient egypt. *Explorations in Economic History*, 34(2):135–154, 1997.

[37] Nathan L Engle. Adaptive capacity and its assessment. *Global environmental change*, 21(2):647–656, 2011.

[38] Scott Heckbert. Mayasim: an agent-based model of the ancient maya social-ecological system. *Journal of Artificial Societies and Social Simulation*, 16(4): 11, 2013.

[39] Maja Schlüter and Claudia Pahl-Wostl. Mechanisms of resilience in common-pool resource management systems: an agent-based model of water use in a river basin. *Ecology and Society*, 12(2), 2007.

[40] Angelos Chliaoutakis, Georgios Chalkiadakis, et al. An agent-based model for simulating inter-settlement trade in past societies. *Journal of Artificial Societies and Social Simulation*, 23(3):1–10, 2020.

[41] Brandon Gower-Winter and Geoff Nitschke. Do harsher environments cause selfish or altruistic behavior? In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion))*, July 2022.

[42] Brandon Gower-Winter and Geoff Nitschke. Societies prefer the middle-ground between selfishness and cooperation. In *4th International Workshop on Agent-Based Modelling of Human Behaviour (ABMHuB'22)*, 2022.

[43] Brandon Gower-Winter and Geoff Nitschke. Extreme environments perpetuate cooperation. In *IEEE Symposium Series On Computational Intelligence (SSCI'22)*, 2022.

[44] Simon Levin. Complex adaptive systems: exploring the known, the unknown and the unknowable. *Bulletin of the American Mathematical Society*, 40(1):3–19, 2003.

[45] Andrew T Crooks and Alison J Heppenstall. Introduction to agent-based modelling. In *Agent-based models of geographical systems*, pages 85–105. Springer, 2012.

[46] Joshua M Epstein and Robert Axtell. *Growing artificial societies: social science from the bottom up*. Brookings Institution Press, 1996.

[47] Nigel Gilbert. *Agent-based models*, volume 153. Sage Publications, Incorporated, 2019.

[48] Margaret A Boden. *Computer models of mind: Computational approaches in theoretical psychology*. Cambridge University Press, 1988.

[49] Christian JE Castle and Andrew T Crooks. Principles and concepts of agent-based modelling for developing geospatial simulations. 2006.

[50] Armano Srbljinović and Ognjen Škunca. An introduction to agent based modelling and simulation of social processes. *Interdisciplinary Description of Complex Systems: INDECS*, 1(1-2):1–8, 2003.

[51] Joshua M Epstein. *Generative social science: Studies in agent-based computational modeling*. Princeton University Press, 2006.

[52] Lars-Erik Cederman. *Emergent actors in world politics: how states and nations develop and dissolve*, volume 2. Princeton University Press, 1997.

[53] Cedric Perret, Emma Hart, and Simon T Powers. From disorganized equality to efficient hierarchy: how group size drives the evolution of hierarchy in human societies. *Proceedings of the Royal Society B*, 287(1928):20200693, 2020.

[54] Joshua M Epstein. Remarks on the foundations of agent-based generative social science. *Handbook of computational economics*, 2:1585–1604, 2006.

[55] Joshua M Epstein. Agent-based computational models and generative social science. *Complexity*, 4(5):41–60, 1999.

[56] Mark W Lake. Explaining the past with abm: on modelling philosophy. In *Agent-based modeling and simulation in archaeology*, pages 3–35. Springer, 2015.

[57] Eugene Ch'ng. Using games engines for archaeological visualisation: Recreating lost worlds. In *11th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games, CGames*, volume 7, pages 26–30, 2007.

[58] Steven M Manson. Agent-based modeling and genetic programming for modeling land change in the southern yucatan peninsular region of mexico. *Agriculture, ecosystems & environment*, 111(1-4):47–62, 2005.

[59] C Michael Barton. Complexity, social complexity, and modeling. *Journal of Archaeological Method and Theory*, 21(2):306–324, 2014.

[60] Timothy A Kohler, R Kyle Bocinsky, Denton Cockburn, Stefani A Crabtree, Mark D Varien, Kenneth E Kolm, Schaun Smith, Scott G Ortman, and Ziad Kobti. Modelling prehispanic pueblo societies in their ecosystems. *Ecological Modelling*, 241:30–41, 2012.

