January 2020

Catgame: A Tool For Problem Solving In Complex Dynamic Systems Using Game Theoretic Knowledge Distribution In Cultural Algorithms, And Its Application (catneuro) To The Deep Learning Of Game Controller

Faisal Waris
Wayne State University

CATGAME: A TOOL FOR PROBLEM SOLVING IN COMPLEX DYNAMIC SYSTEMS USING GAME THEORETIC KNOWLEDGE DISTRIBUTION IN CULTURAL ALGORITHMS, AND ITS APPLICATION (CATNEURO) TO THE DEEP LEARNING OF GAME CONTROLLERS

by

FAISAL WARIS

DISSERTATION

Submitted to the Graduate School of

Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2020

MAJOR: COMPUTER SCIENCE

Approved By:

_____

Advisor Date

_____

_____

_____

_____

DEDICATION

I dedicate this work to my family for their unending support.

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Reynolds for his continual guidance and inspiration. And my committee, Drs Schwiebert, Hua and Chinnam for their feedback. I am also thankful to my wife Ayesha and my two daughters Anam and Tasmia for their patience, understanding and support. The General Motors corporation provided essential financial support. I would like thank my directors at GM, Tom Capotosto and Tom Piegat for their encouragement and flexibility. I have been enlightened by each of my teachers throughout the years and enriched by the interaction with my colleagues in the various places I have worked especially GM and Ford Motor; I am deeply grateful to all.

“Very little grows on jagged rock. Be ground. Be crumbled, so wildflowers will come up where you are.”

— Rumi

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.0 Introduction

The world is changing at an ever-increasing pace around us. The rise in speed of communication and global social connectivity (Figure 1-2); the ubiquitous personal computing through smart phones; and human-level task performance by machines (e.g. autonomous vehicles) etc. are creating unprecedented challenges and new opportunities for society. However, human civilization has survived for thousands of years, relying on social and cultural adaption across myriad periods of duress. One expects that the resiliency and robustness of human kind also will help the species adapt to the current and future challenges.

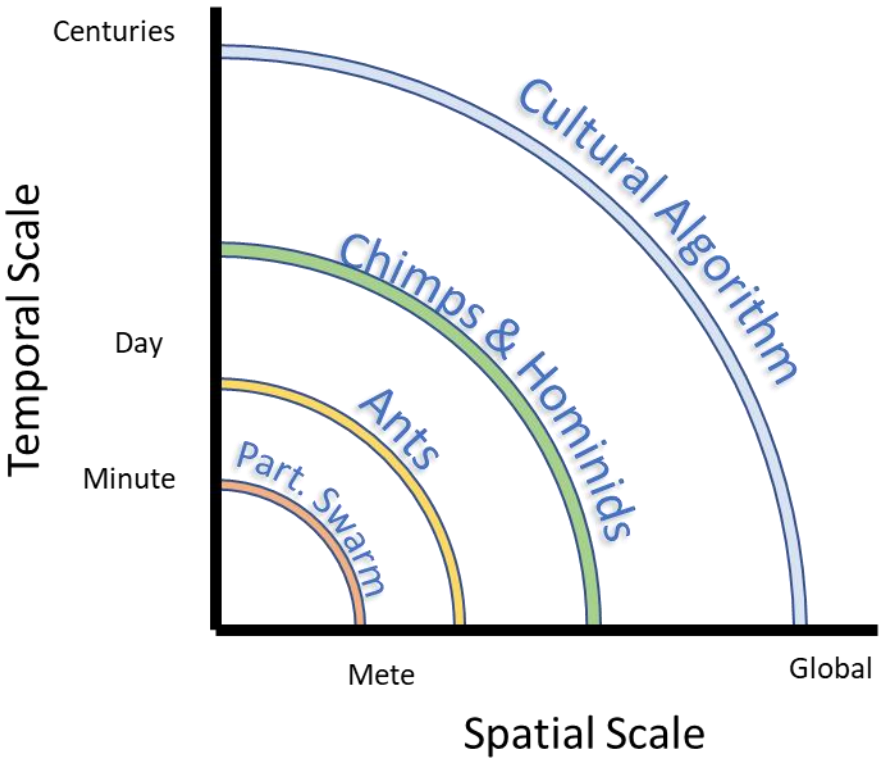
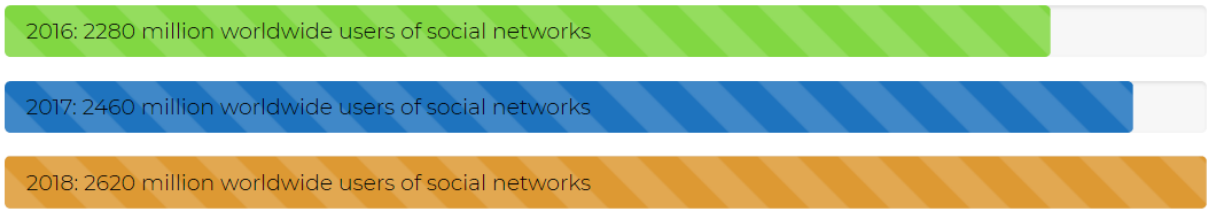


Figure 1-1 Scales of social interaction. The emergent properties depend on the scale at which the interaction takes place



> *With 2.5 billion people using social networks their impact goes beyond one place. In many countries, these networks are the only mechanism for freedom of speech and the only way for citizens to stay connected and organized and to share with the world what is happening.*

Figure 1-2: The Impact of social media: Source givingcompass.org

Cultural Algorithms (CA) are stochastic optimization methods that are modelled after human culture and are suited to finding the solutions to problems embedded in complex environments (Figure 1-1). The CA belong to the class of population-based optimization algorithms but are enhanced in that the population agents are connected by a social network and they share a common Belief Space. Knowledge of various types reside in the Belief Space and are distributed to the population via a *Knowledge Distribution (KD) mechanism* on a periodic basis. Each type of knowledge is a metaheuristic that guides the associated population individuals through the problem search space in a specific manner. A knowledge distribution mechanism serves as a hyper-heuristic that solves the problem of selecting the right metaheuristic at the right time for each population individual.

Hitherto, CA implementations have used ***competitive*** KD mechanisms – i.e. mechanisms where knowledge types are pitted against each other and vie for the control of a population of individuals. Such KD methods resolve to a 'winner' knowledge type for each individual, which then

controls the individual in the next period. Competitive KD methods have performed well for problems embedded in ***static*** environments. Relatively recently, CA research has evolved to encompass ***dynamic*** problem environments – which for immediate purpose can be defined as environments that can change over time. The degree and rate of an environment's changes can be referred to as its *complexity*.

*Given increasing environmental complexity, a natural question arises as to whether the KD mechanisms that also incorporate **cooperation** can perform better in such dynamic environments?*

Stochastic optimization requires a balance between exploration and exploitation (Matej Črepinšek, 2013). Among other projections, each knowledge type or metaheuristic can also be viewed as a point on the exploration-exploitation continuum, due to the nature in which it moves the individual through the search space. All else being equal, in CA the Knowledge Distribution mechanism is the primary allocator of resources between exploration and exploitation. It controls allocations both at the macro (population-level) and the micro (individual-level). Thus, the KD mechanism, through the allocation of knowledge in the population space, is a key determiner of optimization performance.

Game theory is a formal approach for analyzing the behaviors of goal-oriented, interacting entities. Here the term 'entity' should be interpreted broadly. For example, it could refer to software agents, individuals in a population, organizations, countries or even blocs of countries (e.g. NATO vs. Warsaw pact countries). Game theory can inform about both competitive and cooperative situations. For example, alliance formation as in the case of the European Union was a cooperative undertaking whereas a market share tussle between say Ford and GM is competitive.

The application of Game theory for Knowledge Distribution mechanisms in Cultural Algorithms thus seems very apropos, especially in the light of eliciting cooperation among the available knowledge types. Game theory is an established discipline in that it is both broad and deep with applications in myriad fields, e.g. economics, social sciences, marketing, computer science, military strategy, to name just a few. Hence it is a rich source for ideas for game-based knowledge distribution.

The primary question that this research tries to answer is whether cooperative games can be an effective mechanism for Knowledge Distribution in Cultural Algorithms especially in the case of dynamic environments and / or complex domains.

1.1 Cooperative and Completive Games

In this research, CA knowledge distribution is studied in the context of 3 types of games that encompass both cooperation and competition:

- Iterated Prisoner's Dilemma
- Stag-Hunt
- Stackelberg

Prisoner's Dilemma is a well-studied game whose analytical solution settles in favor of non-cooperative actions. However, when played repeatedly cooperation can emerge. Axelrod showed that reciprocity based stable strategies can emerge in iterated game play (e.g. tit-for-tat) (Axelrod & Hamilton, 1981).

Stag-Hunt is a game that models situations where the default is cooperation but can lead to defection (competition) if reward is sufficiently delayed. The "stag hunt" metaphorically is where

a group of hunters must cooperate to hunt a stag – a more difficult task but with a larger reward. The alternate is to hunt a rabbit which each hunter can do alone without needing cooperation from others – but this provides lower reward. Hunters initially wait for a stag to appear but the longer the delay in sighting a stag (which are rarer) the more tempted they are to defect and go after a rabbit (which are plentiful). Among other situations, Stag-Hunt has been used to study alliance formation (Boudreau, Rentschler, & Sanders, 2019) and the evolution of social structure (Skyrms, 2004).

Stackelberg is a model of collusion/competition in Microeconomics that is closely related to Cournot competition. Here a leader firm takes the lead in setting the production-level/price in a market that is being targeted by the leader and a few followers. The leader can take advantage of its first-mover position to set an advantageous production target / price. However, the leader and followers have enough information to implicitly coordinate on prices and production for mutual benefit, given their relative market positions. In Stackelberg, limited cooperation emerges due to the inherent structure in the interaction arena. The Stackelberg model is often applied in product pricing and production strategies. For example, Yu and Hong have studied supply-demand balance in the electricity market in the context of a Stackelberg game (Yu & Hong, 2016).

1.2 CAT Software to Support Competitive / Cooperative Communication

To evaluate game-based knowledge distribution, two new CA software systems were developed that implement several KD mechanisms. The CA default KD mechanism - Weighted Majority (WTD) or “wisdom of the crowd” - is used as a baseline for comparison in both systems.

The two systems, described below, optimize in vastly different domains. This was done in order to more robustly investigate the premise that – in complex and dynamic environments – games

that span both cooperation and competition are a better method of knowledge distribution than Weighted Majority which is akin to “wisdom of the crowd (majority voting). Majority voting is shown to work well when the signal-to-noise ratio is weak or when the environment is relatively static. For example, many animal cultures use a form of voting to make group decisions (Hoole, 2018). By contrast, human cultures are rich and complex where social network games play a vital role in knowledge flow (Jiang, Chen, & Liu, 2014) and eliciting cooperation (Takano, Wada, & Fukuda, 2016).

The first system, ***CATGame***, solves numerical optimization problems in both static and dynamic environments. CATGame supports a variety of game mechanics. Here the KD mechanism is implemented in an abstract manner and serves as the framework that can be used to study concrete game implementations by injecting them into the this mechanism. All three games described above leverage this framework.

The second system, ***CATNeuro***, evolves optimal deep learning models i.e. neural networks (Goodfellow, Bengio, & Courville, 2016) – a domain very different from numerical optimization. Each of the population individuals in CATGame contain a *vector of real numbers*. And those in CATNeuro contain a *directed graph* - a somewhat direct encoding of a deep learning model. CATNeuro takes many aspects from the neuro-evolution discipline (Miikkulainen, et al., 2017) (Stanley & Miikkulainen, 2002) but adapts them to work in the Cultural Algorithms framework.

The CA knowledge sources operate accordingly in each domain, i.e. move individuals through a) a real-valued hyperspace for numerical optimization; and b) the space of directed graph structures for deep learning models. However, in both cases the respective knowledge sources operate on the principles established in the Cultural Algorithms literature.

1.3 Models of Complex Dynamic Environments in CATGame

To evaluate the performance of the various knowledge distribution mechanisms implemented in CATGame, the Cones World test problem generator is used. Cones World is based on the DF1 generator devised by Morrison and De Jong (Morrison & De Jong, 1999); DF1 is specifically designed to evaluate the performance of evolutionary algorithms in dynamic environments. A sample 2D Cones World landscape is shown in Figure 1-3. The optimization goal is to find the global maximum (highest peak) of the landscape within a specified epsilon. There may be a thousand or so cones – i.e. local maxima – making this a relatively hard optimization problem. The Cones World problem generator includes dynamics to periodically modify the landscapes while the performance optimization is still underway. The level of change from landscape-to-landscape – i.e. the dynamic complexity of the environment – is controllable by a system parameter.

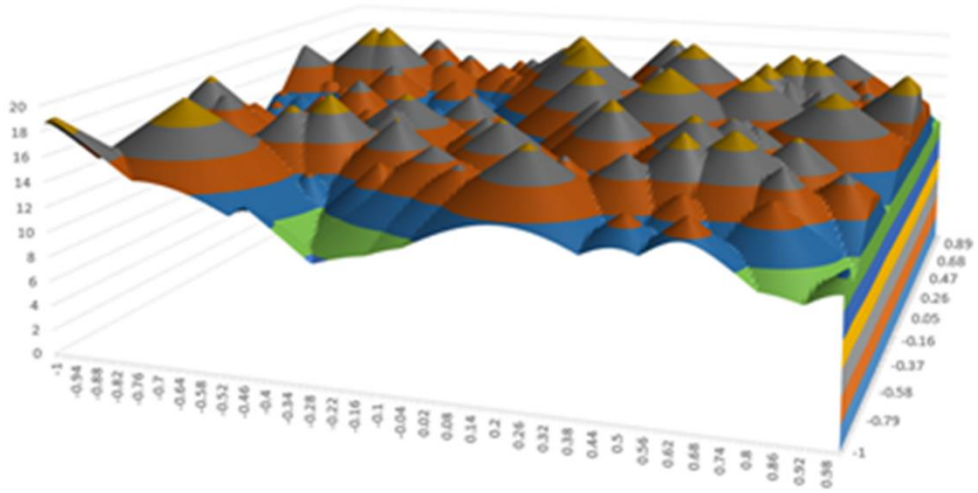


Figure 1-3: Sample 2D Cones World landscape

A key characteristic to note is how quickly the system recovers from a shock. In other words, when the proverbial rug is pulled from under, how many iterations does the system take on average to find the new global maximum. A robust system should recover sooner. A non-robust

system may take longer or could get stuck at a local maximum more of the time. The KD mechanisms are tested at varying levels of dynamic complexity – from linear changes, to non-linear, all the way to near-chaotic changes – to better understand each's responsiveness to different levels of dynamic complexity.

Multiple types of metrics are collected and analyzed to obtain a wholistic, multi-faceted understanding of KD operation and performance. The primary performance metric is the number of generations needed in order to reach solution within the specified epsilon. However, since knowledge distribution operates in the context of a social network, several 'social' metrics are also collected, such as Schelling's segregation index (Schelling, 1971); diffusion; and information related to the dynamics of knowledge flow in the network.

1.4 Models of Complexity in CATNeuro

The notion of complexity to test CATGame is in the form of change over time. By contrast, CATNeuro must solve an inherently complex, multi-layered optimization problem. The top level is that of the overall structure of the model as illustrated in Figure 1-4. At this top level, deep learning models are directed graphs that process input to produce output – the optimization should produce a feasible and hopefully compact top-level structures (often there is no single right answer).

The internal nodes of the top-level graph are modules which are smaller, reusable graphs. The system maintains several module 'species'. Each species is evolved in a separate population so as to protect the members from being eliminated too early and reducing overall available diversity – a process known as speciation in biology (Howard & Berlocher, 1998). Hence, the second level

optimization is to select the right modules for the internal nodes from the available species of modules.

Several parameters associated with both the top-level and module individuals also need tuning. For example, the overall learning rate (top level) and deep learning operation parameters (module-level) e.g. number of dimensions of a dense node; activation type (ReLU, TanH), etc. This is the third level of optimization.

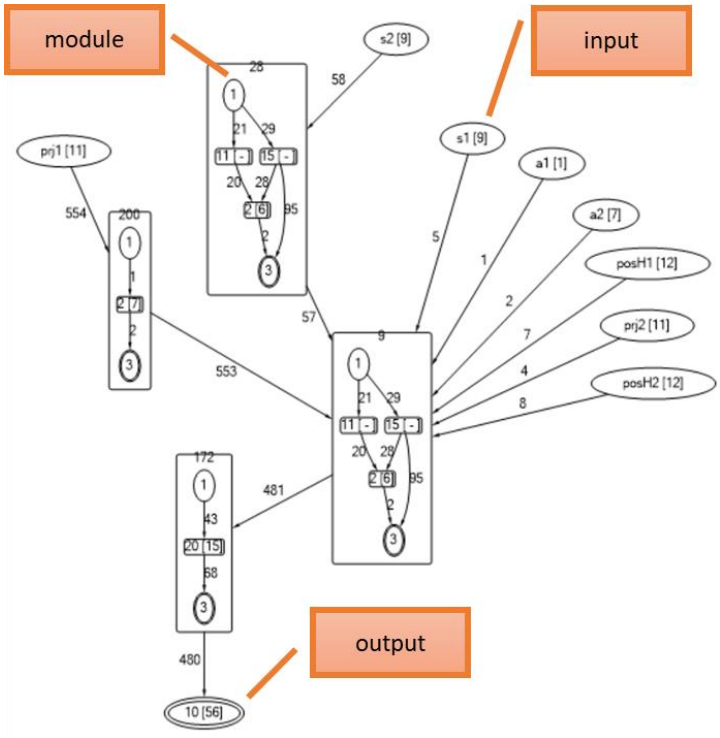


Figure 1-4: Example CATNeuro evolved model - shows overall structure and the structure of selected modules

The structures produced by CATNeuro are translated into concrete deep learning models for the chosen deep learning library (e.g. CNTK, Tensorflow, PyTorch, etc.). The concrete models are then trained using the available training data for the problem. The error or loss obtained from the training step is fed back to CATNeuro as the fitness signal to guide the optimization. CATNeuro is

a general system that can be used to optimize deep learning models for any problem as long as a suitable training dataset is available for the given problem. The training dataset for the CATNeuro test problem is created via a Reinforcement Learning process and is explained next.

Figure 1-5: A frame from FightingICE game

The test problem selected for CATNeuro is the construction of a deep learning model driven controller to play a fighting game called FightingICE (Intelligent Computer Entertainment lab., Ritsumeikan University, 2018). FightingICE is a research testbed for AI, maintained by Ritsumeikan University, Japan. It is a version of Rumble Fish (Dimps Corp.) a 2D street fighting game (Figure 1-5).

The FightingICE research testbed is part of an annual competition for AI-driven controllers. Among other features, the testbed allows programmatic access to internal game state which can be used to create controllers that play the characters in the game. Each player has 56 possible actions to choose from (e.g. jump, hit, block, throw projectile, etc.).

Any deep learning model requires a fair amount of training data to train. To acquire the training data for the deep learning controller model, a Reinforcement Learning (Sutton & Barto, 2018) inspired approach was followed. The main idea is to first learn a mapping (as a table) between game state and action distribution. In Reinforcement Learning parlance this known as a policy table. How such a mapping is learned is described later. If however such a mapping is available, an agent (or controller) can use it to play the game. It can look up the current game state in the table and if there is a match, it can then sample from the found distribution and play the selected action. Otherwise it may take a random action.

For very large game state spaces, as is the case for FightingICE, a table-based approach is not feasible as the table will have either many gaps or will be impractically large. A better approach is to convert the table into a neural network model which would be a compressed representation of the table. A table is discrete mapping whereas the corresponding neural net is a continuous one. The neural network model can be learned from the table by using the table as the training data. The network can "fill in the blanks", i.e. can abstract over the learnings available in the policy table.

How is the policy table learnt? For the FightingICE controller, the process is to play many games and record each frame's non-pixel data to obtain representative samples of states encountered during game play. The table is initialized with uniform action distributions for each recorded state. Then the FightingICE supplied *simulator* is used to play selected actions for each recorded state. If

simulation actions produce a win, the probability of choosing the winning actions for the corresponding states is increased and the table updated accordingly. Initially actions are selected at random but later a mix of random and sampled actions are used. It takes a while for the policy table to converge but eventually it does. Following this process resulted in a 1.3 million row policy table with a disk size of about 1 gigabyte. The table thus learnt provides sufficient data to train a deep learning model to be used as a controller.

Stag-Hunt – for which CATGame experimental results showed as performing the best – and WTD knowledge distribution mechanisms are implemented for CATNeuro. Several deep learning models are evolved using each of the mechanisms, with the training data obtained from the policy table. FightingICE was played for each model and game play statistics recorded. The selected opponents for all test games are a) “Jerry Mizuno” (Chu & Thawonmas, 2017) – an algorithmically-driven AI used as the benchmark for comparison and b) “Thunder” (Intelligent Computer Entertainment lab., Ritsumeikan University) the 2018 FightingICE competition champion.

In addition to game statics (e.g. hits-to-opponent; relative-score; distribution of actions taken; etc.) several aspects of the produced model structures were noted (e.g. the number learnable parameter weights; number of nodes; number of edges; maximum path length; etc.). The game play statistics and model structural properties are analyzed and compared to evaluate the performance of each knowledge distribution mechanism under CATNeuro.

1.5 Outline of the Thesis

The remainder of the thesis is organized as follows. Chapter 2 provides an overview of the Cultural Algorithms framework which is the main subject of this research. Since knowledge

distribution is one of the primary determiners of CA performance, the prior mechanisms and their historical progression is detailed in Chapter 3.

Chapter 4 is a primer on Game Theory. Only the basics are covered in enough detail for one to understand the application of games to CA knowledge distribution. Chapter 5 explores how games can be used for knowledge distribution in CA, and details an abstract mechanism for injecting a wide variety of games into the CA framework for study. The concepts and implementations of the three games mentioned earlier are provided in Chapter 6.

Chapter 7 details the experimental framework used to evaluate game-based knowledge distribution mechanisms implemented in CATGame. The Cones World based dynamic landscape generation mechanism is explained in detail. Also detailed are the various 'social' metrics used for evaluation such as Schelling' Index of segregation; diffusion; and analytical methods based on a Markovian view of knowledge flow across the population network. Data collected from CATGame experiments, conducted as described in Chapter 7, are analyzed and presented in Chapter 8.

The CATNeuro system and the experimental framework to evaluate CATNeuro is explained in Chapter 9. It details the graph operations used to evolved deep learning models and the mapping of those operations to knowledge sources. Chapter 9 also provides a description of the FightingICE game and the reinforcement learning inspired method used to create the training data for deep learning, along with the training regime used to train the models. Chapter 10 presents the performance analysis for CATNeuro knowledge distribution mechanisms from the experimental data collected. Finally, the main conclusions of this research are summarized in Chapter 11. Future work arising from the conducted research are also presented in the chapter.

CHAPTER 2 CULTURAL ALGORITHMS FRAMEWORK OVERVIEW

2.0 Introduction

In the spirit of Ant Colony Optimization (Dorigo, Maniezzo, & Colorni, 1996) (ACO) and Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995), the Cultural Algorithm (CA) (Reynolds R. G., 1978) is a socially motivated knowledge-driven approach that can be used to find optimal solutions in a search landscape. Its applications exist in a diverse set of domains, e.g. numerical optimization (Ali, Suganthan, Reynolds, & Al-Badarneh, 2016); archeology (Jayyousi & Reynolds, 2014), biology (Judeh, Jayyousi, Acharya, Reynolds, & Zhu, 2014); gesture recognition (Waris & Reynolds, 2015), and computer vision (Waris & Reynolds, 2018); to name some. As with ACO and PSO, the CA approach employs a socially interacting population of agents. By contrast however CA also employs a high-level component called the Belief Space that collects and disseminates varied types of knowledge from/to the population. The Belief Space consists of Knowledge Sources (KS), each of which essentially represents a type of search strategy. Some KS are primarily exploratory while others are primarily exploitative and yet others can be like stem cells that can be explorative or exploitative depending on the context. The Cultural Algorithm is a hyper-heuristic that determines what strategies to distribute to the population.

The two components (Population and Belief Space) are connected by an interface; the communications protocol. The interface consists of an acceptance function and an influence function. The acceptance function manages transferring experience from the Population component to the Belief Space. In turn, the influence function is concerned with distribution of knowledge across a social network. One of the keys to the influence function is the knowledge distribution mechanism. This mechanism controls how the search strategies from the Belief Space

are distributed among the population of individuals. The establishment of the right dynamic balance between the influences of the various Knowledge Sources on the population of individuals is a key goal of CA.

Over the years, many knowledge distribution mechanisms have been proposed and studied (Peng, 2005) (Che, 2009) (Reynolds & Kinnaird-Heether, 2013) (Al-Tirawi & Reynolds, 2018). This are detailed in Chapter 3. However, prior research on Cultural Algorithms has focused exclusively on competitive KD mechanisms where each individual in the population is assigned one Knowledge Source (the winner) that influences that individual for the next generation.

Biologically inspired computing was first given impetus by the John Holland with the development of Genetic Algorithms (GA) (Holland, 1992). GA mimic the chromosomal processes of mutation and inheritance that occurs in nature. The philosophical underpinnings go all the way back to Darwin and the aphorism “survival of the fittest”. In fact, the GA structure follows nature’s form quite closely. A population of individuals is evolved through mutation and crossover of ‘genetic material’; individuals are evaluated for fitness as candidate solutions against the problem space under consideration; the fittest individuals receive higher chance of passing their genes on to the next generation.

2.1 The Cultural Algorithm

CA are a computational model of cultural evolution that happens at a faster pace than biological evolution (Perreault, 2012). Dawkins (Dawkins, 1976) proposed the idea of ‘meme’ as carrier of cultural knowledge, analogous to genes in biology. The Belief Space in CA can function as the storehouse of memetic information that can be transcribed on to future generations. The basic CA is given in Figure 2-1.

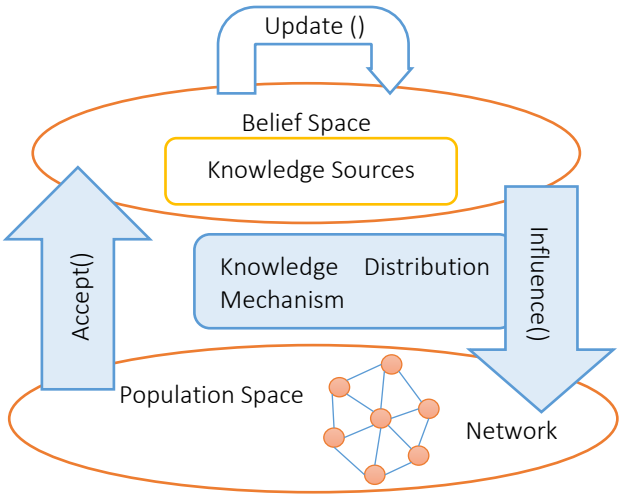


Figure 2-1: Cultural Algorithms Framework (source: CA papers)

The major components of the CA are explained next and the pseudocode for the main loop is given in Figure 2-4:

- **Population Space** – a collection of agents or individuals. Each individual contains information that can determine its action and behaviors. The individuals may be networked together in some topology (see Figure 2-2). Each is associated with a single Knowledge Source (in most cases).

- **Belief Space** – stores and organizes different categories of knowledge into the aforementioned Knowledge Sources. The Knowledge Sources reside in the Belief Space and are organized in a tree structure as shown in Figure 2-3. Knowledge is harvested from the current population generation and stored into the Belief Space. It is then disseminated to the next generation. This knowledge transfer and update occurs via the communications protocol described next.

- **Communication Protocol** – methods for transferring knowledge between the Population Space and the Belief Space. It consists of an **Accept, Update** and **Influence** functions. Every generation, interesting individuals from the current generation are inducted via the **Accept** function into the Belief Space. The **Update** function harvests knowledge from the selected individuals and updates the stored knowledge in the Belief Space. The **Update** function flows accepted individuals as per Figure 2-3. The **Influence** function updates the population to create the next generation by a) first associating each individual with a Knowledge Source using the knowledge distribution mechanism; and b) modifying the individual via its associated KS to move it through the search space in search of better solutions.

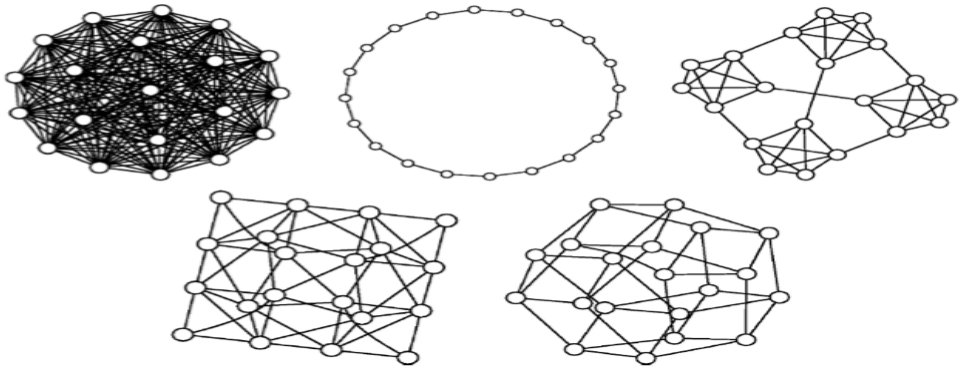


Figure 2-2: Population network topologies (source: CA papers)

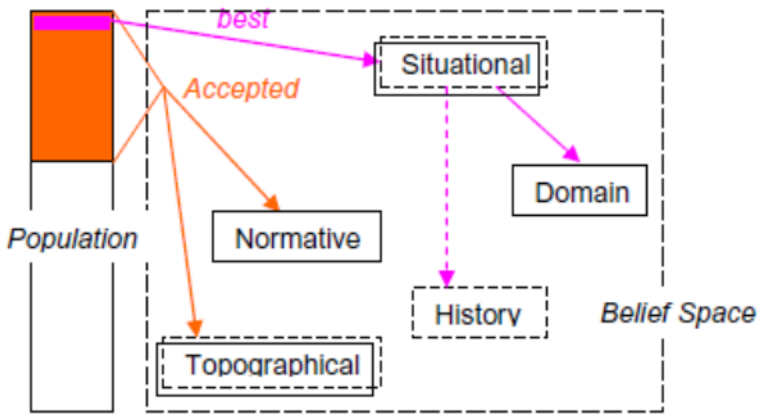


Figure 2-3: Knowledge Source dependency graph (source: CA papers)

```
Begin
t = 0;
initialize BeliefSpace; Pop
repeat
   Pop <- evaluateFitness (Pop)
   Selected-Indvs <- accept(Pop)
   BeliefSpace <- update (BeliefSpace, Selected-Indvs)
   Pop <- influence (BeliefSpace, Pop)
   t <- t + 1
until (termination condition achieved)
End
```

Figure 2-4: Cultural Algorithms Pseudocode (source: CA papers)

Knowledge Sources can be viewed as being exploitative or explorative. Exploitative KS explore a local region of the search space. For example, the Situational KS tracks exemplar individuals in the population. Individuals under its influence are small variations of one of the exemplars stored in the Belief Space. Situational knowledge is therefore considered exploitative. The Topographic KS on the other hand maintains knowledge of diverse regions of the search space by tracking clusters of individuals. Under its influence, individuals are likely to move to a location in or near one of the clusters which span a larger radius than that around a single individual.

It must be noted that Knowledge Sources operate on principles harvested from the dynamics and evolution of cultures and as such are abstract ideas. Their concrete implementation for a

particular domain really determines where on the explorative-exploitative scale they fall. For example, Normative KS as interpreted for numerical problems is considered explorative whereas it is considered exploitative in CATNeuro because it does not alter the structure of the graph and instead only changes the parameters of the graph nodes.

The CA can be configured with either homogenous (fixed) or heterogenous network topologies (Figure 2-5). Reynolds, et al (Reynolds, Gawasmeh, & Salaymeh, 2015) found that homogenous topologies were more efficient in low entropy problems but a variation of heterogenous topologies performed more predictably in higher complexity problems. Below (Figure 2-5) are examples of completely connected graphs. Disjoint graphs can be represented in a co-evolution fashion as multiple populations are may communicate via the Belief Space.

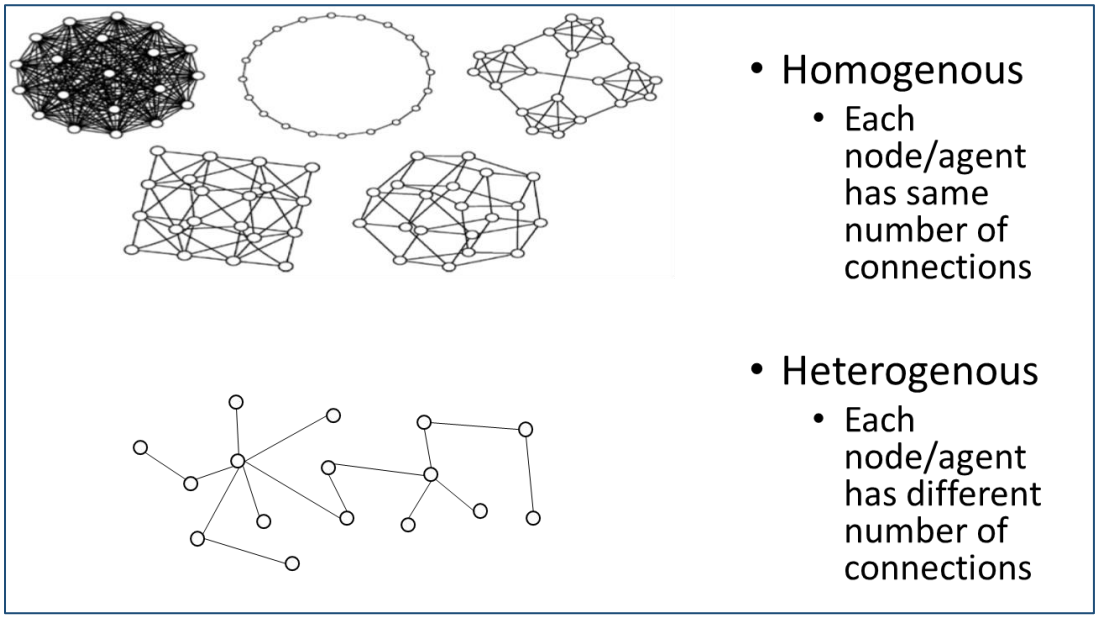


Figure 2-5: Homogenous and Heterogenous topologies (source: CA papers)

The CA has been applied in a vast variety of problem domains. In addition to the applications noted in 1.0, to solve scheduling problems in cloud workflows (Mojab, Ebrahimi, Reynolds, & Lu,

2019); implement security policies (Bhuyan, Lu, Reynolds, Zhang, & Ahmed, 2019); and multi-objective optimization (Stanley S. D., 2020), among others. Several books by Reynolds covers additional applications (Reynolds R. G., 2020), (Reynolds R. G., 2019).

2.2 Knowledge Sources

The CA framework allows for different types of Knowledge Sources to be combined in a synergistic manner. The schematic in Figure 2-6 depicts the flow of knowledge from the Belief Space to the population space.

The number and types of Knowledge Sources is not fixed. However, the CA is typically configured with a default set, namely Situational, History, Domain, Topographical and Normative knowledge. These are described in some detail in Table 2-1. KS usage is selective and new knowledge types can be integrated into the Belief Space, if required by the problem domain. Colon (Colon, 2012) for example, augmented CA with a 'Contextual' rule-based knowledge representation to determine optimal plans for pediatric nursing care. The reader is referred to other CA papers for the detailed explanation of the commonly used Knowledge Source metaheuristic types (Reynolds & Saleem, 2005) (Ali M. , 2008) (Che, 2009).

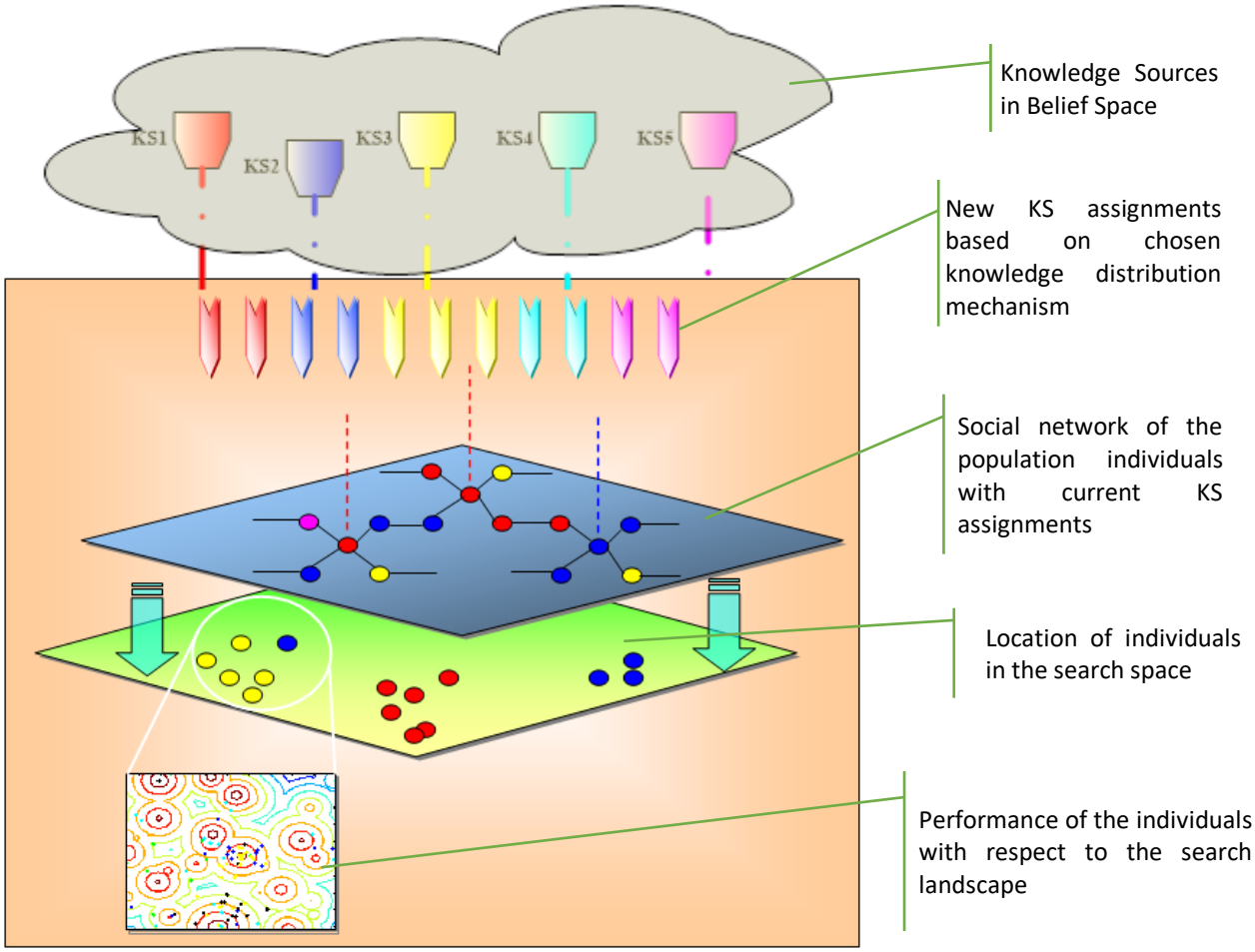


Figure 2-6: CA Knowledge distribution flow (source: CA papers)

Conceptually, each KS represents some process in cultural evolution. These abstract concepts are translated into concrete implementations based on the problem domain. In general, for numerical optimization problems, the KS operate on numerical vectors. Each vector is a point in the problem hyperspace and represents a candidate solution. Each KS modifies the vector elements, in its own way, to guide the associated individual to optimality in the search space. The

The CATGame system is meant for numerical optimization and thus operates on numerical vectors as described above. CATNeuro by contrast operates on directed graphs and hence the KS

need to modify graphs. This is done with the help of graph operations derived from NEAT (Stanley & Miikkulainen, 2002), namely:

- Toggle Connection
- Add Connection
- Add Node
- Crossover
- Mutate Parameter

These operations are described in detail in section 9.1. Here the mapping of these operations to the various KS is described in Table 2-1.

Table 2-1:Default Knowledge Source Types

Knowledge Source & Description	**Acceptance**	**Influence**
Situational Situational knowledge was first introduced by Chung **(Chung, 1997)**. It consists of a set of exemplar individuals along with their parameter values and the fitness value. These individuals represent 'event-based' memories observed within some species. They also serve as examples for other individuals to follow.	Best performing individuals are added to the list of exemplars maintained by Situational KS. The new list is ranked by fitness and truncated to the configured maximum length.	***Numeric***: The parameter values of an individual under the influence of this KS are mutated in the direction of the corresponding parameter values of the exemplars or evolved from their current values. The probability between the two actions depends on a configure probability with bias towards parameter evolution. ***Neuro***: Graph of a randomly selected exemplar is evolved via an operation sampled from the probability distribution over operations associated with Situational (see Figure 2-7). The updated graph is then assigned to the influenced individual.

Normative In cultural terms this knowledge represents the accepted norms of behaviors in a society. Normative knowledge was also introduced by Chung **(Chung, 1997)**. It is a set of intervals for each of the parameters of the problem. The intervals are considered promising range of values for the corresponding parameters.	The interval ranges maintained by Normative KS are updated from the parameter values associated with best performing individuals.	***Numeric***: The parameter values of the population individuals are mutated in accordance with the intervals. If a parameter value falls outside the range, a value is randomly assigned from the range interval. Otherwise the value is mutated around its current value. ***Neuro***: The parameters of the influenced individuals' graph are evolved by sampling from kernel density estimates (Cosma Shalizi CMU, 2009) maintained by Normative KS.
Topographic In cultural terms, this KS represents the knowledge of the landscape or the terrain. The version of Topographic KS used for CATGame and CATNeuro is based on the Brainstorm optimization algorithm **(Shi, 2011)**. The individuals are grouped into clusters (using an algorithm such as K-means). The clustering mechanism by its nature divides the top performers into diverse groups each of which marks a promising region of the search space. Note: Earlier versions of this KS use multi-dimensional trees. Jin **(Jin & Reynolds, 1999)** introduced topographic or regional knowledge. The search space is divided into cells. A list of best cells is maintained. Overtime the cells may be divided into finer grained cells to provide better resolution for optimization.	The top performing individuals are added to the list of individuals maintained by Topographic. The new list is ranked by fitness and truncated to a configured maximum. The updated list is clustered with K-means where each list individual is binned to a fixed number of clusters. Note that to perform K-means clustering some measure of distance is required between two individuals. This is usually the Euclidean distance in the case of numeric optimization. For CATNeuro a measure of distance is defined based on graph similarity (see Chapter 9).	***Numeric***: The parameter values of the influenced individual are derived by evolving those of the centroid individual of a randomly selected cluster. ***Neuro***: Graph of a randomly selected centroid is evolved via an operation sampled from the probability distribution over operations associated with Topographic (see Figure 2-7). The probability distribution is heavily biased towards the crossover operation. The updated graph is then assigned to the influenced individual.
Domain Domain knowledge was introduced by Saleem **(Saleem,**	Accept the current generation of top performers	***Numeric***: The influenced individuals parameter values are mutated in the direction of the

2001). The idea is to leverage knowledge specific to the problem domain – such as by consulting an expert in that domain. For numerical problems, Domain guides the individuals in a direction determined from local gradients. David Colon **(Colon, 2012)** used business rules as the basis of Domain knowledge. For Domain, CATNeuro is the primary KS for adding new nodes to a graph. CATNeuro encodes knowledge of graph structures to evolve structurally sound graphs for deep learning models however this applies for all KS except Normative (which does not modify structure).		gradient to achieve better performance (for high dimensional problems, the determination of local gradients may be expensive so CATGame also implements an alternate, Differential Evolution (Storn & Price, 1997) based version of Domain KS). ***Neuro***: The influenced individuals' graph is updated either by evolving its own current graph or that of a top performer of the current generation. Neuro evolution is elitist (Stanley & Miikkulainen, 2002). With configured probability, a top performer's graph is chosen over the individuals own. Graph evolution is strongly biased towards addition of a new node to the graph.
Historical History knowledge was also introduced by Saleem **(Saleem, 2001)**. In cultural terms it represents 'episodic' memories. History maintains the trail of best performers over time. This allows for backtracking or branching from a prior best – helpful if the optimization is stuck in local optima.	Add new best, if any, to the history list. Truncate the list to a configured maximum	***Numeric***: If the influenced individual's fitness is worse than that of a randomly selected best from the history list, the influenced individual is assigned mutated parameters of the selected history individual. Otherwise the influenced individual's parameters are evolved around their current values. ***Neuro***: Here the mechanism used is similar to the one used for numeric problems except that the mutation applied is a graph operation which is sampled from the distribution given in Figure 2-7. For History, equal weight is given to mutate parameter, toggle connection and add connection operations.

To enable better exploration and to prevent premature convergence, the Knowledge Sources are not deterministically mapped to graph operations in the CATNeuro system. Rather the KS are associated via probability distributions. When a KS influences a popluation indvidual, it samples from the associated distribution and selects a graph operation to apply. A high level view of this mapping is in Figure 2-7 and more details are given in Chapter 9. In Figure 2-7, each row visually depicts the probabilty of selecting a graph operation for the corresponding KS. Each 'block' of each row corresponds to a graph operation. The graph operations are shown on the x-axis. The color of the block represents probability. The color key on the right side provides a mapping from color to proability value. As an examle Topographic is biased to select the Crossover operation but can select Add Connection or Add Node operations with some probability. If a KS is under represnted in the population then, under a determinsitic mapping, the corresponding graph operation could be under applied. The randomized association helps maintain diversity of graph operations in the population.

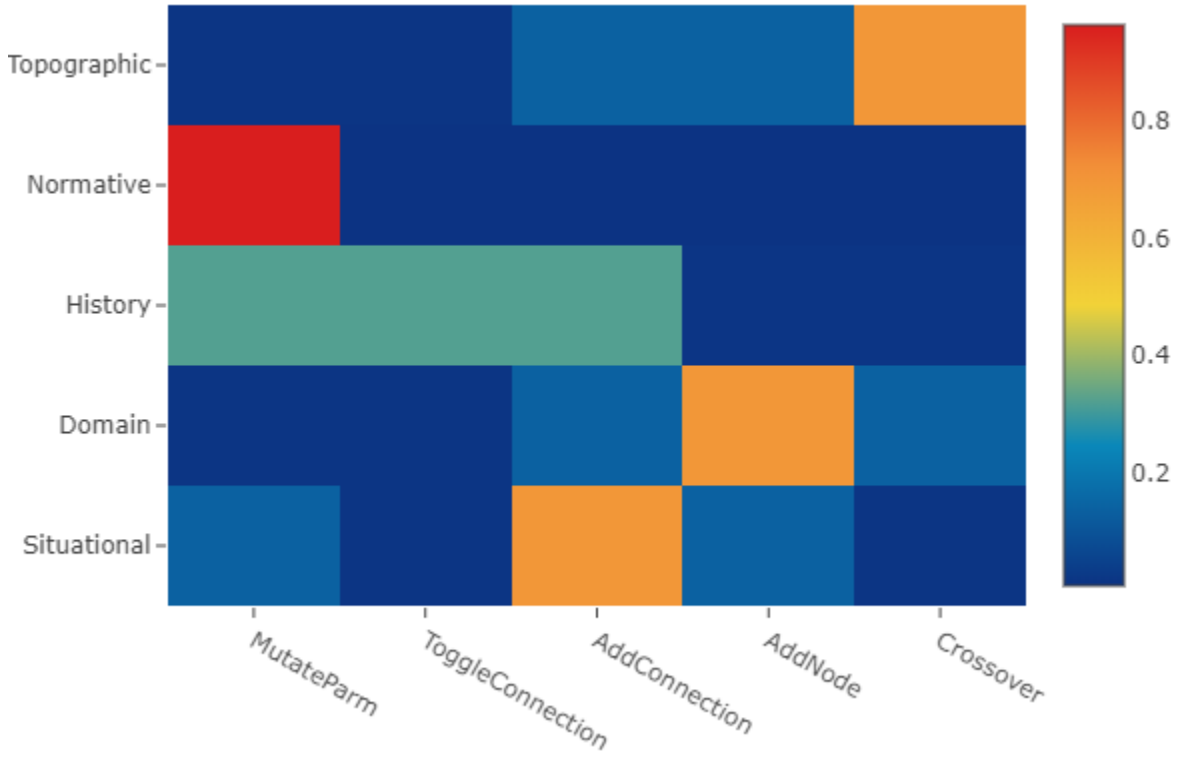


Figure 2-7: Knowledge Source to graph operation heat map - colors represent probability; row probabilities sums to 1 (source: CA papers)

2.3 Chapter Summary

This chapter provided an overview of Cultural Algorithms in terms of its architecture, major components and operation, namely the Belief Space and the contained Knowledge Sources; the Population Space and its network configuration; Accept, Update and Influence functions and the influence of KS on population individuals depending upon the problem domain (e.g. numerical optimization or graph evolution).

The role of knowledge distribution was briefly mentioned but since KD mechanisms are central to the research question, the entire next chapter is dedicated to describing the history of the CA distribution mechanisms and their detailed operation.

CHAPTER 3 FROM COMPETITION TO COOPERATION: A PERSPECTIVE ON CA KNOWLEDGE FLOW MECHANISMS

3.0 Introduction

The **Influence** function is the main driver of knowledge distribution in Cultural Algorithms. It has two major roles a) to associate a Knowledge Source with each population individual – i.e. *distribute knowledge*; and b) to update the parameters of each population individual by letting the associated Knowledge Source operate on the said individual. The pseudocode of the Influence function is given Figure 3-1. First the direct influence for the individual is determined. The exact determination is performed by the underlying distribution mechanism. It could be as simple as taking the currently assigned KS. Or it may be a more involved calculation. Then the network neighbors of the individual are collected. And finally, a new KS assignment is arrived upon by utilizing the direct influence; state of the individual & its neighbors; and possibly other relevant information identified as "System State" in the referenced pseudocode. The newly determined KS is then used to influence the individual to obtain the offspring for the next generation. The new KS is associated with the offspring.

```
import Population, Network, DirectInfluence, DetermineKS, SystemState
init NewPopulation
Begin
   For individual in Population
     ksDirect <- DirectInfluence(individual)
     neighbors <- Network(individual)
     ksNew = DetermineKS(ksDirect, individual, neighbors, SystemState)
     offspring <- ksNew.Influence(individual)
     offspring.KnowledgeSource <- ksNew
     NewPopulation[individual.Id] <- offspring
   End For
   Return NewPopulation
End
```

Figure 3-1: Pseudocode for the Influence function (source: CA papers)

Knowledge distribution emerged as a focused area of research over the evolution of the CA. Knowledge distribution generally does not affect how the Knowledge Sources modify the associated individuals. It is primarily concerned with the association of knowledge to individuals. Knowledge distribution mechanism is another heuristic (Figure 3-2). It is the primary allocator of compute resources to search strategies (metaheuristics) and is therefore an important operation in CA. The CA thus is a 'hyperheuristic' algorithm.

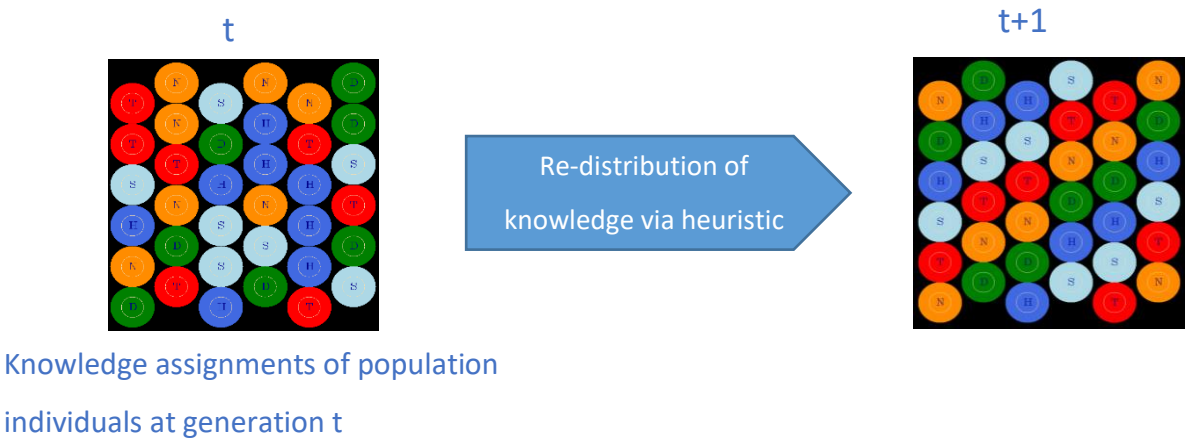


Figure 3-2: Knowledge distribution operation

Over the years, many distribution mechanisms have been studied. Figure 3-3 displays the various distribution mechanisms relative to the fidelity of the environmental signal available in the optimization problem at hand. The left end corresponds to a completely noisy signal. As one moves to the right, the fidelity of the signal gets stronger and less noisy. Majority win is a good strategy when there is a signal but some background noise. The voting process filters out much of the noise. As the signal gets stronger particular knowledge sources may be better at tracking it and therefore can begin to carry more weight. Once the signal is strong enough that it is visible to most then an auction or bidding mechanism becomes useful to identify the individuals most

attracted to the signal. When the signal takes a more precise value the fixed price solution is possible. Now the agents have a precise set of moves, they can pay the price or not. At the far right, agents can make specific moves to support cooperation and competition or both.

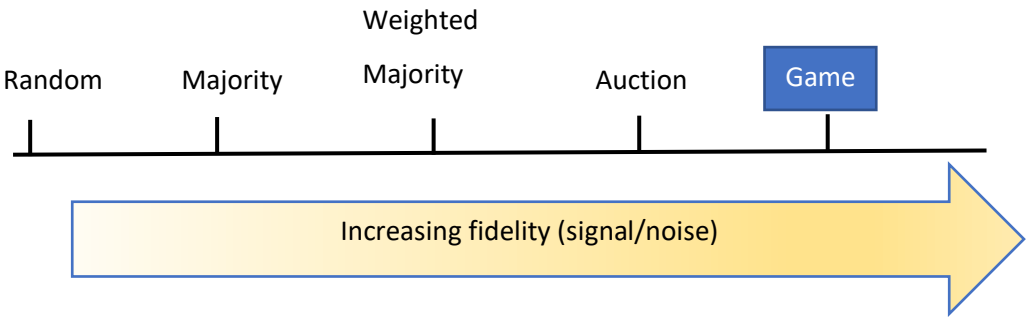


Figure 3-3: Spectrum of Knowledge Distribution mechanisms (source: CA papers)

The following few sections describe the commonly used mechanisms for knowledge distribution.

3.1 Random Selection

The earliest CA knowledge distribution was random, i.e. individuals were randomly assigned a KS with equal probability. Reynolds and Saleem (Reynolds & Saleem, 2005) introduced Random knowledge distribution in 2005. Later mechanisms took account of individual and aggregate KS fitness, the social network structure, and other relevant factors. Peng (Peng, 2005) devised Knowledge Source assignment proportional to KS performance (see Figure 3-4) based on Charnov's Marginal Value Theorem for predator/prey dynamic (Charnov, 1976). Ali (Ali M. , 2008) extended Peng's work with the idea of a social fabric and flow of knowledge between connected individuals and introduced the Majority Win Knowledge Distribution mechanism. Che (Che, 2009) introduced a variety of homogeneous network topologies taken from the Particle Swarm literature and introduced the Weighted Majority win distribution mechanism.

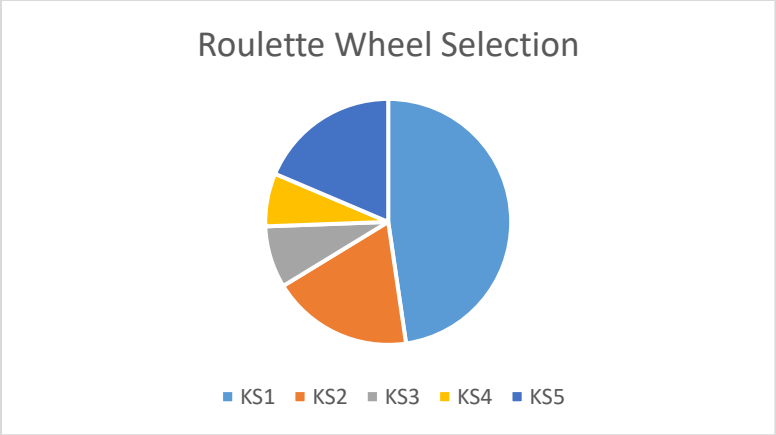


Figure 3-4: KS selection roulette wheel (source: CA papers)

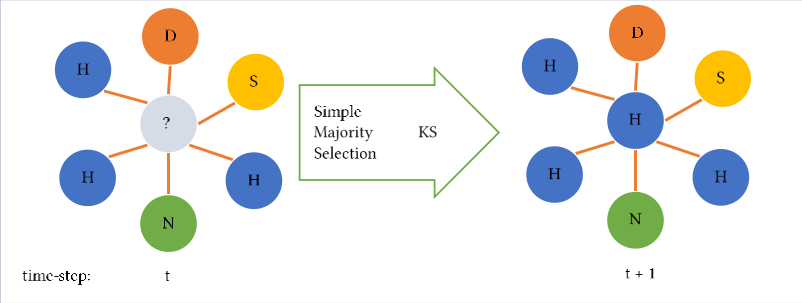


Figure 3-5: Simple Majority Knowledge Distribution (source: CA papers)

3.2 Majority Win

The difference between (simple) Majority and Weighted Majority is shown in Figure 3-5 and Figure 3-6, respectively. The diagrams show the KS assignment process for a single individual going from time t to t + 1. The surrounding circles represent neighbors which are coded with color and letter to represent their assigned KS (H=History, S-Situational, etc.). Each of these nodes has its own direct knowledge source. First the direct influence for the center node is arrived at. Then the Knowledge Sources of the neighboring nodes are considered. In the case of Simple Majority, the center individual gets the KS that is most frequent considering the direct influence and neighboring KS (with supplementary mechanisms to handle ties).

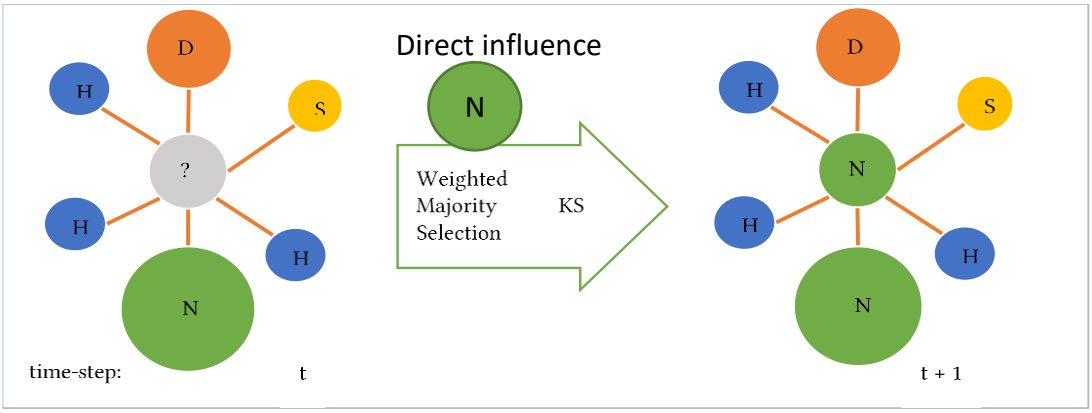


Figure 3-6: Weighted Majority Knowledge Distribution (source: CA papers)

Since Weighted Majority is the most commonly used mechanism and used as the benchmark for knowledge distribution, it is described more formally next using mathematical notation. $K = \{H, S, N, T, D\}$ *is the set of* KS. Let i index population members. $W_{k \in K} = \frac{1}{\sum_i fit_i} \sum_i fit_i | \, k_i = k$ is the relative weight of each KS in the population, where fit_i is the fitness (assuming maximization objective) and k_i is the KS of i and $\sum_k W_k = 1$. Let j index $i's$ network neighbors then $k_{i1}, k_{i2}, \ldots, k_{ij}, \ldots$ be their KS. Also, $k_{ig} \in K$ be a randomly selected KS with selection probability W_k – i.e. the direct influence. The KS assigned to i in the next generation is k_{inew}, given in Eq **3-1**.

$$\boldsymbol{k_{i\,new} = \frac{argmax}{k} \left(\sum_j w_{k_{ij}} \mid k_{ij} = k \right) + \begin{cases} W_{k_{ig}} & k_{ig} = k \\ 0 & othewise \end{cases}} \qquad \textbf{Eq } 3\text{-}1$$

Weighted majority denotes "wisdom of the crowd" and is a form of voting. Voting is also exhibited by animal cultures (Hoole, 2018). When the signal-to-noise ratio is relatively a simpler voting mechanism is often is as effective any alterative.

3.3 Auctions

More recent developments in CA's have started to explore auctions for Knowledge Distribution (Reynolds & Kinnaird-Heether, 2019), (Al-Tirawi & Reynolds, 2019), (Al-Tirawi & Reynolds, 2018)

(Reynolds & Kinnaird-Heether, 2013) . In the Auction KD mechanism devised by Reynolds and Kinnaird-Heether, 'bidding wheels' are first constructed for each KS. A pie or slice of the wheel proportionately represents the fitness of one of the individuals that the KS had influenced in some specified number of past generations. Each slice is a token for bidding with value proportional to its thickness. A mechanism solicits bids from the relevant KS for a contender individual. The KS cast bids with probability proportional to the value of the tokens they hold. The bidding mechanism determines a winner and the winning KS is assigned to the contender. The winning KS then removes the winning token so that it is no longer available for subsequent bids (see Figure 3-7)

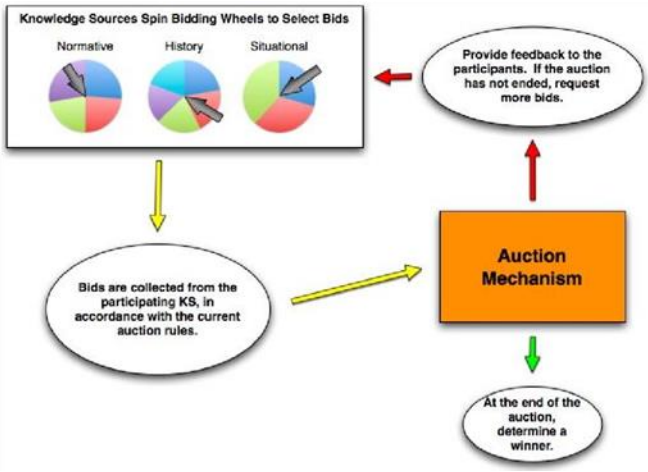


Figure 3-7: Auction mechanism, [source: (Reynolds & Kinnaird-Heether, 2013)]

3.4 Chapter Summary

Knowledge distribution is a critical part of CA. It's the primary allocator of compute resources and currently an active area of research in Cultural Algorithms. This chapter summarizes the prior work and currently active research in distribution mechanisms – from random, all the way to auction mechanisms. While auctions are a kind of a game, they are still a competitive mechanism.

Departing from prior tradition, this research focuses on mechanisms that span both cooperation and competition. The new mechanisms are sourced from classical and evolutionary game theory.

The next chapter is a brief primer on game theory designed to re-acquaint the reader with some terms and concepts that will be used later. Then in Chapter 5, an abstract framework for injecting arbitrary games for knowledge distribution is described. The three specific game-based KD mechanisms, introduced earlier, are described in detail in Chapter 6. These are concrete instantiations of the abstract mechanism described in Chapter 5.

CHAPTER 4 A PRIMER ON GAME THEORY

4.0 Introduction

Game theory is a well-studied discipline that is both broad and deep. Any extensive treatise of Game Theory is beyond the scope of this document but the curious reader is pointed to the comprehensive online book "Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations" by Shoham et al. (Shoham & Leyton-Brown, 2009). This section describes some relevant terms and ideas – just enough to support the approach presented here.

4.1 Basic Game Formulation

A game can be represented in what is known as 'normal' form as a 3 tuple:

$\phi = (N, A, u)$
N = Set of Actors (players)
A = Set of actions available to actors – usually discrete choices but may be continuous (Veelen & Spreij, 2009)
u = Set of utility functions (u1, u2, …, un) corresponding to each player.

The utility function determines the payout for the corresponding player given the actions taken by all players in the game. Often two-player games are represented in matrix form. An example of the well-studied, Prisoner's Dilemma game is presented below:

N = {1, 2}
A = {Cooperate, Defect}
u = Utilities as shown in Table 4-1

Table 4-1: Prisoner's Dilemma Payout

	Player 2: Cooperate	Player 2: Defect
Player 1: Cooperate	1, 1	-1, 2
Player 1: Defect	2, -1	0, 0

As an example, if player 1 Cooperates and player 2 Defects then player 1 gets utility -1 and player 2 gets utility 2. Game theorists are often concerned with 'solving' a posed game, i.e. determining the actions (or mix of actions) that players should play, if they are rational. Each player is expected to maximize its utility with a view that other players will do the same. In the Prisoners Dilemma game, rational players are expected to Defect. If either player Cooperates then the other player can maximize its payoff by Defecting (Axelrod & Hamilton, 1981) (Holland, 1992).

4.2 Competitive and Cooperative Games

There are several possible taxonomies for games in Game Theory however from the perspective of application to Cultural Algorithms dividing games into Competitive and Cooperative categories is useful (see Figure 4-1). Cooperative games are also referred to as Coalitional games and Competitive games are also referred to as Non-cooperative games in the literature.

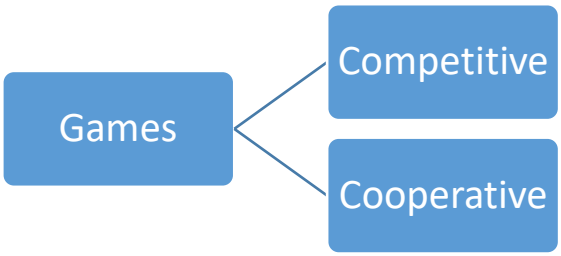


Figure 4-1: Competitive and Cooperative games – a useful categorization of games for the application of games to Cultural Algorithms

Broadly speaking, in Competitive games each agent or player is trying to maximize its own utility, i.e. the agents behave in a selfish manner. In Cooperative games, agents may form teams or coalitions and work together to maximize the utility of the team. In the extreme case, all agents may form a single team (the so called 'Grand Coalition') and maximize the overall utility.

From the perspective of Cultural Algorithms, the various Knowledge Sources can be seen as agents in a game that can influence a particular population individual. If the game is formulated as a competitive game, the agents will compete against each other but there may be only one 'winner' that acquires the individual and affects its parameters via the Influence function. Recall that population individuals exist in a (social) fabric or network, and that each individual is currently under the 'influence' of a Knowledge Source. An interplay between Knowledge Sources can develop via this network.

Alternatively, the interplay between Knowledge Sources can be posited as a Cooperative game. The Knowledge Sources involved can form a single team or multiple competing teams. If multiple teams are formed then these teams can compete with one another to win the opportunity to influence of the current population individual. This interaction would be similar to what happens in a competitive game. The difference is that the winning team (or the grand coalition) would have multiple members that can collectively influence the population individual.

The CA knowledge distribution mechanisms applied up until now associate a single KS with a population individual at each time step. From a Cooperative game perspective, this stipulation can be relaxed to allow multiple Knowledge Sources to influence a single individual at each time step.

In Cooperative games, there are two additional problems to solve (when compared with Competitive games); 1) how stable coalitions or teams can form from the individual agents and 2) how to divide the total utility gained by a team among its members, at the end of the game.

4.3 Some Common Competitive Games

There are several well-known classes of games that have been given names (sometimes multiple names). A new game situation may be analyzed more quickly by mapping to an existing scenario, thereby harnessing the existing knowledge, solution strategies, etc. and applying them to the new game situation. This section describes some common classes of games. Most of the information in this section is sourced from the book Shoham et al. (Shoham & Leyton-Brown, 2009).

Zero-Sum or Constant-Sum Games

In such games the total payoff is a constant regardless of the strategies chosen by the players. One player's gain is another player's loss. Such games were studied very early on in GT development by van Neumann and Morgenstern (von Neumann & Morgenstern, 1947). The minmax theorem was first published in 1928 by von Neumann that proved that the best strategy for each player is independent of the strategies of other players. A sample payoff matrix is given in Table 4-2. Zero sum games can be solved more easily using minmax strategies and are Pareto optimal. From a computational efficiency perspective it might be desirable to model a knowledge distribution mechanism as a zero-sum game.

Table 4-2: Zero-sum Payoff Matrix Conducive to Minmax Solutions

	Left	Right
Left	1, -1	1,-1
Right	-1,1	0, 0

Battle of the Sexes

This is a game of both coordination and competition. Both players lose if they choose different actions. Both are better off if they chose the same action but the utility of the action differs for each player.

This game is often posed as a husband and wife deciding on what movie to go to. Each has a preference for a different movie but above all they both want to be together. A possible payoff matrix is show in Table 4-3.

Table 4-3: Battle of the Sexes Payoff Table

	Left	Right
Left	2,1	0,0
Right	0,0	1,2

In studying climate change negotiations DeCanio et al show the applicability of games similar in structure to the Battle of the Sexes (DeCanio & Fremstad, 2011).

The fact that this game models both coordination and competition makes it interesting for Cultural Algorithms where a coordinated outcome may be desired for KS selection.

Matching Pennies

This is a zero-sum game where one player wins by matching the action of another player whereas the other player wins by picking something different. In the canonical version of the game each player chooses either heads or tails from a coin with the payoff matrix shown in Table 4-4.

Table 4-4: Matching Pennies Payoff Matrix

	Head	Tail
Head	1,-1	-1,1
Tail	-1,1	1,-1

This game is important in the study of behavior over repeated play. For example, an innovative company such as Apple can do well by trying to define new products and markets. Established players such as Microsoft can do well by matching the innovations of its competitor.

Hawk – Dove

In this game, the players may adopt two types of actions predatory (Hawks) or peaceful (Doves) when competing over the same resource. This game was first described by Maynard Smith (who is well known for biological games) and Price (Maynard Smith & Price, 1973) and has been used to model aspects of animal behavior in many species. A possible payoff matrix is as follows in Table 4-5.

Table 4-5: Hawk-Dove Payoff Matrix

	Hawk	Dove
Hawk	-2, -2	6,0
Dove	0,6	3,3

Many competitive business situations can be modeled as a Hawk-Dove game, for example a competitor entering a market currently dominated by an existing player. The existing player can chose to fight by lowering prices or other marketing spending or acquiesce some market share to the new player. Anderton (Anderton, 2003) has studied competitive behavior in developing economies using Hawk-Dove models.

4.4 Repeated Games

The taxonomy of games is extensive. One important categorization is that between one-shot and repeated games. In repeated games, players remember the history of interaction and respond accordingly. For example, in the iterated version of Prisoners' Dilemma, a strategy that often

emerges is tit-for-tat where a player will defect if the opponent defected in the past otherwise cooperate. Unlike the one-shot version, the Iterated Prisoners' Dilemma can lead the players to cooperation, resulting in higher utility for both (Holland, 1992). The notion of repeated games with possibly continuous actions is utilized as the basis for Knowledge Distribution in CA and is discussed in the next section.

Repeated games are an important subclass of Competitive games that can prove useful in modeling many real-world phenomenon (Harrington & Zhao, 2012). Repeated games pose a challenge because the strategy space can be very large or even infinite. At any game stage, the players know the history of the game thus far; i.e. the actions all players have taken to get to the current stage. Thus, the actions taken at the current stage can depend on the history of the game that can become intractably large very quickly.

An approach to solving repeated games is based on average or payoff. The Folk[1] Theorem (Shoham & Leyton-Brown, 2009) is useful for determining an equilibrium based on feasible and enforceable payoff profiles. The basic idea is that in an infinitely repeated game the average payoff attainable in an equilibrium is the same as the Nash equilibrium in a single stage game. The constraint is that each player must obtain at least the minmax payoff.

Formally,
let $G = (N, A, u)$ be a normal form game and $r = (r_1, r_2, \dots, r_n)$ be any payoff profile.
$v_i = \min\max u_i(s^{-i}, s_i)$ is the player $i's$ minmax value, i.e. the payoff i receives when other players play minmax strategies against i
r is feasible i.e. it can be constructed from the individual payoffs in the game
r is enforceable $r_i \geq v_i \; \forall \; i$

[1] The Folk Theorem is named as such because like a folk song it has been generally known for a long time but no one knows who originally authored it. Nonetheless, it is widely accepted in the game theory community, see (Shoham & Leyton-Brown, 2009).

By the Folk Theorem[1] r is the payoff profile for some Nash equilibrium of the infinitely repeated game G with average payoffs.

Consider the knowledge distribution game, one can take the position that the same game will be played repeatedly between the same network individuals, over many generations, so one can analyze the game as an infinitely repeated game to find a solution or an equilibrium.

4.5 Continuous Action Games

The vast majority of games in Game Theory are restricted to discrete action choices, as they are easier to analyze, however, game actions can also be continuous real values. Veelen and Spreij have analyzed games in the continuous action space (Veelen & Spreij, 2009).

To study a wide variety of possible game-based knowledge distribution mechanisms, it was deemed necessary to support both discrete as well as continuous action spaces. For example, a game may be structured such an individual could choose an action from a discrete set, e.g. Cooperate or Defect. Whereas in a different game, the chosen action can be continuous, e.g. the fitness of the player. The same general mechanism can be used for both discrete and continuous action games.

4.6 Chapter Summary

Game Theory is a vast and deep subject area. The core ideas of this research are inspired by games in classical and evolutionary game theory. This chapter provided a brief overview of Game Theory with enough terminology and formalism to understand the abstract game framework presented in the next chapter. This framework is the basis of the games implemented and tested for knowledge distribution.

CHAPTER 5 GAMES AS MECHANISMS FOR KNOWLEDGE FLOW IN CULTURAL ALGORITHMS

5.0 Introduction

The CATGame system was developed with a general mechanism for injecting arbitrary games for knowledge distribution. The general mechanism is abstract and requires a concrete game to be operational.

Recall that each Knowledge Source is a certain type of search strategy (Chapter 2) and that the Cultural Algorithm functions as a hyper-heuristic in order to select the appropriate metaheuristic in each context. At the macro level, a desirable property of the KD mechanism is to achieve a dynamic balance between the various KS, as the CA proceeds in exploring the problem landscape. One notion of balance is that the mix or proportion of KS in the population can be varied in order to best facilitate search space exploration and exploitation at a given phase of the problem-solving process. Another assumption is that all of the KS maintain at least some presence in the population and is not crowded out completely by a dominant KS.

Games are a convenient way to balance the conflicting demands of exploration and exploitation at the macro and micro levels. The general premise here is that individuals in the population have the ability to either cooperate or compete for knowledge. The next section describes the abstract mechanism for injecting games for knowledge distribution. The structure defined here is leveraged in Chapter 6 that defines the mechanisms of the concrete games studied for this research.

5.1 Abstract Game Mechanism

The minimal terminology given in the previous section is leveraged now to describe the abstract structure of the game theoretic KD mechanism. This mechanism is structured in terms of the following components: Actor; Play; Action; Payoff; Payout; and Outcome:

Actor Population individual linked to other players via network. (terms 'individual', 'agent', 'player' and 'actor' are used interchangeably in this document).

Play function A function that produces the Action or 'hand' that a player plays against its neighbors. This function may take utilize the current and historic states of all players in addition to other available environmental information.

Payoff function A function that produces the Payout structure – which represents the utility to a player – given its own Action and those of its neighbors in the game.

Outcome function Given the population's collective Actions and Payouts, this function produces the updated population where each individual is assigned a KS based on the results of the collective game play.

The general mechanism abstracts out the common steps in game play. Specific game types can be injected into the mechanism by supplying a game instance which is a record of three functions: ***Play, Payoff and Outcome***. In other words, a concrete game can be injected if mutually compatible

implementations of these three functions are defined and grouped into a record structure. Such a record can be accepted by the general game mechanism. Mutual compatibility means that the data structures produced and consumed by these functions are consistent. For example, the Play function produces a list of Actions that are consumed by the Payoff function. The general game does not care about the details of the Action data structure as long as the Payoff function can accept it. The Action and Payout data structures produced and consumed by these functions are defined by the concrete game. The general game mechanism abstracts over these details and operates at a higher level and is depicted schematically in Figure 5-1.

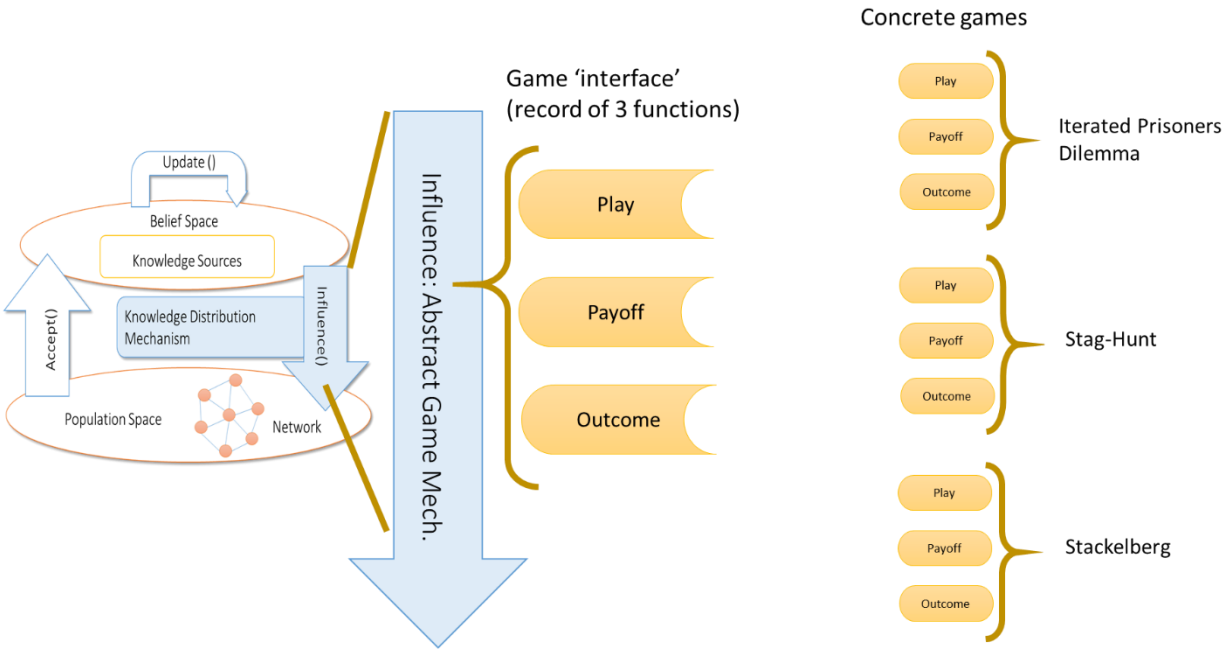


Figure 5-1: Abstract game interface

The high-level flow of the mechanism is described next and the corresponding pseudocode is given in Listing 5-1. There are two distinct phases. In the first phase, the Play function, supplied by the concrete game, is used to play games between an individual and its network neighbors. The neighbors for each individual are obtained by using the Network function configured in the current

instantiation of the CA. This phase produces a list of Actions for each player; each Action in this list is directed towards a neighbor. The first phase where an agent plays its hands against each of its neighbors is pictorially depicted in Figure 5-2.

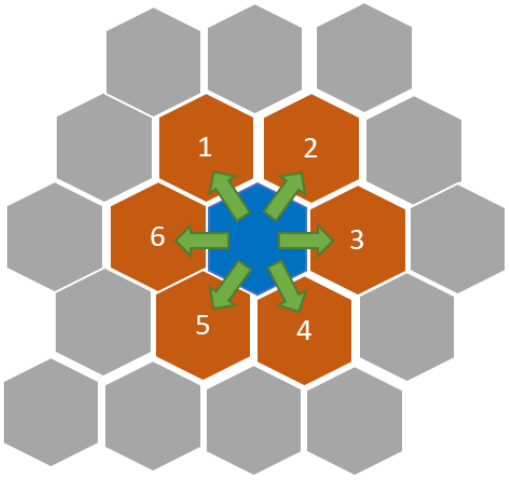


Figure 5-2: In the 1st phase each network individual plays actions against its neighbors, utilizing current and historic information

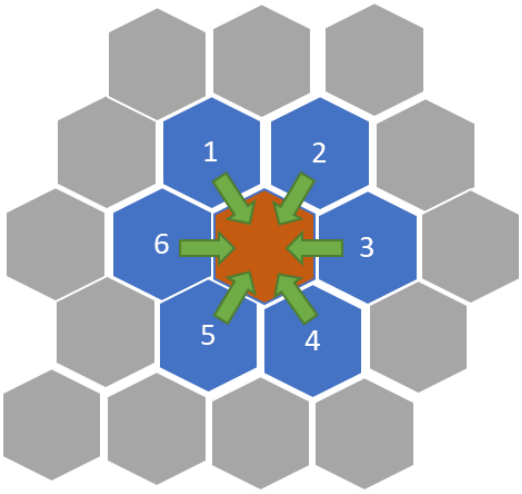


Figure 5-3: In the 2nd phase, actions played by an individual's neighbors are collected to determine payout

In the second phase, all Actions are aligned so that all Actions pertaining to a single individual are grouped together. These are the Actions the individual played against each of its neighbors and the Actions the neighbors played only against the said individual (Figure 5-3). From each such group the Payout for each individual is determined by using the Payoff function given by the concrete game. The general game mechanism then takes the Payouts for all individuals and passes them to the Outcome function (also supplied by the concrete game). The Outcome function produces a new population of individuals, each with a KS assignment determined by the associated Payout.

```
game = {Play, Payoff, Outcome}

GameKD (Pop, Network, game)
  Actions ← Empty
  Payouts ← Empty

  "1st phase – determine and play actions against each neighbor"

  FOR p in Pop DO
     neighbors ← Network(p)
     Actions[p] ← game.Play (p, neighbors)
  END FOR

  "2nd phase – determine payout based on actions neighbors played against
   indidvidual in 1st phase"

  FOR p in Pop DO
     neighbors ← Network(p)
     neighborActions ← [FOR j in ns → Actions[j]]
     Payouts[p] ← game.Payoff(p, pAction, neighborActions)
  END FOR

  Pop ← game.Outcome(Pop, Payouts, Actions)
  Return Pop
```

Listing 5-1 Pseudocode for a general game mechanism for Knowledge Distribution

To enable a truly abstract mechanism, the data types of the Action and Payout structures are also determined by the injected game. This is achieved via generic programming capability of the implementation language F# (Microsoft Corporation). The abstract mechanism is able to deal with any data structures that the injected game chooses to use. For example, the Actions produced by Play may be single values (e.g. fitness of the individual) or may be tuples (e.g. fitness with some other score). The general game mechanism is unaffected as long the corresponding Playoff function can accept the same structure. The main rationale for using generic programming is to allow for a wide variety of games to be studied under the general mechanism. This is the strength of this approach. The alternative would be to specify a more rigid (and less abstract) mechanism. Such a mechanism could be easier to use but it will not allow an expansive set of games to be studied.

The pseudocode in Listing 5-1 is now elaborated with a simple game for knowledge distribution. The game is posed as follows. Each individual plays the action 0 or 1 against its neighbors based on whether the individual's fitness improved since the previous generation or not. Each individual looks at its neighbors and adopts the KS that is the most frequent among neighbors who played 1. If no such neighbors exist, the individual retains its current KS. Ties are broken with random selection. Listing 5-2 is the pseudocode for injecting this game into the generic mechanism. The central idea is to define the Play, Payoff and Outcome functions and then package them up into a game record for injection.

Listing 5-2: Pseudocode for a sample game

```
//definition: supporting function definition needed by Play
//returns 1 if fitness improved from the previous generation, 0 otherwise
fitnessImproved : Indivdual -> {0,1}

//defintion: return the most frequent entry in the list
//if there are multiple entries with the same count
//then a random selection from the top most frequent is returned
//if the list is empty, null is returned
mostFrequent : (KS list) -> KS

//define Play function
def Play (indivdual, neighbors) =
   action <- fitnessImproved (indvidual)    //decide which action to take
   actions <- []
   //play action against each neighbor (each item is 4-tuple)
   For n In neighbors Do
     actions <- actions :: (individual.Id, n.Id, action, indivdiual.KS)
   End For
   Return actions      //Action structure is 4-tuple list

//define Payoff function
def Payoff (indvidual, indvidualAction, neighborActions) =
   ksSelected <- []
   For (id,nId,action,KS) In neighborActions Do
      if action = 1 then
         oneActions <- oneActions :: KS     //collect KS for action=1
   End For
   maxKS <- mostFrequent (ksSelected)
   newKS <- if maxKS = null then individual.KS else maxKS
   Return (individual.Id,newKS)            //Payout is tuple of id and KS

//define Outcome function
def Outcome(pop,payouts) =
   pop' <- []
   For p in pop do
     (id,newKS) <- payouts[p.Id]
     p.KS <- newKS
     pop'[id] <- p'
   End For
   Return pop'

game = {Play, Payoff, Outcome}       //game is a record of 3 functions
                                     //ready to be injected into the
                                     //generic mechanism in Listing 5-1
```

Note the generic mechanism is flexible because instead of accepting just a data structure (e.g. a payoff table) the mechanism can accept both code plus data structures. This makes it versatile and able to handle a diverse set of scenarios.

Much of game theory is concerned with finding a solution to a posed game as the strategy adopted by each player. The collective set of player strategies is the solution of the game. Assuming rational players, the action played by an agent should be the best response to the best responses of its peers, determined jointly, i.e. the Nash equilibrium (Nash, 1950). For many games, finding a solution is a computationally hard, especially for more than 2 players. In the proposed framework, each agent is concurrently participating in many games, with usually more than 2 players per game. To clarify, each agent is playing a game against its neighbors that are in turn playing different games with their neighbors. The final result is a set of interlinked games. There are no known (computationally tractable) analytical methods for solving such a complex set of interlinked games. To address this situation, one takes the view that each agent unilaterally decides to take action based on its and its neighbors' current and prior states, without regard to the actions that other agents make take in the current round (i.e. take a hedonic approach). Here the interest is in the emergent properties of the system given bounded rationality decisions of the agents.

5.2 Chapter Summary

This brief chapter explained the abstract game framework that can be used to inject arbitrary games for knowledge distribution. It provides the background to understand the concrete games that use this framework and are described in the next chapter. The mechanism described is

general and more-or-less can handle any arbitrary game for knowledge distribution. As such, it serves as way of facilitating future research on this topic.

CHAPTER 6 **CATGAME – COOPERATIVE/COMPETITIVE GAMES FOR KNOWLEDGE DISTRIBUTION**

6.0 Introduction

The abstract game mechanism presented in Chapter 5 is exercised with 3 specific games, namely:

- N-Player Iterated Prisoner's Dilemma
- Stag-Hunt
- Stackelberg

Each of these games is described in this chapter. For each, the historical background and perspective is given first followed by the implementation in relation to the abstract game mechanism described in Chapter 5.

The definition of 'player' is required for any game. In CATGame, two perspective can be taken: a) the player is a population individual who plays against its neighbors as determined by the network topology; or b) the player is a Knowledge Source in the Belief Space playing against other Knowledge Sources. Both perspectives are covered by the three studied games here. *Competitive* knowledge distribution is relatively easy to grasp – viz. a 'winner' Knowledge Source gets to influence the population individual. But what does it mean to cooperate in this context? The notion of cooperation is not as clear but is developed and clarified for the 3 games, in the respective sections.

All of the tested game mechanisms implement a concept from simulated annealing where the influence level (temperature or 'exploratiVeness') of the "stem cell" knowledge – Domain – is

reduced the longer an individual retains Domain from generation to generation up to a configured minimum. The influence level is reset whenever an individual is assigned Domain to replace a non-Domain knowledge.

6.1 Iterated Prisoners Dilemma – cooperation emerges over repeated interaction

The Prisoner's Dilemma (PD) game is probably the most studied game of the three as its can be applied in a number of scenarios (see 4.1) in the social sciences, military strategy, business and economics. The analytical solution for a single-shot (not repeated) Prisoner's Dilemma is to Defect (Axelrod & Hamilton, 1981). However, cooperation can emerge if the game is played repeatedly. Evidence that Iterated Prisoner's Dilemma (IPD) leads to cooperation comes from the fact that the winning strategy in Axelrod's famous tournament was tit-for-tat. Essentially the tit-for-tat strategy is: cooperate in the current round if the opponent cooperated in the prior round, defect otherwise. Glossing over the many nuances uncovered by years of research on this subject, one can justify IPD as a viable game for knowledge distribution when both cooperation and competition are desired.

IPD is usually analyzed as a two-player game but here the multiplayer version – n-player Prisoner's Dilemma (NPD) – is required as the number of players is more than two for any realistic knowledge distribution scenario in CA. A version of NPD is the well-known "Tragedy of the Commons" (Chappelow, 2019) that is often used to explain the depletion of common resources.

The payoff matrix for an NPD game is shown in Table 6-1 (Yao & Darwen, 1994) where the top row is number of cooperating players starting. The 2nd and 3rd rows are rewards for cooperation and defection, respectively. Both depend on the number of cooperating players in the game.

Table 6-1: Payoff Matrix for n-Player Prisoner's Dilemma

No. of Cooperators	0	1	…	x	…	N-1
Cooperate	C_0	C_1	…	C_x	…	C_{n-1}
Defect	D_0	D_1	…	D_x	…	D_{n-1}

The matrix is the same for every player (i.e. the game is symmetric). N is the total number of players in the game and so N-1 is the number of players other than the player whose perspective is represented by the matrix, denoted as P_{curr}. Each of the N players play their hand: Cooperate or Defect. If x other players cooperate (where x is between 0 & N-1) then the payoff for P_{curr} will be C_x if P_{curr} decides to cooperate and D_x otherwise. The following conditions need to hold for the game to be considered NPD:

- $D_x > C_x$ (defection provides a higher reward than cooperation)
- $D_{x+1} > D_x$ (the more cooperators there are the higher the reward for defecting)
- $C_{x+1} > C_x$ (same applies for cooperators)
- $C_{n-1} > D_0$ (if everyone cooperates, the reward is higher for each than if everyone defects)

To summarize, the reward for both cooperators and defectors increases with the number of cooperators but for any individual player the reward for defection is always higher. And total cooperation is more rewarding than total defection.

While such a payoff matrix may be constructed for any given scenario, usually the matrix is compressed into a payoff function by making some simplifying assumptions. The key consideration is that depending on how C_x and D_x are determined, there may be x, y $(x < y)$ where $C_x > D_0$ and $D_y < C_{n-1}$ which implies that the number of cooperators are between x and y for such a scenario.

The scheme for IPD based knowledge distribution is adapted from the NPD game. However, to use IPD for knowledge distribution, further details are required such as who are the players in the game; how should the players choose their actions; what happens when cooperating or defecting players receive their payoff (i.e. what should the knowledge distribution outcome be); etc. These are provided in the next section.

6.1.1 IPD Adaption for CATGame Knowledge Distribution

The IPD game for knowledge distribution is structured as follows:

- **Players**: Population individual and its immediate (1-hop) network neighbors
- **Action**: Cooperate if player's fitness was worse from prior generation, defect otherwise
- **Outcome**: Player choose defection for knowledge distribution if average defection reward is above a certain threshold, cooperation otherwise.

Note that in the case of CA population network, each player is playing a different game with its neighbors. As well, each player is participating in as many different games as there are neighbors (i.e. the degree of the regular network). In other words, there does not exist complete symmetry and reciprocity.

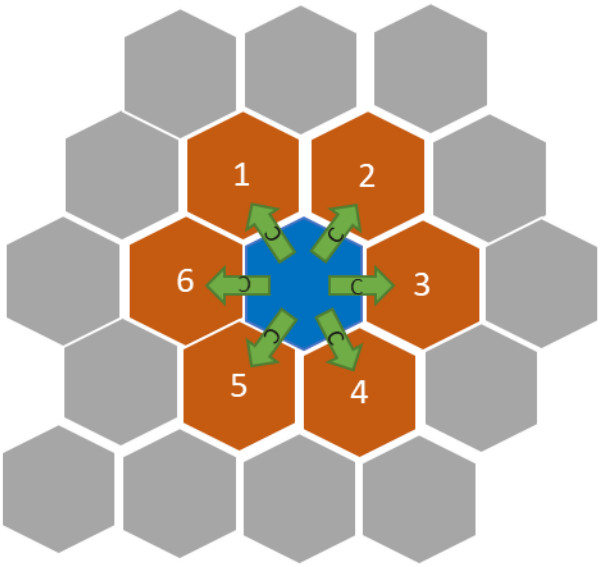


Figure 6-1: In n-player Iterated Prisoners Dilemma each individual chooses to either Cooperate (C) or Defect (D) against all its network neighbors

Recall the phases of game play outlined in Chapter 5. In the first phase, a population individual chooses to play D (for defect) if its fitness improved since prior generation, C (for cooperate) otherwise. A player can choose only one of the two actions. D implies that an individual will want retain a Knowledge Source as it is improving – there is no incentive to cooperate. Otherwise the player will want to cooperate but the final outcome for the player is determined by the player's action and those of its neighbors. This is where the payoff matrix in Table 6-1 comes into play. How many of the neighbors also want to cooperate? The detail decision process is described later in Listing 6-2 but in general if enough neighbors are cooperative, the player will be classified as cooperator. Otherwise it will be classified as a defector. Knowledge assignment (i.e. the outcome) depends on this classification (i.e. the payout). As an illustration, Figure 6-1 depicts an individual playing the C (cooperate) action against its neighbors. In the second phase, the actions played by

an individual's neighbors (C or D) (Figure 6-2) are collected to make the final determination about the payoff and the its implication for knowledge distribution.

If the individual is classified as a defector, it retains its current knowledge assignment. If is classified as cooperator then it gets an assignment based on its 'social rank' among its peers. If the player is performing relatively well in terms of fitness as compared to its peers (neighbors), it will be assigned a relatively exploitative KS otherwise a relatively exploratory one. All knowledge sources are ordered on the explorative-exploitative scale. The 'social rank' of the player (relative to its neighbors) determines which KS is chosen from the ordered set of KS. Thus, cooperation in the context of IPD means behaving according to "your rank in society". If the individual is not doing well (relative to its neighbors) then it should try harder by adopting a relatively exploratory strategy. If the individual is doing well relatively then it should adopt a more exploitative strategy. Under cooperation the individual does not make a unilateral (egoistic) decision. It looks at the (bounded) context and decides on what is the best for the collective as a whole.

To utilize the generic mechanism defined in Listing 5-1, a concrete game needs to be supplied that is a tuple (or record) of three functions – Play, Payoff and Outcome. These functions are formally defined using mathematical notation in this section. Listing 6-1 is a set of supporting definitions for specifying the game functions that are also referenced in later sections.

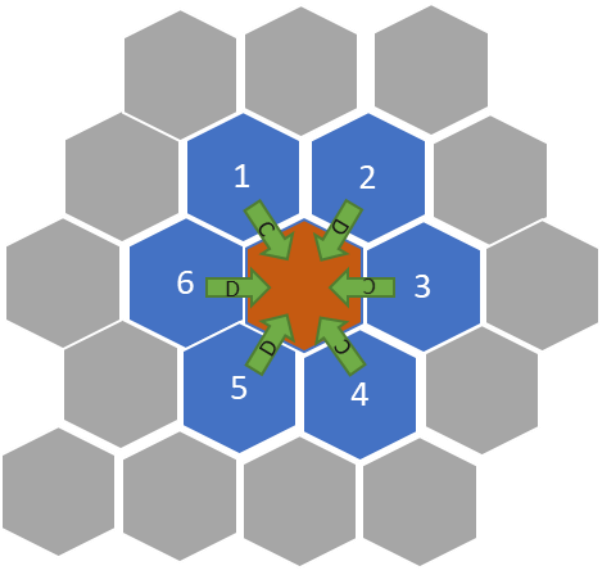


Figure 6-2: The payout for the individual is jointly determined by the individual and neighbors' actions. The payout determines whether the individual will adopt cooperative or comptetitive behavior for knowledge distribution with respect to its neighbors

Listing 6-1: Definitions to support game-based knowledge distribution

$game = (Play, Payoff, Outcome)$	From Listing 5-1 a game is a triple of 3 functions
$K = \{H, S, N, T, D\}$	Set of Knowledge sources
$Pop = \{(k, f, j) \mid k \in K, f \in \mathbb{R}, j \in \mathbb{Z}+\}$ $Pop' = powerset\ of\ Pop$	Population is a triple of k=Knoweldge Source, f=fitness and j=number of generations the indivudal had the same k. Assume the triple has a unique identity not shown here
$PriorFit : Pop \rightarrow \mathbb{R}$	Function that provides the fitness in the prior generation of a population individual (its implementation is context dependent)
$Network : Pop \rightarrow Nbrs, \quad Nbrs \subseteq Pop'$	Network function definition
$Play : Pop \rightarrow Network \rightarrow Actions$	Play function defintion. The function takes a population indivdual and the network function to produce a set of actions played by the individual against its neighbors

$Payoff : Pop \rightarrow Action \rightarrow Actions' \rightarrow Payout$	Payoff function definition - computes the payout for a population indvidual given its own action and those of its neighbors towards the individual
$Outcome : Pop' \rightarrow Payout' \rightarrow Actions' \rightarrow Pop'$	Outcome function definition - takes the population the payouts and actions for all individuals and returns an updated popultation with new KS assignements
$Actions = < generic >$ $Actions' = poweset\ of\ Action$ $Payout = < generic >$ $Payout' = powerset\ of\ Payout$	The Actions are Payout structures for each game is definded by each game individually. Here they are defined generically to complete the type signatures required for injected games.