[61] Gaku Yamamoto. Agent server technology for managing millions of agents. In *International Workshop on Massively Multiagent Systems*, pages 1–12. Springer, 2004.

[62] Hazel R Parry and Andrew J Evans. A comparative analysis of parallel processing and super-individual methods for improving the computational performance of a large individual-based model. *Ecological Modelling*, 214(2-4):141–152, 2008.

[63] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K Heinz, Geir Huse, et al. A standard protocol for describing individual-based and agent-based models. *Ecological modelling*, 198(1-2):115–126, 2006.

[64] Volker Grimm, Uta Berger, Donald L DeAngelis, J Gary Polhill, Jarl Giske, and Steven F Railsback. The odd protocol: a review and first update. *Ecological modelling*, 221(23):2760–2768, 2010.

[65] Volker Grimm, Gary Polhill, and Julia Touza. Documenting social simulation models: the odd protocol as a standard. In *Simulating social complexity*, pages 349–365. Springer, 2017.

[66] Birgit Müller, Friedrich Angermueller, Romina Drees, Gunnar Dressler, Jürgen Groeneveld, Christian Klassert, Maja Schlüter, Jule Schulze, Hanna Weise, and Nina Schwarz. Describing human decisions in agent-based social-ecological models-odd+ d an extension of the odd protocol. *Available at SSRN 2044736*, 2012.

[67] Claudio Cioffi-Revilla and Mark Rouleau. Mason rebeland: An agent-based model of politics, environment, and insurgency. *International Studies Review*, 12(1):31–52, 2010.

[68] Gianluca Manzo and Toby Matthews. Potentialities and limitations of agent-based simulations. *Revue française de sociologie*, 55(4):653–688, 2014.

[69] Volker Grimm, Eloy Revilla, Uta Berger, Florian Jeltsch, Wolf M Mooij, Steven F Railsback, Hans-Hermann Thulke, Jacob Weiner, Thorsten Wiegand, and Donald L DeAngelis. Pattern-oriented modeling of agent-based complex systems: lessons from ecology. *science*, 310(5750):987–991, 2005.

[70] Jeffrey S Dean, George J Gumerman, and Joshua M Epstein. Understanding anasazi culture change through agent-based modeling. *Dynamics in human and primate societies: Agent-based modeling of social and spatial processes*, pages 179–205, 2000.

[71] Friedrich Recknagel. Applications of machine learning to ecological modelling. *Ecological modelling*, 146(1-3):303–310, 2001.

[72] Claudio Cioffi-Revilla. A methodology for complex social simulations. *Journal of Artificial Societies and Social Simulation*, 13(1):7, 2010.

[73] Bruce Hannon and Matthias Ruth. *Dynamic modeling*. Springer Science & Business Media, 2001.

[74] Anand S Rao, Michael P Georgeff, et al. Bdi agents: from theory to practice. In *ICMAS*, volume 95, pages 312–319, 1995.

[75] Sander van der Hoog. Deep learning in (and of) agent-based models: A prospectus. *arXiv preprint arXiv:1706.06302*, 2017.

[76] Benoît Calvez and Guillaume Hutzler. Automatic tuning of agent-based models using genetic algorithms. In *International Workshop on Multi-Agent Systems and Agent-Based Simulation*, pages 41–57. Springer, 2005.

[77] Ronald C Arkin, Ronald C Arkin, et al. *Behavior-based robotics*. MIT press, 1998.

[78] Wenwu Tang. Simulating complex adaptive geographic systems: A geographically aware intelligent agent approach. *Cartography and Geographic Information Science*, 35(4):239–263, 2008.

[79] Christophe Deissenberg, Sander Van Der Hoog, and Herbert Dawid. Eurace: A massively parallel agent-based model of the european economy. *Applied Mathematics and Computation*, 204(2):541–552, 2008.

[80] M Scheutz, P Schermerhorn, R Connaughton, and A Dingler. Swages-an extendable distributed experimentation system for large-scale agent-based alife simulations. *Proceedings of Artificial Life X*, pages 412–419, 2006.

[81] Nicholson Collier and Michael North. Repast hpc: A platform for large-scale agent-based modeling. *Large-Scale Computing*, pages 81–109, 2012.