Given the supporting definitions in Listing 6-1, the implementation of the IPD knowledge distribution is in Listing 6-2.

Listing 6-2: IPD game definition

$A = \{C, D\}$	The set of actions players play in the 1st phase
$Actions = \{(p1, p2, a) \mid p1 \in Pop, p2 \in Pop, a \in A)\}$ $Actions' = powerset\ of\ Actions$	IPD Actions is a triple of values representing the action played by an individual against its neighbor
$Decision = \{Cooperate, Defect\}$ $Payout = \{(p, d) \mid p \in Pop, d \in Decision\}$ $Payout' = powerset\ of\ Payout$	Payout structure is a set of tuples that maps to a decision taken by each indivdual in the 2nd phase, after all games have been played and all actions known
$Play\ (p)(network) =$ $a \leftarrow \begin{cases} D & : p.f > PriorFit(p) \\ C & : otherwise \end{cases}$ $nbrs \leftarrow network(p)$ $actions \leftarrow \{(p, n, a) \mid n \in nbrs\}$ $return\ actions$	Play implementation for IPD p.f is short hand for the fitness value of a population individual triple
$Payoff\ (p)\ (a)\ (neighborActions) =$ $actions \leftarrow \{a\} \cup neighborActions$	Payoff implemetnation for IDP

$defections \leftarrow \{d \mid d \in actions \text{ and } d = D\}$ $avgDefection \leftarrow \frac{\lvert defections \rvert * ALPHA}{\lvert actions \rvert}$ $decision \leftarrow \begin{cases} Defect & : avgDefection > 1.0 \\ Cooperate & : otherwise \end{cases}$ $return\ decision$	ALPHA = 1.5 If enough neighbors are defecting (i.e have played action D) then defect otherwise cooperate. By using ALPHA > 1 the game is biased towards defection
$Outcome\ (network)\ (pop)\ (payouts)\ (actions) =$ $newPop \leftarrow \{Dist(p)\ (Dscn(p)\ (payouts)), network(p)) \mid p \in pop\}$ $return\ newPop$	Outcome function implementation. It relies on supporting functions Dist and Dscn that are given below
$Dscn : Pop \rightarrow Payout' \rightarrow Decision$	Function definition to return a decision (cooperate or defect) made by population individual given the decisions (payouts) for all individuals
$Dist : Pop \rightarrow Decision \rightarrow Nbrs \rightarrow Pop$	Function definition for returning an updated population individual given its decision (to cooperate or not) and its neighbors
$Dist\ (p)(d)(nhbrs) =$ $(k, f, j) \leftarrow p$ $newK = \begin{cases} CoopDist\ (p)\ (nhbrs) & : d = Cooperate \\ CompDist(p)\ (nhbrs) & : d = Defect \end{cases}$ $newJ = \begin{cases} j + 1 & : k = newK \\ 0 & : otherwise \end{cases}$ $newP = (newK, f, newJ)$ $return\ newP$	Dist function implementation. It relies on several functions that are defined and described below
$SocialRank : Pop \rightarrow Nbrs \rightarrow \mathbb{Z}$	Given a population indivdual and its neighbors this funtion returns the rank of the indivdual among its neighbors based on relative fitness
$KSforRank : \mathbb{Z} \rightarrow K$	Given an integer rank from the SocialRank function, this function returns a Knoweldge Source. The Knowledge sources are ordered from explorative to exploitative. A low rank is associated with (relatively) explorative KS and high rank with

	exploitative. The premise is that if an individual is performing comparatively well, it should continue to exploit the local region it is in. This is a configurable value to enable the provision of different rankigns of KS for different problem types.
$CoopDist = SocalRank \circ KSForRank$	Cooperative distribution: Function composed of SocialRank and KSForRank determines K when individal decides to be Cooperative
$CompDist : Pop \rightarrow Nbrs \rightarrow K$	Competitive distribution: Function that determines K when individual decides to be Competitive. Here the locally dominant Knowledge Source is returned using weighted average fitness to rank the Knowledge Sources of individual and its neighbors.
$game = (Play, Payff, Outcome \circ Network)$	The game tuple for IPD game injection. Note that to match the required definition for the Outcome function for game injection, the Outcome function defined above is composed with the Network function so the type signatures match.

In summary, IPD knowledge distribution is an adaption of the n-player IPD. In the 1st phase, players unilaterally play defect (A=D) if their current fitness is better than prior fitness. In the 2nd phase, if enough players did defect in a player's neighborhood, the outcome for the player is the Defect decision, otherwise its Cooperate. If the final decision is to Cooperate, the Knowledge

Source assigned is based on the ‘social rank’ of the agent and the exploitative-to-explorative ordering of the Knowledge Sources. For Defect, the agent just keeps the current assignment. Reusing some of the definitions in Listing 6-1, the Stag-Hunt game is defined next.

6.2 Stag-Hunt – Cooperation by default

Stag-Hunt can be considered an extension of IPD with an explicit notion of time involved. The players by default cooperate to hunt a stag but as time goes by and no stag is sighted, the players become impatient and can defect to hunt a rabbit. Stag-Hunt comes from evolutionary game theory (Weibull, 1995) whereas IPD is well studied in both classical and evolutionary game theory. Contemporary evolutionary game research is usually performed with computer simulation (Dong, Xu, & Fan, 2019) (Wang, Luo, Ding, & Wang, 2018) because the dynamics can be complex and not always capturable analytically, as is possible for classical games. A payoff matrix type formulation thus is not very instructive for n-player evolutionary games.

While IPD is a series of single-shot games, Stag-Hunt is a game that is played over some units of time. The useful notion of time in CA is generations (i.e. when a new population is generated). For CA knowledge distribution, the Stag-Hunt game is played continuously across generations till the optimization run is terminated. For CA knowledge distribution, there is a configured number of cooperative generations followed by an evaluative one. This is pictorially depicted in Figure 6-3; here there are 3 cooperative generations followed by one evaluative one.

Figure 6-3: In Stag-Hunt individuals cooperate for a fixed number of generations and then evaluate – a pattern that is repeated till max number of generations is reached

During the cooperative phase, all individuals cooperate – where the notion of cooperation is very similar to the altruistic one used in IPD (section 6.1) i.e. one based on 'social rank'. Note that Stag-Hunt is cooperative by default. All individuals are cooperating in the cooperative generations whereas in IPD the decision to cooperate or defect is made individually by each player at each generation.

In an evaluative generation, each individual evaluates whether to continue cooperation or to defect. If the individual's fitness is improved since the last evaluative generation (or the initial fitness), it defects by keeping its current knowledge assignment. Otherwise it cooperates but instead of choosing a KS based on social rank, it chooses the locally dominant KS. Here "locally dominant" means the KS that has the highest weighed average fitness among the individual's current KS (direct influence) and those of its neighbors.

Under Stag-Hunt there are two types of cooperation. First is based on social rank and is similar to the one in IPD. This type is used by all individuals in a cooperative generation. Second is based on adopting a strategy that is performing the best overall in the local neighborhood (bounded context). It is used in an evaluative generation under the decision to cooperate. The structure of the Stag-Hunt is given next followed by its detailed formulation using mathematical notation.

6.2.1 Stag-Hunt Adaptation for CATGame

The Stag-Hunt game is structured as follows:

- **Players**: The population individual and its immediate (1-hop) network neighbors
- **Action**: The players play their fitness value as the (continuous) action in the 1st phase, regardless of cooperative or evaluative generation. The actions are used later in the 2nd phase to determine how knowledge is distributed.
- **Outcome**: In a cooperative generation, a cooperative strategy is used for every player where the KS assigned is determined from the player's social rank (similar to IPD), as depicted in Figure 6-4. Each player receives the fitness values of the neighbors as the actions taken by the neighbors. It then determines its social rank by comparing its own fitness with those of the neighbors'. Based on the rank it adopts a KS from an ordered set. If the rank is relatively low it will adopt a relatively exploratory KS and vice-a-versa. In an evaluative generation a player defects by keeping its current KS, if the player's fitness improved since prior evaluation otherwise the player accepts the locally dominant KS (Figure 6-5). As explained earlier, the locally dominant KS is the one with the highest weighted average fitness in the neighborhood. This is still a type of cooperation but one that is done in the evaluative generation.

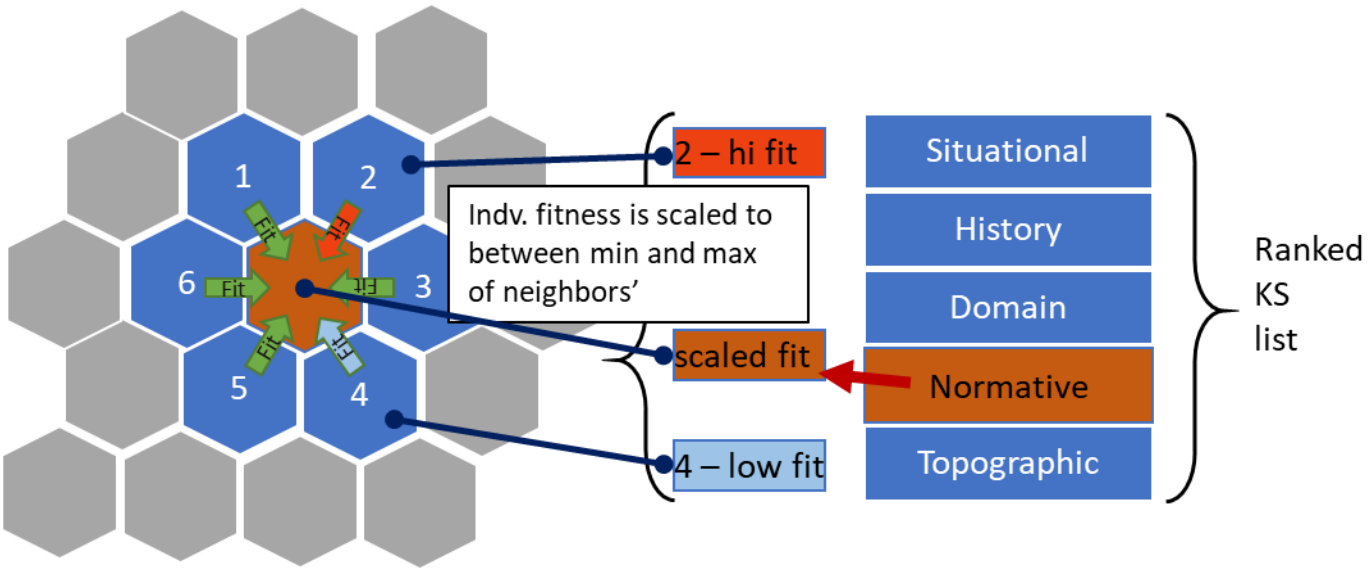
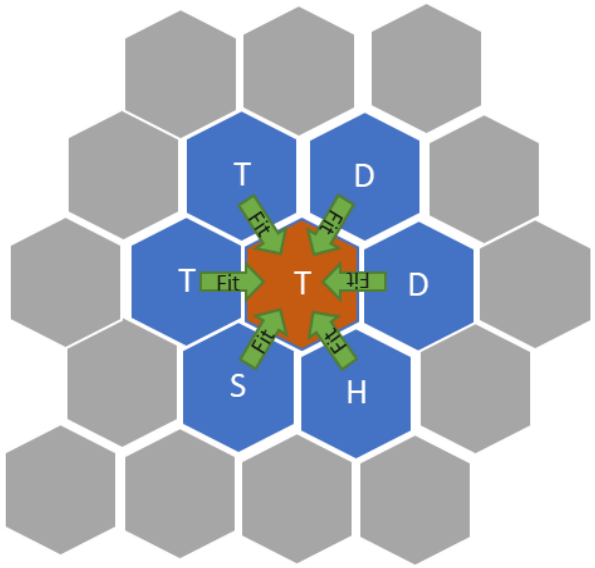


Figure 6-4: In a cooperative generation, the individual is ranked between the fitness of its highest and lowest fit neighbors on a continuous scale. Based on its rank, the individual is assigned the best matched KS from a ranked list



$$Fit(ks) = \sum_i fit_i * \begin{Bmatrix} 1: KS(i) = ks \\ 0: otherwise \end{Bmatrix}$$

$$Selected\ KS = argmax_{ks}(Fit\ (ks))$$

Figure 6-5: In an evaluative generation, individual is assigned a KS that has the best weighted fitness among individual and neighbors, if the individuals fitness has not improved

A formal definition of the Stag-Hunt game for CA knowledge distribution is given in Listing 6-3.

Listing 6-3: Stag-Hunt game definition

$Actions = \{(p1, p2, a) \mid p1 \in Pop, p2 \in Pop, a \in \mathbb{R})\}$ $Actions' = powerset\ of\ Actions$	Stag-Hunt Actions structure is a triple of values representing the action played by an individual against its neighbor. Here the 1st phase action is the fitness value (which is continuous)
$Payout = Actions$ $Payout' = Actions'$	The payout structure is the same as the Actions structure in Stag-Hunt game.
$Play\ (p)\ (network) =$ $a \leftarrow p.f$ $nbrs \leftarrow network(p)$ $actions \leftarrow \{(p, n, a) \mid n \in nbrs\ \}$ $return\ actions$	Play implementation for Stag-Hunt p.f is short hand for the fitness value of a population individual triple
$Payoff\ (p)(a)(neighborActions) =$ $payout \leftarrow \{a\} \cup neighborActions$ $return\ payout$	Payoff function in Stag-Hunt passes the collected actions for each individual as the Payout structure to be used in the outcome function. These are actions that the individual played against each of its neighbors and those that the neighbors played against just this individual. It represents all the actions pertaining to a single individual
$Outcome\ (network)\ (pop)\ (payouts)\ (actions) =$ $newPop \leftarrow \{Dist\ (p)\big(Indvp\ (p)(payouts)\big)(network) \mid p \in pop\}$ $return\ newPop$	Outcome function implementation. It relies on supporting functions Dist and IndvPayouts that are given below
$IndvP \rightarrow Pop \rightarrow Payout' \rightarrow Payout$	Function definition to return the Payout for the given individual

	from the Payouts collected for all users. The generic game mechanism (Chapter 5) packages all payouts into a single collection for all users. This function seperates out the ones for the supplie d user as that is required in the Outcome function
$Dist : Pop \rightarrow Payout \rightarrow Network \rightarrow Pop$	Function definition for returning an updated population individual given its Payout structure
$Dist\ (p)(payout)(network) =$ $(k, f, j) \leftarrow p$ $newK = \begin{cases} CoopDist(p)(payout)(network) & : IsCoopGen( ) \\ EvalDist(p)(payout)(network) & : otherwise \end{cases}$ $newJ = \begin{cases} j+1 & : k = newK \\ 0 & : otherwise \end{cases}$ $newP = (newK, f, newJ)$ $return\ newP$	Stag-Hunt Dist function implementation. It relies on several functions that are defined and described below
$IsCoopGen : \{true, false\}$	Supporting function that returns true if the current generation is cooperative, false otherwise (i.e. evaluative)
$SocialRank : Pop \rightarrow Nbrs \rightarrow Payout \rightarrow \mathbb{Z}$	Stag-Hunt version of social rank Given a population indivdual and its neighbors and the payout, this funtion returns the rank of the indivdual among its neighbors based on relative fitness (see Figure 6-4)
$KSforRank : \mathbb{Z} \rightarrow K$	Given an integer rank from the SocialRank function, this function returns a Knoweldge Source. The Knowledge sources are ordered from explorative to exploitative. A low rank

	is associated with (relatively) explorative KS and high rank with exploitative. The premise is that if an individual is performing comparatively well, it should continue to exploit the local region it is in. This is a configurable value to enable the provision of different rankigns of KS for different problem types.
$CoopDist = Pop \rightarrow Payout \rightarrow Network \rightarrow K$	Cooperative distribution function defintion for Stag-Hunt. Implmentation given next
$CoopDist\ (p)(payout)(network) =$ $nbrs \leftarrow network(p)$ $newK\ \leftarrow (SocalRank \circ KSforRank)(p)(nbrs)(payout)$ $return\ newK$	Implementation of the cooperative distribution function defined above
$EvalDist : Pop \rightarrow Payout \rightarrow Network \rightarrow K$	Evaluative distribution: Function that determines K in an evalutive generation
$EvalDist\ (p)(payout)(network) =$ $nbrs\ \leftarrow network(p)$ $(k, f, j) \leftarrow p$ $newK\ \leftarrow \begin{cases} k & : PriorFit(p) < f \\ DominantK(pop)(nbrs)(payout) & : othewise \end{cases}$ $return\ newK$	Evaluative distribution function implementation
$DominantK \rightarrow Pop \rightarrow Nbrs - Payout \rightarrow K$	Function defintion to return the locally dominant K given the indviduals, it neighbors and Payout (see Figure 6-5)
$game = (Play, Payff, Outcome \circ Network)$	The game tuple for Stag-Hunt game injection. Note that to match the required definition for the Outcome function for game injection, the Outcome function defined above is composed with the Network function so

	that the type signatures match.

Stag-Hunt can be considered a version of IPD where the element of time is explicitly considered. Unlike IPD where cooperation emerges over repeated interaction, Stag-Hunt is cooperative by default. Individuals periodically check to see if they want to defect or not but generally cooperate. In terms of knowledge distribution, cooperative behavior is very similar to that for IPD's in that an individual will obtain a new KS as a function of its social rank among peers.

6.3 Stackelberg – A structured model for cooperation

The structure of this game is modeled after Stackelberg pricing model in microeconomics (Evans, 2014). Stackelberg, Cournot, and Bertrand are related models of oligopoly market competition. Unlike perfect competition (where participants have no control over prices) or monopoly (where there exists complete pricing power), in oligopoly, the firms have a degree of pricing power, determined by their "strategic complementarities" (Julien, 2011). But importantly for CA knowledge distribution, the market interaction can give rise to implicit cooperation as production (and pricing) decisions emerge from the inherent market structure.

In the classic Stackelberg model (Julien, 2011) there are two firms – a leader (or first-mover) and a follower. The leader firm moves first to set production and target price by taking into account the reaction of the follower firm. It knows how the follower will react and so it sets production and price that is at the expected equilibrium between the two firms. The leader has the first-mover advantage and will be able to command a higher price as a result, especially if the cost of entry is high (Annen, 2019).

Unlike the previously discussed two games (IPD and Stag-Hunt), the Stackelberg game is played between the Knowledge Sources that reside in the Belief Space. The adaption of Stackelberg for CA knowledge distribution is described next.

6.3.1 Stackelberg Adaptation for CATGame

The application of the Stackelberg concept for CA knowledge distribution is as follows:

- **Players**: Knowledge Sources in the Belief Space
- **Action**: Knowledge Source play their strength in terms of their current weighted average fitness represented in the population
- **Outcome**: The strongest Knowledge Source moves first and acquires the top n/k players in the population where n is the number of population individuals and k is the number of Knowledge Sources. The next KS takes the next strongest n/k player and so on

Here the Knowledge Sources act as firms in the Stackelberg model. The first-mover is the Knowledge Source that is the strongest in the current generation; it acquires the best performing individuals. Other Knowledge Sources take turns in order of their strength and acquire remaining individuals in a step-by-step manner. The overall population is equally divided among the available Knowledge Sources. This process is conceptually represented in Listing 6-4 and schematically in Figure 6-6.

Listing 6-4 Knowledge Sources assignments under Stackelberg

$P = < p_1, p_2, \dots, p_i, \dots, p_n >$	`is ranked list of population indviduals by fitness`
fit_i	`is the fitness of the` i^{th} `individual`
$K = \{H, S, N, T, D\}$	`is the set of KS used`

$q = \|K\|$	is the number of KS in the K set (5 in most cases)
$W_{k \in K} = \frac{1}{\sum_i fit_i} \sum_i fit_i \mid k_i = k$	is the relative weight of each KS in the population
$\sum_k W_k = 1$	the total weight for all KS sums to 1
$R = < k_1, k_2, \dots, k_q >, \quad k \in K$	is a ranking of KS by weight; strongest KS first
$S = < P_1, P_2, \dots, P_q >$	is a partitioning of the population into q portions. Each partition is of size $1/q$ [or $floor\left(\frac{1}{q}\right)$ to be exact] except for the last one that consists of the remaining individuals after $q-1$ partitions have been taken, respecting the ranking in P
$for\ p\ in\ P_1 \rightarrow p.KS \leftarrow R_1$ $for\ p\ in\ P_2 \rightarrow p.KS \leftarrow R_2$ … $for\ p\ in\ P_q \rightarrow p.KS \leftarrow R_q$	The top performing indviduals get the top KS and so forth for each partition

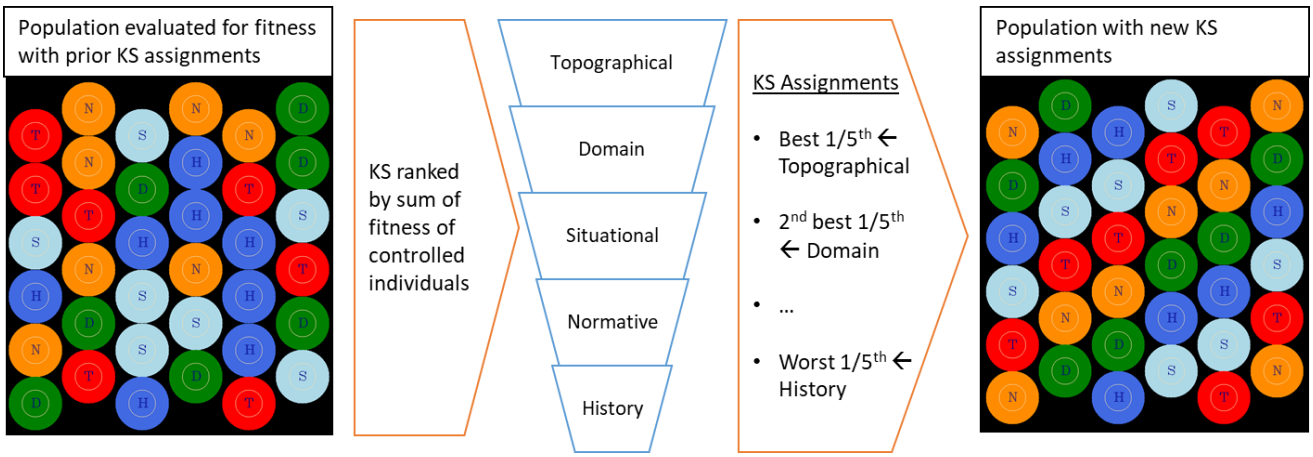


Figure 6-6: In Stackelberg the fittest KS takes the top 1/kth of the population and so on, where k is the number of configured KS

While the true players are Knowledge Sources in the Stackelberg game, the game still has to be structured in a manner to be injectable into the generic game described in Chapter 5. The formal specification for Stackelberg follows a model similar to the specifications of other games described earlier. The formal specification for the Stackelberg game is given in Listing 6-5.

Listing 6-5: Stackelberg game specification

$Actions = \{(p1, p2, a) \mid p1 \in Pop, p2 \in Pop, a \in \mathbb{R})\}$ $Actions' = powerset\ of\ Actions$	Stackelberg Actions structure is a triple of values representing the action played by an individual against its neighbor. Here the 1st phase action is the fitness value (which is continuous). Eventually these will be funneled into calculating the weighted fitness for Knowledge Sources. The game framework requries information to come from population individuals, which this structure contains.
$Payout = Actions$ $Payout' = Actions'$	The payout structure is the same as the Actions structure in Stackelberg.
$IsDistributionGen: \{true, false\}$	In Stackleberg, knowledge distribution is performed after a configured number of generations. This utility function provides whether the current generation is for distribution
$Play\ (p)(network) =$ $a \leftarrow p.f$ $nbrs \leftarrow network(p)$ $acts \leftarrow \{(p, n, a) \mid n \in nbrs\}$ $actions \leftarrow \begin{cases} acts & : IsDistributiveGen() \\ \{\} & : otherwise \end{cases}$ $return\ actions$	Play implementation for Stackelberg p.f is short hand for the fitness value of a population individual triple. Actions are only collected during a distributive generation
$Payoff\ (p)\ (a)\ (neighborActions) =$ $pout \leftarrow \{a\} \cup neighborActions$ $payout \leftarrow \begin{cases} pout & : IsDistributiveGen() \\ \{\} & : otherwise \end{cases}$ $return\ payout$	Payoff function in Stackelberg returns an empty set if its not a distributive generation. Othewise it passes the collected actions for the given individual as the Payout structure (to be used in the outcome

	function). These are actions that the individual played against each of its neighbors and those that the neighbors played against just this individual. It represents all the actions pertaining to a single individual
$Outcome\ (pop)\ (payouts)\ (actions) =$ $newPop \leftarrow \begin{cases} Dist(pop, payouts) & : IsDistributiveGen(\,) \\ pop & : otherwise \end{cases}$ $return\ newPop$	Outcome function implementation. It relies on the Dist supporting function given below
$Dist : Pop' \rightarrow Payout' \rightarrow Pop'$	Fuction definition to return an updated population (with new Knowledge Source assignments) given current population and all the payouts from the game
$RankKS : Payout' \rightarrow < k \mid k \in K >$	Function definition to provide an ordered set of Knowledge Sources – denoted with <…> - given the Payouts from all individuals in the population. The Knowledge Sources are ranked by the sum of the fitness values of the indivduals they control
$RankPop: numKs \rightarrow Pop' \rightarrow < sp \mid sp \subseteq Pop' >$	Function definition to return ordered set of subsets (chunks) of the population. The number of chunks is the number of KS in the system. The first chunk contains the most fit individuals and so on
$AssingKS: Pop' \rightarrow K \rightarrow Pop'$ $AssignKS(pop, k) = return\ \{SetKS\ (p)\ (k) \mid p \in pop\}$	Function definition and implementation to assign new Knowledge Source to a population chunk

$SetKS: Pop \rightarrow K \rightarrow Pop$ $SetKS(p)(newK) =$ $(k, f, j) \leftarrow p$ $newj \leftarrow \begin{cases} j+1 & : k = newK \\ 0 & : otherwise \end{cases}$ $return\ (newK, f, newJ)$	Function definition and implementation to assign a new Knowledge Source to a population individual
$Dist\ (pop)$(payouts) = rankedKs $\leftarrow$ RankKS(payouts) rankedChnks $\leftarrow$ RankPop($\|rankedKs\|$, pop) $rankedKs = < k1, k2, \dots, ki, \dots \mid\ i = 1\,..\,\|rankedKs\| >$ $rankedChnks = < c1, c2, \dots, ci, \dots \mid\ i = i\,..\,\|rankedKs\| >$ $assignedChnks \leftarrow < AssignKS(ci)(ki) \mid$ i = 1..|rankedKs| $>$ pop2 $\leftarrow \bigcup assignedChnks$ newPop $\leftarrow \begin{cases} pop2 & : IsDistributiveGen(\ \ ) \\ pop & : otherwise \end{cases}$ return newPop	Stag-Hunt Dist function implementation. It relies on several functions that are defined and described below
$game = (Play, Payff, Outcome)$	The game tuple for Stackelberg game injection.

6.4 Chapter Summary

The three game-based knowledge distribution mechanisms that are implemented and tested in this research were described in detail in this section. These mechanisms are inspired by the following: Iterated Prisoner's Dilemma from classical Game Theory; Stag-Hunt from evolutionary Game Theory; and Stackelberg from microeconomics. All use the game framework defined in Chapter 5. Stackelberg is a model where implicit cooperation emerges as a property of the market structure. The Stackelberg inspired game for CA knowledge distribution has Knowledge Sources taking turns to claim the best performing individuals in order or their strength. When compared to IPD and Stag-Hunt, Stackelberg is more structured in that the population individuals are generally evenly divided among the Knowledge Sources. It can be likened to a more centrally planned economy where resource allocation decisions are made at a higher level. By contrast in

IPD and Stag-Hunt, the decisions are made locally by each individual based on the individual's social rank among its peers. From an economic perspective, IPD and Stag-Hunt – in the context of CA knowledge distribution – behave more like a free market system.

CHAPTER 7 EXPERIMENTAL FRAMEWORK FOR UNDERSTANDING SYSTEM LEARNING IN DYNAMIC ENVIRONMENTS

7.0 Introduction

CA is meant for knowledge-driven problem solving in complex environments. An example of such is the multi-objective systems modelling of prehistoric environments (Stanley S. D., 2020). For the numerical optimization system CATGame, the Cones World dynamic environment generator is used. Cones World is an adaptation of the DF1 generator (Morrison & De Jong, 1999) for use in Cultural Algorithms. The landscapes generated by Cones World are periodically modified with a sequence generator based on the logistic equation (Eq **7**-**2**). This serves as a mechanism to create dynamic environments. The complexity is controlled by the 'a' multiplier (henceforth referred to as A or A value) of the logistic equation. This process is explained in greater detail in section 7.1.

The testbed allows the performance landscaped to be replaced with a new one while the optimization is underway. This is akin to pulling the proverbial rug from under the system. The A values control how hard the rug is pulled (on a periodic basis). A robust system should be able adapt quickly to changing environmental conditions. A resilient system should be able to withstand even large shocks without leading to system collapse (Figure 7-1). The testbed system is run with varying degrees of shocks applied to understand the behavior of the knowledge distribution mechanisms under different levels of dynamic complexity. The metric that captures the overall system performance is the number of generations to find optimum after each landscape change. Section 7.2 explains this performance metric and the core experimental setup.

Operation of the Knowledge Sources – i.e. how they guide the population individual through the search landscape is constant across the different distribution mechanisms tested. Since the performance difference is only due to how knowledge is distributed via the network, several metrics related to characteristics of the 'social' network are collected. These include diffusion (section 7.3); Schelling's segregation index (section 7.4); and others related to capturing the flow of knowledge in a graph e.g. Page-Rank (Wills, 2006)(section 7.5). The social metrics are intended to shed a brighter light into the emergent patterns of knowledge flow with respect to complexity changes.

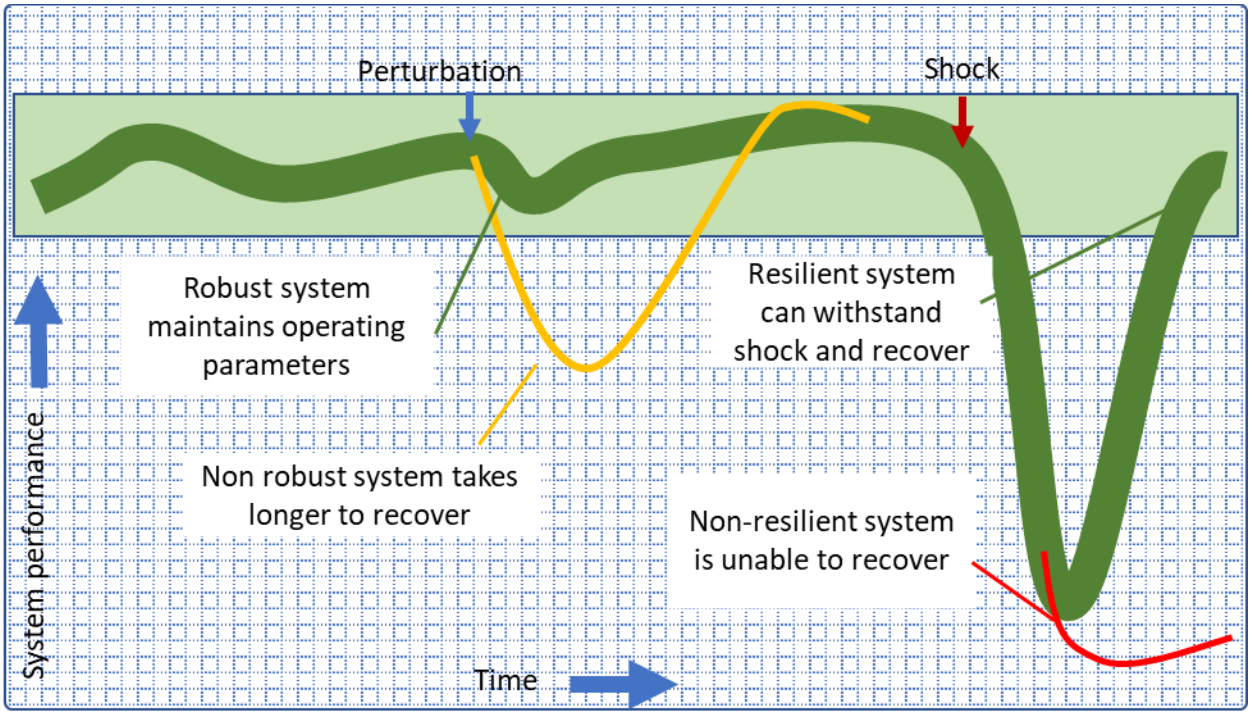


Figure 7-1: Resilience and robustness of complex systems

The main question that this research addresses is whether the hitherto unapplied cooperative approach to CA knowledge distribution is effective for certain categories of complex problems. As noted earlier there are two notions of complexity being addressed – dynamic complexity for

numerical optimization problems and hierarchical complexity for deep learning models; the former is addressed by CATGame and later by the CATNeuro system. As this chapter relates to numerical problems, the hypotheses arising from the primary question for numerical problems are posed in this chapter. A labeling scheme is defined to refer to hypotheses with the pattern '**Hy Chapter-#**'. To start, hypothesis Hy **7-1** reflects the primary research question with respect to dynamically complexity for numerical problems.

Cooperative knowledge distribution is effective for problem solving in dynamically complex environments	**Hy** 7-1

The following sections describe the experimental testbed and the metrics collected in more detail. Several other hypotheses are posited in the following sections in proximity with the description of the said metrics. The analysis and interpretation of the experimental data collected with the testbed, are presented in Chapter 8.

7.1 Cones World with Dynamic Landscapes

The Cones World test problem generator is a relatively simple method of constructing real-valued optimization problems of arbitrary complexity for benchmarking purposes. Complexity is controlled via the various parameters of the Cones World such as number of cones; the ranges of their heights; range of the radii; and the number of dimensions. A sample 2-D Cones World landscape is presented in Figure 7-2.

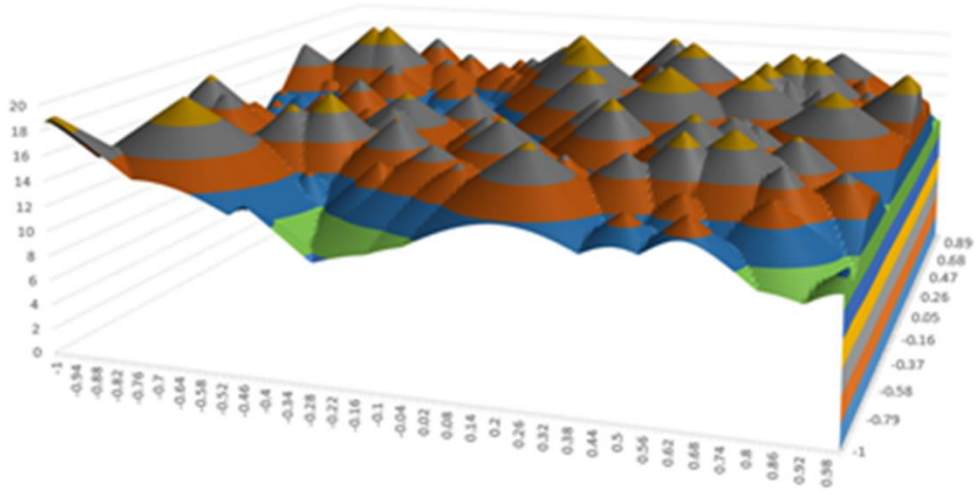


Figure 7-2: Cones World sample (source: CA papers)

The Cones World landscape consists of a set of superimposed cones. The height of the landscape is the value of the fitness function; it is given by **Eq 7-1**. The input to the fitness function is an n-dimensional point location in the problem hyperspace. Thus, the number input parameters is equal to the number of dimensions chosen for the Cones World testbed. All testing was done with 2D landscapes that are easier to visualize and therefore analyze than higher dimensional ones. The problem can always be made sufficiently complex by choosing appropriate values of other parameters such as the number of cones (i.e. the number of local maxima).

Cones World surface height at the given location:	$$f(\langle x_1,x_2,\ldots,x_n\rangle)=\max_{j=1,k}\left(H_j-R_j\cdot\sqrt{\sum_{i=1}^{n}(x_i-C_{j,i})^2}\right)$$	**Eq 7-1**
	where f returns the height of the landscape surface at the given coordinates $x_1, x_2, \ldots, x_n$; k is the number of cones; n is the dimensionality; H_j is the height of the cone j; R_j is the radius of cone j; and C_{ij} is the coordinate of cone j in dimension i.	

To create a sequence of landscapes needed for the testbed, the Cones World uses a sequence generator based on the logistic equation (Eq **7**-**2**) (Langton, 1990). The logistic equation is a recursive function.

Logistic equation:	$x_i = A * x_{i-1} * (1 - x_{i-1})$	**Eq** 7-2

Figure 7-3 shows the sequence of values generated from the logistic equation. To get the next value in the sequence, the previous value is fed back into the equation. The change between the two values is controlled by the A multiplier (also known as the 'r' multiplier). In Figure 7-3, the x-axis is the sequence number, y-axis is the value of the output at that position and each ribbon corresponds to a different 'r' or A values. Values of A between 1 and 3, produce almost flat ribbons – i.e. there is linear change between successive values. As r increases the change between successive values becomes larger and more unpredictable. The system switches to non-linear at A=3.1 and becomes chaotic at A=3.9 (not shown).

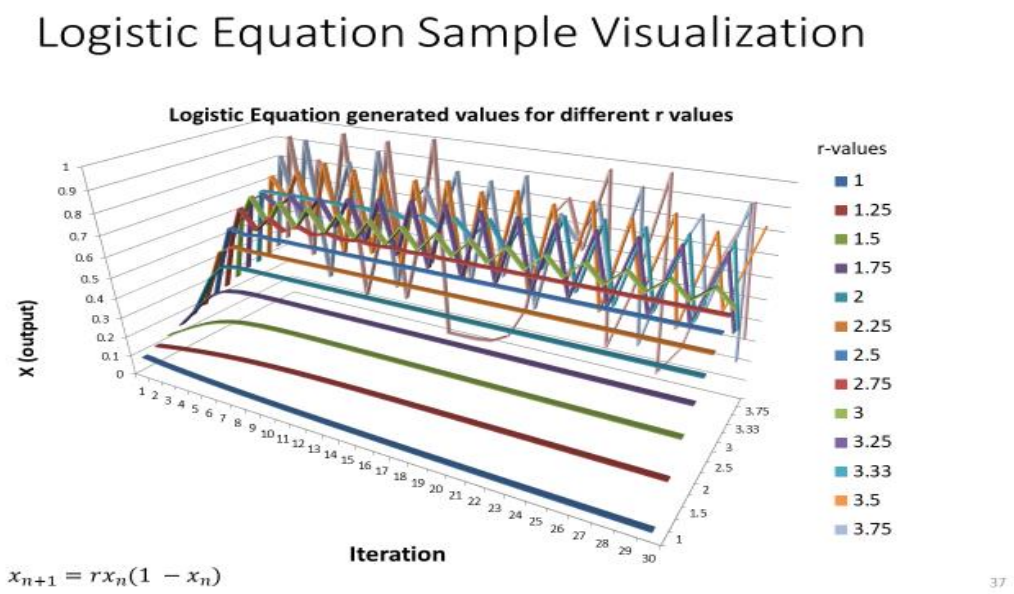


Figure 7-3: Logistic function behavior by A-Value (source: CA papers)

The values generated from the logistic sequence generator are used to modify the heights of the cones in the Cones World landscape to produce a new landscape for the testbed when required. For a particular run, the A value is kept fixed for landscape generation. To test with different levels of complexity – i.e. shocks to the system – the experimental runs are conducted with four different A values namely 1.0, 3.1, 3.6, and 3.9.

7.2 Generations-to-Solution the Basic Performance Metric

Using the dynamic landscape generation system described earlier, the basic metric to measure the response of the system to change or shock is the number of generations to solution. Say the system is in some state. Now change is introduced. The peak point (optimum) shifts to some unknown position. The CA system scrambles to locate the new peak. How long does it take? The number of generations to solution or G2S is the primary measure of performance used to compare the performance of the tested knowledge distribution mechanism vis-à-vis the dynamic

complexity factors. Cooperative mechanisms are expected to show better resilience than competitive mechanisms (Hy **7-2**).

Cooperative knowledge distribution exhibits better robustness than competitive distribution	**Hy** 7-2

In general, the shorter the average G2S value the more robust the system. However, it could be more instructive to plot the G2S values obtained from a sequence of landscape changes, to better understand performance over time.

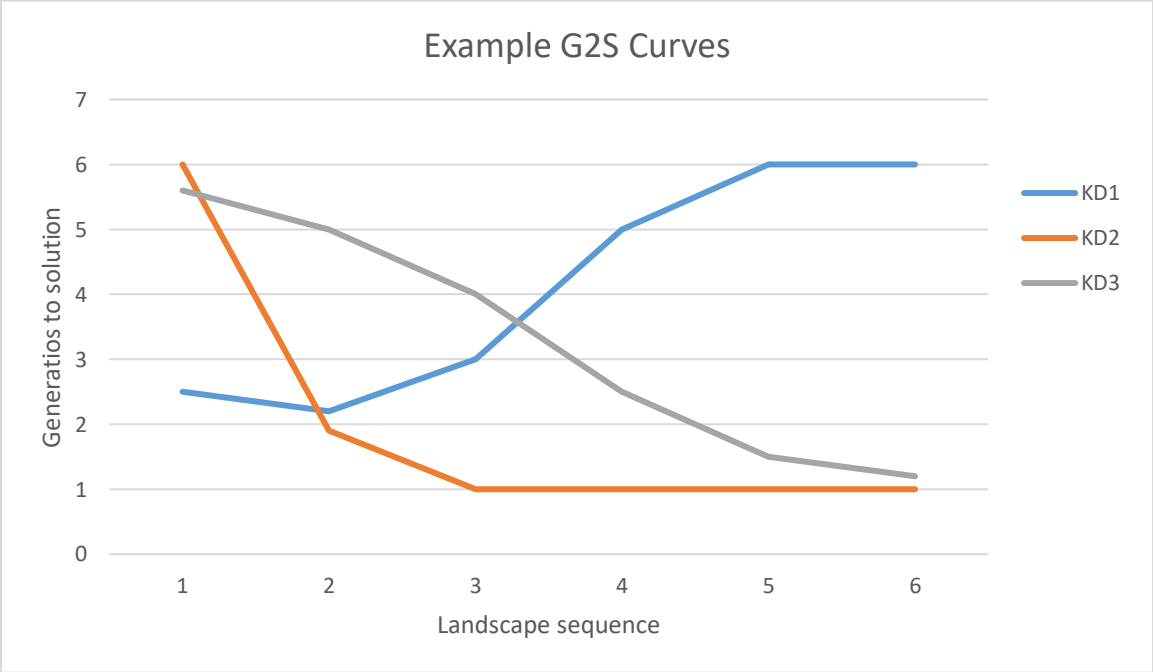


Figure 7-4: Generations-to-solution curve depicting possible responses to system change

Figure 7-4 shows the possible responses of three different hypothetical knowledge distribution mechanisms to illustrate the performance patterns that may emerge. The KD1 system starts out well but is unable to track the changes thrown at it and its performance (in terms of G2S) becomes

worse over the progression of landscape changes (i.e. it takes increasingly longer to find the solution). Such a mechanism may be well suited to problem solving in static environments. The KD2 mechanism seems to adapt quickly and tracks the changes well. By contrast KD3 is a slow learner; its performance gradually becomes better over the progression.

The way a mechanism responds to change over time is a useful characteristic as it can provide guidance for where best the mechanism may be applicable.

7.3 Social Stress or Diffusion

Social Stress or Diffusion, as the name implies, is a social metric. It is an attempt to measure the duress in the system. In all knowledge distribution mechanisms, the immediate neighbors have a bearing on each other and hence the distance between them in search space is of interest.

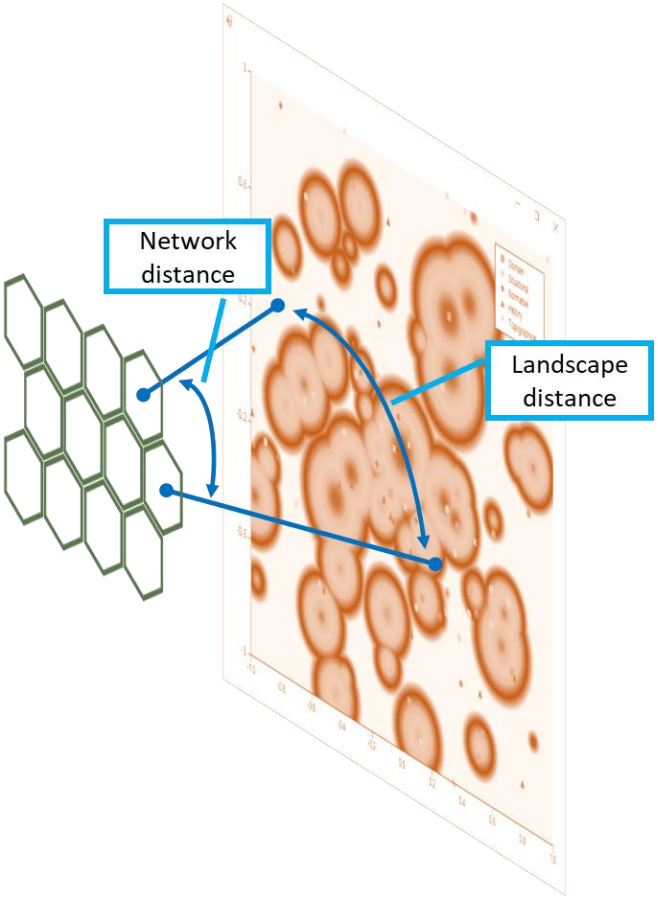


Figure 7-5: Social tension is reflected by how far apart are network neighbors in parameter space

Diffusion is measured as the Euclidean distance between two neighbors averaged over the population. Pictorially, distance is represented in Figure 7-5 and the distance equation is Eq **7-3**.

Euclidean distance in search space:

$$d(p1, p2) = \sqrt{1/n\left(\sum_i (p1_i - p2_i)^2\right)}$$

Eq 7-3

Where:
p1 and p2 are the parameters of two population individuals
i = 0,1, 2, … (n-1) indexes the parameter array, n is the number of dimensions

As with G2S (previous section) the Diffusion metric for each run is plotted over the progression of landscape sequences to obtain a temporal view of diffusion. We can expect diffusion to be

higher for more complex environments – i.e. landscape sequences generated with higher A values (Hy **7**-**3**). The maximum parameter distance is 2 [range of landscape location in each dimension is (-1, +1) and n=2] and therefore the maximum value is 2.0 for two neighbors by Eq **7**-**3**. However, the observed average Diffusion values for the population are in the 0.5-1.0 range.

Diffusion is higher for more dynamically complex environments	**Hy** 7-3

7.4 Segregation Index

In a seminal work, Thomas Schelling (Schelling, 1971) showed that the racial segregation in large cities (like Chicago) could be explained by slight biases in peoples preferences about what type of neighborhood, in terms of racial mix, they would prefer to reside in. Computer simulations conducted by Schelling showed that even slight biases in racial preferences lead to stark segregation at the city level.

Following Schelling's work, a test knowledge distribution was constructed to see if such segregation could be observed in the CA population space. The test knowledge distribution run results are shown Figure 7-6. Under this mechanism, each individual has a slight bias to be surrounded by individuals that have the same KS as the individual does. This results in a highly segregated population overall. Figure 7-6 is a view of the population arranged in a regular hexagon topology. Each individual thus has six neighbors and the topology is toroidal (i.e. it wraps around to form a sphere). Each circle is an individual where the color represents the Knowledge Source acquired via the Schelling-like rule. The key at the bottom of the figure provides a mapping from color to KS. This side experiment provides visual evidence that segregation as studied by Schelling for real-life social systems can also emerges in artificial social systems like the one CA has.

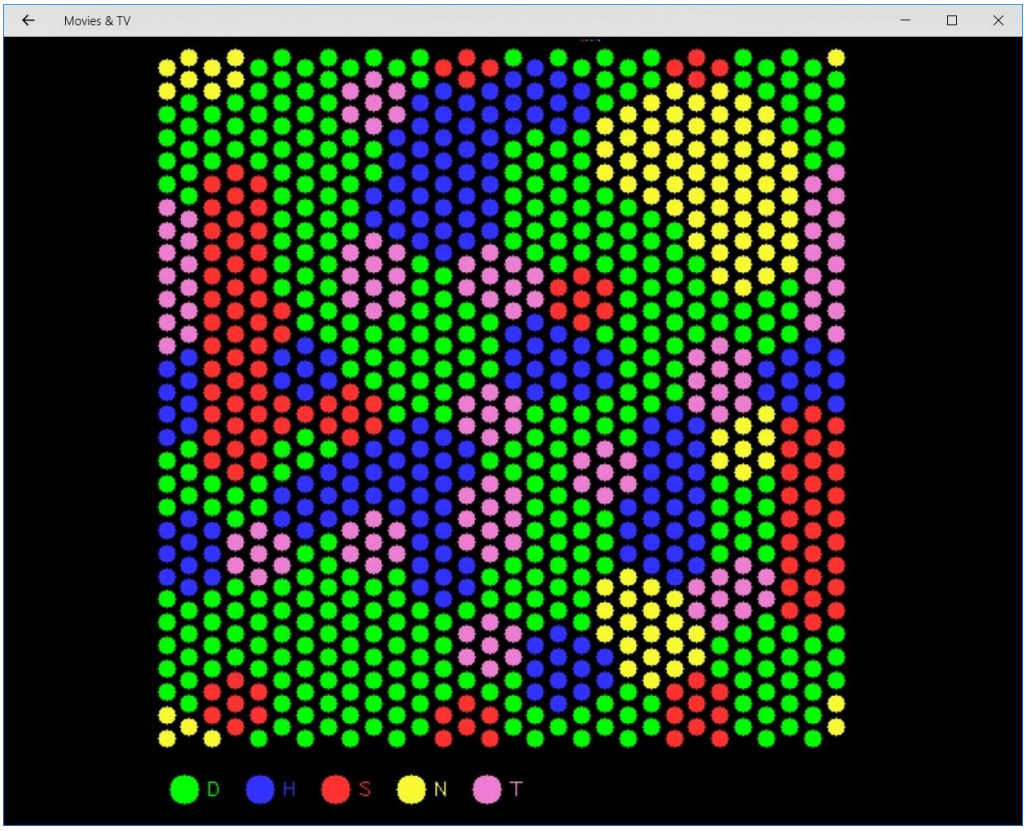


Figure 7-6: Stark segregation of Knowledge in population due to application of Schelling-like rule

The core idea behind Segregation Index is measuring the imbalance in proportions of subgroups in a local neighborhood with respect to proportions in the population at large. For example, say US population can be divided into two subgroups R and D where the overall proportion is 0.50/0.50. Divide US into smaller regions geographically. For each region can test how far the proportions of R and D are from the ideal 0.50/0.50. This will be the measure of segregation in that region. Table 7-1 shows the calculation of Schelling's Segregation Index for different regional proportions. R, D are population proportions and R_n, D_n represent regional proportions.

Table 7-1: Calculation of Regional Segregation with Different Subgroup Proportions

Neighborhood proportions R_n and D_n		Segregation	Comment
R_n	D_n	$\|R - R_n\| + \|D - D_n\|$	
0.50	0.50	|.5 – .5| + |.5 – .5| = 0	The neighborhood proportion is ideal
0.0	1.0	|.5 – 0.0 | + |.5 – 1.0| = 2.0	Max value is 2
.70	.30	|.5 - .7| + |.5 - .3| = 0.4	

The first example has the R_n/D_n at 0.5/0.5 same as for the overall population and segregation index works out to be zero. The second example represents extreme segregation where R_n/D_n is 0.0/1.0 and that produces a value of 2.0. Thus, the Segregation Index ranges from 0.0 to 2.0. The third example with a 0.7/0.3 split produces an intermediate value of 0.4. The calculation of Segregation Index for a CA population configuration is given in Eq 7-4.

CA Population segregation:

$$\textbf{Pop Seg} = \frac{1}{n}\sum_{r}\sum_{i}|p_{ri} - P_i| \qquad \textbf{Eq } 7\text{-}4$$

Where:
i indexes the Knowledge Source
r indexes the population individual
P_i is the proportion of the i^{th} Knowledge Source in the CA run (usually its 1/5 as there are 5 KS)
p_{ri} is the proportion of the i^{th} Knowledge Source in $r's$ neighborhood
n is the number of population individuals

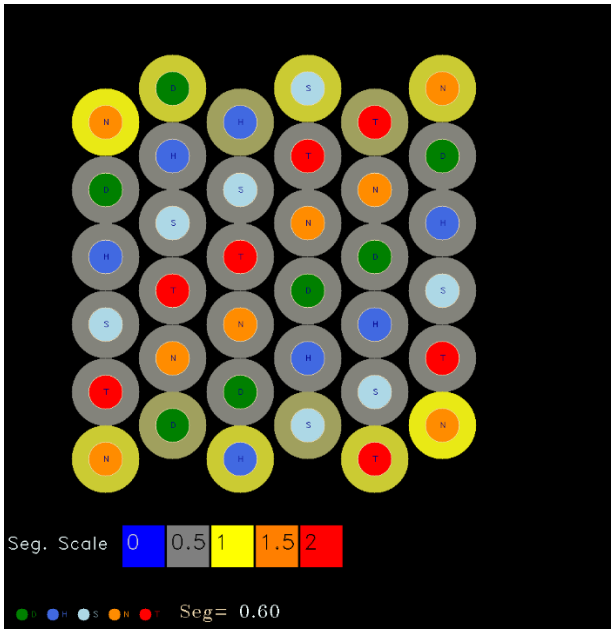


Figure 7-7: An example of low segregation network

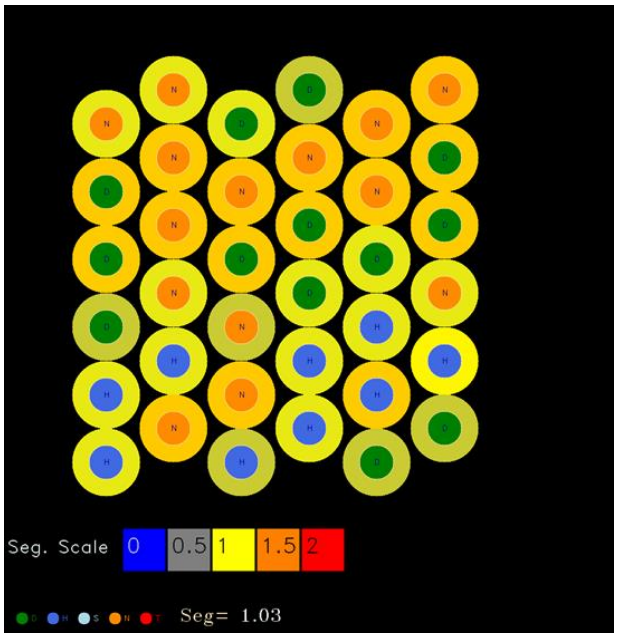


Figure 7-8: An example of a high segregated network

Also, Figure 7-7 (low segregation population) and Figure 7-8 (high segregation population) are examples of segregation index represented visually over a population of 36 individuals. The inner circle represents the population individual and its color the Knowledge Source. The outer ring corresponds to the segregation index for that individual's neighborhood. "Seg. Scale" is the key for decoding Segregation Index color. As expected, low segregation population individuals are surrounded by a greater variety of Knowledge Sources, in contrast with high segregation individuals.

It is difficult to judge a-priori the levels of segregation manifested by the tested knowledge distribution mechanisms with respect to the complexity factors. However, expect that the segregation level will be higher for higher complexity environments due to the greater level of stress placed on the individuals to solve a more complex problem (Hy **7-4**).

Segregation is higher for higher complexity environments	**Hy** 7-4

Also, in general it is expected that Stag-Hunt and IPD mechanisms will produce higher segregation than WTD (Hy **7**-**5**). This comes down to the mechanics of these mechanisms. The WTD mechanism requires compensation for lost Knowledge Sources that evens out the knowledge assignments somewhat. If this is not done, some Knowledge Sources entirely could be removed from the population, never to be regained. The WTD mechanism injects back Knowledge Sources that have been driven out after each distribution. Twenty percent of randomly selected individuals (from the total population) receive Knowledge Sources that were excluded in the natural assignment step. There are no such compensating mechanisms for IPD or Stag-Hunt – assignments are all due to the natural process followed. Both mechanisms allow Knowledge Sources to be regained even if they are driven out in some generation. As a result, the segregation will likely be higher as some Knowledge Sources may be absent in some generations.

Stag-Hunt and IPD distributions will produce higher segregation than WTD distributions	**Hy** 7-5

Following similar reasoning, it is expected Stackelberg should produce the lowest segregation of all (Hy **7**-**6**). Stackelberg – for CA knowledge distribution – is a somewhat structured approach where each Knowledge Source takes turns to acquire a piece of the evenly divided population pie and so, as a result, all Knowledge Sources are expected to be present in every generation. Note that even if all Knowledge Sources are represented in a population, segregation could still be high if they are clustered together into local groups.

Stackelberg produces lowest segregation among the mechanisms tested **Hy** 7-6

Both Diffusion and Segregation are metrics that pertain to entire populations – i.e. are aggregate metrics. While these are analyzed with respect to time in Chapter 8, information could be lost due to aggregation at each timestep. To understand the mechanism of knowledge distribution at a finer grained level, an approach is required that captures the dynamics of knowledge flow over time. The next section explains some methods derived from network analysis that should provide further insight into the inner workings of the knowledge distribution mechanisms.

7.5 Understanding the Dynamics of Communal Knowledge Flow

The CA population exists in the context of a social network (or fabric) of some topology (usually regular). The knowledge distribution mechanisms leverage the social connections for distributing knowledge. The flow of knowledge over the network, driven by the workings of the knowledge distribution mechanisms, is thus of high interest. Patterns of knowledge flow should highlight the differences between how the mechanisms operate.

The tools used for analyzing knowledge flow come from a variety of disciplines. First off is the Frequent Pattern Growth algorithm (Agrawal, Imieliński, & Swami, 1993). It is used to find communities of Knowledge Sources in a population. Its formulation and use are explained in section 7.5.1 . Then, a method related to visualizing weighted graphs is discussed in 7.5.2. Finally, the famous Page Rank (Wills, 2006) algorithm from Google is discussed in 7.5.3 ; and its use for CA knowledge distribution analysis clarified.

7.5.1 Frequent Pattern Growth - A mechanism for Community Detection in Social Networks

There are many community detection methods in graphical networks (Lancichinetti & Fortunato, 2009), however a simple and effective approach is to use market basket type analysis to detect local communities in the network. Kumar, et al from IBM (Kumar, Raghavan, Rajagopalan, & Tomkins, 1999) first described the use of such methods for mining communities in cyber space. For CA community detection, a method is required that can find clusters for Knowledge Sources. This is based on the ***type*** of the node rather than the ***link strength*** between nodes – the premise for most other community detection methods.

There are two primary algorithms for market basket analysis – apriori (Wu, et al., 2007) and frequent pattern growth (FPG). FPG is faster than apriori as it first constructs a tree and then mines it for frequent items.

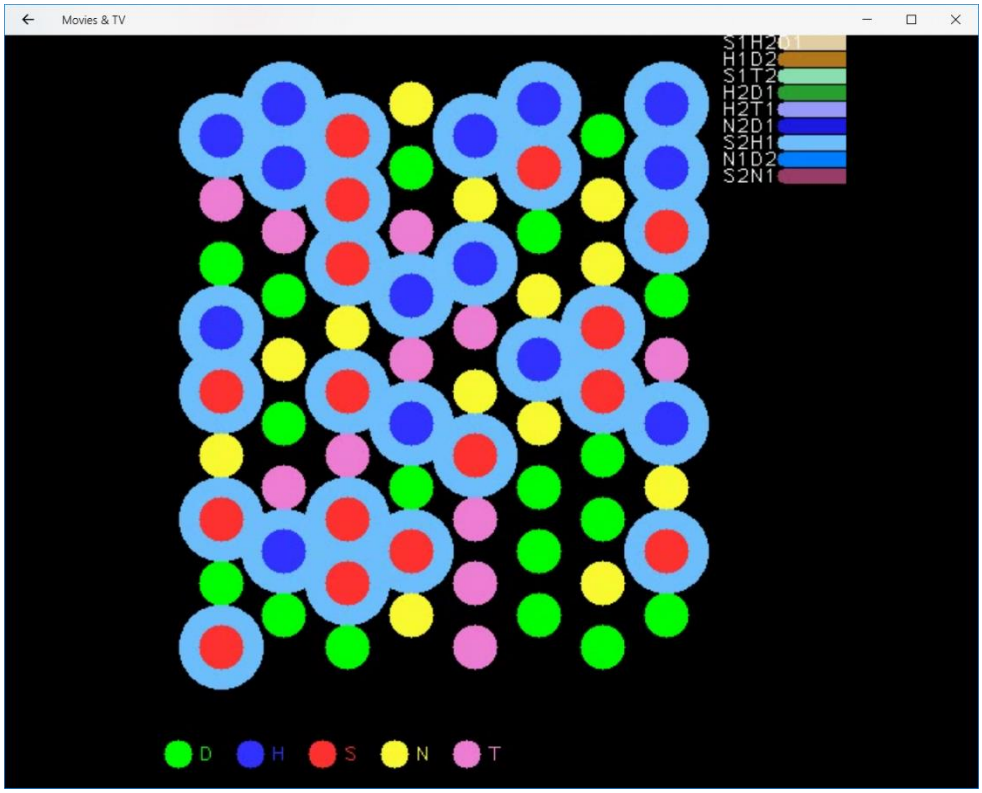


Figure 7-9: Combination of History (Dark Blue) and Situational (Red) Knowledge forms the dominant community type is this network as indicated by the colored outer ring

The idea of using FPG for community detection is to find what Knowledge Sources are present in a local neighborhood – i.e. around a population individual. For the population, count all the instances of the unique patterns that occur around each individual and take the top n patterns as the strongest communities.

FPG is an efficient way of counting such the patterns (called frequent itemsets). Figure 7-9 is an example of the application of this method for community detection in graphs. Here only the top community detected, comprised of History and Situational Knowledge Sources, is marked with a light blue outer ring. Note that this method does not preclude an individual to be part of several communities at the same time. For example, in Figure 7-9 another dominant pattern is pairing of Domain and Situational KS. Many individuals who are in History-Situational communities are also

in Domain-Situational. One example of an individual who is part of both communities is 3rd from the left in the top row.

Also, community size (number of unique Knowledge Sources in the community) may be between 1 to the number of Knowledge Sources in the system. For example, if all individuals have the same Knowledge Source then there is only one community comprising of the single Knowledge Source. The possible number of distinct communities is 32 for 5 Knowledge Sources.

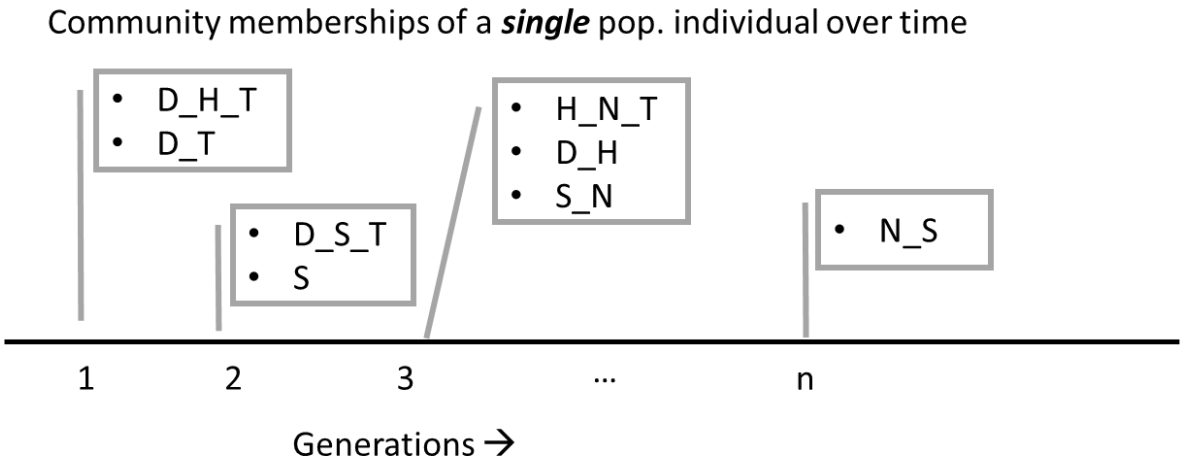


Figure 7-10: Timeline of community memberships of a single pop. Individual

The emergence of communities, especially if communities persist over time, is an insight into how a knowledge distribution mechanism is functioning for a given level of complexity. However, community analysis can be taken a step further by tracking change over time. It is very likely that an individual is part of several communities in one generation, several others in the generation after, and so forth. This pattern is pictorially represented in Figure 7-10. The letter pattern "D_H_T" represents a combination of Domain-History-Topographical community. Other letter

patterns can be similarly deciphered. Figure 7-10 is a view from the point of view of single individual over time. It shows the community-to-community transitions for that individual.

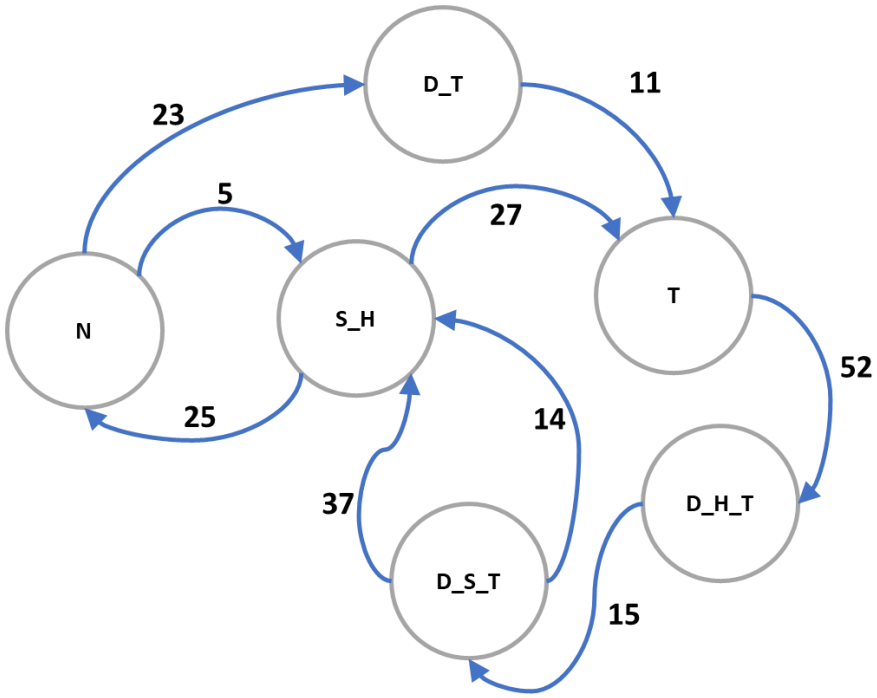


Figure 7-11 - Community membership transitions for a single individual over n generations can be folded into a weighted graph where the arc weights represent the number of corresponding transitions observed in the run

The sequence of community-to-community transitions for an individual can be folded into a single weighted graph (Figure 7-11). The nodes represent communities and arcs the transitions. The weight of the arcs is the number of times that transition occurred in the sequence of generations for a particular run. The graph is a compact view of the entire dynamic process of knowledge flow for a single individual. This construction is really a stepping stone towards the construction of a population-wide weighted graph and is not very useful by itself. All such graphs, each corresponding to a population individual, can be merged together to form a single view. This is described further in the next section.

7.5.2 Crystalizing Knowledge Flow Dynamics as a Weighted Graph

If one merges the community-to-community transition graphs across all individuals, one can obtain a weighted graph that is a compact, emergent view of a knowledge distribution mechanism's operation.

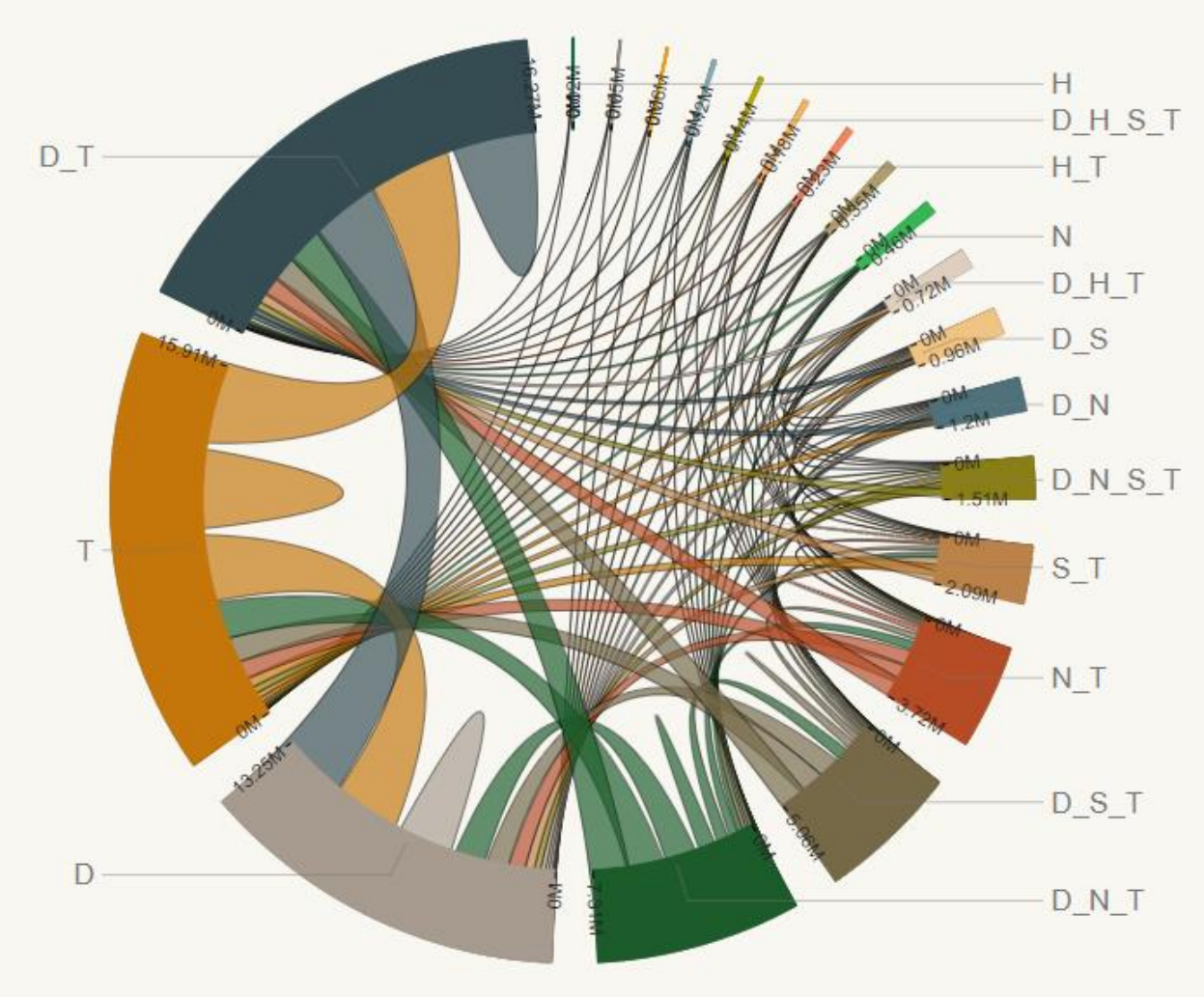


Figure 7-12: A weighted graph of community-to-community transitions aggregated across entire population presented as a 'chord' diagram. Such a diagram captures the dynamics of communal knowledge flow in a single view

Figure 7-12 is a representation of a weighted graph as a 'chord' diagram (Jalali, 2016). This particular view was generated from the Microsoft Power BI (Microsoft, 2019) tool using the Chord diagram extension from the chart gallery. The color scheme is selected by the tool itself and is

based on the ordering of the labels (communities in this case) arranged in order of importance in a counter-clockwise manner, starting from the top. If two similar charts have slightly different node orderings, the colors selected for the two charts will be very different. Thus, the node colors are way of differentiating charts quickly.

The community labeling scheme for the chord diagrams, and others explained later, is as follows:

- 'D' = Domain
- 'H' = History
- 'N' = Normative
- 'S' = Situational
- 'T' = Topographic
- Composite communities are labeled with the letter assignments joined by underscores ('_'), e.g. 'D_N_T' = combination of Domain, Normative and Topographic
- For composite communities, alphabet ordering is maintained so 'D_N_T' will always appear as such and not as 'T_D_N', for example.

In the chord diagram, the graph nodes are the segments around the circumference. The greater the importance of the node, the longer the length of the corresponding segment. The arcs are represented as the 'chords' between the nodes. The thickness of the arc represents its combined weight (i.e. both ways). The color of arc matches (is closer to the color of) the node with the higher

inflow. Self-loops are represented as stubby arcs. The arcs connected to a node (segment) are ordered in a counter-clockwise manner relative to the outgoing weight.

In Power BI the chord diagrams are interactive. Charts allow for interactive filtering that can be used to remove infrequent transitions. This feature can considerably reduce visual clutter. Also hovering the mouse over different parts of the chart provides more detail.

The weighted community transition graph, represented by the chord diagram, can be considered to be the ***signature*** of a distribution mechanism. As such one can expect the diagrams to appear to be quite different for each of the distribution mechanisms tested (Hy **7**-**7**). While it is hard to hypothesize about any particular feature of the graphs as these are emergent phenomenon, it can be argued that if the mechanisms were to operate more or less in the same way, their graph signatures would also be similar.

The community transition weighted graphs for the tested knowledge distribution mechanisms are visibly distinguishable from each other	**Hy** 7-7

This reasoning can be extended to the reaction of the distribution mechanism when it is subjected do environments of varying complexity – as controlled by the A value. For different levels of complexity one can expect the mechanism to respond differently with discernable manifestations in the corresponding weighted transition graphs (Hy **7**-**8**).

The community transition weighted graphs for a tested knowledge distribution mechanism are appreciably different for different A values	**Hy** 7-8

Weighed transition graphs, as visualized by chord diagrams, may be useful but still a weak differentiator of knowledge distribution mechanisms as one is asked to rely on somewhat subjective visual judgement. Using additional methods, the case for signature-based differentiation can be made stronger. This is discussed in the next section.

7.5.3 Page Rank for measuring Community Influence in Knowledge Flow Graphs

Community-to-community transition graphs, presented in the previous section, are seen as a means of providing a unique signature for each of the tested distribution mechanisms. However apart from visual differentiation, these views do not provide much useful information.

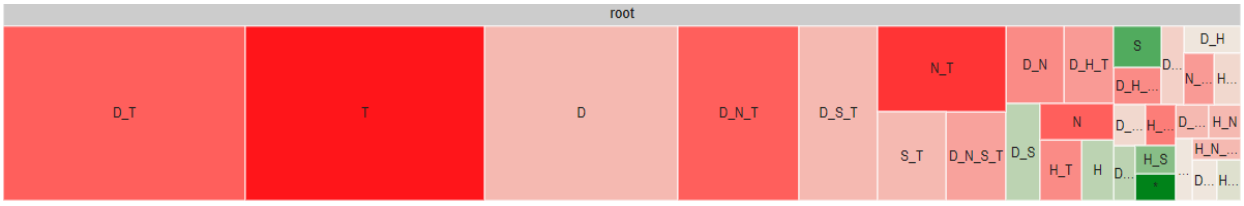


Figure 7-13: The importance of communities in a communal knowledge-flow graph can be extracted via Google's Page-Rank algorithm. Here, example results are presented in a 'tree' chart

A further refinement of the weighted graph approach helps to extract more useful information for comparative analysis. The arc weights in the prior weighted graphs are transition counts. These weights can be normalized so that they represent transition probabilities. This means that the weights for all outgoing arcs for any node sum to 1.0. Such a graph can be treated as a Markov chain and as such is amenable to analysis via the Page Rank (Wills, 2006) (Wu, et al., 2007) algorithm.

Page Rank is an iterative method of computing the stationary distribution of a Markov Chain. The significant outcome, however, is that the graph nodes are ranked in terms of importance. Google devised Page Rank to rank pages in search results. Since then, the algorithm has been

widely used in many areas where the problem involves graph analysis, e.g. biology (Gong, et al., 2014); natural language processing (Pershina, He, & Grishman, 2015); and sociology (Lu, Wang, Gao, & Liu, 2015).

Applying Page Rank to a normalized community transition graph nets the ranked list of communities. Such a list can be visualized as a 'tree' chart as shown in Figure 7-13. The communities are ordered by importance. Here importance means the proportion of time the individuals in a population are found to be in such a community. The area of the box representing a community in the tree diagram is proportional to the importance weight calculated by Page Rank. The color of the box represents the exploratory factor of the community. Red hues represent communities comprised of exploratory knowledge sources and Green hues represent exploitative ones.

The Page Rank derived tree chart provides a clearer view of the signature of a distribution mechanism than the chord diagram. First the communities are clearly ranked in order and it should be easy to spot the differences between diagrams of different mechanisms, if such differences exist. Second, the information contained in the tree charts for a given mechanism, for different A values can be stacked together into another useful view. Figure 7-14 is an example of such a view. It is a 'parallel chords' diagram. It shows the rank of each community with respect to each A value. Horizontal lines connect each community across the A value vertical lines. As such it allows one to easily spot changes in community rank ordering with respect to environmental complexity.

Apart from providing a visual signature for the distribution mechanisms, the tree diagram also allows one to assess the explorative-exploitative nature of the distribution mechanisms. Features

from such views can be extracted and related to the G2S performances of different distribution mechanisms, to provide further insight. This graph can be used to measure whether the community rankings for the tested distribution mechanisms are different from each other, reflecting their different internal mechanisms (Hy **7**-**9**).

Community importance weights for the tested distribution mechanisms are different from each other, reflecting their different operational characteristics	**Hy** 7-9

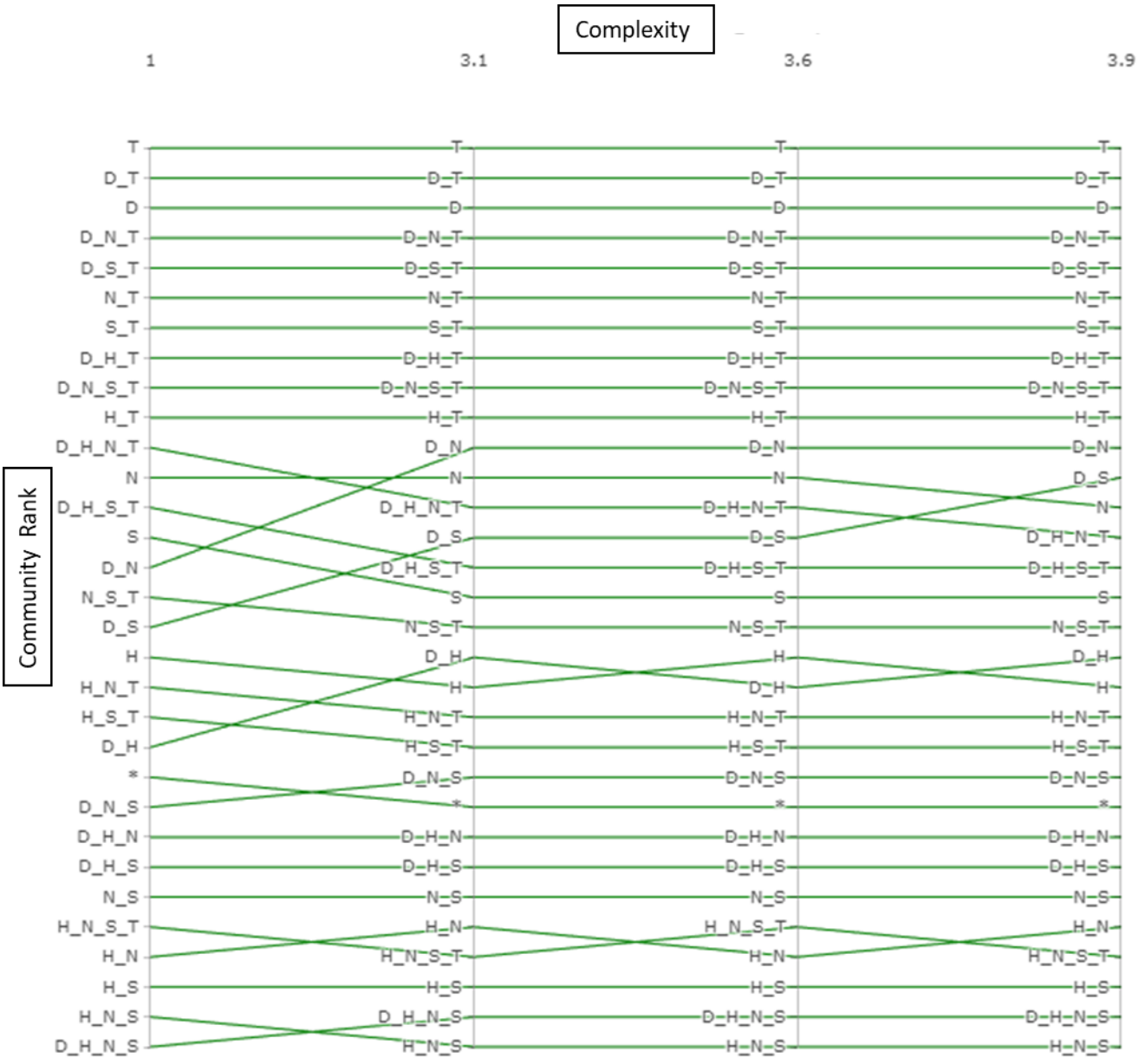


Figure 7-14: Changes in community rank with respect to change in A-value (complexity) - presented as a 'parallel coordinates' chart.

Further, the tested distribution mechanisms should respond differently to varying levels of environmental complexity. This should be reflected in the community rankings across the tested A values for each mechanism (Hy **7**-**10**).

Community importance rankings for the tested distribution mechanisms vary by environmental complexity **Hy** 7-10

Another view of responsiveness to varying complexity can be created based on changes in the explorative-exploitative balance of each mechanism with respect to changes in A values. Here the 32 possible communities are collapsed into 3 categories, Explorative, Neutral and Exploitative. The method used is as follows:

- Assign each Knowledge Source a numerical rank based on where it falls on the explorative-exploitative scale
- Average the ranks for Knowledge Sources within a community to obtain the community rank
- Bin each community into 1 of 3 categories based on the calculated rank and set thresholds for Explorative, Neutral and Exploitative categories.

Listing 7-1 : Community categorization

$T = 5, \quad N = 4, \quad D = 3, \quad H = 2, \quad S = 1, \quad * = 0$	Rank assigned to each Knowledge Source based on where it places on the explorative-exploitative continuum. '*' represents the no community found case (very rare)
`D_H_T = (3 + 2 + 5) / 3 = 3.33`	Example calculation of exploratory-exploitative ranking for a community comprimising of 3 Knowledge Sources
$Category(r) = \begin{cases} Explorative & r > 3 \\ Neutral & r = 3 \\ Exploitative & r < 3 \end{cases}$	Function to categorize a commnity as Explorative, Neutral or Exploitative based on its calculated rank

Listing 7-1 shows the supporting calculations for the method of ranking and categorization described above. With just 3 categories to work with, it is easier to perform meaningful statistical tests for the response of a distribution mechanism in the face of environmental complexity – a process that is explained next.

The underlying community-to-community transition counts are aggregated into category-to-category transitions (Figure 7-15). For a particular A value then there are only three nodes in the network namely, Explorative, Neutral, and Exploitative. An arc between say Explorative → Neutral represents the transitions from all explorative communities to Domain (which is the only neutral category); and so forth for the other arcs.

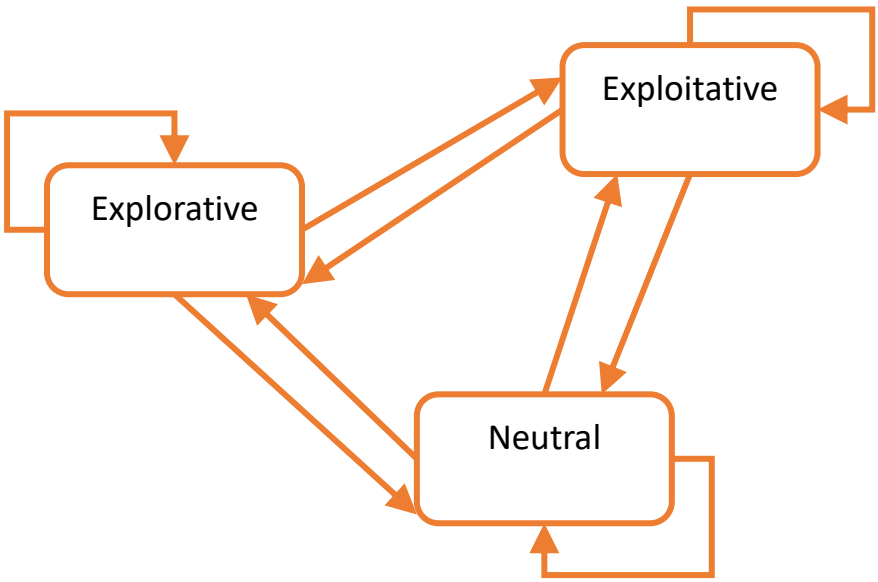


Figure 7-15: Category-to-category transition graph constructed by aggregating community-to-community transition counts

The counts underpinning an arc are collected from several sample runs. Hence, for each arc, the mean and standard deviation are available. Consider the arc Explorative → Neutral for some distribution mechanism when A = 1. Now consider the corresponding arc for the transition graph when A = 3.1 – the next level up for A. The means and standard deviations are available for both.

Are the means significantly different for the Explorative → Neutral arcs when A = 1 and when A = 3.1? This is an answerable question; the data are available.

Two-sample t-tests are performed to test for statistical differences between corresponding arcs, for successive A values – 1 → 3.1; 3.1 → 3.6; and 3.6→3.9. These are significant transitions in complexity. The 1 → 3.1 transition is from linear changes to non-linear changes; 3.1 → 3.6 is from non-linear to highly non-linear; and 3.6→3.9 is from highly non-linear to near chaotic changes. Such tests can answer the question whether or not the distribution mechanisms are responsive to varying levels complexity, with statistical rigor.

The information from a) statistical significance testing and b) changes in arc transition weights can be combined into a single, compact view with the help of a 'Sankey' diagram - see Figure 7-16. Admittedly, this chart is a little confusing at first glance, so detailed explanation is provided next.

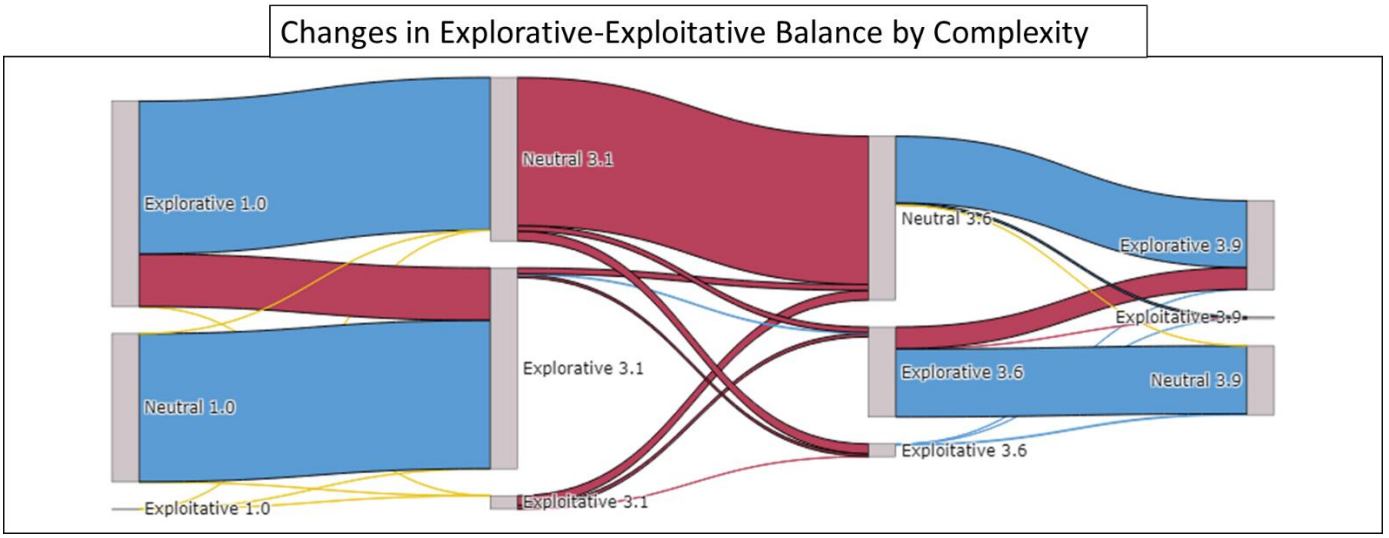


Figure 7-16: Changes in communal explorative-exploitative balance driven by complexity changes viewed as a 'Sankey' diagram

First a quick word about Sankey diagrams (Riehmann, Hanfler, & Froehlich, 2005). These are generally used to visualize flow but really are a way of visualizing multiple linked graphs simultaneously. Figure 7-16 has 3 distinct sections separated by vertical 'posts' (gray bars). The left most set of posts are for A=1 and from left-to-right the posts are for A=3.1, A=3.6 and finally A=3.9. There are 3 posts in each set; these are for the three types of nodes Explorative, Neutral and Exploitative. The posts are labeled, e.g. "Explorative 1.0", "Neutral 3.1", etc. The first part of the name is the node type and second part the A value.

The links between vertical posts represent the ***change*** in the transition rates between corresponding arcs for adjacent A values. For example, consider the arc from "Explorative 1.0" to "Neutral 3.1". It represents the change in the Explorative→Neutral arc weight between A=1.0 graph and A=3.1 graph. If the change is positive, the arc is Blue otherwise its Red. The width of the arc represents the amount of change. Finally, if the change is not statistically significant, the arc is drawn as a thin line.

The Sankey chart as conceived for this analysis is information rich. It is a compact way of capturing a distribution mechanism's response to environmental complexity. The Sankey chart, while informative for a single distribution mechanism, is not suitable for comparing multiple mechanisms together. One issue with the chart is that the scale is relative to the chart so different charts are not comparable. However, a Sankey chart is still useful for judging whether or not a mechanism is responsive to changes in environmental complexity (Hy **7**-**11** & Hy **7**-**12** below). For example, if most of the arcs are thin lines (i.e. the changes are not statistically significant) the conclusion can be drawn that the associated mechanism is not responsive to changes in environmental complexity.

The tested distribution mechanisms are responsive to changes in environmental complexity	**Hy** 7-11

Cooperative distribution mechanisms are more responsive to changes in environmental complexity than competitive mechanism	**Hy** 7-12

The Sankey chart depicts the significant changes in exploration/exploitation balance via arc thickness however it is difficult to determine the exact magnitude of the change – especially net effects. And because of relative scale, it is difficult compare the different mechanisms together. Nevertheless, the information contained by the Sankey chart is transformable into a shape that makes the goal comparing distribution mechanisms achievable.

From the information developed for the Sankey chart, the net changes in the explorative-exploitative balance can be tracked. Consider for example the Explorative category and the change 1.0 → 3.1 in A. The net change to Explorative is calculated by summing the significant flows in and out of Explorative category as shown in Listing 7-2. Generalizing, the net flows for all categories, for all adjacent A value can thus be calculated.

Listing 7-2: Example - net changes to Explorative A: 1.0 → 3.1

$$E = \{Explorative, Neutral, Exploitative\}$$

$$Net\ change\ to\ Explorative_{a1 \to a2} = \triangle\ Explorative\ Inflows_{a1 \to a2} - \triangle\ Explorative\ Outflows_{a1 \to a2}$$

$$= \sum_{e \in E} e_{a1} \to Explorative_{a2} - \sum_{e \in E} Explorative_{a1} \to e_{a2}$$

$$a1 = 1.0, \qquad a2 = 3.1, \qquad (\to) = change\ in\ flow$$

Net changes are simpler quantities, easier to visualize and compare. A sample visualization is provided in Figure 7-17.

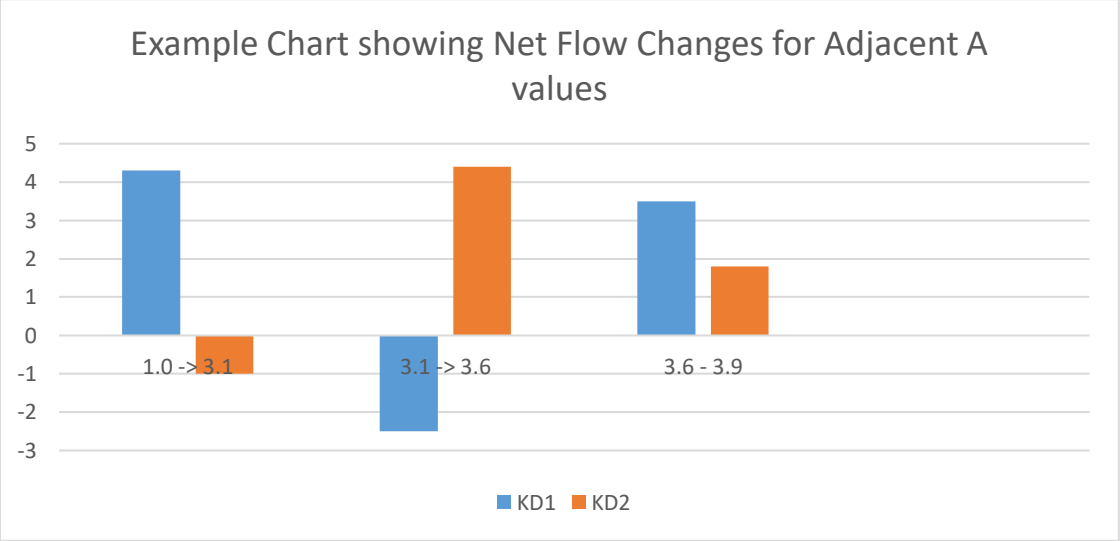


Figure 7-17: An example of Net flow changes by adjacent A values for a hypothetical distribution mechanisms

The tracking of net flows (Figure 7-17) is one way to relate the performance of distribution mechanisms to how these mechanisms respond to changes in environmental complexity. For example, a mechanism that responds by consistently increasing resources to explorative communities as A is increased can expected to perform better (Hy **7-13**).

Better performing distribution mechanisms will exhibit consistent responses to changes in environmental complexity	**Hy** 7-13

Net flow thus is one of the analytical tools, along with other tools discussed in this chapter, to peer into the workings of the distribution mechanisms and draw insights that might be useful for improving the mechanism in future. Thus far this chapter has focused on analytical methods to

drive the design of experiments at the conceptual level. The detail setup of experiments, the configurations used and the data collected is described next.

7.6 Experimental Setup

The primary goal of the thesis is to understand the performance of knowledge distribution mechanisms that encompass cooperation with those that are purely competitive. The inspiration for cooperative mechanisms comes from classical and evolutionary Game theory. The purely competitive mechanism selected is Weighted Majority, which is the default for Cultural Algorithms.

Table 7-2 provides the detail settings of the experimental parameters employed to test the hypotheses given above.