[82] Francesco Lamperti, Andrea Roventini, and Amir Sani. Agent-based model calibration using machine learning surrogates. *Journal of Economic Dynamics and Control*, 90:366–389, 2018.

[83] Alexander Forrester, Andras Sobester, and Andy Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[84] Graupe Daniel. *Principles of artificial neural networks*, volume 7. World Scientific, 2013.

[85] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[86] David J Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989.

[87] Wenchao Yi, Jinghui Zhong, Singkuang Tan, Wentong Cai, and Nan Hu. Surrogate assisted calibration framework for crowd model calibration. In *2017 Winter Simulation Conference (WSC)*, pages 1216–1227. IEEE, 2017.

[88] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[89] Yifeng Zhang and Siddhartha Bhattacharyya. Effectiveness of q-learning as a tool for calibrating agent-based supply network models. *Enterprise Information Systems*, 1(2):217–233, 2007.

[90] Ammar Jalalimanesh, Hamidreza Shahabi Haghighi, Abbas Ahmadi, Hossein Hejazian, and Madjid Soltani. Multi-objective optimization of radiotherapy: distributed q-learning and agent-based simulation. *Journal of ExpErimEntal & thEorEtical artificial intElligEncE*, 29(5):1071–1086, 2017.

[91] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[92] David E Edward Goldberg. Genetic algorithms in search, optimization & machine learning. 1989.

[93] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[94] Claudio Cioffi-Revilla, Kenneth De Jong, and Jeffrey K Bassett. Evolutionary computation and agent-based modeling: biologically-inspired approaches for understanding complex social systems. *Computational and Mathematical Organization Theory*, 18(3):356–373, 2012.

[95] Francis Oloo and Gudrun Wallentin. An adaptive agent-based model of homing pigeons: A genetic algorithm approach. *ISPRS International Journal of Geo-Information*, 6(1):27, 2017.

[96] Robert G Reynolds. An introduction to cultural algorithms. In *Proceedings of the third annual conference on evolutionary programming*, pages 131–139. World Scientific, 1994.

[97] Chan-Jin Chung and Robert G Reynolds. A testbed for solving optimization problems using cultural algorithms. In *Evolutionary programming*, pages 225–236, 1996.

[98] Mahamed GH Omran. A novel cultural algorithm for real-parameter optimization. *International Journal of Computer Mathematics*, 93(9):1541–1563, 2016.

[99] Kaiqiao Yang, Kenjiro Maginu, and Hirosato Nomura. Cultural algorithm-based quantum-behaved particle swarm optimization. *International journal of computer mathematics*, 87(10):2143–2157, 2010.

[100] Ziad Kobti, Robert G Reynolds, and Tim Kohler. Agent-based modeling of cultural change in swarm using cultural algorithms. *Funded by NSF grant BCS-0119981*, 2004.

[101] Robert G Reynolds and Mostafa Z Ali. The social fabric approach as an approach to knowledge integration in cultural algorithms. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 4200–4207. IEEE, 2008.

[102] Robert G Reynolds and Bin Peng. Cultural algorithms: Modeling of how cultures learn to solve problems. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*, pages 166–172, 2004.

[103] Colin D WRen, Chloe ATWATeR, Kim Hill, Marco Janssen, Jan de Vynck, and Curtis W MAReAn. An agent-based approach to weighted decision making in the spatially and temporally variable south african palaeoscape. 2019.

[104] Carolina Cucart-Mora, Sergi Lozano, and Javier Fernández-López de Pablo. Bio-cultural interactions and demography during the middle to upper palaeolithic transition in iberia: An agent-based modelling approach. *Journal of Archaeological Science*, 89:14–24, 2018.

[105] Sonia Schulenburg and Peter Ross. An adaptive agent based economic model. In *International Workshop on Learning Classifier Systems*, pages 263–282. Springer, 1999.

[106] Bing Zhang, Yongliang Zhang, and Jun Bi. An adaptive agent-based modeling approach for analyzing the influence of transaction costs on emissions trading markets. *Environmental Modelling & Software*, 26(4):482–491, 2011.

[107] Robert N Bernard et al. Using adaptive agent-based simulation models to assist planners in policy development: The case of rent control. *Rutgers University, Department of Urban Planning and Policy Development*, 1999.