Table 7-2: Parameter Settings for Experimental Runs

Parameter	**Value**	**Comment**
Knowledge Distribution Mechanisms	Stag-Hunt (SHS) Iterated Prisoner Dilemma (IPD) Stackelberg (STK) Weighted Majority (WTD)	In the experimental analysis section, the KD mechanisms are referenced via the abbreviations used here
A – values	1.0, 3.1, 3.6, 3.9	
Population size	36	

Network topology	Hexagonal	
Cones World number of cones per landscape	1000	All KD mechanisms were tested on the exact same landscapes generated in the sequence, to make the performance differences more meaningful
Number of generations per landscape	2500	The system was allowed to run for a fix number of generations per landscape.
Number of landscapes in sequence per run	50	Number of generations per run = 50*2500 = 125000
Number of runs (sample size)	200 per KD-A combination	
Threshold distance for solution	0.001	

Cone parameters modified for landscape sequence	Height	Modifying height is equivalent to relocation cones as the peak can move

Detailed data was collected for each generation into a log file. The format of the log file is in Table 7-3. Over 1 Terabyte of log data was collected and analyzed.

Table 7-3: CATGame Log File Format

Column	Description
Sample	Sample number, 1, 2, …, 200
KD	Distribution mechanism WTD, SHS, IPD, STK
EnvSnsty	Not used
LandscapeNum	The sequence number of the landscape in the sequence, 1, 2, …, 50
A	1.0, 3.1, 3.6, 3.9
GenCount	Population generation counter resets after landscape change, 1,2, …, 2500
Best	Height of the best cone found thus far
Max	Ground truth best for the landscape
Seg	Average segregation index of the population at the end of the current generation

Dffsn	Average Diffusion for the population at the end of the current generation
Net	Network topology (Hexagon)
IndvSeg	Segregation index of each population individual, delimited by '|'
IndvDffsn	Diffusion of each population individual, delimited by '|'
IndvKS	Knowledge source for each population individual, delimited by '|'

All experimental runs were performed on the Wayne State's grid computing environment. Details of the configuration used are provided Table 7-4.

Table 7-4: Wayne State Grid Computing Environment Particulars

Category	**Value**
Environment	Wayne State grid computing environment
Number of jobs (that can run in parallel)	1600 = num KD * num A * num samples = 4 * 4 * 200
Hardware requirements for grid resource	4 cores with 500MB RAM
Size of log data collected	1.04 Terabyte

Experiments were conducted in line with the experimental framework presented in this section and the required data recorded in log files. The analysis of the log file results is presented in the next chapter.

7.7 Chapter Summary

This chapter described the experimental framework used to test and compare the three game distribution mechanisms vs. the stalwart Weighted Majority, for the CATGame system. The core idea is to observe the performance and the 'social' behavior of the CATGame system when configured with each of the four mechanisms. The testbed is a dynamic environment generator created by hybridizing the Cones World system with the logistic equation (Eq **7**-**2**). The logistic equation is used as a sequence generator to modify the height of the cones periodically to generate new landscapes, while the performance optimization is still underway. The main performance metric is the average number of generations to reach solution for each change in the landscape.

CA has a social aspect due to the networked population. The social network is leveraged by the KD mechanisms. The 'social' response of the system to varying levels of dynamic complexity can be studied with the help of social metrics. Diffusion and Segregation capture static aspects of the network. To understand dynamic aspects, Markovian methods that track patterns of communal knowledge flow over time, are also described. The next chapter analyzes the data collected under the experimental framework described in this chapter.

CHAPTER 8 CATGAME EXPERIMENTAL RESULTS FOR CONES WORLD BENCHMARK

8.0 Introduction

Following the experiment designs detailed in Chapter 7, experiments were conducted on the Wayne State's grid computing environment. A total of 1600 jobs were run and over a terabyte of log data collected for analysis. Log data analysis results are presented in the sections of this chapter. In the charts presented the tested KD mechanisms are referred to by short names. The mapping is a follows: WTD → Weighted Majority; IPD → Iterated Prisoner's Dilemma; SHS → Stag-Hunt; and STD → Stackelberg.

Section 8.1 contains the results of basic performance analysis – mean generations-to-solution (G2S) – for each distribution mechanism, by A value and by landscape sequence number. Note: the sample size is ***200*** – i.e. each combination of A value and distribution mechanism is run 200 times to obtain statistically significant results. For each A value, first a chart is presented that compares the mean G2S over the 50-landscape sequence. And then a table that gives detailed values by landscape and also provides the two-sample t-test results for the hypotheses that IPD, SHS and STK mean G2S are less that WTD mean G2S, respectively. Section 8.2 presents analysis for the Diffusion statistic for each A value-KD combination. The diffusion statistics are also obtained from log results. Section 8.3 looks at the Segregation patterns obtained from the log data for A value-KD combinations. Representative samples of high and low Segregation population snapshots are also provided for qualitative assessment. Charts showing overall trends of Segregation by landscape are presented. The aggregate view of Segregation for A value-KD combinations is presented as the final chart of this section. Section 8.4 focuses on the dynamic view of knowledge flow resulting from the distribution strategy used by each of the KD

mechanisms and in response to changes in environmental complexity. Here an analysis of the community-to-community transition graphs is presented. Several types of visualizations are discussed:

- Chord diagram views of weighed graphs
- Tree diagrams for Page Rank results
- Parallel-chords diagrams for tracing rank changes with respect to A values
- Sankey charts for statistically significant changes in knowledge flow in response to environmental complexity
- Net flow changes in explorative-exploitative balance by A values

Finally, section 8.5 presents the summary of the analytical results and draws conclusions about the hypotheses posed in Chapter 7.

8.1 Performance Analysis

The base performance results are presented for each of the A-value-KD combinations in this section. The results are organized by A value so the performance of the distribution mechanisms can be directly compared for a given level of environmental complexity.

Trend charts for each A value are presented in Figure 8-1, Figure 8-2, Figure 8-3 and Figure 8-4. All charts are drawn to same scale for easier comparative analysis. Detailed numerical data for each A value is provided in Table 8-1, Table 8-2, Table 8-3 and Table 8-4. The tables include two-sample T-Test results performed for each landscape in the sequence (1 … 50). As mentioned before, the sample size for each T-Test is 200. The hypotheses are that the mean G2S for each of

the game KDs is less than that of WTD – i.e. one-tail. In other words, the performance of game KD is expected to be better than WTD for every landscape in the sequence. The T-Test columns in the table show the outcomes based on p<=0.05 (95% confidence) for the tests. If T-Test p <= 0.05 the column contains +1 otherwise it contains -1. These column values are converted to 'check mark' and 'cross' icons (using Excel's conditional formatting option) to visually highlight the results.

Analysis by landscape sequence is more interesting and informative as the performance of the mechanisms varies over the progression of the sequences. For A = 1, the environment's dynamic complexity is very low. The change is gradual. The trend lines in Figure 8-1 show that WTD settles down to competitive performance after about 10 landscapes. WTD's G2S value equals that for IPD after 10 landscapes. STK on the other starts out well but then steadily its performance worsens – i.e. STK is not tracking environmental changes well. SHS (Stag-Hunt) starts well and then performs best all the way through.

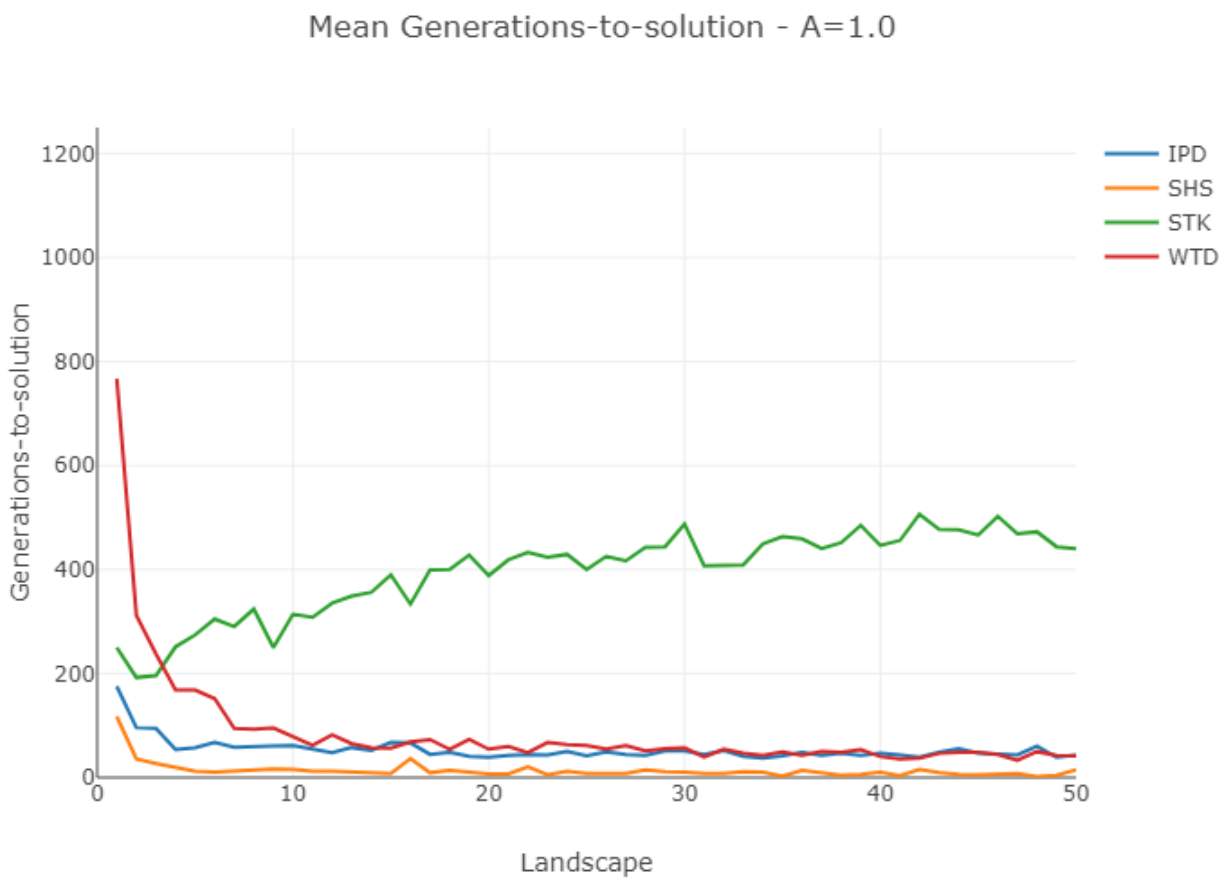


Figure 8-1: Mean generations to solution A=1

The data underlying the chart in Figure 8-1 is listed in Table 8-1. As can be expected from inspecting the chart, IPD performs significantly better in early landscapes but then is on par with WTD. This is borne out by the T-Test results for IPD vs WTD. STK is statistically significantly better than WTD in the first 2 generations but then the one-tail T-Test is not significant. And in fact, the performance is much worse than WTD as depicted in the corresponding chart. SHS is statistically better than WTD for all landscapes except for landscape #16. The corresponding chart also shows a minor up tick at #16 for SHS.

Table 8-1: A=1, Two Sample T-Tests (P<0.05), Mean G2S: {IPD,SHS,STK} < WTD, by Landscape Sequence #

Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD < WTD	SHS Mean	SHS Stdv.	TTest SHS < WTD	STK Mean	STK Stdv.	TTest STK < WTD
1	766.915	809.699	175.380	353.271	✓ 1	117.235	269.411	✓ 1	250.085	394.635	✓ 1
2	311.985	587.532	96.130	277.886	✓ 1	35.590	218.089	✓ 1	192.425	494.579	✓ 1
3	237.930	508.694	94.685	234.787	✓ 1	27.250	196.698	✓ 1	195.955	478.257	✗ -1
4	168.480	394.045	53.795	97.606	✓ 1	20.500	103.186	✓ 1	251.495	560.880	✗ -1
5	168.320	454.257	57.310	83.744	✓ 1	11.860	60.101	✓ 1	274.315	638.700	✗ -1
6	151.880	405.286	67.180	158.407	✓ 1	10.700	58.607	✓ 1	304.785	681.366	✗ -1
7	94.400	219.313	57.985	159.700	✓ 1	12.605	52.419	✓ 1	290.540	665.886	✗ -1
8	92.975	242.981	60.195	120.779	✓ 1	14.315	53.889	✓ 1	324.040	679.655	✗ -1
9	95.585	261.194	61.195	186.537	✗ -1	16.290	76.520	✓ 1	250.010	607.030	✗ -1
10	77.645	222.615	61.325	198.225	✗ -1	15.555	78.984	✓ 1	314.220	703.343	✗ -1
11	62.150	109.071	53.545	101.937	✗ -1	12.000	56.169	✓ 1	308.000	689.352	✗ -1
12	81.850	239.262	47.620	77.715	✓ 1	11.815	91.045	✓ 1	335.430	726.295	✗ -1
13	65.135	151.502	57.685	132.556	✗ -1	10.275	36.392	✓ 1	348.850	755.467	✗ -1
14	57.510	99.048	52.365	105.558	✗ -1	7.075	50.772	✓ 1	356.295	739.290	✗ -1
15	56.565	94.609	67.760	164.902	✗ -1	7.810	44.638	✓ 1	389.800	788.860	✗ -1
16	68.255	195.146	66.735	194.164	✗ -1	36.850	209.586	✗ -1	333.425	723.865	✗ -1
17	73.015	214.724	44.735	67.405	✓ 1	9.230	50.401	✓ 1	399.040	794.970	✗ -1
18	54.155	95.076	48.665	100.362	✗ -1	14.120	58.049	✓ 1	400.135	799.080	✗ -1
19	73.515	206.798	40.900	54.909	✓ 1	10.650	45.635	✓ 1	427.960	821.356	✗ -1
20	54.595	92.785	39.035	54.829	✓ 1	7.125	29.240	✓ 1	388.375	778.656	✗ -1
21	59.445	122.924	42.720	80.201	✗ -1	6.770	25.162	✓ 1	418.800	829.890	✗ -1
22	48.065	87.186	44.635	85.559	✗ -1	20.415	111.899	✓ 1	432.670	827.644	✗ -1
23	67.695	210.112	43.730	76.334	✗ -1	5.050	22.180	✓ 1	423.580	819.541	✗ -1
24	63.110	153.568	50.590	113.320	✗ -1	12.170	76.801	✓ 1	429.130	822.740	✗ -1
25	61.825	181.462	42.130	58.530	✗ -1	7.590	38.211	✓ 1	400.265	788.196	✗ -1
26	54.865	131.300	49.635	118.743	✗ -1	8.930	49.148	✓ 1	425.455	835.117	✗ -1
27	61.175	193.732	44.640	64.723	✗ -1	7.830	46.940	✓ 1	416.540	833.842	✗ -1
28	51.260	96.820	42.400	57.954	✗ -1	14.530	93.808	✓ 1	442.715	836.251	✗ -1
29	55.780	184.012	51.110	107.175	✗ -1	10.895	80.185	✓ 1	443.605	853.336	✗ -1
30	56.850	188.138	51.005	86.323	✗ -1	10.165	44.853	✓ 1	487.470	887.364	✗ -1
31	39.645	56.592	44.675	66.962	✗ -1	7.535	37.571	✓ 1	407.220	793.622	✗ -1
32	54.805	187.584	52.100	107.935	✗ -1	8.020	56.719	✓ 1	409.035	809.620	✗ -1
33	46.670	81.155	40.930	73.247	✗ -1	10.935	56.621	✓ 1	409.100	827.707	✗ -1
34	42.555	63.685	37.035	47.971	✗ -1	10.110	57.554	✓ 1	449.255	831.573	✗ -1
35	49.650	142.008	41.435	76.443	✗ -1	2.415	10.846	✓ 1	463.365	848.689	✗ -1
36	42.175	94.815	48.745	85.739	✗ -1	14.065	107.412	✓ 1	458.950	854.459	✗ -1
37	50.325	180.067	42.640	74.990	✗ -1	8.585	51.533	✓ 1	440.250	828.141	✗ -1
38	48.830	97.362	46.545	85.299	✗ -1	4.250	15.639	✓ 1	451.650	851.897	✗ -1
39	54.005	138.554	42.680	62.202	✗ -1	5.700	30.381	✓ 1	485.135	883.463	✗ -1
40	40.335	66.468	46.980	75.602	✗ -1	10.020	50.878	✓ 1	446.300	856.528	✗ -1
41	35.815	53.205	43.045	82.252	✗ -1	3.790	13.968	✓ 1	455.805	858.635	✗ -1
42	37.520	63.592	39.100	80.622	✗ -1	15.275	123.612	✓ 1	506.525	907.526	✗ -1
43	46.445	114.691	48.595	125.638	✗ -1	9.695	48.977	✓ 1	477.570	877.761	✗ -1
44	48.500	145.577	55.315	118.305	✗ -1	5.910	22.290	✓ 1	476.375	886.260	✗ -1
45	48.725	115.337	46.805	78.842	✗ -1	5.085	22.909	✓ 1	466.600	877.711	✗ -1
46	44.400	70.695	43.885	93.234	✗ -1	6.795	31.698	✓ 1	502.450	906.193	✗ -1
47	33.210	51.469	43.565	82.566	✗ -1	7.980	36.514	✓ 1	468.965	883.215	✗ -1
48	49.800	127.121	60.550	189.036	✗ -1	2.065	8.790	✓ 1	472.620	874.147	✗ -1
49	42.195	63.700	38.755	68.907	✗ -1	4.550	23.560	✓ 1	443.865	819.473	✗ -1
50	41.580	59.575	43.120	75.250	✗ -1	14.615	122.639	✓ 1	440.055	853.218	✗ -1

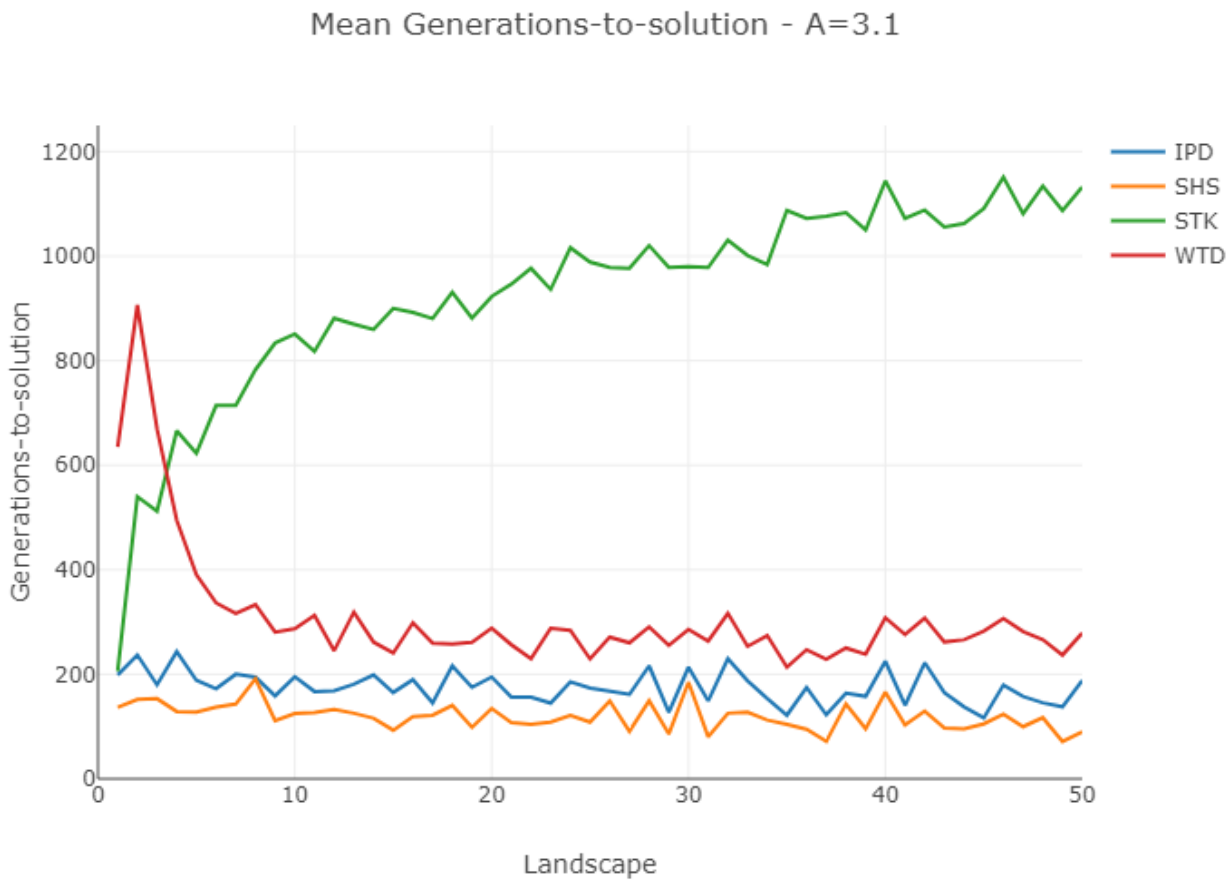


Figure 8-2: Mean generations to solution A=3.1

For A=3.1 the corresponding chart and table are in Figure 8-2 and Table 8-2. At A=3.1 a simple cycle is introduced. Here again WTD settles down to stable performance but it takes longer – landscape 20 (in contrast to 10 for the previous A value). Also, the settled G2S value is higher than for A=1.

At A = 3.1, IPD separates itself from WTD. As shown in Table 8-2, the T-Test results for IPD are all positive except for a single landscape - #18. Although IPD is biased towards defection, cooperation can emerge over repeated interactions. The fruits of limited cooperation become apparent when environmental change is moderately complex.

Table 8-2: A=3.1, Two Sample T-Tests (P<0.05), Mean G2S: {IPD,SHS,STK} < WTD, by Landscape Sequence #

Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD < WTD		SHS Mean	SHS Stdv.	TTest SHS < WTD		STK Mean	STK Stdv.	TTest STK < WTD	
1	634.640	753.930	198.815	358.242	✓	1	136.905	315.152	✓	1	206.245	314.660	✓	1
2	906.265	919.400	237.005	283.683	✓	1	152.055	347.956	✓	1	539.820	783.832	✓	1
3	669.555	865.619	179.920	343.975	✓	1	153.360	345.446	✓	1	512.210	804.292	✓	1
4	494.730	689.906	243.295	430.012	✓	1	128.800	243.500	✓	1	665.680	926.064	✗	-1
5	390.150	585.016	188.510	331.550	✓	1	127.955	283.396	✓	1	622.845	935.369	✗	-1
6	336.685	511.467	172.520	307.199	✓	1	137.245	312.991	✓	1	714.145	974.085	✗	-1
7	316.385	494.257	200.165	381.172	✓	1	143.165	328.881	✓	1	714.660	976.989	✗	-1
8	333.235	473.662	194.675	359.921	✓	1	191.235	410.370	✓	1	782.630	1009.315	✗	-1
9	280.735	417.607	158.660	230.399	✓	1	111.010	195.150	✓	1	833.985	1045.860	✗	-1
10	287.480	436.747	195.440	348.006	✓	1	124.605	278.760	✓	1	851.115	1041.465	✗	-1
11	312.550	382.584	167.085	250.802	✓	1	126.385	253.439	✓	1	817.960	1029.492	✗	-1
12	245.170	328.444	168.605	309.381	✓	1	132.865	298.523	✓	1	881.270	1061.412	✗	-1
13	318.970	438.738	180.805	317.272	✓	1	125.415	312.027	✓	1	870.005	1064.697	✗	-1
14	261.585	394.754	198.670	338.686	✓	1	115.880	273.901	✓	1	859.420	1045.630	✗	-1
15	240.635	338.770	165.185	261.520	✓	1	93.055	200.881	✓	1	900.260	1076.293	✗	-1
16	298.365	408.174	189.860	333.687	✓	1	118.965	315.876	✓	1	892.520	1061.965	✗	-1
17	259.615	355.368	145.315	290.919	✓	1	121.610	300.744	✓	1	880.500	1070.456	✗	-1
18	257.780	313.938	216.045	395.194	✗	-1	140.485	341.639	✓	1	930.925	1058.405	✗	-1
19	261.330	267.105	174.920	293.128	✓	1	98.520	216.401	✓	1	881.575	1089.975	✗	-1
20	288.005	358.051	194.745	380.384	✓	1	134.355	295.908	✓	1	922.680	1089.942	✗	-1
21	256.540	328.159	156.505	310.191	✓	1	107.205	303.285	✓	1	946.325	1095.166	✗	-1
22	230.010	294.379	156.485	263.583	✓	1	103.915	232.261	✓	1	976.625	1115.042	✗	-1
23	288.375	372.178	145.010	199.659	✓	1	108.710	262.208	✓	1	936.410	1117.774	✗	-1
24	284.210	379.634	185.455	286.977	✓	1	121.085	251.163	✓	1	1016.125	1109.854	✗	-1
25	229.200	307.376	173.825	300.785	✓	1	108.525	218.248	✓	1	988.855	1090.862	✗	-1
26	270.995	333.403	167.200	265.443	✓	1	149.105	375.619	✓	1	978.450	1109.479	✗	-1
27	260.125	337.739	161.385	344.419	✓	1	90.695	244.261	✓	1	976.850	1134.055	✗	-1
28	290.840	378.233	216.130	316.582	✓	1	149.745	386.325	✓	1	1020.145	1113.572	✗	-1
29	255.415	299.437	126.885	267.769	✓	1	85.575	178.614	✓	1	978.540	1112.065	✗	-1
30	285.845	363.202	213.520	373.054	✓	1	184.945	423.834	✓	1	980.095	1110.086	✗	-1
31	263.360	341.957	149.130	299.261	✓	1	80.195	195.042	✓	1	978.195	1120.450	✗	-1
32	316.450	404.714	229.550	419.036	✓	1	125.450	267.915	✓	1	1030.410	1127.200	✗	-1
33	253.415	339.655	187.690	389.285	✓	1	127.250	344.475	✓	1	1001.610	1137.683	✗	-1
34	274.080	361.972	153.825	307.644	✓	1	112.425	303.851	✓	1	984.125	1129.468	✗	-1
35	213.850	274.986	121.690	230.028	✓	1	103.945	275.889	✓	1	1087.690	1158.428	✗	-1
36	247.105	325.090	174.490	239.626	✓	1	94.815	199.958	✓	1	1071.925	1144.100	✗	-1
37	228.960	284.929	122.420	241.249	✓	1	71.680	108.891	✓	1	1076.190	1142.302	✗	-1
38	250.200	276.726	164.080	278.756	✓	1	142.800	354.376	✓	1	1083.085	1149.561	✗	-1
39	238.390	308.158	157.650	286.012	✓	1	95.860	153.230	✓	1	1050.255	1149.602	✗	-1
40	308.155	367.380	224.705	443.522	✓	1	166.000	436.341	✓	1	1143.940	1147.126	✗	-1
41	276.150	371.125	140.755	200.309	✓	1	103.250	255.636	✓	1	1072.080	1135.136	✗	-1
42	307.670	413.476	222.065	418.457	✓	1	129.280	313.803	✓	1	1088.015	1133.865	✗	-1
43	261.770	367.902	165.005	352.106	✓	1	96.870	225.662	✓	1	1055.780	1121.037	✗	-1
44	266.070	320.002	137.500	253.560	✓	1	95.450	169.421	✓	1	1062.575	1152.055	✗	-1
45	282.525	327.201	116.605	217.012	✓	1	105.165	253.010	✓	1	1090.665	1134.082	✗	-1
46	306.870	405.512	179.355	336.770	✓	1	123.585	311.118	✓	1	1151.260	1163.228	✗	-1
47	281.740	368.820	157.645	318.990	✓	1	99.840	223.751	✓	1	1081.025	1147.262	✗	-1
48	266.390	297.291	145.355	214.023	✓	1	116.885	284.526	✓	1	1133.885	1134.958	✗	-1
49	236.920	297.533	137.930	219.518	✓	1	71.525	130.835	✓	1	1087.180	1142.061	✗	-1
50	278.635	360.957	188.345	327.316	✓	1	90.145	166.059	✓	1	1132.210	1140.896	✗	-1

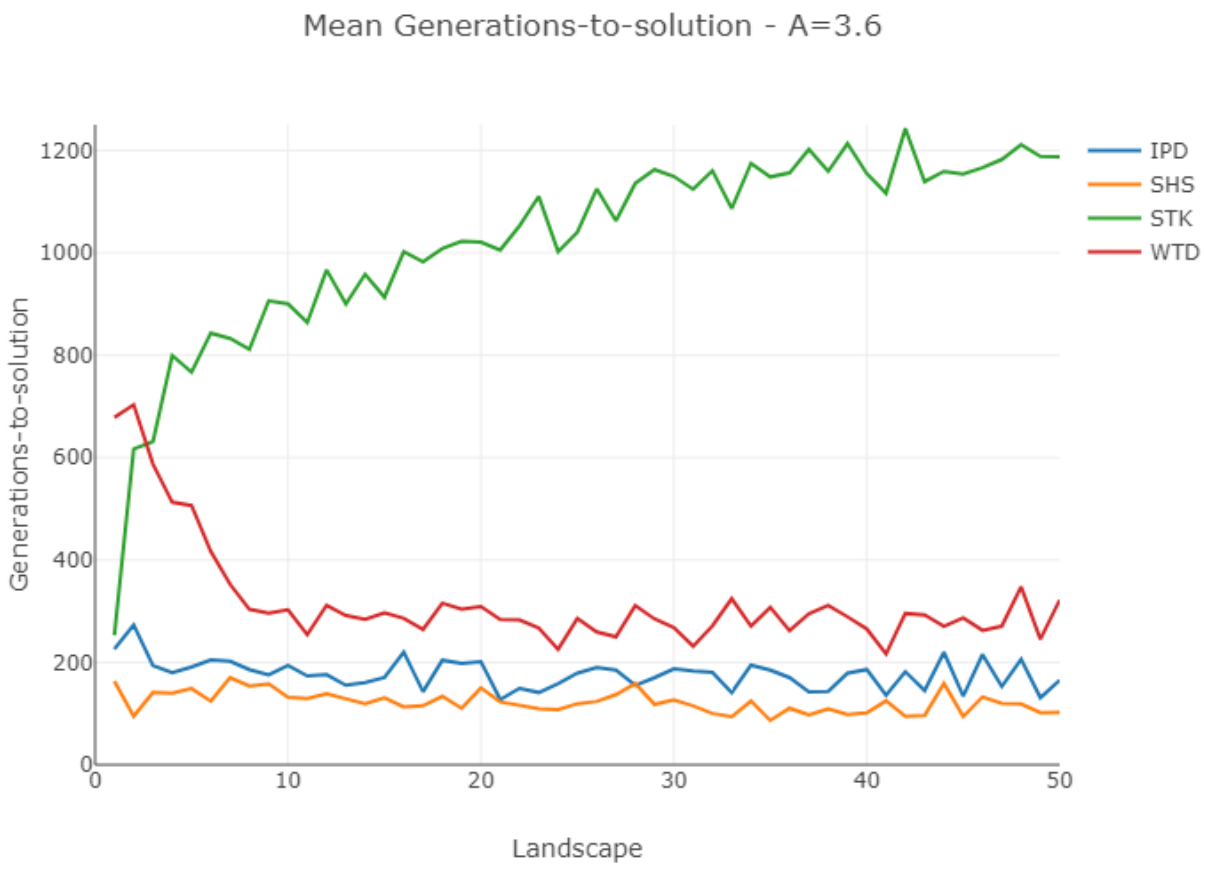


Figure 8-3: Mean generations to solution A=3.6

For A=3.1 the corresponding chart and table are in Figure 8-2 and Table 8-2. At A=3.1 a simple cycle is introduced. Here again WTD settles down to stable performance but it takes longer – landscape 20 (in contrast to 10 for the previous A value). Also, the settled G2S value is higher than for A=1.

At A = 3.1 when the signal becomes non-linear, IPD separates itself from WTD. As shown in Table 8-2, the T-Test results for IPD are all positive except the for the 18th landscape in the sequence. Although IPD is biased towards defection, cooperation can emerge over repeated interactions. The fruits of limited cooperation become apparent when environmental change is

moderately complex. SHS still performs the best at A=3.1, without any caveats. Also, all distribution mechanisms settle at a higher G2S as expected due to higher level of dynamic complexity at A=3.1 vs. A=1.0.

The general trend, seen with A=1 and A=3.1 continues, with A=3.6 (Figure 8-3 & Table 8-3). At A=3.6 the change is nonlinear and thus the environmental dynamic complexity is much higher than with A=3.1. IPD clearly performs better than WTD but the difference seems to be a little less than with A=3.1. In A=3.6, IPD is not significantly better than WTD in 3 out of 50 landscapes (#31, #44 & #46). Whereas, with A=3.1, IPD was significantly better than WTD in all but 1 landscape. SHS is significantly better across the board. STK is better for the first landscape and then progressively its performance deteriorates. As well, due to greater environmental complexity, WTD's downward adjustment is a little slower than with A=3.1.

Table 8-3: A=3.6, Two Sample T-Tests (P<0.05), Mean G2S: {IPD,SHS,STK} < WTD, by Landscape Sequence #

Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD < WTD		SHS Mean	SHS Stdv.	TTest SHS < WTD		STK Mean	STK Stdv.	TTest STK < WTD	
1	678.355	806.743	226.205	443.161	✓	1	163.535	414.050	✓	1	252.820	408.300	✓	1
2	703.350	781.262	272.985	420.238	✓	1	94.875	135.917	✓	1	616.600	857.887	✗	-1
3	587.275	767.052	193.685	349.120	✓	1	141.070	303.592	✓	1	631.115	885.759	✗	-1
4	512.735	714.996	179.980	276.945	✓	1	139.545	277.266	✓	1	799.085	980.432	✗	-1
5	506.420	700.092	191.200	316.653	✓	1	149.040	318.002	✓	1	766.490	969.206	✗	-1
6	416.230	567.251	204.725	367.145	✓	1	124.205	201.621	✓	1	843.140	1031.689	✗	-1
7	352.415	513.681	202.275	366.366	✓	1	170.265	352.338	✓	1	832.755	1019.471	✗	-1
8	303.515	432.498	185.925	318.015	✓	1	153.735	341.717	✓	1	811.630	1034.849	✗	-1
9	295.675	403.588	175.635	325.565	✓	1	158.250	378.039	✓	1	906.250	1065.279	✗	-1
10	302.820	440.483	194.080	347.544	✓	1	131.720	287.249	✓	1	899.990	1070.563	✗	-1
11	254.050	359.478	173.585	295.989	✓	1	129.085	299.050	✓	1	863.675	1071.202	✗	-1
12	311.440	418.643	175.825	241.652	✓	1	138.790	215.337	✓	1	966.600	1089.462	✗	-1
13	291.435	421.853	155.245	262.907	✓	1	128.940	283.490	✓	1	899.915	1063.163	✗	-1
14	284.130	399.136	160.580	257.073	✓	1	118.770	235.150	✓	1	958.110	1078.424	✗	-1
15	296.345	398.085	170.640	300.263	✓	1	130.595	359.534	✓	1	913.245	1091.829	✗	-1
16	285.925	370.783	220.230	358.542	✓	1	113.245	280.344	✓	1	1001.805	1095.050	✗	-1
17	264.370	357.814	142.815	199.473	✓	1	115.165	278.183	✓	1	982.515	1105.724	✗	-1
18	315.440	459.060	204.245	352.107	✓	1	133.645	285.919	✓	1	1008.450	1100.266	✗	-1
19	303.905	390.982	197.725	341.859	✓	1	110.270	206.846	✓	1	1022.515	1100.841	✗	-1
20	309.225	377.867	201.330	295.030	✓	1	150.195	329.300	✓	1	1020.780	1110.377	✗	-1
21	283.635	384.917	127.525	179.687	✓	1	122.810	299.968	✓	1	1005.320	1110.695	✗	-1
22	282.675	360.463	149.510	207.418	✓	1	116.670	238.737	✓	1	1052.775	1112.949	✗	-1
23	267.210	326.100	141.190	291.094	✓	1	109.745	262.474	✓	1	1109.990	1145.379	✗	-1
24	225.360	285.573	158.550	237.865	✓	1	107.630	252.929	✓	1	1001.910	1131.247	✗	-1
25	285.830	375.117	179.035	306.953	✓	1	119.315	229.931	✓	1	1040.365	1135.070	✗	-1
26	259.550	307.056	190.360	341.653	✓	1	123.645	306.516	✓	1	1124.675	1142.822	✗	-1
27	249.730	329.600	184.705	308.088	✓	1	137.030	321.605	✓	1	1062.240	1133.042	✗	-1
28	310.700	430.214	154.845	177.569	✓	1	158.740	351.032	✓	1	1135.610	1134.730	✗	-1
29	284.815	359.878	170.175	295.158	✓	1	117.895	259.020	✓	1	1162.385	1159.136	✗	-1
30	268.275	334.069	187.245	338.320	✓	1	126.335	279.549	✓	1	1149.165	1148.168	✗	-1
31	231.540	293.565	182.805	300.537	✗	-1	114.995	271.305	✓	1	1124.195	1148.748	✗	-1
32	271.400	343.463	180.565	336.429	✓	1	100.270	257.214	✓	1	1160.180	1148.057	✗	-1
33	324.445	355.557	140.840	248.271	✓	1	93.585	244.944	✓	1	1086.490	1134.842	✗	-1
34	270.765	383.771	194.760	297.568	✓	1	124.620	343.534	✓	1	1174.625	1158.750	✗	-1
35	307.525	385.738	184.925	324.526	✓	1	86.495	145.250	✓	1	1148.200	1146.618	✗	-1
36	261.755	356.352	170.130	286.679	✓	1	110.450	290.062	✓	1	1156.165	1154.519	✗	-1
37	294.745	399.426	142.475	258.205	✓	1	97.725	254.225	✓	1	1202.240	1162.205	✗	-1
38	311.000	404.321	143.375	228.957	✓	1	109.140	170.767	✓	1	1159.020	1159.761	✗	-1
39	289.030	390.274	178.905	376.293	✓	1	98.230	232.736	✓	1	1213.340	1173.599	✗	-1
40	265.245	364.760	185.985	286.676	✓	1	101.380	235.880	✓	1	1154.605	1148.657	✗	-1
41	216.425	279.654	135.590	211.719	✓	1	124.835	389.038	✓	1	1116.180	1136.277	✗	-1
42	295.485	390.751	181.745	322.566	✓	1	94.755	218.543	✓	1	1242.485	1150.237	✗	-1
43	292.635	355.951	144.755	242.122	✓	1	96.305	202.990	✓	1	1138.920	1149.368	✗	-1
44	270.245	330.704	219.555	389.882	✗	-1	158.765	346.930	✓	1	1159.085	1142.769	✗	-1
45	287.020	372.742	134.385	174.694	✓	1	94.205	213.422	✓	1	1153.400	1152.112	✗	-1
46	262.460	317.709	215.275	417.042	✗	-1	132.375	258.831	✓	1	1166.145	1156.556	✗	-1
47	270.810	366.029	152.710	238.173	✓	1	119.710	287.569	✓	1	1182.225	1159.406	✗	-1
48	347.495	474.985	206.270	374.717	✓	1	118.890	259.253	✓	1	1211.020	1156.954	✗	-1
49	245.190	292.021	130.855	253.260	✓	1	101.625	183.823	✓	1	1187.960	1163.997	✗	-1
50	321.775	376.985	164.925	265.311	✓	1	102.145	209.826	✓	1	1187.890	1168.143	✗	-1

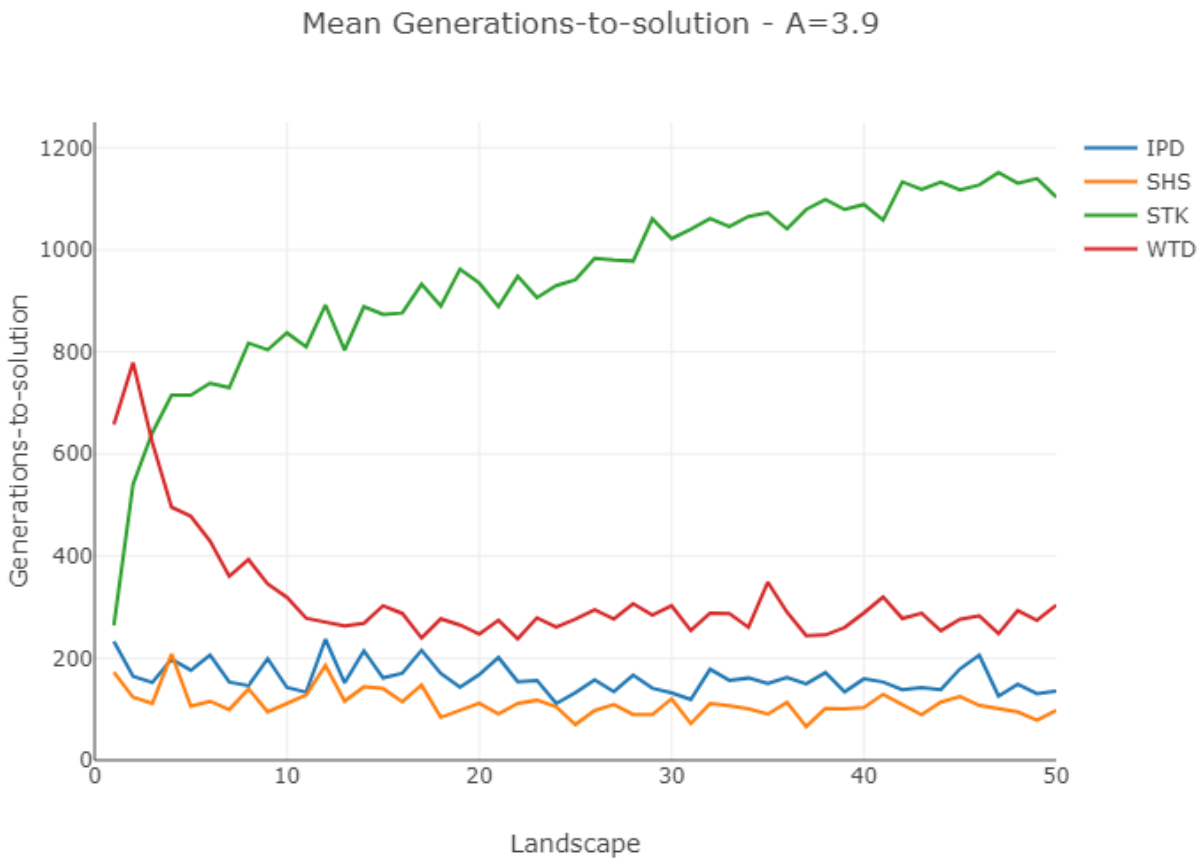


Figure 8-4: Mean generations to solution =3.9

At A = 3.9 the environment's dynamic complexity is on the edge of chaos. The trends established with lower values of A continue with A=3.9 (Figure 8-4., Table 8-4). Statistical tests in Table 8-4 show that IPD mostly performs better than WTD at A=3.9. STK performs statistically better than WTD very early on then its performance progressively worsens. SHS consistently performs better. WTD settles down at about the same rate with A=3.9 as with A=3.6.

Table 8-4: A=3.9, Two Sample T-Tests (P<0.05), Mean G2S: {IPD,SHS,STK} < WTD, by Landscape Sequence #

Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD < WTD		SHS Mean	SHS Stdv.	TTest SHS < WTD		STK Mean	STK Stdv.	TTest STK < WTD	
1	658.125	738.335	232.470	469.185	✓	1	172.670	395.331	✓	1	264.490	457.034	✓	1
2	778.990	852.226	164.525	224.830	✓	1	123.285	280.323	✓	1	541.000	809.937	✓	1
3	622.170	750.684	152.090	202.376	✓	1	111.360	200.598	✓	1	641.685	882.553	✗	-1
4	495.915	685.381	198.140	365.820	✓	1	207.895	480.219	✓	1	715.150	942.862	✗	-1
5	478.155	661.925	176.405	279.389	✓	1	105.860	233.076	✓	1	714.845	955.912	✗	-1
6	429.610	612.780	205.630	363.744	✓	1	115.675	244.865	✓	1	738.905	982.533	✗	-1
7	360.280	478.720	153.150	221.530	✓	1	99.090	145.590	✓	1	729.710	979.655	✗	-1
8	393.185	574.334	146.210	229.752	✓	1	139.885	360.769	✓	1	817.100	1022.064	✗	-1
9	345.265	512.110	198.655	378.997	✓	1	94.750	183.120	✓	1	803.995	1046.507	✗	-1
10	319.010	431.579	142.550	268.464	✓	1	109.780	273.393	✓	1	837.310	1052.451	✗	-1
11	278.080	415.575	133.185	181.987	✓	1	127.500	303.290	✓	1	809.780	1031.316	✗	-1
12	269.840	384.726	236.990	379.933	✗	-1	185.945	423.787	✓	1	892.145	1056.615	✗	-1
13	263.475	314.551	151.560	221.917	✓	1	115.795	260.736	✓	1	803.750	1036.633	✗	-1
14	268.400	372.968	214.550	396.312	✗	-1	143.785	378.652	✓	1	888.950	1078.127	✗	-1
15	302.710	410.775	161.150	310.784	✓	1	140.335	378.433	✓	1	873.145	1053.503	✗	-1
16	287.490	353.976	170.605	322.213	✓	1	114.315	262.136	✓	1	876.180	1064.710	✗	-1
17	239.310	356.938	215.390	424.734	✗	-1	147.170	349.422	✓	1	932.710	1071.828	✗	-1
18	276.985	352.255	170.040	309.080	✓	1	83.860	127.849	✓	1	890.005	1067.872	✗	-1
19	264.955	337.042	143.350	209.021	✓	1	97.005	213.145	✓	1	962.115	1096.631	✗	-1
20	247.170	285.171	167.810	274.940	✓	1	111.780	237.390	✓	1	934.825	1084.664	✗	-1
21	274.505	366.359	201.155	428.644	✓	1	91.120	260.109	✓	1	888.800	1092.058	✗	-1
22	238.035	301.154	153.995	271.352	✓	1	111.480	282.977	✓	1	948.195	1094.160	✗	-1
23	278.875	338.613	156.360	241.431	✓	1	117.555	230.563	✓	1	906.185	1077.485	✗	-1
24	260.810	344.514	111.085	214.082	✓	1	104.805	284.772	✓	1	929.700	1088.808	✗	-1
25	276.235	341.515	132.690	223.165	✓	1	69.845	112.674	✓	1	941.165	1091.861	✗	-1
26	295.100	372.720	157.620	281.807	✓	1	97.385	182.604	✓	1	983.525	1110.013	✗	-1
27	276.600	381.810	134.570	180.922	✓	1	108.955	294.696	✓	1	979.830	1099.626	✗	-1
28	306.950	379.238	166.965	274.936	✓	1	89.615	202.504	✓	1	977.950	1113.934	✗	-1
29	284.425	353.618	141.015	226.911	✓	1	89.280	138.102	✓	1	1060.655	1130.608	✗	-1
30	302.830	385.269	131.905	235.492	✓	1	120.450	277.559	✓	1	1021.735	1119.163	✗	-1
31	254.365	328.916	118.810	166.633	✓	1	71.730	161.467	✓	1	1040.530	1144.690	✗	-1
32	287.945	335.622	178.445	306.456	✓	1	111.310	263.640	✓	1	1061.255	1106.323	✗	-1
33	287.270	420.029	156.395	320.850	✓	1	106.940	170.897	✓	1	1045.715	1136.995	✗	-1
34	260.505	339.803	160.980	297.446	✓	1	100.945	247.726	✓	1	1065.185	1136.721	✗	-1
35	348.905	489.428	150.080	295.892	✓	1	90.230	230.574	✓	1	1072.615	1147.182	✗	-1
36	290.055	373.703	161.900	237.958	✓	1	113.165	262.112	✓	1	1041.025	1145.788	✗	-1
37	244.125	264.003	149.480	315.497	✓	1	65.775	91.103	✓	1	1079.160	1145.213	✗	-1
38	245.295	342.091	171.650	267.471	✓	1	101.475	254.092	✓	1	1098.910	1142.119	✗	-1
39	259.855	315.545	134.090	188.440	✓	1	100.305	251.171	✓	1	1078.815	1150.752	✗	-1
40	288.695	374.682	159.515	253.543	✓	1	103.190	265.159	✓	1	1088.695	1135.440	✗	-1
41	319.800	466.599	153.255	321.555	✓	1	129.180	330.752	✓	1	1058.355	1155.785	✗	-1
42	277.655	347.013	137.935	184.991	✓	1	108.430	296.037	✓	1	1133.150	1154.890	✗	-1
43	288.045	440.326	141.895	223.071	✓	1	89.155	232.627	✓	1	1118.665	1144.351	✗	-1
44	253.755	310.214	138.020	226.154	✓	1	113.710	325.007	✓	1	1132.665	1141.417	✗	-1
45	276.335	345.992	179.095	321.316	✓	1	124.640	313.564	✓	1	1117.275	1139.332	✗	-1
46	282.660	351.702	205.875	380.063	✓	1	107.465	252.657	✓	1	1127.210	1133.731	✗	-1
47	248.065	317.842	125.925	186.678	✓	1	101.925	288.568	✓	1	1151.380	1158.025	✗	-1
48	293.170	386.899	149.055	251.481	✓	1	94.430	244.561	✓	1	1130.625	1152.795	✗	-1
49	274.050	292.353	130.600	192.523	✓	1	78.605	91.004	✓	1	1139.400	1150.857	✗	-1
50	303.725	435.204	135.605	252.836	✓	1	97.180	237.694	✓	1	1103.525	1144.508	✗	-1

In summary, game-based distribution mechanisms that encompass cooperation, generally perform better than WTD – the purely competitive mechanism – however, the game-based mechanisms don't get a blanket pass. STK cannot seem to be able to track changes well in a dynamic environment, even if its initial performance is always better than WTD. The WTD mechanism takes some time to learn to respond to environmental changes. Its initial performance is quite off the mark but it settles down into a steady rhythm in later stages. However, from the observed data the best mechanism SHS is statistically better than WTD in almost all cases. SHS finds the solution in about 100 less generations than WTD does, for A>=3.1. Subsequent sections in this chapter analyze the properties of the distribution mechanisms from different perspectives to try to develop insights into the observed performance characteristics.

8.2 Social Stress / Diffusion Analysis

In section 8.1, the focus was on performance characteristics of the distribution mechanisms which are analyzed qualitatively and with statistical rigor. The rest of the sections are focused on the emergent properties manifested by the distribution mechanisms, reflecting their inner workings in some way. Qualitative & quantitative analysis and a variety of visualization techniques are leveraged to try to derive useful insights in these mechanisms.

The charts presented in Figure 8-5, Figure 8-6, Figure 8-7, and Figure 8-8 are Diffusion trends for A = 1, 3.1, 3.6 and 3.9, respectively. Each chart contrasts the values of Diffusion for each distribution mechanism, over the landscape sequence. From the charts, it is clear that each mechanism operates within a set range. STK has the lowest diffusion, followed by WTD, SHS and then IPD. These patterns persist for all tested A values. Statistical tests for the difference in means between WTD and the rest of the KDs are given in Appendix II. The data are consistent with the

charts. The mean value of WTD for each A and each landscape in the sequence is statistically different from the corresponding means of the other mechanisms, for almost all landscapes. Some exceptions are seen for the first few landscapes in each sequence. While statistical tests are not performed for the pairwise difference between two mechanisms, for all possible pairs, the sample size is large enough at 200 to impart confidence in the mean values. This is because the standard deviations are fairly low compared to the means, in the tables presented in Appendix II.

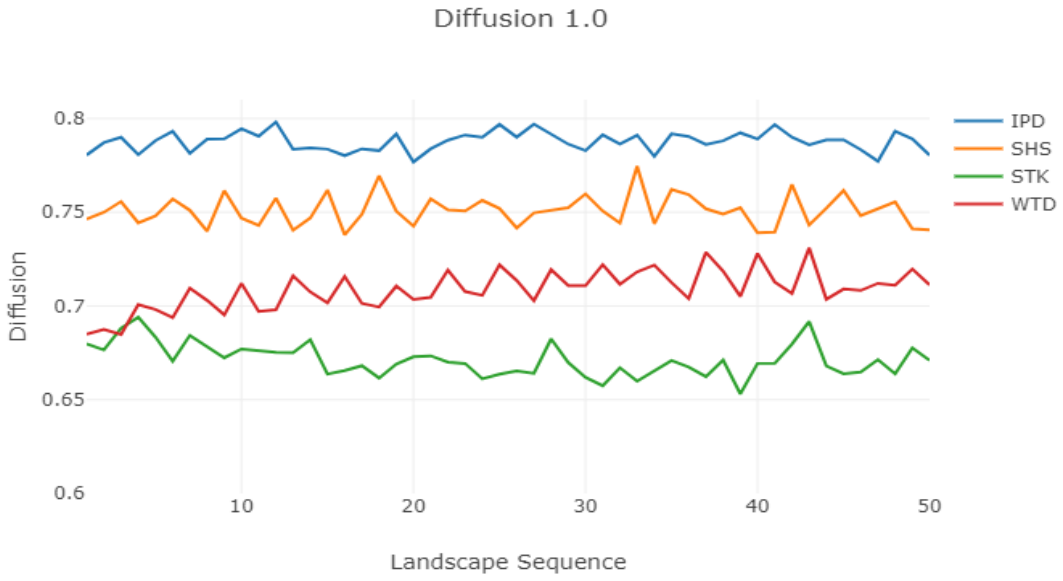


Figure 8-5: Diffusion A=1.0

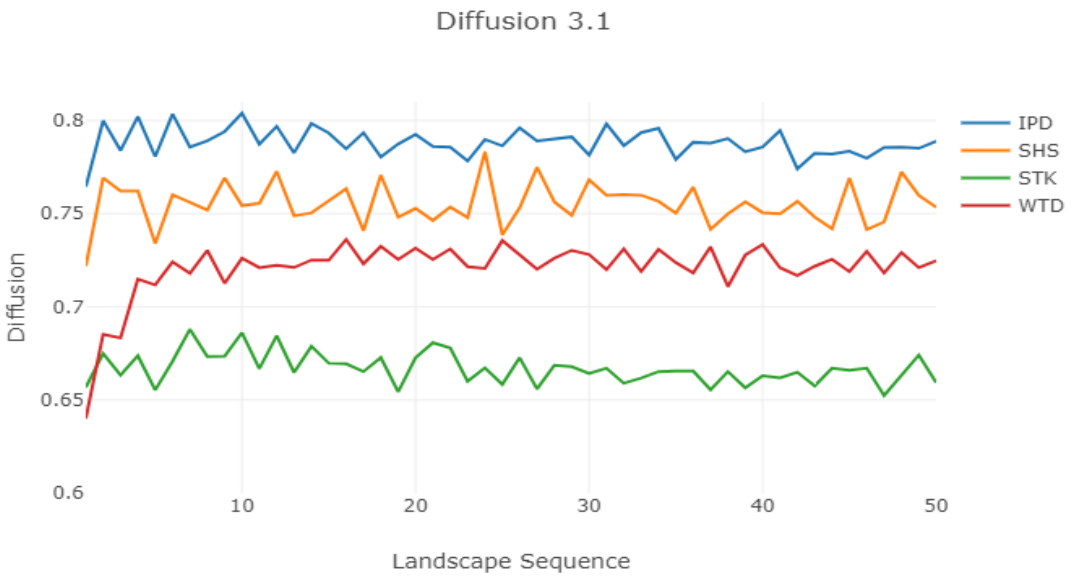


Figure 8-6: Diffusion A=3.1

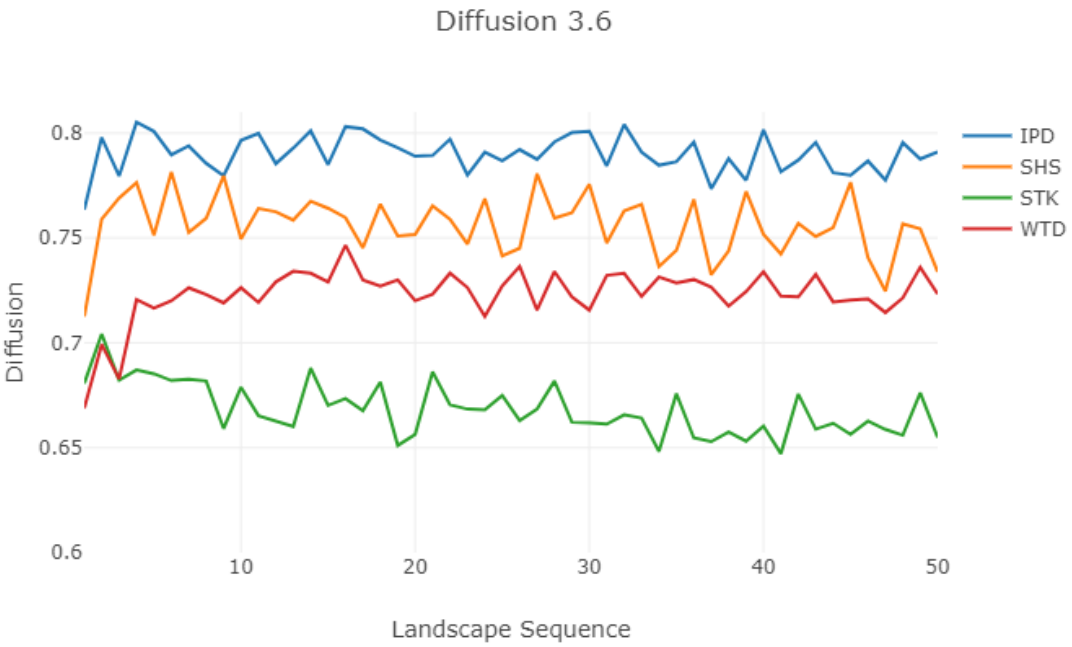


Figure 8-7: Diffusion A=3.6

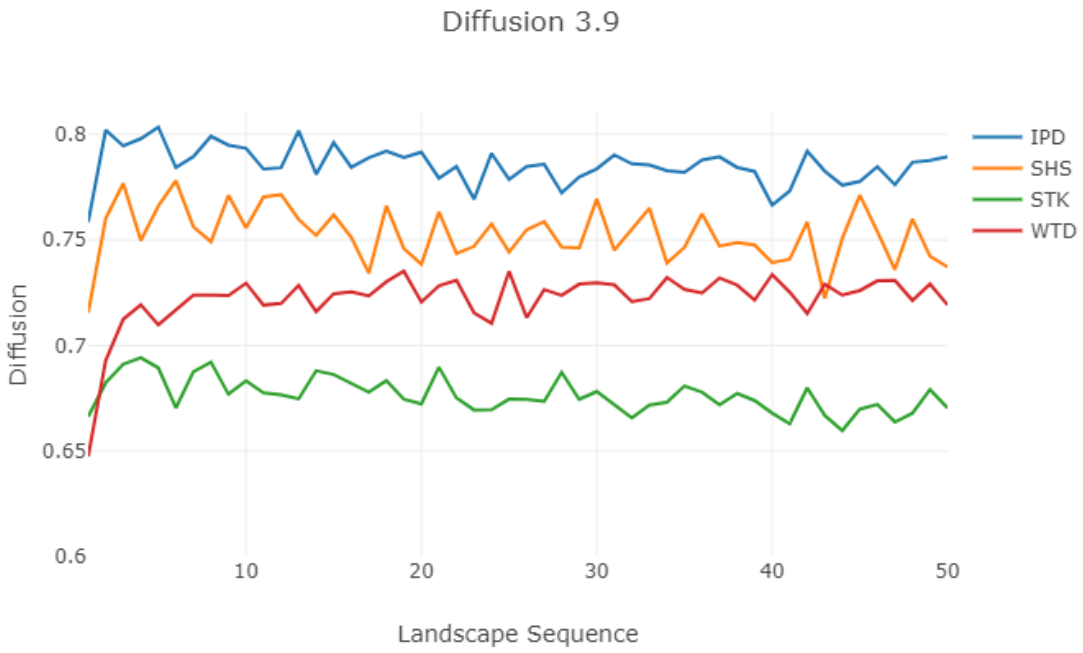


Figure 8-8: Diffusion A=3.9

One can infer a weak relationship between Diffusion and performance characteristics of the distribution mechanisms. STK seems to stress its population individuals the least and it also performs the worst in terms of G2S as discussed in the previous section. The better performing mechanisms IPD and SHS place the highest stress on individuals. However, there may be an optimal level of stress as IPD places greater stress than SHS but still performs worse in terms of G2S than SHS.

The charts in Figure 8-9, Figure 8-10, Figure 8-11 and Figure 8-12 are Diffusion values grouped by distribution mechanism to allow comparison of Diffusion for the same mechanism at different A. Diffusion is not appreciably different for the different A values for the same mechanism. For each mechanism, two-sample t-tests in general are not significant for the difference in means between successive A values, e.g. 1→3.1; 3.1→3.6; and 3.6→3.9. One exception is for WTD

(Figure 8-9). Here the Diffusion for A = 1 is statistically than for A=3.1 for 18 out of 50 landscapes (see Appendix III.V). Another observation is that for WTD the Diffusion starts low and settles to a steady level at about landscape 10. This corresponds to the initial learning by WTD as seen in the G2S charts presented in the previous section.

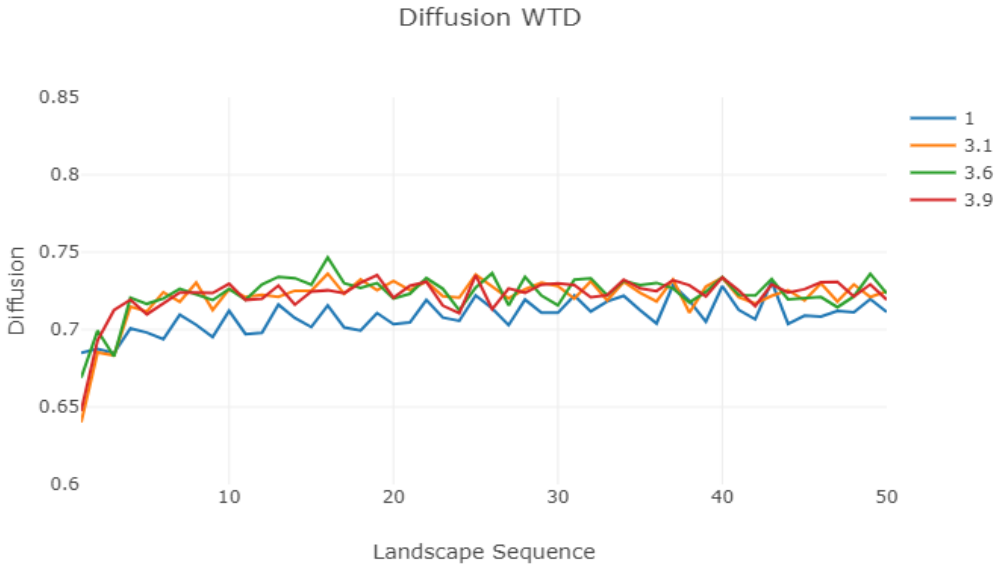


Figure 8-9: WTD diffusion by complexity

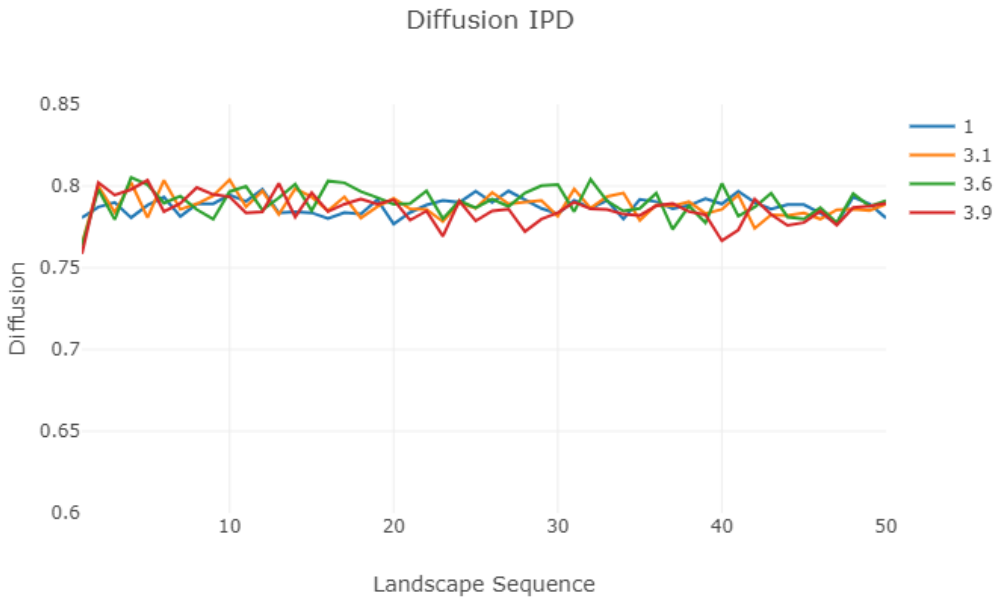


Figure 8-10: IPD diffusion by complexity

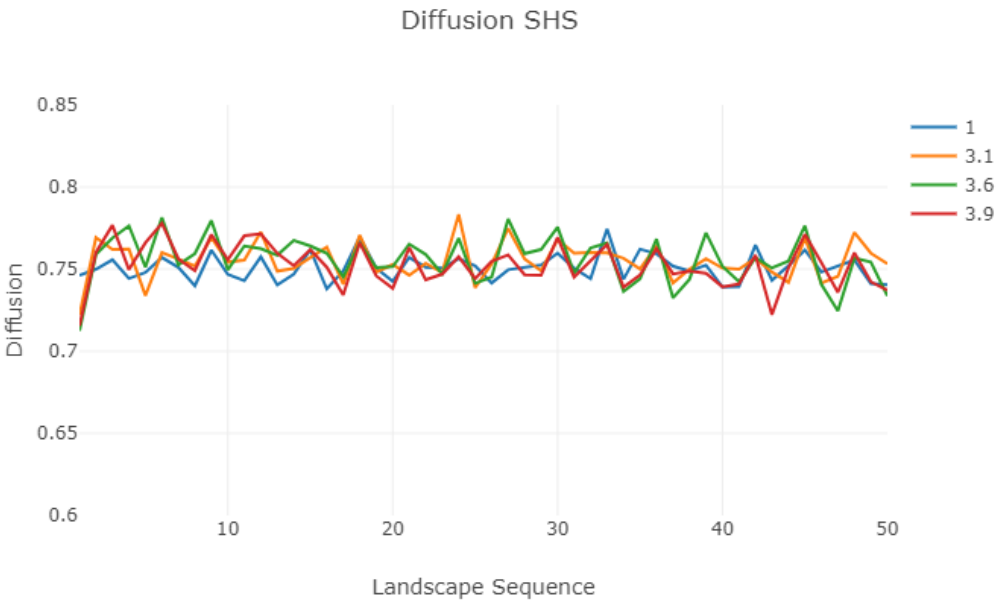


Figure 8-11: Stag-Hunt diffusion by complexity

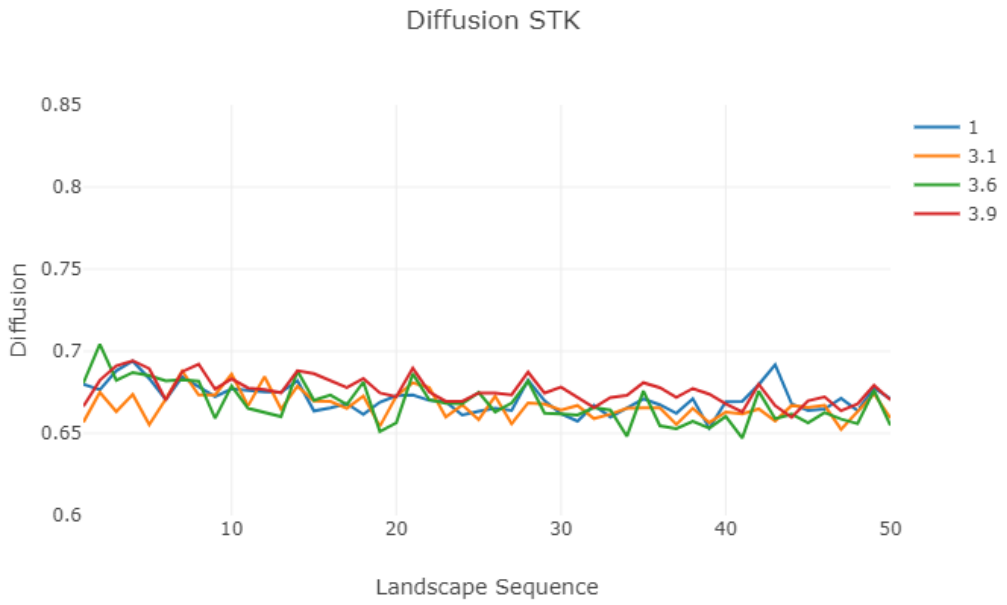


Figure 8-12: Stackelberg diffusion by complexity

For SHS, there is a marked cycle in the Diffusion values over the landscapes. The cycle corresponds to the existence of cooperative and evaluative generations in the SHS mechanism's repertoire (see 6.2).

Diffusion for STK is not only the lowest among all tested distribution mechanisms, there is also a slight downtrend in Diffusion across all A. Otherwise, there is no appreciable difference between Diffusion for the different A values.

To summarize, the Diffusion values for the game mechanisms are statistically different from those for WTD (with minor exceptions – see Appendix II). For each mechanism, Diffusion is not statistically different between successive A changes (for almost all landscapes). The exception is for WTD for A=1→A-3.1 (linear to non-linear transition) where statistical difference exists for 18/50 landscapes (see Appendix II.e). As Diffusion is a measure of stress in the system, these

results show that each mechanism places a different level of stress but that level does not vary much by changes in environmental complexity.

8.3 Segregation Analysis

Diffusion is a measure of the spatial diversity of a node's neighbors. In contrast Segregation Index (also referred to as segregation in this document) is a measure of clustering or grouping of Knowledge Sources among neighbors (7.4). The next four charts – Figure 8-13, Figure 8-14, Figure 8-15 and Figure 8-16 – show segregation by the tested A values, allowing one to compare the distribution mechanisms together for each A. Statistical testing shows that the segregation values for the game mechanisms are statistically different from WTD's for the majority of the landscapes, across the tested A values (Appendix III).

The general pattern followed by the distribution mechanisms for each A value is about the same for segregation. On average, WTD shows the lowest segregation. STK shows a steady increase from very low segregation to high of about 0.8 (range is 0.0-2.0, Table 7-1). SHS reaches highest segregation levels and shows a pronounced cycle due to periodicity of cooperative and evaluative distributions (section 6.2). The SHS peaks correspond to evaluative generations when some individuals may defect to keep their current knowledge assignments causing higher segregation. In cooperative generations the knowledge assignments are more varied due to the 'social rank' based allocation, leading to comparatively lower segregation. IPD shows steady segregation around 0.7 with a slight upward trend for A > 1.0. WTD is purely competitive and gives a rather flat line across all A values. In contrast, the cooperative mechanisms show higher segregation. Although not evident here, the analysis of social dynamics presented in the next section shows that IPD and SHS tend to favor exploration and STK favors exploitation. The higher

segregation shown by the game mechanisms is due the aforementioned biases as these result in preponderance of certain types of KS over others.

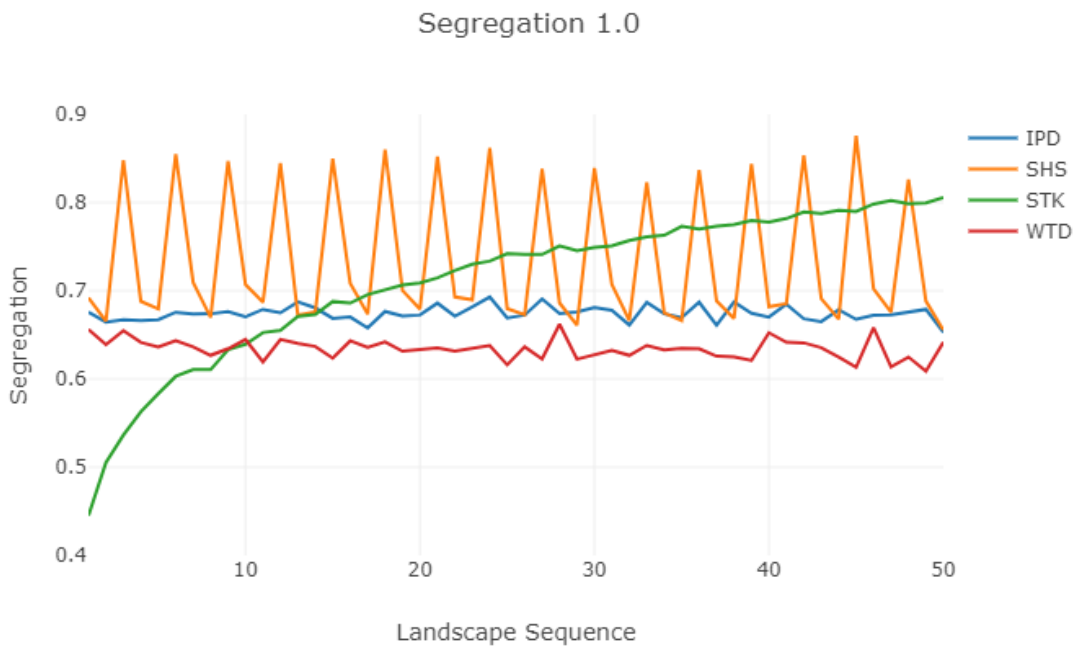


Figure 8-13: Segregation A=1

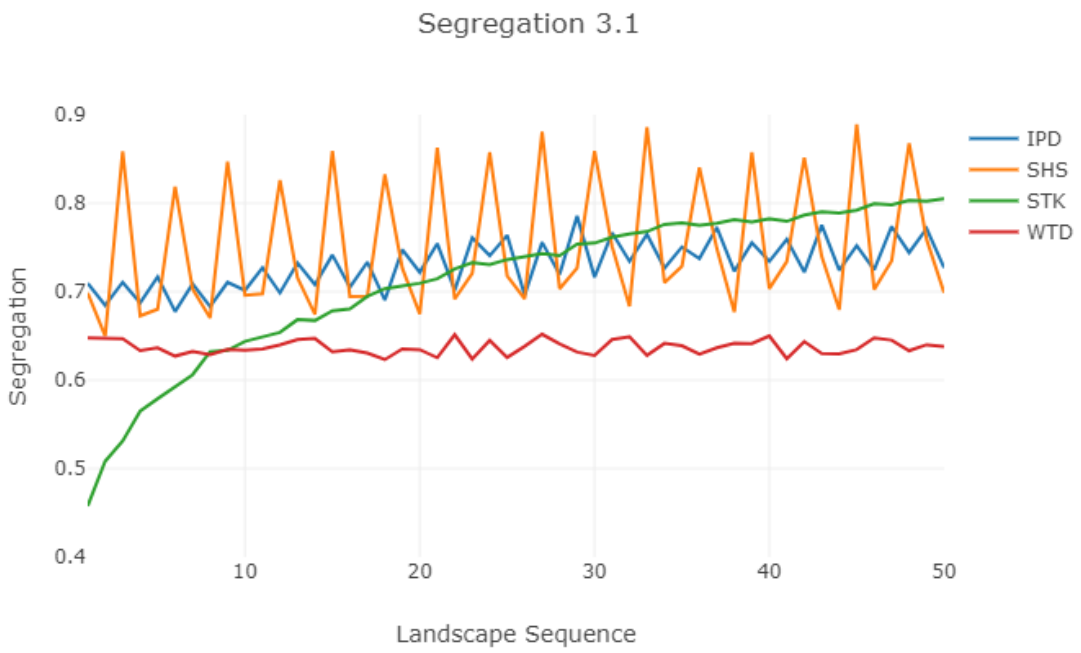


Figure 8-14: Segregation A=3.1

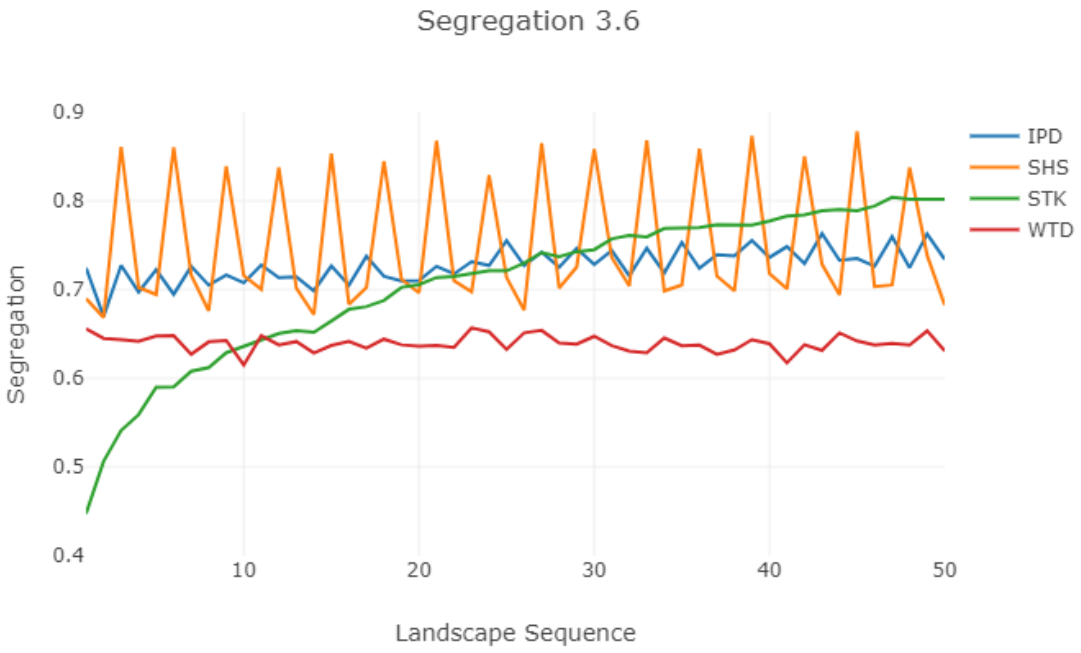


Figure 8-15: Segregation A=3.6

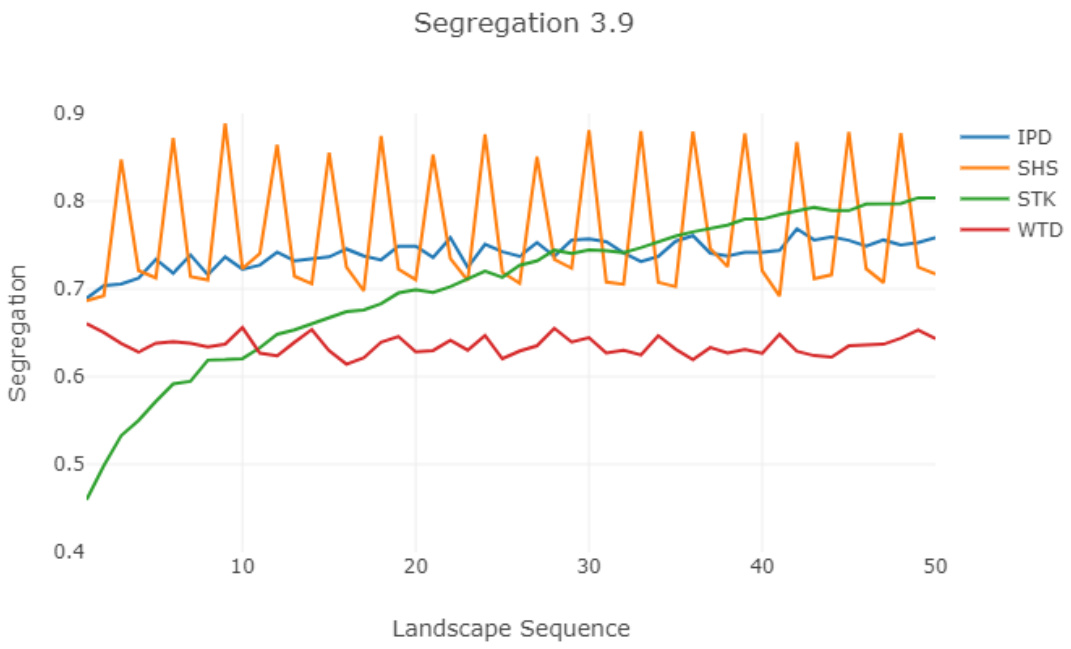


Figure 8-16: Segregation A=3.9

Given that the range is between 0.0 and 2.0 for Segregation, at the aggregate level (i.e. averaged over 200 samples) Segregation stays below 0.9 for all distribution mechanisms. This shows that all mechanisms maintain a healthy diversity of knowledge in local neighborhoods. Segregation close to 0.0 means all Knowledge Sources are evenly distributed in the population. Whereas a value close to 2.0 means that only one or two Knowledge Sources cover the entire population.

The fact that SHS performs the best in terms of G2S, implies that a segregation level towards 0.8 may be ideal. In addition, however, maintaining a steady level of segregation may not be best as WTD does that and still does not perform as well (in terms of G2S) as IPD or SHS. It may well be that segregation that modulates between a range is required for best performance.

The next set of charts are organized by distribution mechanism. Segregation trend for each distribution mechanism by A shows each mechanism's response to A in terms of the level of segregation. Also provided for each distribution mechanism are representative samples of population snapshots of low and high segregation, by A. These are provided for qualitative assessment and to show some examples of the mechanisms at work. The snapshots are randomly selected from population sets that are +/- 1 standard deviations away from the mean segregation of the mechanism.

As a baseline, WTD (Figure 8-17) shows a steady level of segregation around 0.65 that does not seem to vary much as A increases. Statistical testing for the difference in segregation levels between successive A values (e.g. A=1 → A=3.1, etc.) shows that for almost all landscapes there is no statistical difference. WTD's low segregation and unresponsiveness to A is likely because the Knowledge Sources that have been driven out by the voting mechanism are added back in to 20% of the population, at random (section 7.4). Samples of population snapshots showing low and high Segregation are given in Figure 8-17 and Figure 8-18.

Figure 8-17: WTD segregation by complexity and landscape sequence

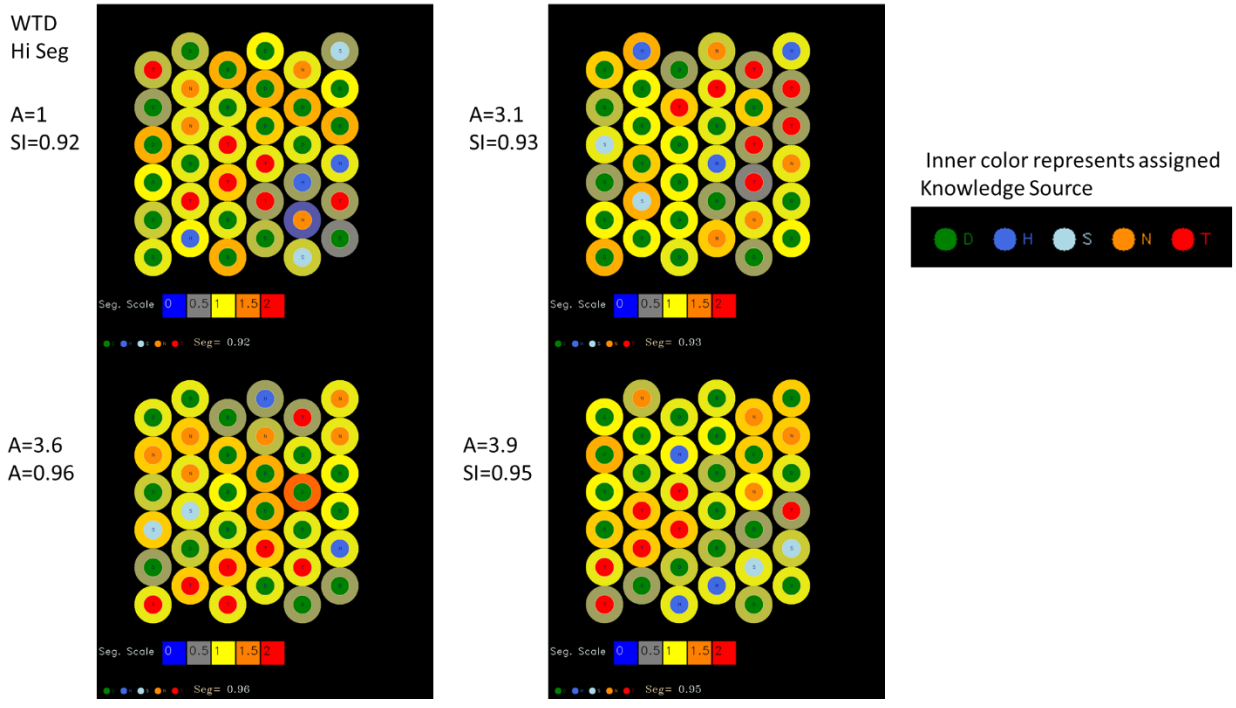


Figure 8-18: WTD high segregation landscape examples

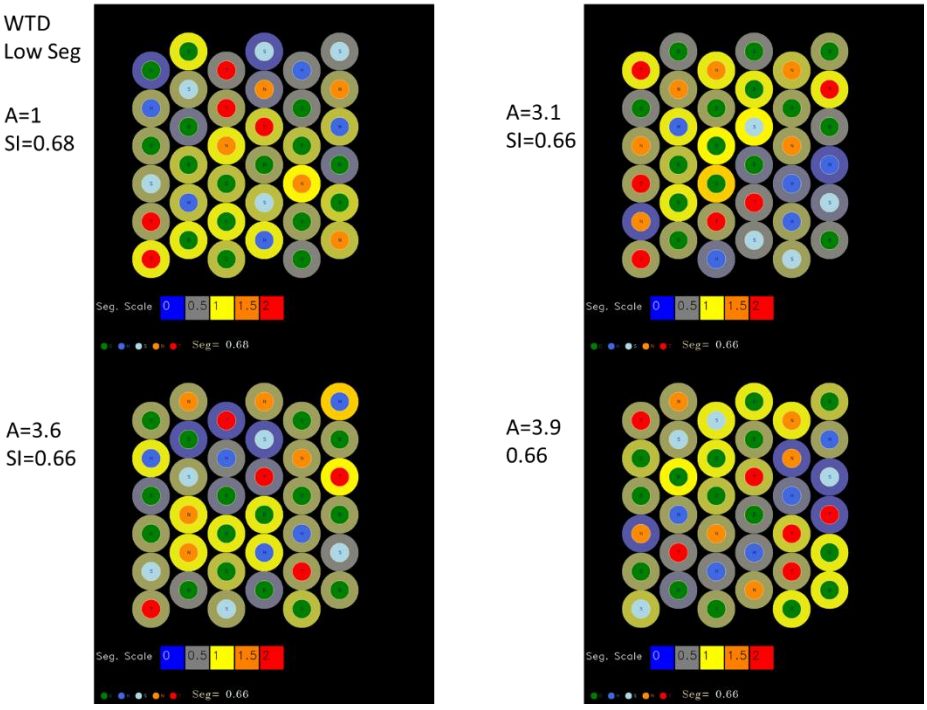


Figure 8-19: WTD low segregation landscape examples

Figure 8-20 show the Segregation Index trends as A increases for IPD. Unlike WTD, here there is a clear separation between segregation at A = 1 and A > 1. It shows that IPD is responsive when environmental complexity changes from linear (A=1) to non-linear (A =3.1). This difference is statistically significant (see Appendix III.e) for almost all landscapes. Changes in segregation levels for other transitions (i.e. A=3.1→3.6; A=3.6→3.9) are not statically significant (by landscape) except for a few landscapes (Appendix III.e). Also, for non-linear environmental complexity IPD shows a slight positive slope in the trend lines.

High and low segregation snapshots given in Figure 8-21 and Figure 8-22, respective, show a dominance of Topographic knowledge. By contrast, equivalent figures for WTD (Figure 8-18, Figure 8-19) show dominance of Domain knowledge that can be exploratory or exploitative based

on the context (like stem cells in Biology). These show that IPD allocates more resources to exploration because Topographic is an explorative Knowledge Source.

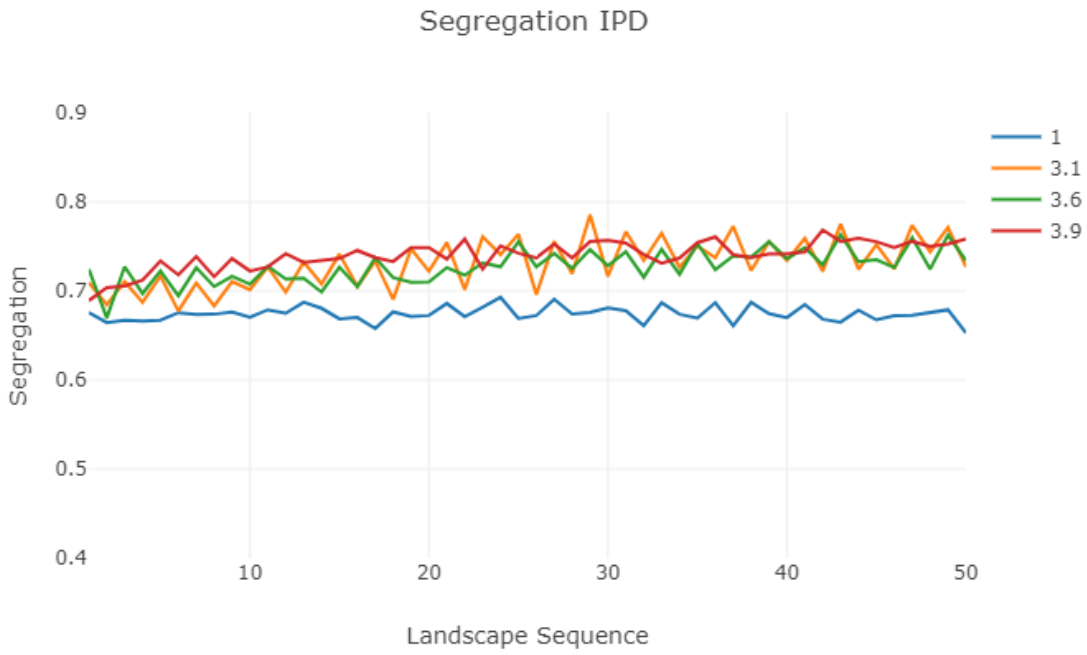


Figure 8-20: IPD segregation by complexity and landscape sequence

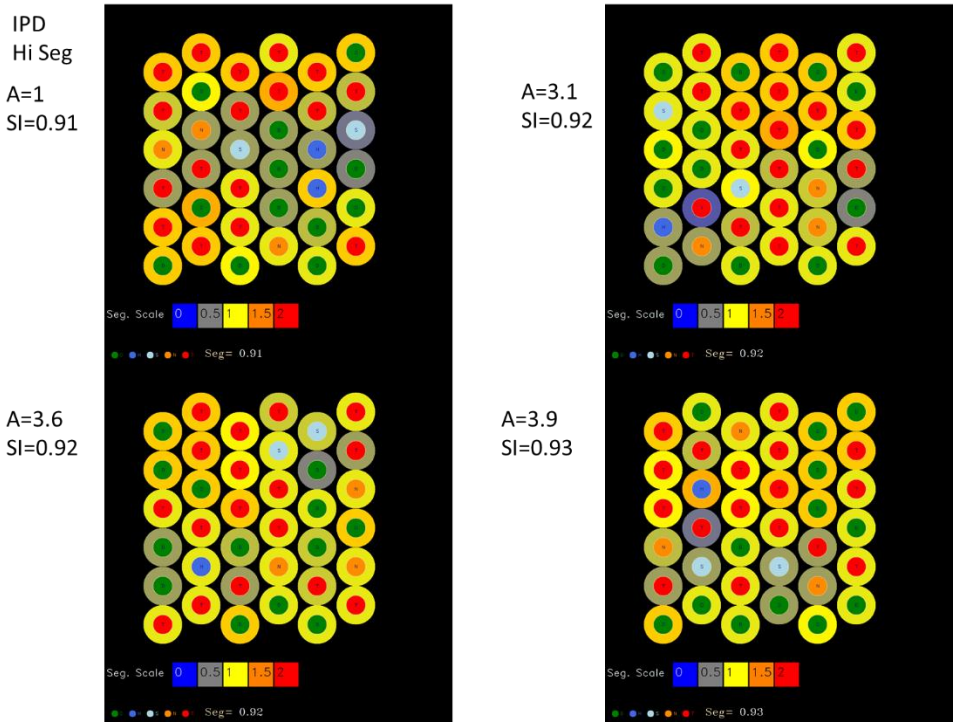


Figure 8-21: IPD high segregation landscape examples

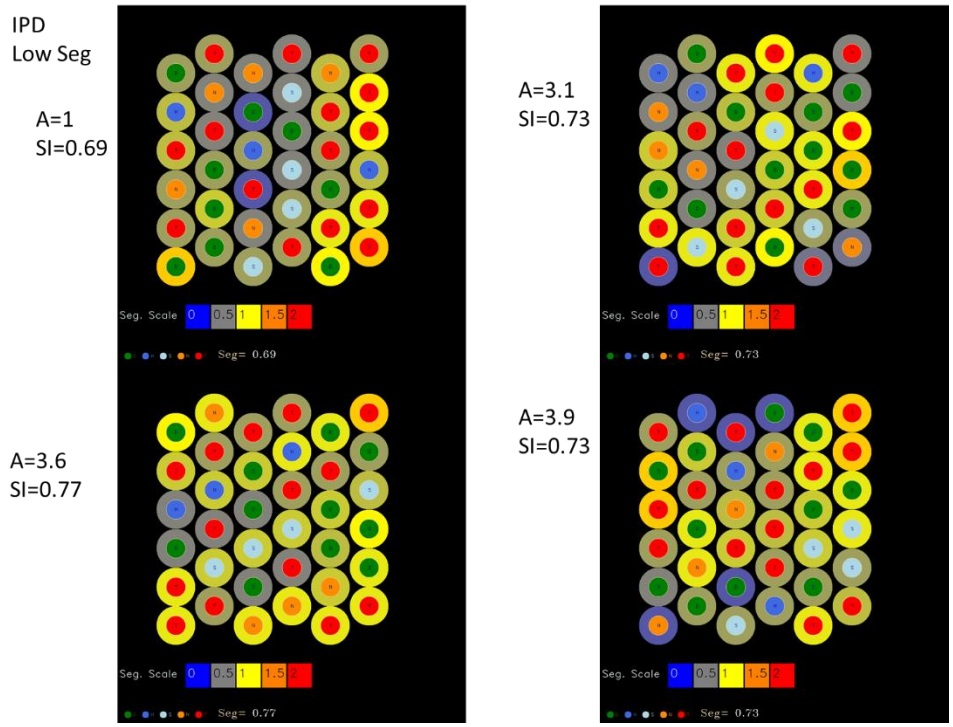


Figure 8-22: IPD low segregation landscape examples

Segregation trends for SHS (Figure 8-23) shows that segregation levels for all A values is are close. Statistical testing showed no difference for segregation levels between successive A, per landscape, except for a few landscapes. High and low Segregation snapshot examples (Figure 8-24, Figure 8-25) show dominance of Topographic and Domain knowledge in the population. This shows neutral to exploratory bias for SHS resource allocation.