[108] Blake LeBaron. Agent-based computational finance: Suggested readings and early research. *Journal of Economic Dynamics and Control*, 24(5-7):679–702, 2000.

[109] Josh C Bongard. Evolutionary robotics. *Communications of the ACM*, 56(8):74–83, 2013.

[110] Marco A Janssen, Lilian Na'ia Alessa, Michael Barton, Sean Bergin, and Allen Lee. Towards a community framework for agent-based modelling. *Journal of Artificial Societies and Social Simulation*, 11(2):6, 2008.

[111] Robert Axelrod. *The complexity of cooperation: Agent-based models of competition and collaboration*, volume 3. Princeton University Press, 1997.

[112] Bruce Edmonds and Scott Moss. From kiss to kids–an 'anti-simplistic'modelling approach. In *International workshop on multi-agent systems and agent-based simulation*, pages 130–144. Springer, 2004.

[113] Federico Bianchi, Francisco Grimaldo, Giangiacomo Bravo, and Flaminio Squazzoni. The peer review game: an agent-based model of scientists facing resource constraints and institutional pressures. *Scientometrics*, 116(3):1401–1420, 2018.

[114] James Andreoni and John H Miller. Auctions with artificial adaptive agents. *Games and economic behavior*, 10(1):39–64, 1995.

[115] David A Ostrowski, Troy Tassier, Mark Everson, and Robert G Reynolds. Using cultural algorithms to evolve strategies in agent-based models. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*, volume 1, pages 741–746. IEEE, 2002.

[116] Lei Xue, Changyin Sun, Donald Wunsch, Yingjiang Zhou, and Fang Yu. An adaptive strategy via reinforcement learning for the prisoner's dilemma game. *IEEE/-CAA Journal of Automatica Sinica*, 5(1):301–310, 2017.

[117] Iza Romanowska, Stefani Crabtree, Kathryn Harris, and Benjamin Davies. Agent-based modeling for archaeologists: Part 1 of 3. 2019.

[118] Benjamin Davies, Iza Romanowska, Kathryn Harris, and Stefani A Crabtree. Combining geographic information systems and agent-based models in archaeology: Part 2 of 3. *Advances in Archaeological Practice*, 7(2):185, 2019.

[119] Stefani A Crabtree, Kathryn Harris, Benjamin Davies, and Iza Romanowska. Outreach in archaeology with agent-based modeling: Part 3 of 3. *Advances in Archaeological Practice*, 7(2):194, 2019.

[120] Noel Llopis. Data-oriented design (or why you might be shooting yourself in the foot with oop), Dec 2009. URL `https://gamesfromwithin.com/data-oriented-design`.

[121] Timo Szczepanska, Patrycja Antosz, Jan Ole Berndt, Melania Borit, Edmund Chattoe-Brown, Sara Mehryar, Ruth Meyer, Stephan Onggo, and Harko Verhagen. Gam on! six ways to explore social complexity by combining games and agent-based models. *International Journal of Social Research Methodology*, pages 1–15, 2022.

[122] Thibault Raffaillac and Stéphane Huot. Polyphony: Programming interfaces and interactions with the entity-component-system model. *Proceedings of the ACM on Human-Computer Interaction*, 3(EICS):1–22, 2019.

[123] Carl Folke, Stephen R Carpenter, Brian Walker, Marten Scheffer, Terry Chapin, and Johan Rockström. Resilience thinking: integrating resilience, adaptability and transformability. *Ecology and society*, 15(4), 2010.

[124] Andy Pike, Stuart Dawley, and John Tomaney. Resilience, adaptation and adaptability. *Cambridge journal of regions, economy and society*, 3(1):59–70, 2010.

[125] Maria Chli and Philippe De Wilde. The emergence of knowledge exchange: An agent-based model of a software market. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(5):1056–1067, 2008.

[126] Ben Fitzhugh, S Colby Phillips, and Erik Gjesfjeld. Modeling hunter-gatherer information networks: an archaeological case study from the kuril islands. *Information and its role in hunter-gatherer bands*, pages 85–115, 2011.

[127] Matteo Giuliani and A Castelletti. Assessing the value of cooperation and information exchange in large water resources systems by agent-based optimization. *Water Resources Research*, 49(7):3912–3926, 2013.