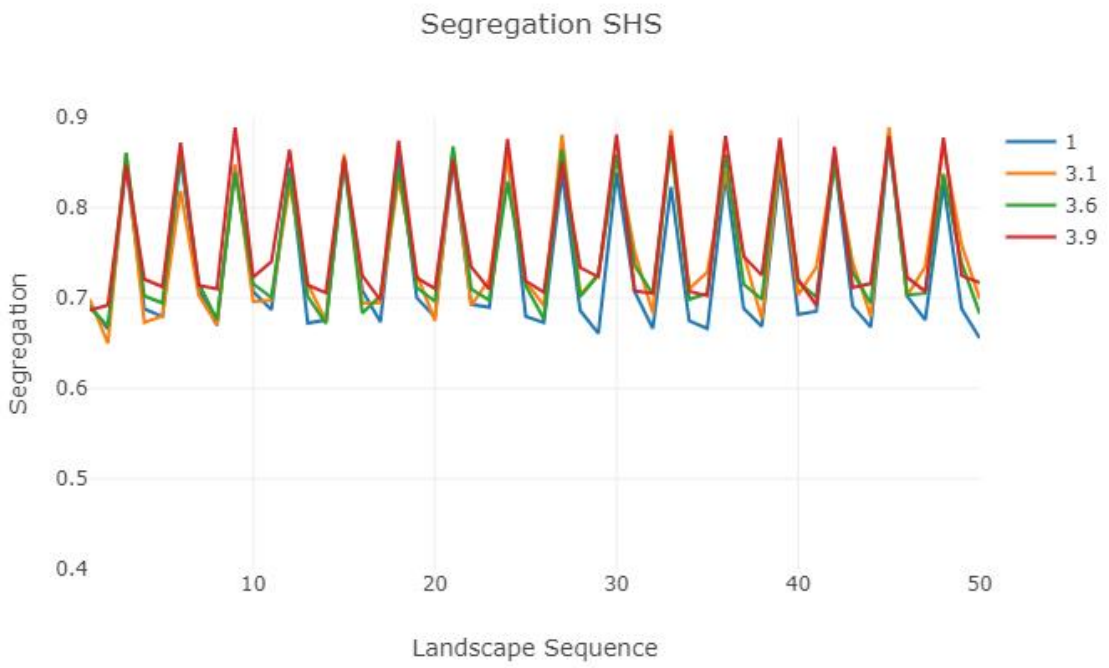


Figure 8-23: Stag-Hunt segregation by complexity and landscape sequence

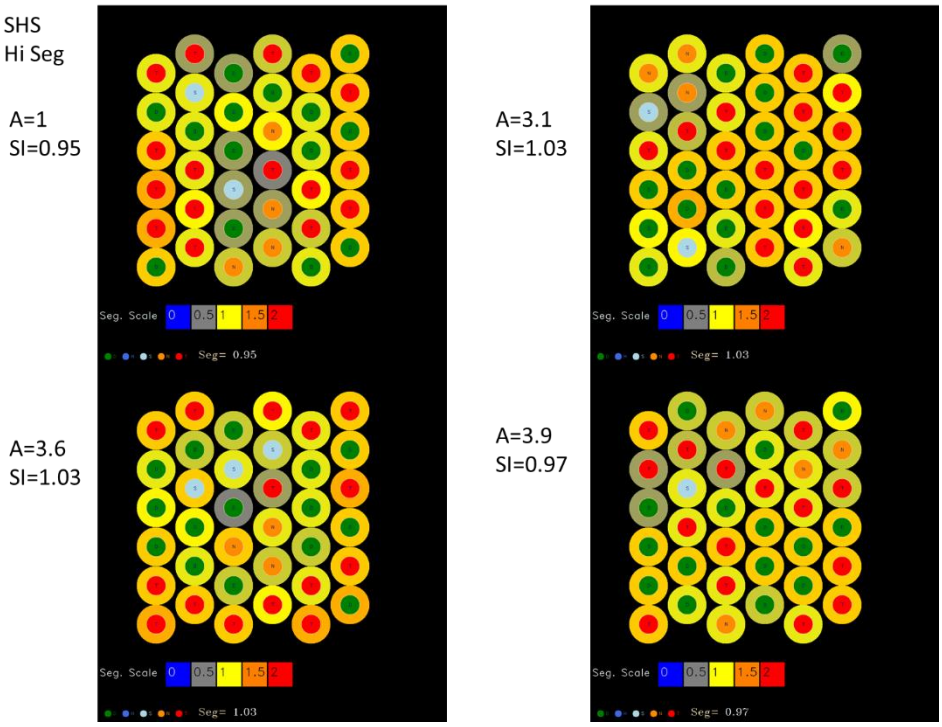


Figure 8-24: Stag-Hunt high segregation landscape examples

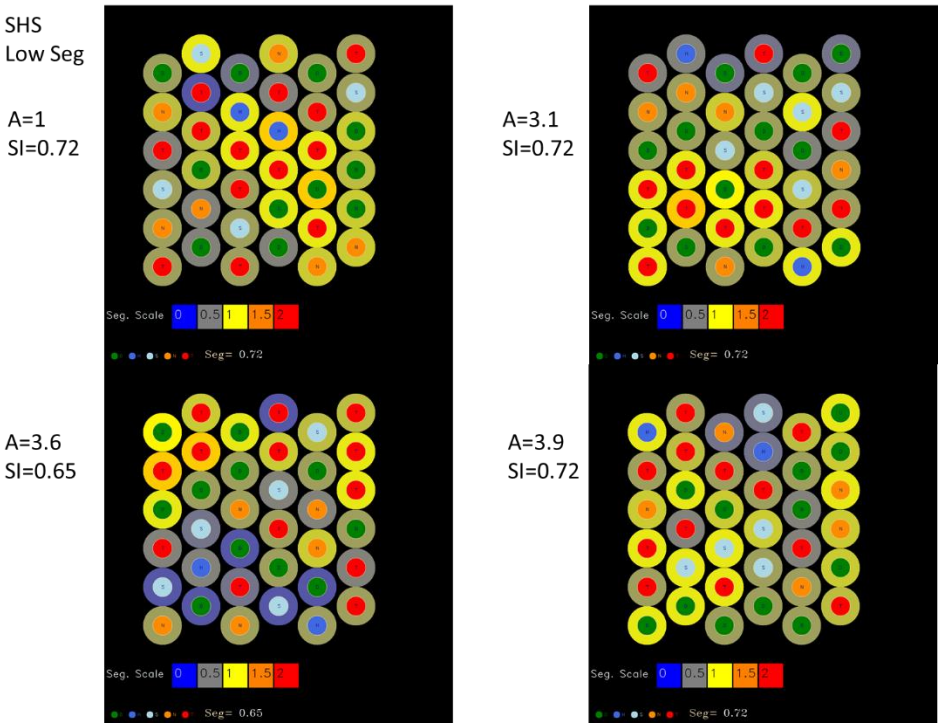


Figure 8-25: Stag-Hunt low segregation landscape examples

STK (Figure 8-26) starts with very low Segregation Index at 0.45 and then steadily rises to 0.8. Segregation is seen as plateauing at 0.8 as the slopes of the trend lines are flatting out towards the end of the landscape sequence. This is no discernable difference in segregation levels by landscape for the different A values. Statistical testing also validates this visual assessment. For STK no landscapes were found where segregation is significantly different between successive A values. This shows that STK is not responsive to changes in environmental complexity. From the population snapshots (Figure 8-27, Figure 8-28) for STK, it can be seen that STK is allocating much resources to exploitation in high segregation cases as compared to WTD (Figure 8-18).

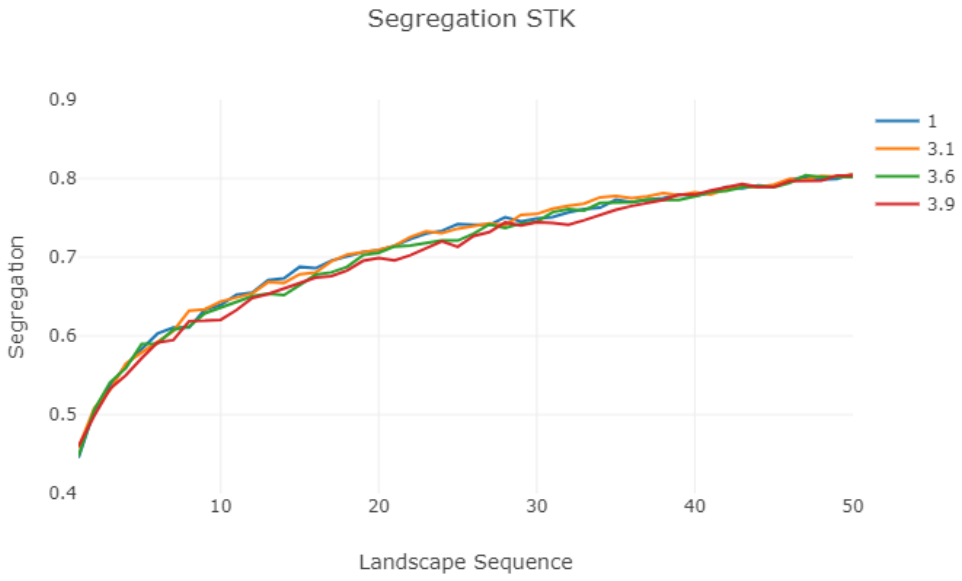


Figure 8-26: Stackelberg segregation by complexity and landscape sequence

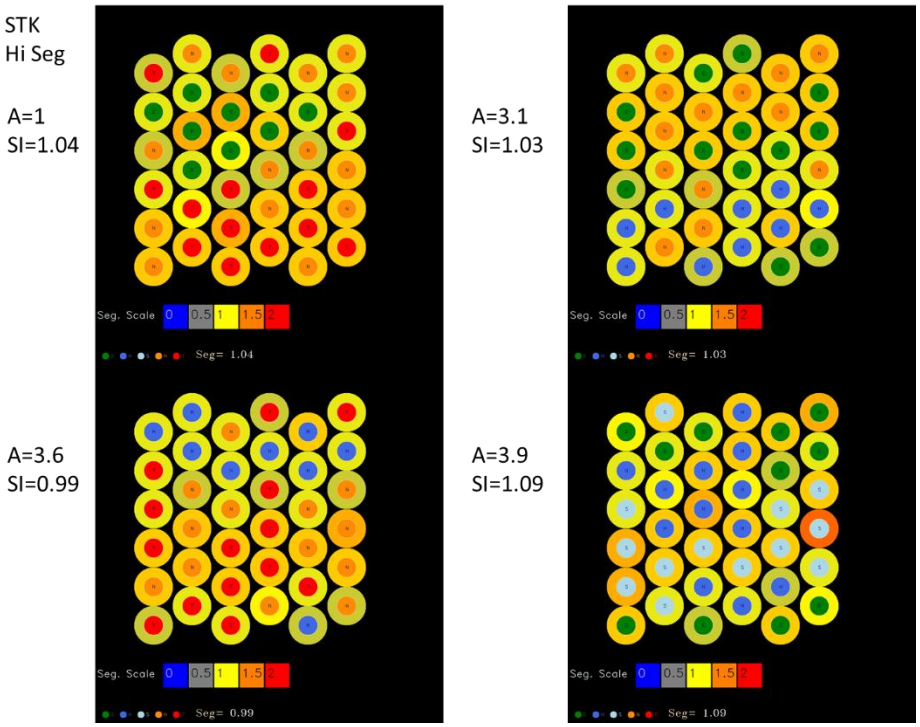


Figure 8-27: Stackelberg high segregation landscape examples

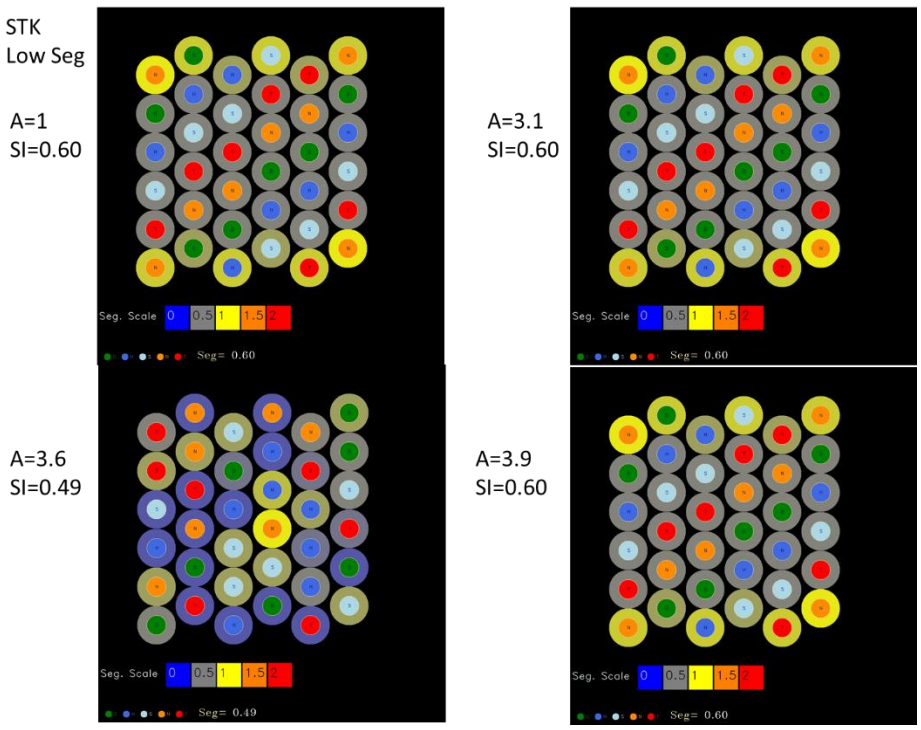


Figure 8-28: Stackelberg low segregation landscape examples

Thus far the analysis was focused on Segregation Index trends by landscape sequence. It was found that cooperative mechanisms have higher segregation in general on a per landscape basis. Also, in general, except for the IPD A=1 → 3.1 case, none of the mechanisms show statistically significant changes in segregation levels for successive A transitions, on a per landscape basis. The trend lines still provide useful insights such as the cyclic nature of segregation in SHS (Figure 8-23), flat and low segregation for WTD (Figure 8-17) and the increase in segregation levels by landscape for STK (Figure 8-26).

Now the analysis is focused on overall segregation by KD-A ignoring the landscape sequence, i.e. the *aggregate* segregation levels for each KD-A combination. The sample size for each mean value (segregation by KD-A) is very large. Each mean is based on close to a billion data points (number of runs * population size * number of generations per landscape * number of landscapes

in sequence = 200 * 36 * 2500 * 50 = 900M). And therefore, difference between any two mean values is statistically significant. The mean segregation values are shown in Table 8-5 and t-tests for significance between successive A for each KD in Table 8-6. Figure 8-29 shows average segregation with +/- 1 standard deviation bands for each KD-A combination.

Table 8-5: Average Segregation by KD and A

KD	A	Mean	Std. Deviation
WTD	3.9	0.63804	0.136789738
WTD	3.6	0.637074	0.136375862
WTD	3.1	0.637499	0.136509734
WTD	1	0.635276	0.135036359
STK	3.9	0.698276	0.161027065
STK	3.6	0.703175	0.158131383
STK	3.1	0.707844	0.157221999
STK	1	0.706842	0.157270233
SHS	3.9	0.727804	0.155380488
SHS	3.6	0.71921	0.154454508
SHS	3.1	0.719512	0.154879267
SHS	1	0.702398	0.150377916
IPD	3.9	0.740624	0.131153711
IPD	3.6	0.726136	0.131002041
IPD	3.1	0.731641	0.132611557
IPD	1	0.676802	0.127004235

Table 8-6: T-Tests for Difference in Segregation between Successive A values by KD

KD	From A	To A	p value, t-test for difference in means
WTD	1	3.1	0.000
WTD	3.1	3.6	0.002
WTD	3.6	3.9	0.000
STK	1	3.1	0.000
STK	3.1	3.6	0.000
STK	3.6	3.9	0.000
SHS	1	3.1	0.000
SHS	3.1	3.6	0.051

SHS	3.6	3.9	0.000
IPD	1	3.1	0.000
IPD	3.1	3.6	0.000
IPD	3.6	3.9	0.000

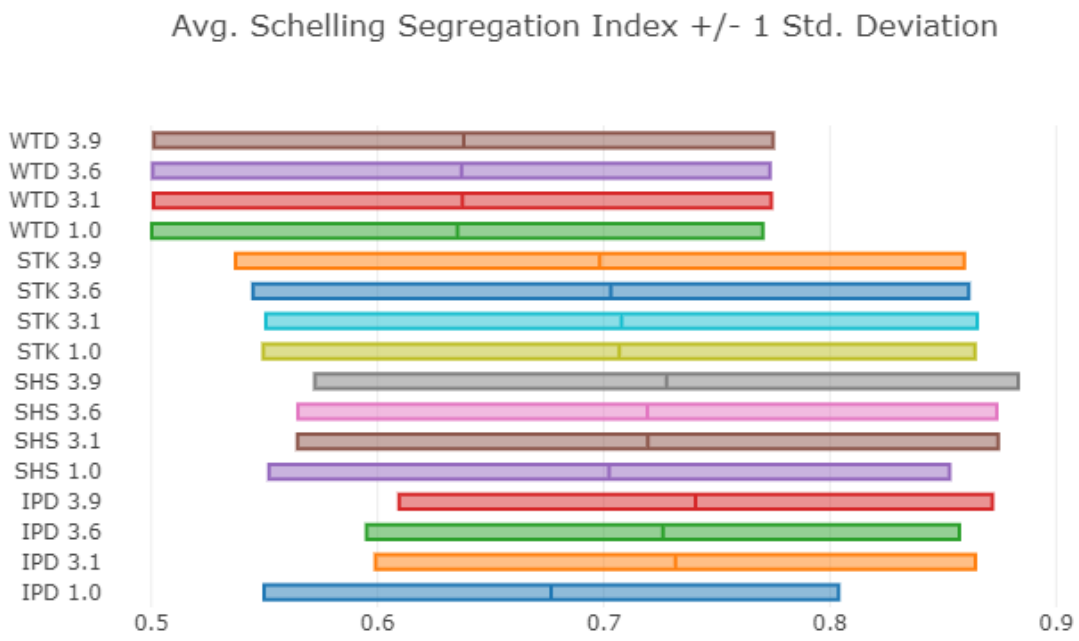


Figure 8-29: Schelling index summary for each KD and A value combination

Due to the large sample size, all means are close to true means and any differences are significant statistically. As per Table 8-5 all mechanisms show a slight decrease in segregation for the transition A=3.1 to A=3.6. The is likely due to the underlying harmonics of the data generating process. Since segregation is an emergent property the exact reason for this anomaly cannot be fully understood. Apart from this anomaly, there are some clear trends.

WTD shows a very slight but overall positive increase in segregation levels as A increases. As do IPD and SHS. IPD shows the most difference, followed by SHS. STK on the other hand shows an overall negative relationship with A. The data shows that better performing mechanisms (in terms of G2S) are responsive to changes in A by increasing segregation levels in the population for the distribution of knowledge. In contrast, the worst performer STK decreases segregation in response to increase in environmental complexity. However, the aggregate level data masks other useful features such as the modulation in segregation levels by the best performing mechanism SHS, which is apparent in the landscape-sequence view of the data. One can surmise that under conditions of stress (high environmental complexity), increased segregation is the valid response. This is borne out by the analysis performed in the next section.

8.4 Communal Knowledge Flow Analysis

In this section the focus is on the dynamics of knowledge flow, as outlined in section 7.5. This section starts with the chord diagrams for the weighted graphs representing community-to-community transitions. Then Page Rank based charts are presented – tree and parallel chords charts by A. After that are presented Sankey charts that show statistically significant changes or deltas in knowledge flow due to A. Finally, a summary of view of knowledge flow deltas captures useful insights relating knowledge flow to performance characteristics.

Note that the analysis presented here has a natural progression; data underlying chord diagrams represents raw transition counts. For Page Rank and community rank this data is normalized to a Markov Chain. For the Sanky charts, the raw count data is converted to mean and standard deviation of change in transitions over the 200 samples collected for each KD-A combination.

For the chord diagrams, the transition weights are the count of transitions across all 200 samples with 50 landscapes each. The total number of transitions for a KD-A combination is in the order 1 billion: $transitions = 200\ samples * 50\ landscapes * 2500\ genertions * 36\ \ pop. \cong 1B$. Note that an individual may be part of several communities at the same time (Figure 7-10). The transitions are counted from community-to-community. So multiple transitions may be generated from one individual depending on the before and after communities the individual is part of. To make the charts more readable, only links with transition weights over 100K are shown.

The labeling scheme described in section 7.5 is used but as an example, 'D_T' is a community composed of Domain and Topographic Knowledge Sources. The WTD chord diagrams for community-to-community transitions are shown in Figure 8 30, Figure 8 31, Figure 8 32, and Figure 8 33. Each chart is for a specific A value. The charts are in order of decreasing community importance, in a counter-clockwise arrangement. For WTD, the strongest community is 'D' (for Domain), then 'D_T', 'D_N' and so on. These three combined all represent the emphasis on exploration.

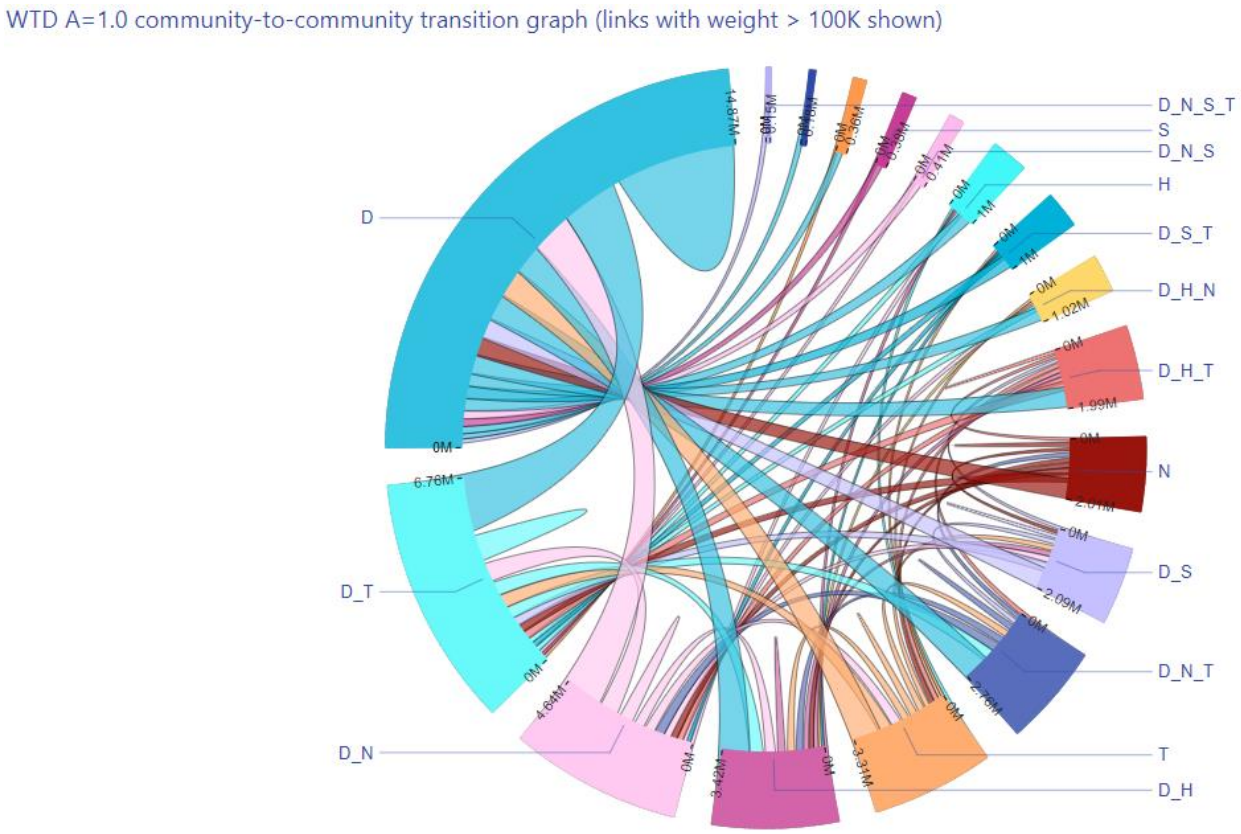


Figure 8-30: WTD A=1 communal knowledge flow graph

All the charts for WTD show more-or-less the same pattern of community dominance. The 'D' community has a large self-loop that indicates that individuals tend to retain 'D' across generations. The self-loops of the other major communities ('D_T', 'D_N', 'D_H', etc.) are much smaller that tells that individuals in these tend to switch to other communities in the next generation, relatively speaking. The top 5 transitions (arcs) all involve Domain knowledge, 'D'. This indicates that WTD allocates considerable resources to Domain. Domain is placed in the middle of the exploratory-exploitative scale and can function on both sides of the divide. However, excessive reliance on Domain may be one of the reasons that WTD does not perform as well as IPD or SHS.

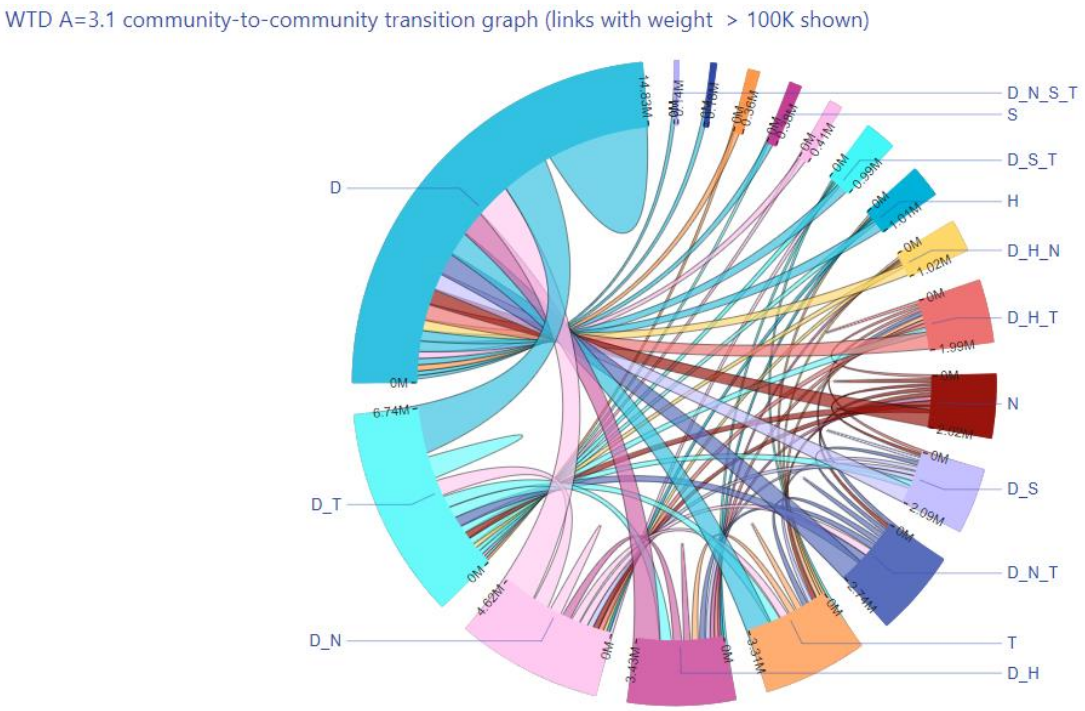


Figure 8-31: WTD A=3.1 communal knowledge flow graph

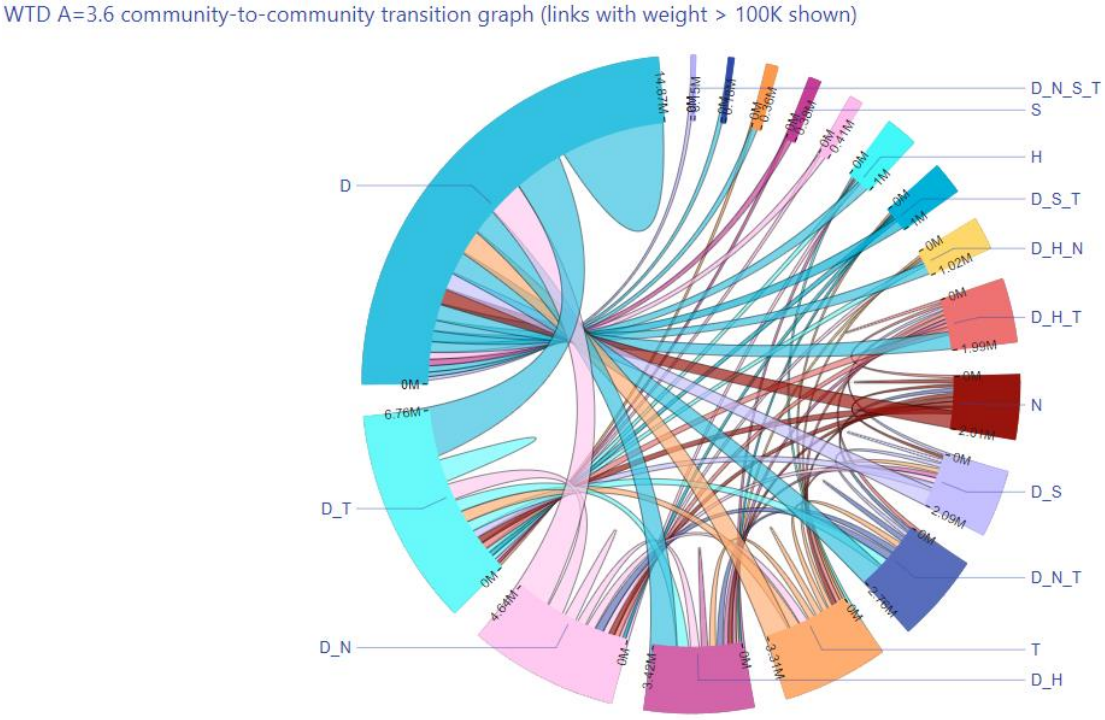


Figure 8-32: WTD A=3.6 communal knowledge flow graph

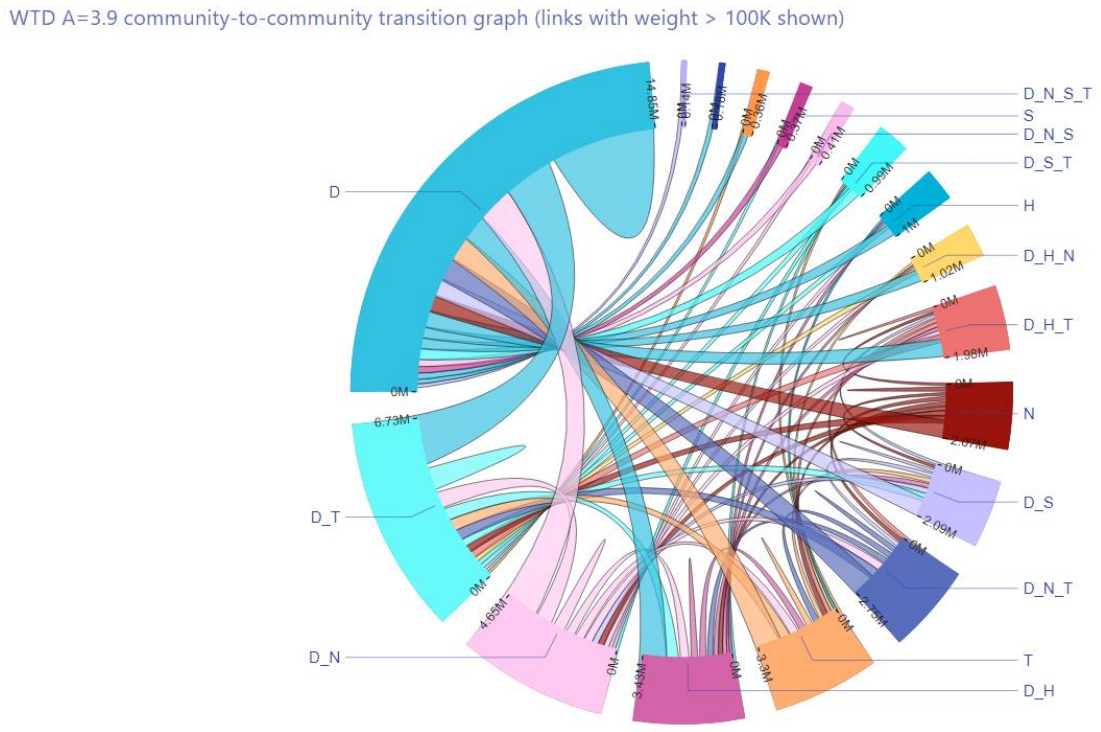


Figure 8-33: WTD A=3.9 communal knowledge flow graph

The chord diagrams for IPD are in Figure 8-34, Figure 8-35, Figure 8-36 and Figure 8-37. Here the dominant communities are 'T' (Topographic) and 'D_T'. Both have large self-loops and are also strongly inter-connected. The next one down is 'D' by itself. It is strongly connected to the other two as well. IPD allocates relatively more resources to Topographic and Domain knowledge with a bias towards Topographic.

Unlike, WTD, IPD shows some sensitivity to A as the community ordering changes from A=1.0 to A=3.1. The change is detectable towards the tail end of the chord diagram. From A=3.1 and onward there is no change in ordering.

Note that the chord diagrams are drawn using Microsoft PowerBI tool. The color scheme is based on the counter-clockwise order of the labels (communities). Charts with the same pattern will have the same scheme. This is evident upon comparison of IPD A=1 and A={3.1, 3.6, 3.9}

charts. The color scheme is a quick way of assessing if two charts have the same order of community dominance.

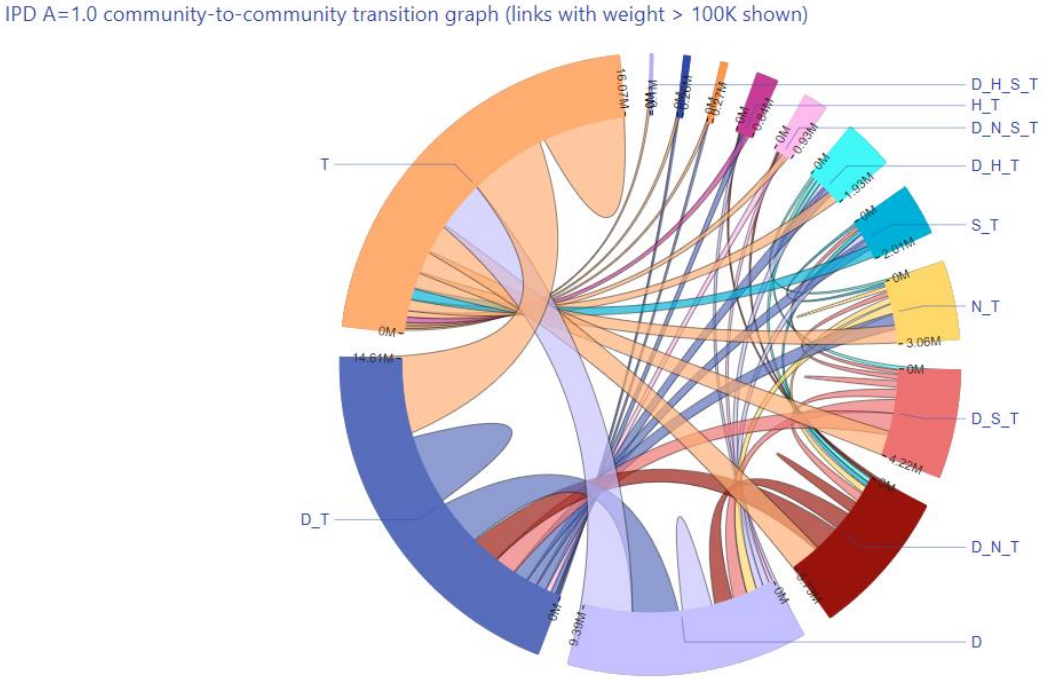


Figure 8-34: IPD 1.0 communal knowledge flow graph

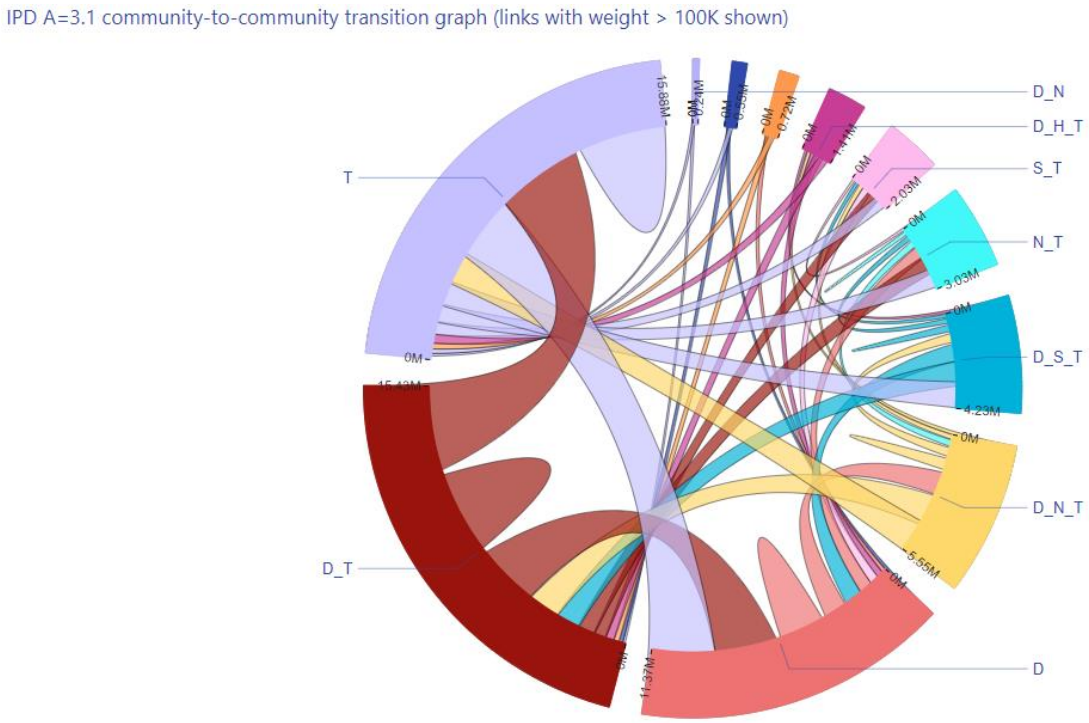


Figure 8-35: IPD A=3.1 communal knowledge flow graph

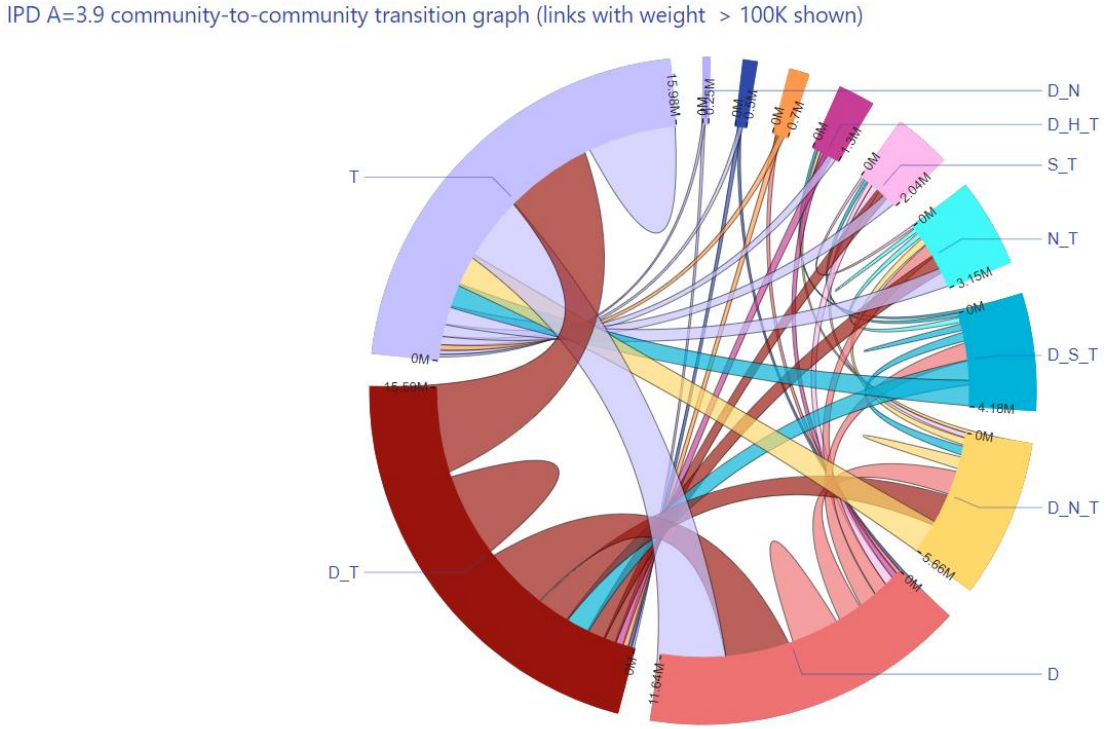


Figure 8-36: IPD A=3.9 communal knowledge flow graph

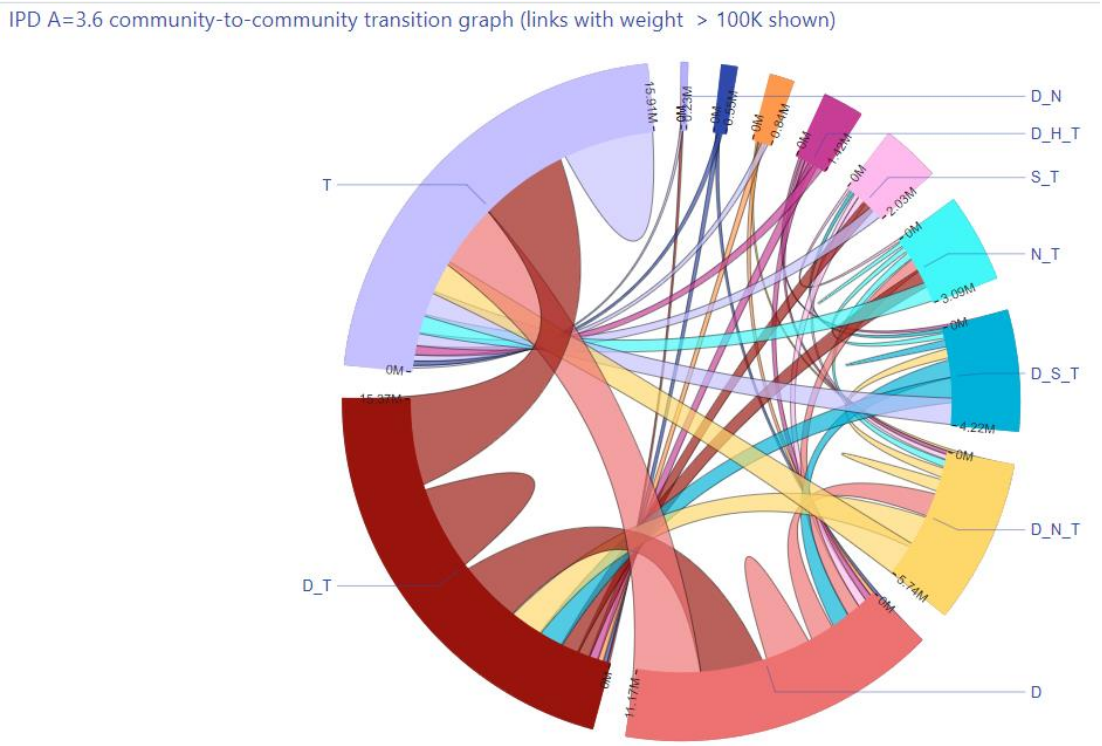


Figure 8-37: IPD A=3.6 communal knowledge flow graph

For SHS, the community-to-community transition charts are given in Figure 8-38, Figure 8-39, Figure 8-40 and Figure 8-41, for the different A values. The charts show a clear dominance Domain knowledge paired with Topographic Figure 8-41('D_T'). Unlike for WTD and IPD, where a single letter community is dominant ('D' for WTD and 'T' for IPD), a paired community is dominant for SHS.

As with IPD, SHS also shows strong inter-connections between 'D_T', 'T' and 'D' communities. However, the self-loops are slightly smaller for SHS than IPD, means that there are more transitions between communities there. Also, for SHS the links from the top 3 communities to the next level down – 'D_N_T' - are also stronger when compared with IPD. These facts indicate that SHS is mixing the Knowledge Sources at a higher rate than IPD. SHS allocates most resources to

regional exploration (Topographic) and local exploration (Domain). Considering the 'social rank' based knowledge distribution for SHS (section 6.2), the indication is that individuals are moved over long distances in the search space under Topographic influence and then (if they are performing relatively well) explore the local landscape under the Domain influence to find the local peak. This seems to suggest that SHS is balancing exploration and exploitation well.