[128] Ron Sun. The importance of cognitive architectures: An analysis based on clarion. *Journal of Experimental & Theoretical Artificial Intelligence*, 19(2):159–193, 2007.

[129] Wander Jager and Marco Janssen. An updated conceptual framework for integrated modeling of human decision making: The consumat ii. In *paper for workshop complexity in the Real World@ ECCS*, pages 1–18, 2012.

[130] Carlos Miguel Reis Silva de Oliveira e Lemos. *On Agent-Based Modelling of Large Scale Conflict Against a Central Authority: from Mechanisms to Complex Behaviour*. PhD thesis, Instituto Universitário de Lisboa, 12 2016.

[131] Elva JH Robinson, Francis LW Ratnieks, and M Holcombe. An agent-based model to investigate the roles of attractive and repellent pheromones in ant decision making during foraging. *Journal of theoretical Biology*, 255(2):250–258, 2008.

[132] Shengxiang Yang, Yong Jiang, and Trung Thanh Nguyen. Metaheuristics for dynamic combinatorial optimization problems. *IMA Journal of Management Mathematics*, 24(4):451–480, 2013.

[133] H Van Dyke Parunak et al. " go to the ant": Engineering principles from natural multi-agent systems. *Annals of Operations Research*, 75:69–101, 1997.

[134] Kelly Rendón Rozo, Julian Arellana, Alcides Santander-Mercado, and Maria Jubiz-Diaz. Modelling building emergency evacuation plans considering the dynamic behaviour of pedestrians using agent-based simulation. *Safety science*, 113:276–284, 2019.

[135] Jeff Jones. Characteristics of pattern formation and evolution in approximations of physarum transport networks. *Artificial life*, 16(2):127–153, 2010.

[136] Taylor M Anderson and Suzana Dragićević. Network-agent based model for simulating the dynamic spatial network structure of complex ecological systems. *Ecological Modelling*, 389:19–32, 2018.

[137] Stan Hendrickx. Predynastic—early dynastic chronology. In *Ancient Egyptian Chronology*, pages 53–93. Brill, 2006.

[138] Fekri A Hassan. The predynastic of egypt. *Journal of World Prehistory*, 2(2): 135–185, 1988.

[139] Barry J Kemp. *Ancient Egypt: anatomy of a civilisation*. Routledge, 2007.

[140] Kent V Flannery. Process and agency in early state formation. *Cambridge Archaeological Journal*, 9(1):3–21, 1999.

[141] Robert L Carneiro. The chiefdom: precursor of the state. *The transition to statehood in the New World*, pages 37–79, 1981.

[142] Henry T Wright and DJ Meltzer. The evolution of civilizations. *IN American Archaeology Past and Future. DJ*, 1986.

[143] Mark Lehner. Fractal house of pharaoh: Ancient egypt as a complex adaptive system, a trial. *Dynamics in human and primate societies: Agent-based modelling of social and spatial processes*, pages 275–353, 2000.

[144] Sarah Symons and Derek Raine. Agent-based models of ancient egypt. *Proceedings of Informatique et Égyptologie. Piscataway, NJ. http://www. physics. le. ac. uk/ComplexSystems/papers/AgentBasedModelsEgypt2008. pdf (accessed 28/5/12)*, 2008.

[145] Geoff Nitschke, Jessica Nitschke, Alexander Furman, and Matthew Cherry. Modeling patterns of wealth disparity in predynastic upper egypt. In *Artificial Life Conference Proceedings 14*, pages 322–323. MIT Press, 2017.

[146] Robert Axelrod. The dissemination of culture: A model with local convergence and global polarization. *Journal of conflict resolution*, 41(2):203–226, 1997.

[147] Atesmachew Hailegiorgis, Andrew Crooks, and Claudio Cioffi-Revilla. An agent-based model of rural households' adaptation to climate change. *Journal of Artificial Societies and Social Simulation*, 21(4), 2018.

[148] James E Maddux and Ronald W Rogers. Protection motivation and self-efficacy: A revised theory of fear appeals and attitude change. *Journal of experimental social psychology*, 19(5):469–479, 1983.

[149] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[150] Duanyang Xu, Alin Song, Hefeng Tong, Hongyan Ren, Yunfeng Hu, and Quanqin Shao. A spatial system dynamic model for regional desertification simulation–a case study of ordos, china. *Environmental Modelling & Software*, 83:179–192, 2016.

[151] Charles Warren Thornthwaite. An approach toward a rational classification of climate. *Geographical review*, 38(1):55–94, 1948.

[152] Christopher S Potter, James T Randerson, Christopher B Field, Pamela A Matson, Peter M Vitousek, Harold A Mooney, and Steven A Klooster. Terrestrial ecosystem production: a process model based on global satellite and surface data. *Global Biogeochemical Cycles*, 7(4):811–841, 1993.

[153] KE Saxton, W_J Rawls, J Sv Romberger, and RI Papendick. Estimating generalized soil-water characteristics from texture. *Soil science society of America Journal*, 50(4):1031–1036, 1986.

[154] Christopher B Field, James T Randerson, and Carolyn M Malmström. Global net primary production: combining ecology and remote sensing. *Remote sensing of Environment*, 51(1):74–88, 1995.

[155] Rimjhim Aggarwal, Sinaia Netanyahu, and Claudia Romano. Access to natural resources and the fertility decision of women: the case of south africa. *Environment and Development Economics*, 6(2):209–236, 2001.

[156] Peter Bellwood and Marc Oxenham. The expansions of farming societies and the role of the neolithic demographic transition. In *The Neolithic demographic transition and its consequences*, pages 13–34. Springer, 2008.

[157] Andreas Angourakis, José Ignacio Santos, José Manuel Galán, and Andrea L Balbo. Food for all: An agent-based model to explore the emergence and implications of cooperation for food storage. *Environmental Archaeology*, 20(4):349–363, 2015.

[158] T Tadono, H Ishida, F Oda, S Naito, K Minakawa, and H Iwamoto. Precise global dem generation by alos prism. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(4):71, 2014.

[159] Matthew Rodell, PR Houser, UEA Jambor, J Gottschalck, Kieran Mitchell, C-J Meng, Kristi Arsenault, B Cosgrove, J Radakovich, M Bosilovich, et al. The global land data assimilation system. *Bulletin of the American Meteorological society*, 85 (3):381–394, 2004.

[160] EM Gallagher. *Evolutionary Models for the Origins of Agriculture*. PhD thesis, UCL (University College London), 2017.

[161] Barbara Bell. The oldest records of the nile floods. *The Geographical Journal*, 136 (4):569–573, 1970.

[162] Fekri A Hassan. The dynamics of a riverine civilization: a geoarchaeological perspective on the nile valley, egypt. *World archaeology*, 29(1):51–74, 1997.

[163] Guus Ten Broeke, George Van Voorn, and Arend Ligtenberg. Which sensitivity analysis method should i use for my agent-based model? *Journal of Artificial Societies and Social Simulation*, 19(1):5, 2016.

[164] Magdalena Maria Nowak et al. Results of the preliminary analysis of lower egyptian settlement discovered on the central kom in tell el-farkha. *Studies in Ancient Art and Civilization*, (15):49–63, 2011.

[165] Ofer Bar-Yosef and Anna Belfer-Cohen. The origins of sedentism and farming communities in the levant. *Journal of world prehistory*, 3(4):447–498, 1989.

[166] Wei Zhang, Andrea Valencia, and Ni-Bin Chang. Synergistic integration between machine learning and agent-based modeling: A multidisciplinary review. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[167] Minae Kwon, Erdem Biyik, Aditi Talati, Karan Bhasin, Dylan P Losey, and Dorsa Sadigh. When humans aren't optimal: Robots that collaborate with risk-aware humans. In *2020 15th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 43–52. IEEE, 2020.

[168] Milica Kašanin-Grubin, Francesca Vergari, Francesco Troiani, and Marta Della Seta. The role of lithology: Parent material controls on badland development. In *Badlands Dynamics in a Context of Global Change*, pages 61–109. Elsevier, 2018.

[169] L Shanan, NH Tadmor, M Evenari, and P Reiniger. Runoff farming in the desert. iii. microcatchments for improvement of desert range 1. *Agronomy Journal*, 62(4): 445–449, 1970.