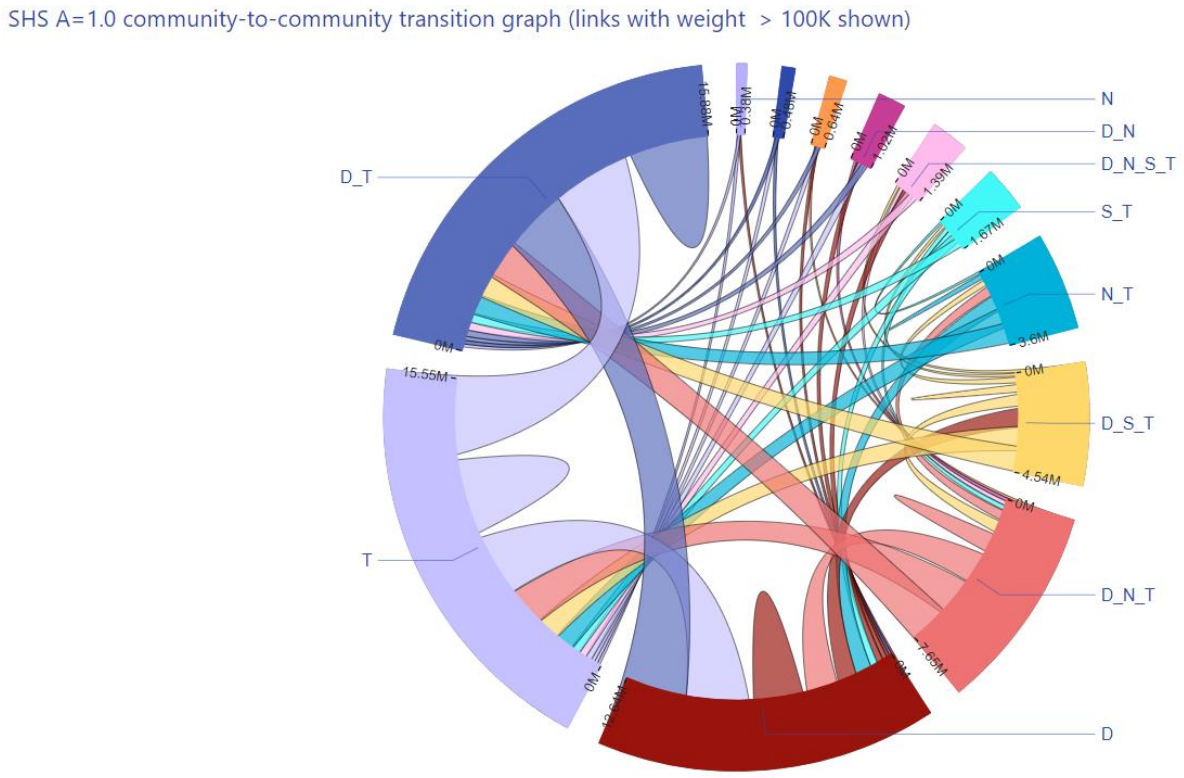


Figure 8-38: Stag-Hunt A=1 communal knowledge flow graph

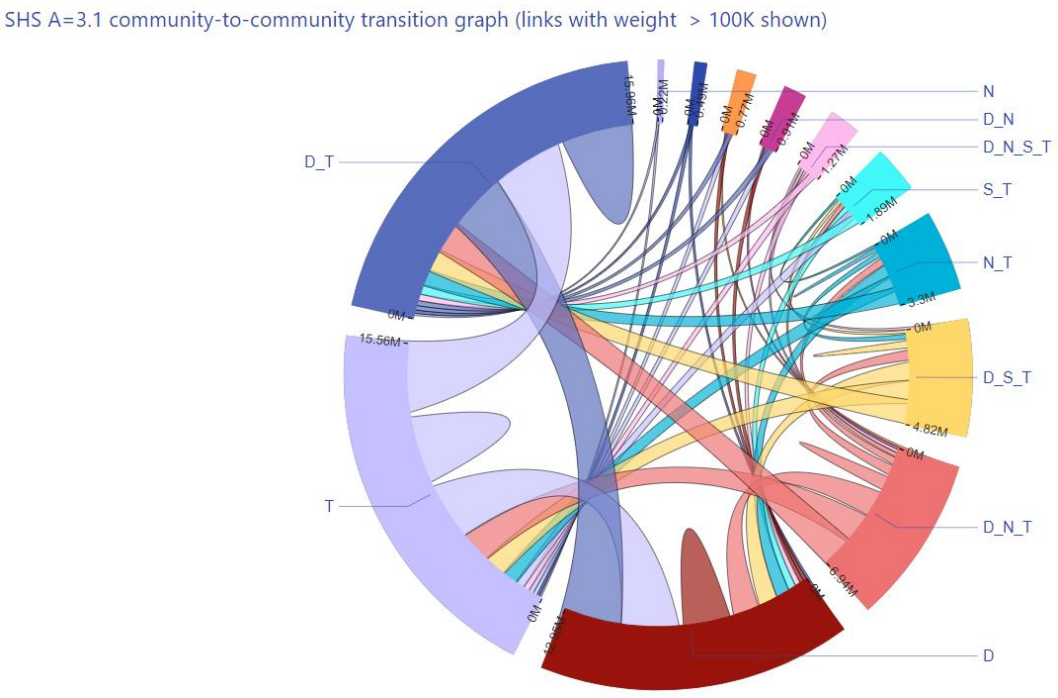


Figure 8-39: Stag-Hunt A=3.1 communal knowledge flow graph

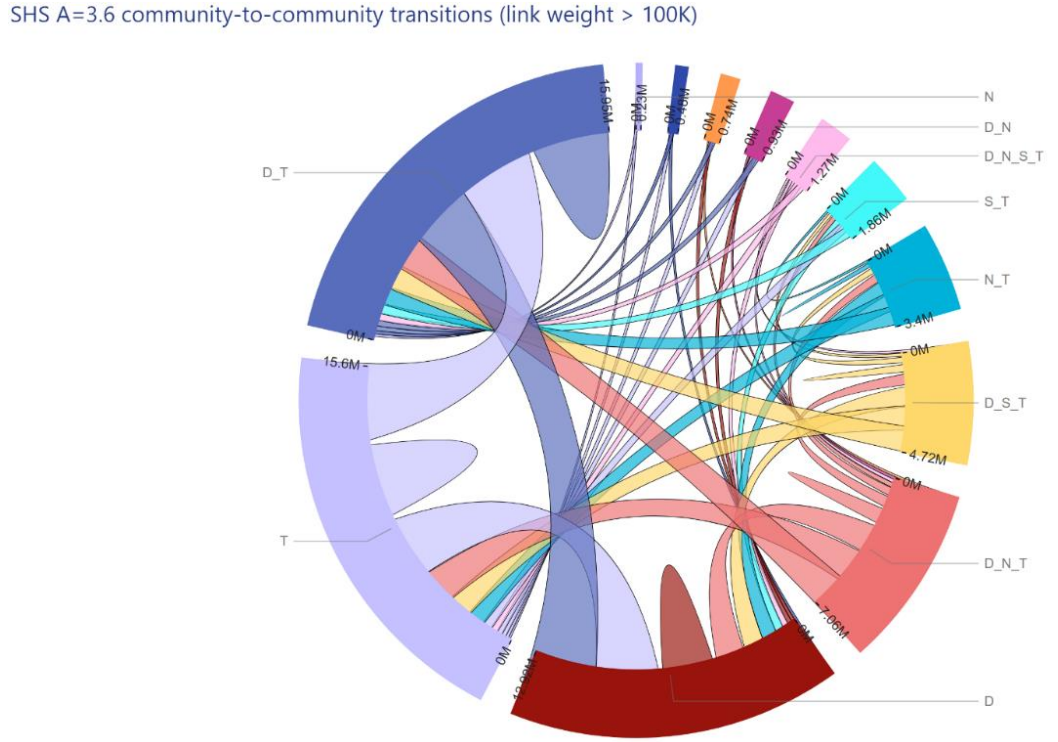


Figure 8-40: Stag-Hunt A=3.6 communal knowledge flow graph

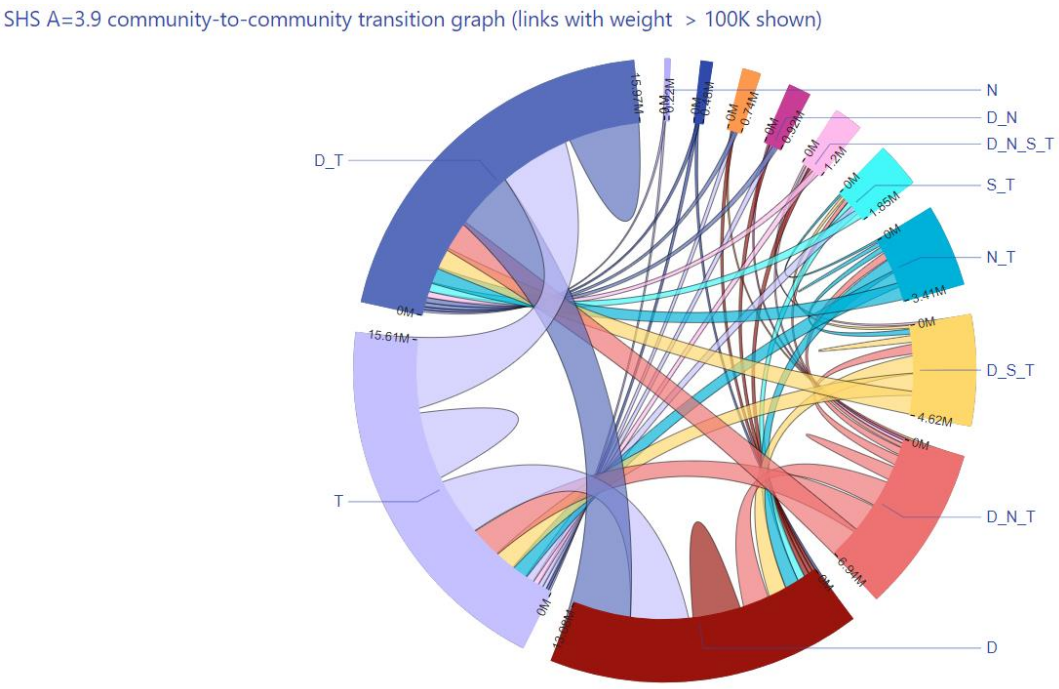


Figure 8-41: Stag-Hunt A=3.9 communal knowledge flow graph

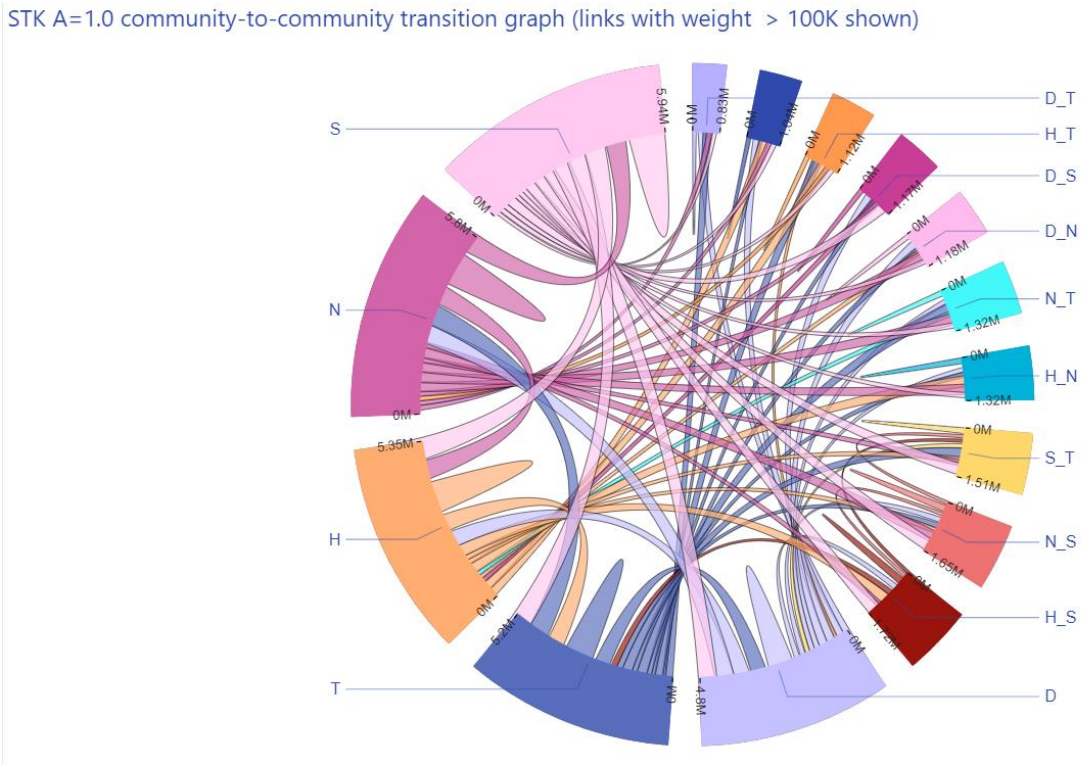


Figure 8-42: Stackelberg A=1.0 communal knowledge flow graph

The STK chord diagrams (Figure 8-42, Figure 8-43, Figure 8-44, Figure 8-45) visually are very distinct from those for WTD, IPD and SHS. Situational is now the dominant knowledge but another exploitative knowledge, History, is also ranked high. The diagrams show that STK allocates relatively more resources to exploitation when compared with WTD, IPD and SHS, respectively. Like WTD and SHS, STK also does not show any differentiation with respect to A values, in the chord diagrams.

Another aspect for STK is that the allocation of resources is more or less evenly divided between the top communities – 'S', 'N', 'H', 'T' and 'D'. The allocation among the top communities of other distribution mechanisms is more varied. For STK, the self-loops and the arcs between communities are relatively even. This indicates that rates of transitions between top communities are quite balanced.

While the STK chord charts indicate a more even allocation between Knowledge Sources (with slight priority for exploitation), this does not translate to better performance in terms of generations-to-solution. A plausible explanation is that under STK, knowledge distribution is structured like in a centrally planned system. Thus, the allocation of resources is not based on local needs and as a result the whole system is relatively inefficient when interacting with a dynamic environment. IPD and SHS by contrast make adjustments by utilizing local knowledge and are more akin to market-based systems and therefore provide more efficient utilization of resources.

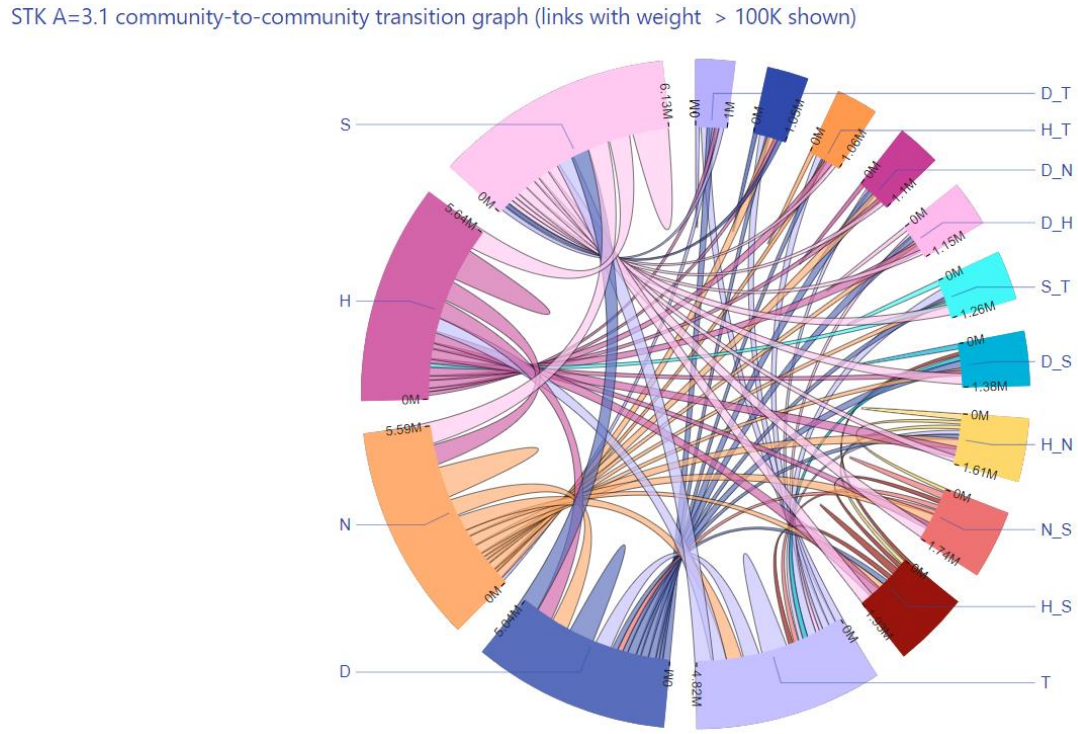


Figure 8-43: Stackelberg A=3.1 communal knowledge flow graph

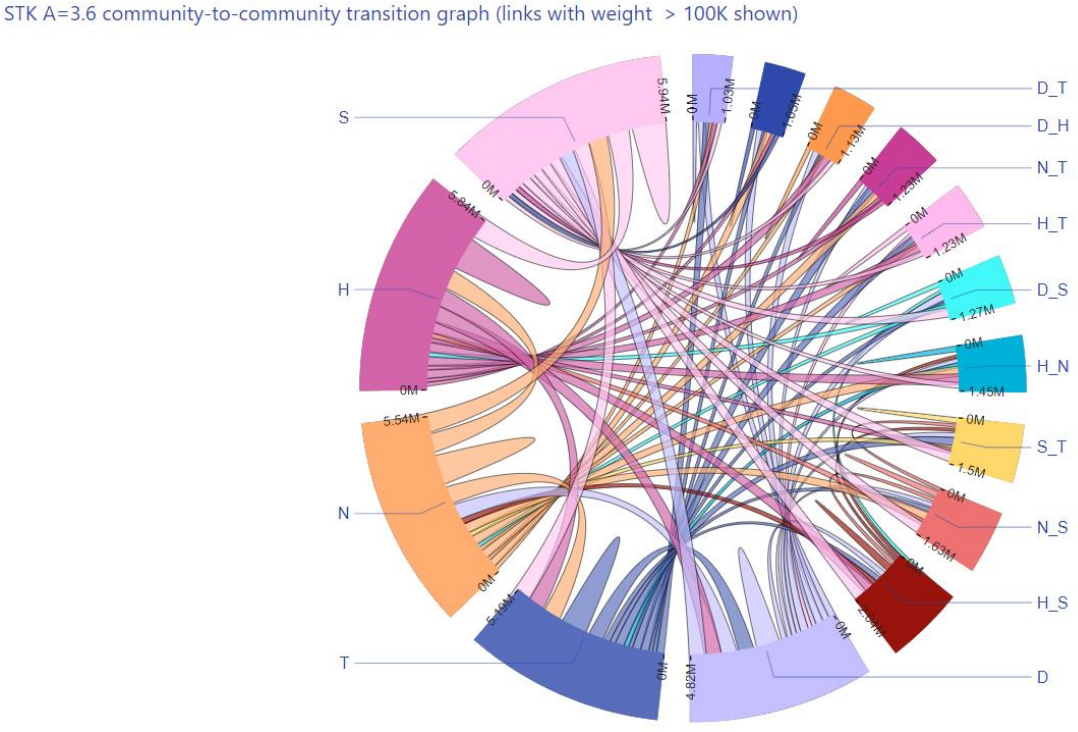


Figure 8-44: Stackelberg A=3.6 communal knowledge flow graph

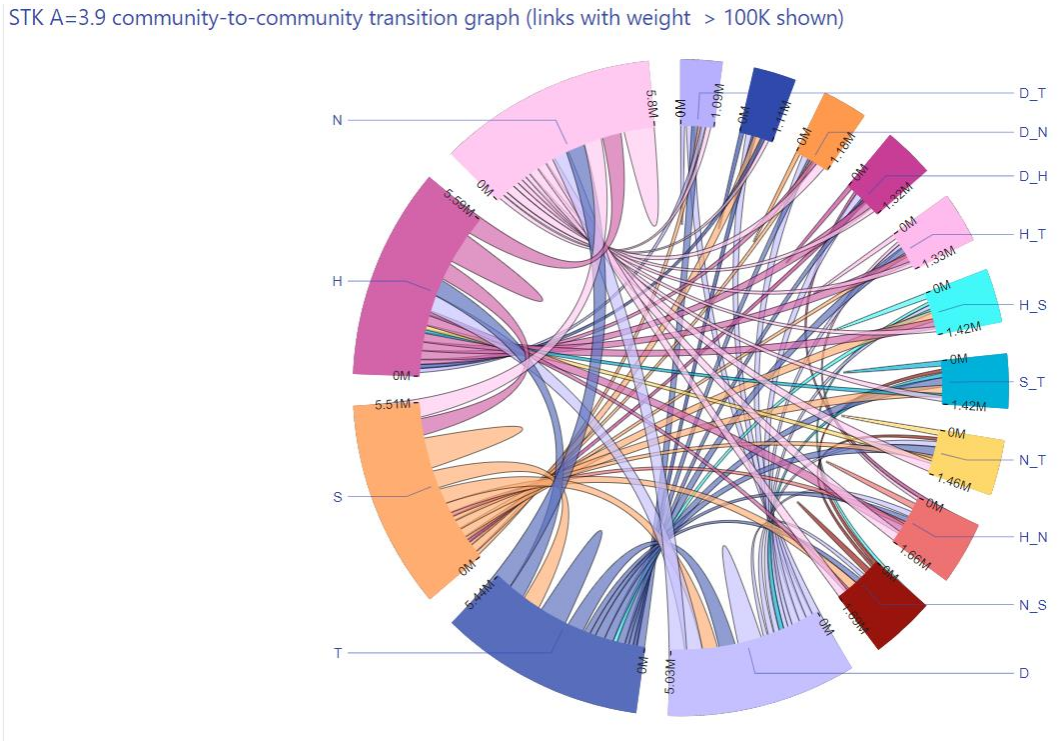


Figure 8-45: Stackelberg 3.9 communal knowledge flow graph

The chord diagrams provide a good sense of community dominance and interaction among communities (arc links). Page Rank, however, provides a more robust answer to community importance questions. The iterative Page Rank algorithm settles the weight or importance of each community with convergence and so is more definitive in this regard. Page Rank determined community weights can be visualized as tree charts – that depict relative mass – and as parallel chord charts – that, combined with A values, provide a clear view of the changes in community rank by A. Chord charts capture the dynamics more explicitly; by contrast, Page Rank output is a static view that implicitly incorporates dynamics.

As explained in section 7.5, the data for Page Rank is derived from the data used by the chord charts. The chord charts are based on raw transition counts. These are normalized such that the weighted graph becomes a Markov Chain – i.e. weights of all outgoing edges of a node sum to 1.

Also as noted earlier (section 7.5) Red hues in the tree charts indicate relatively exploratory communities and Green relatively exploitative. The boxes in a tree chart are ordered by weight, left to right.

Comparing the chord charts with Page Rank derived charts, it is evident that Page Rank is a more sensitive mechanism for ranking communities. As an example, consider the chord charts for WTD (Figure 8-30, Figure 8-31, Figure 8-32, Figure 8-33) with the corresponding tree charts in Figure 8-46 and parallel chord chart in Figure 8-47. The Page Rank based charts capture community rank changes that are not evident in the chord charts. The parallel chord chart shows the changes in rank only without considering the relative weight or importance of each community – which is captured by the tree chart. Combined, the two charts provide a useful view of the inner workings of distribution mechanisms in terms of allocation of compute resources and sensitivity to A.

For WTD, the tree charts in Figure 8-46, show a strong dominance of Domain ('D') knowledge. Almost 25% of the mass is allocated to 'D' alone. Combined with Normative and Topographic, Domain occupies almost 45%-50% of the total allocation. The color hues show that WTD allocates about 50%-60% of the resources to explorative communities. Hues Redder than that for 'D' are considered exploratory.

WTD - Tree charts of community rank by A-value

Figure 8-46: WTD Page-Rank determined community importance by A value

Some A-driven community rank changes are discernable in WTD tree charts (Figure 8-46) but these are progressively harder to spot moving from left to right. The parallel chord diagram in Figure 8-47 shows all changes clearly.

WTD shows considerable sensitivity in terms of community rank changes between A=1 and A=3.1 i.e. transition from linear to non-linear phase. It is relatively inert between A=3.1 and A=3.6 (non-linear → highly-non-linear). And then shows sensitivity between A=3.6 and A=3.9 (highly non-linear → chaotic) but less than that for the linear → non-linear transition. Since all A value changes are accompanied by community rank changes, it is surmised that WTD is a mechanism that is responsive to changes in environmental complexity. Note that even low-weight community changes are meaningful since the total for all transitions is in the order of a billion.

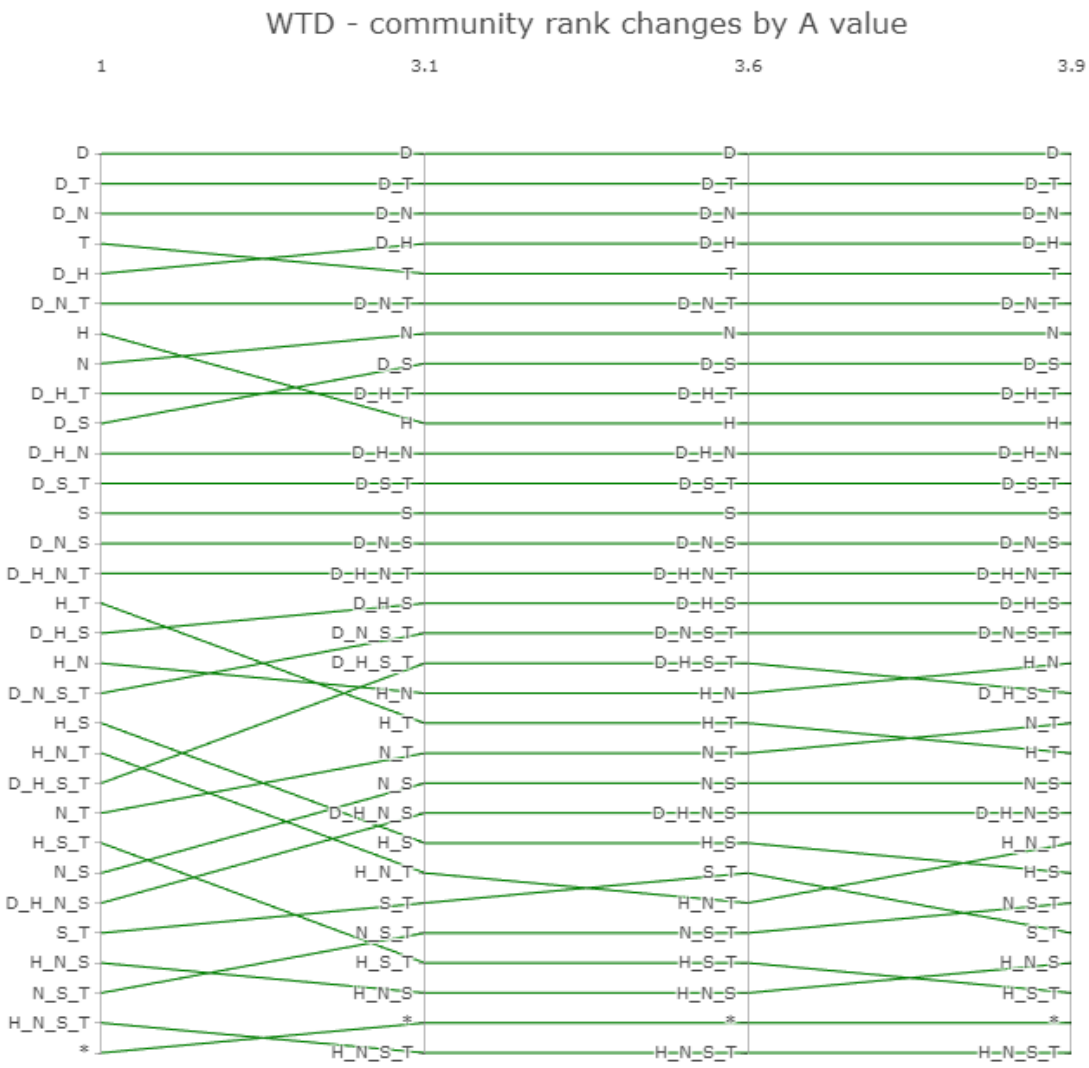


Figure 8-47: WTD changes in community rank by A value

IPD and Stag-Hunt are related since Stag-Hunt is an evolutionary game theory variant of IPD that is from classical game theory. Also, both use 'social rank' as a mechanism to distribute knowledge. IPD is biased toward competitive behavior where Stag-Hunt is more cooperative inf function. Due to their similarities, the two are considered together for this part of the analysis.

IPD - Tree charts of community rank by A-value

Figure 8-48: IPD Page-Rank determined community importance by A value

SHS - Tree charts of community rank by A-value

Figure 8-49: Stag-Hunt Page-Rank determined community importance by A value

The tree charts for IPD and SHS are in Figure 8-48 and Figure 8-49, respectively; the parallel chords charts are in Figure 8-50 and Figure 8-51. Domain and Topographic knowledge are dominant in both, with IPD allocating slightly more to Topographic. As noted earlier, SHS provides more mixing of knowledge due to having smaller self-loops than IPD, noticeable in the corresponding chord charts (Figure 8-34, Figure 8-38) – even though overall allocation is similar.

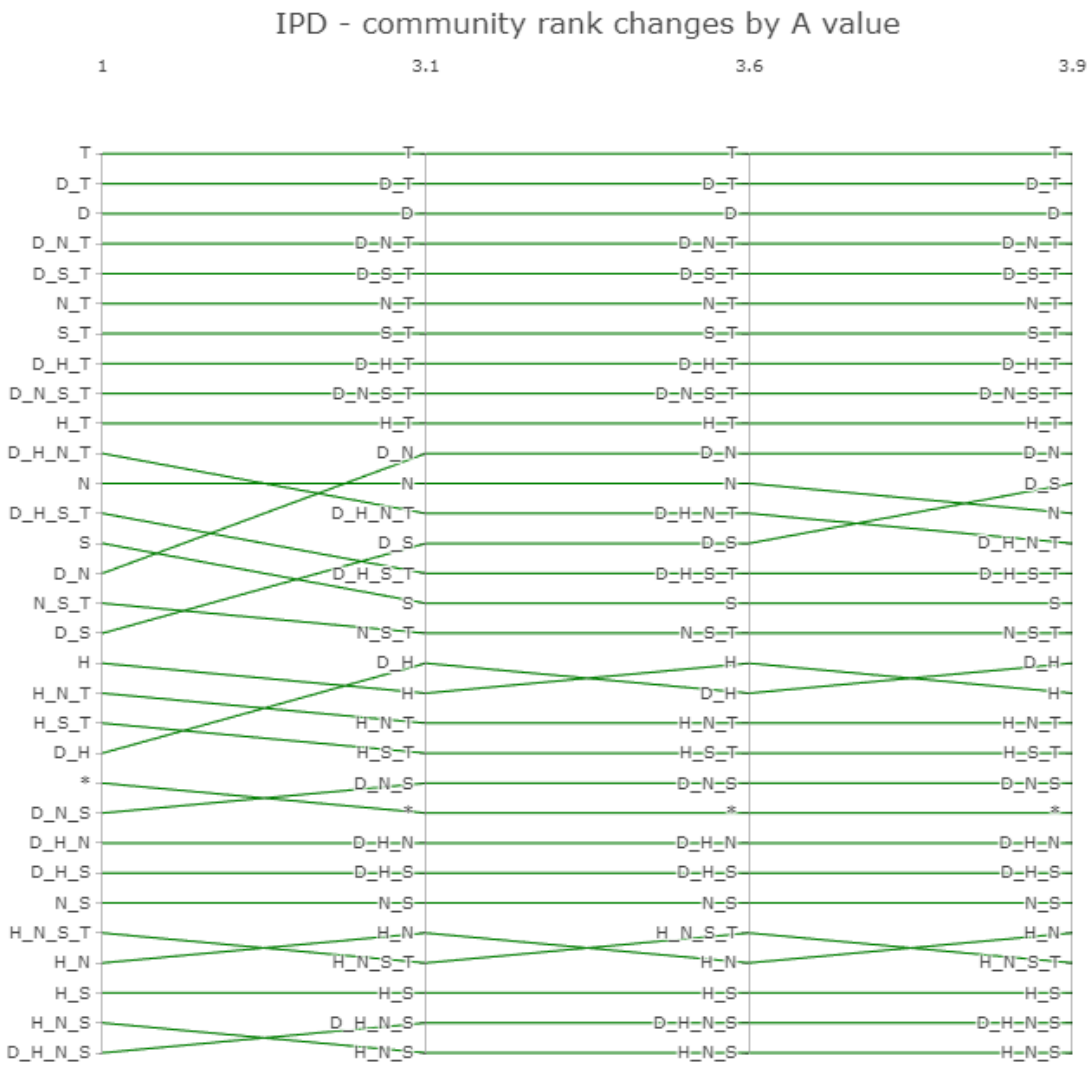


Figure 8-50: IPD changes in community rank by A value

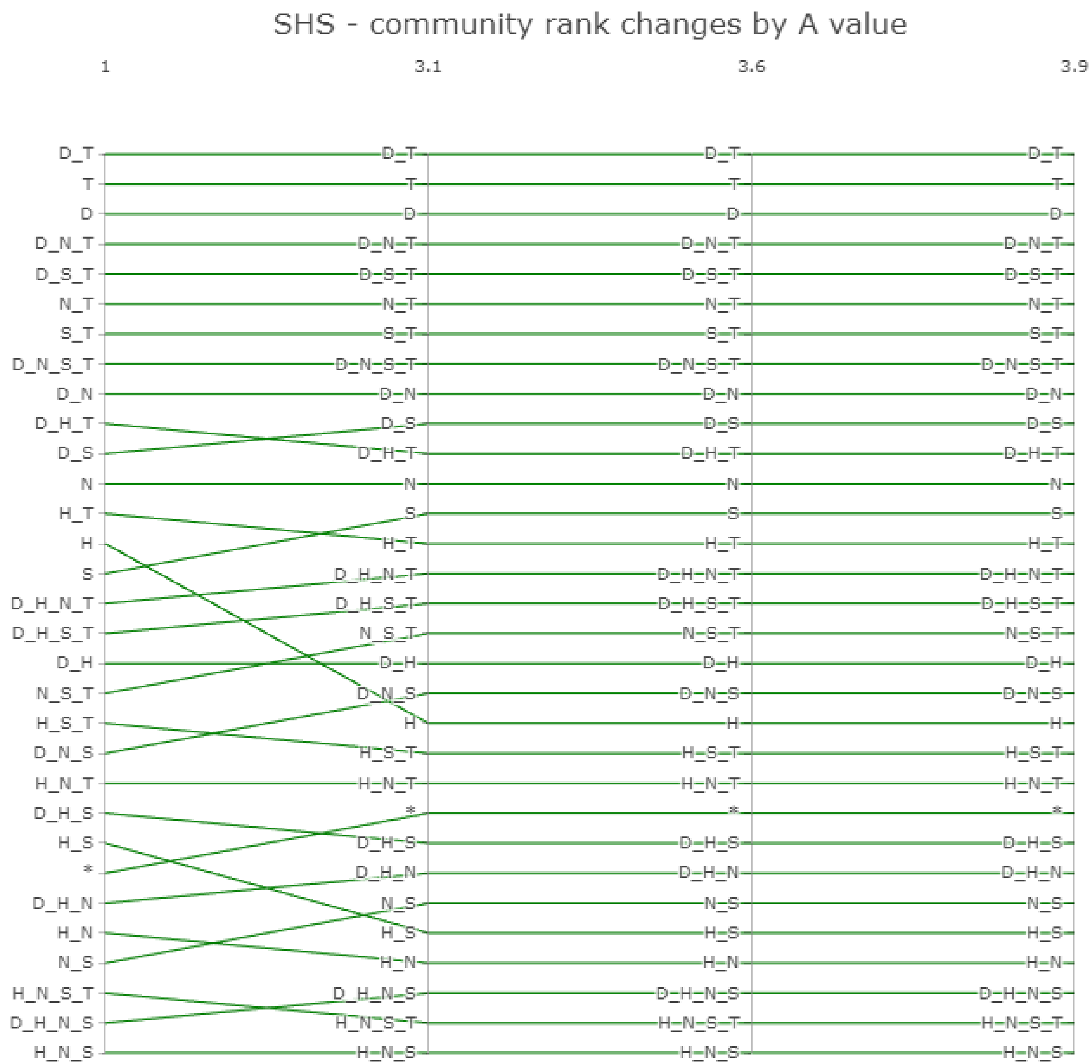


Figure 8-51: Stag-Hunt changes in community rank by A value

When compared with WTD (Figure 8-46), both IPD (Figure 8-48) and SHS (Figure 8-49) allocate more resources to exploration. IPD and SHS show 60%-70% allocation to exploratory communities where WTD is in the 50%-60% range.

From the community rank perspective, IPD shows greater sensitivity to increases in A (Figure 8-50) than SHS (Figure 8-51). SHS shows changes in community rank for A=1 to A=3.1 (linear → non-linear) transition but thereafter it is not responsive; no rank changes are present for other A transitions. How can it be that SHS performs well in terms of generations-to-solution but is relatively inert to A? The answer is that rank changes are but one view into responsiveness to A. Another method of gauging responsiveness is through Sankey chart analysis, which is discussed later in this section. SHS ***is*** sensitive to A but not enough to affect community rankings at higher levels of A.

For both IPD and SHS, the ranks of the top communities remain stable as A increases (Figure 8-50, Figure 8-50). This indicates that IPD and SHS have found stable allocations for top communities that work well for the tested levels of environmental dynamic complexity.

Figure 8-52: Stackelberg Page-Rank determined community importance by A value

As seen earlier, STK (Figure 8-52) knowledge allocations are quite different from the rest. Here the single letter communities are dominant and the allocations are relatively even between them. The changes in community ranking due to A are quite discernable even with the tree chart view.

By contrast to WTD, IPD and SHS, STK allocates more resources to exploitation. The tree chart (Figure 8-52) for STK shows a roughly even split between explorative and exploitative communities. And for all but A=3.9, the exploitative Situational knowledge is the dominant community.

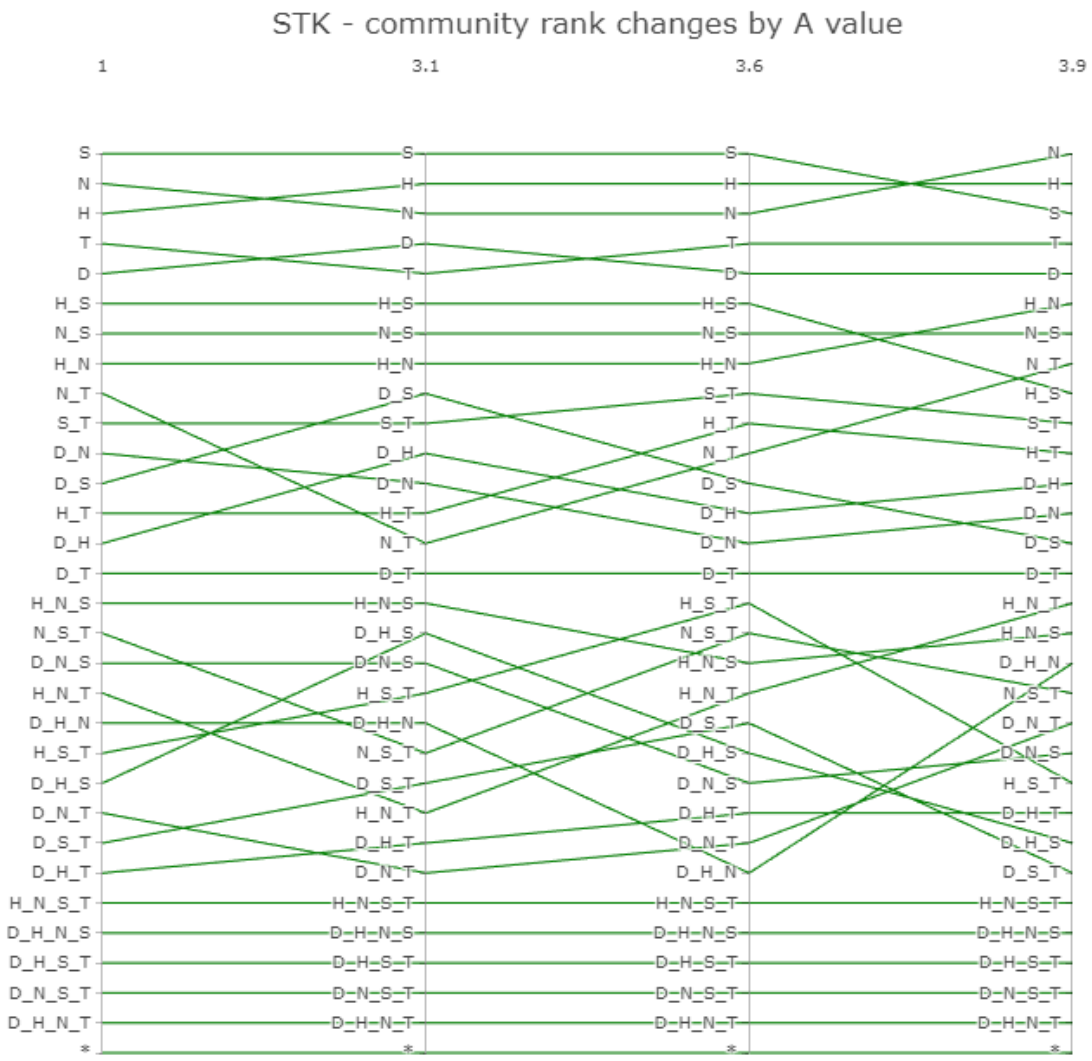


Figure 8-53: Stackelberg changes in community rank by A-value

The parallel chord chart (Figure 8-53) shows that STK is very sensitive to A as is evident by the significant number of rank changes across the board. While STK seems to be responsive to environmental dynamic complexity, its diminutive generations-to-solutions performance indicates that STK's responsiveness is not supporting its performance. The higher performing IPD

and SHS mechanisms show stable ranks, at least for the top communities. It can thus be concluded that STK is overly responsive and is not able to find a stable footing to tackle dynamic environments.

Community rank order changes is one gauge of responsiveness to A increases. However, as was discovered for SHS, it is not a sensitive enough gauge. SHS seems inert to A at higher levels – as per the parallel chords diagram for SHS (Figure 8-51). SHS performs the best in terms generations-to-solution and hence a more sensitive mechanism to track responsiveness is required. Going back to the raw transition counts collected for the chord diagrams, the counts are transformed in the following ways (see section 7.5 for details):

- The transitions counts are grouped into Explorative, Neutral and Exploitative 'categories' (for each A-KD combination) using the explorative index for each community (see Listing 7-1). Three categories are easier to reason with than 32 communities, for sensitivity analysis
- Sample mean and standard deviation (over the 200 samples per A-KD combination) are calculated to enable statistical significance testing. For each A-KD combination there are 9 means (and associated standard deviations) as follows:
    - Explorative → Explorative
        - I.e. count of transitions from Explorative communities back to Explorative communities in the next generation
    - Explorative → Neutral
    - Explorative → Exploitative

- Neutral → Neutral
- Neutral → Explorative.
- …

- Statistically significant deltas or change in transition counts are calculated with respect to each A increment
  - Consider the transition Explorative→Neutral, for A=1 → A=3.1
  - Are the means for Explorative→Neutral transition significantly different from under A=1 and A=3.1?
  - This is answerable by performing a two-sample t-test (for the difference in means)
  - For each A step (e.g. A=1 → A=3.1; A=3.1 → A=3.6; …) measure the changes in transition counts for each of the 9 transitions where the change is statistically significant
- The above transformations yield a series of weighted graphs – one for each A increment
  - The nodes are categories: Explorative, Neutral Exploitative
  - The arcs represent changes in transition counts (or net change in flow)
  - Each arc weight is the actual difference in counts for adjacent A values, if the change was statistically significant, otherwise its zero

The 3 graphs for the 3 increments (1 → 3.1; 3.1 → 3.6; 3.6 → 3.9) can be viewed in a single Sankey chart as some of the nodes are shared between the graphs. For example, the "Explorative 3.1" node in 1→3.1 is the same node for 3.1→3.6. The Sankey chart is set of chord diagrams that

are flattened out. Because the graphs share some nodes, they can be compactly viewed in the same chart.

The Sankey charts, constructed as described above, are shown in Figure 8-54, Figure 8-55, Figure 8-56 & Figure 8-57 for WTD, IPD, SHS & STK, respectively. These are all considered together as they are useful gauges for comparing responsiveness to A changes, for the tested distribution mechanisms. Statistically significant changes to increased entropy are represented by colored arcs. Red arcs represent decrease in transition counts and Green increase. For not statistically significant changes the corresponding arcs are drawn as thin Black lines. The magnitude of the change (either positive or negative) is represented by an arc's thickness.

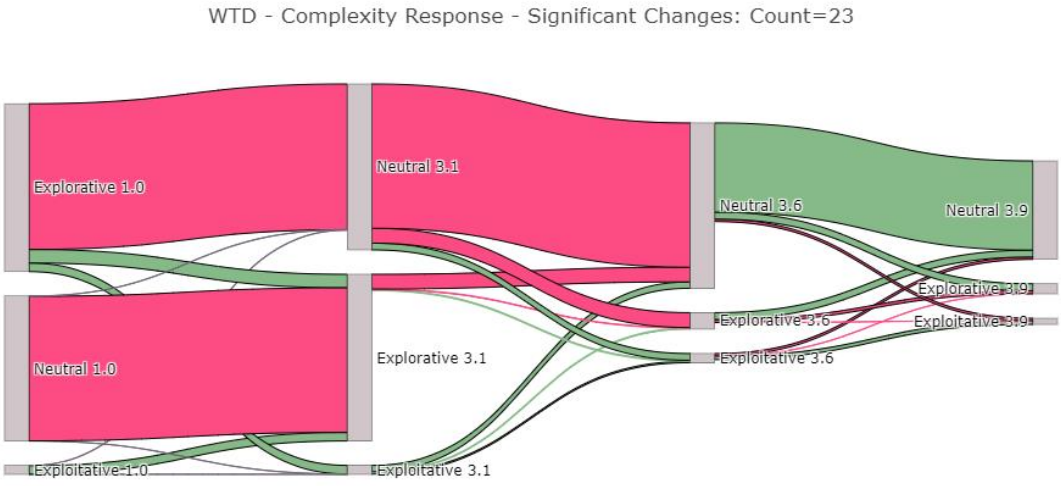


Figure 8-54: WTD - changes in explorative-exploitative balance due to complexity changes

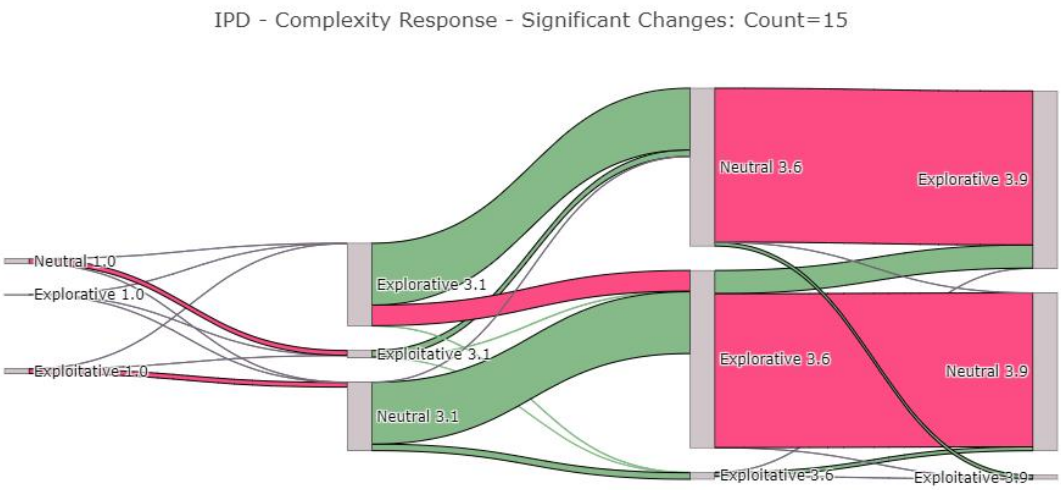


Figure 8-55: IPD - changes in explorative-exploitative balance due to complexity changes

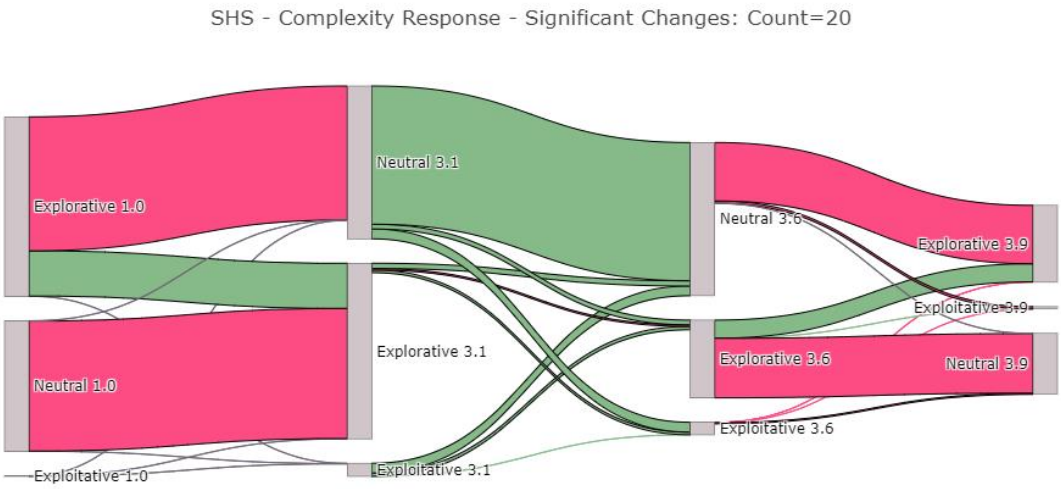


Figure 8-56: Stag-Hunt - changes in explorative-exploitative balance due to complexity changes

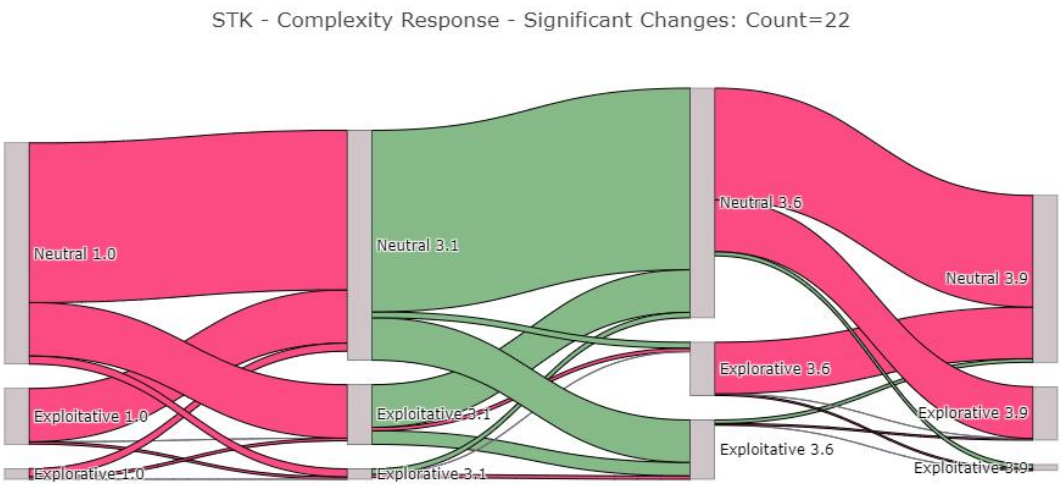


Figure 8-57: Stackelberg - changes in explorative-exploitative balance due to complexity changes

The title of each chart shows the total number of statistically significant changes made as A increases. Each Sankey chart is the profile of responsiveness to A for the corresponding distribution mechanism. The charts show that all tested mechanisms are responsive to changes in environment dynamic complexity. Somewhat surprisingly WTD is the most responsive with 23 statistically significant changes followed by STK (22), SHS (20) and IPD (15). Sankey chart for SHS (Figure 8-56) shows that SHS is responsive to all A increments; this is not picked up by community rank changes in Figure 8-51.

There is a responsiveness story embedded in the deltas of the knowledge flows as depicted by the Sankey charts but it's hard to extract that as such. One reason is that the charts are not comparable with each other as each chart is scaled relative to itself. Squeezing the information contained in the Sankey diagrams a little further provides additional insights.

The Sankey charts show changes in transition rates with respect to A increments, between all combinations of the 3 categories (Explorative, Neutral, Exploitative). Projecting the net changes (in and out) for each category onto a separate view allows one to compare the responsiveness behavior of the different mechanisms on an equal footing (see section 7.5 for additional details).

The net flow changes for Explorative, Exploitative and Neutral categories are provided in Figure 8-58, Figure 8-59 and Figure 8-60, respectively. Compare the net change in transition counts for Explorative communities (Figure 8-58) across the A increments and for the tested distribution mechanisms. SHS at 1→3.1 exhibits the largest influx. This shows that when environmental complexity increases from static to linear, SHS responds by diverting the most resources to exploration. Also, for other increments, (3.1→3.6 and 3.6→3.9) SHS is consistent in further increasing allocation to exploration. All of the other mechanisms are not consistent in that they do not consistently increase allocation to exploration with increasing environment dynamic complexity.

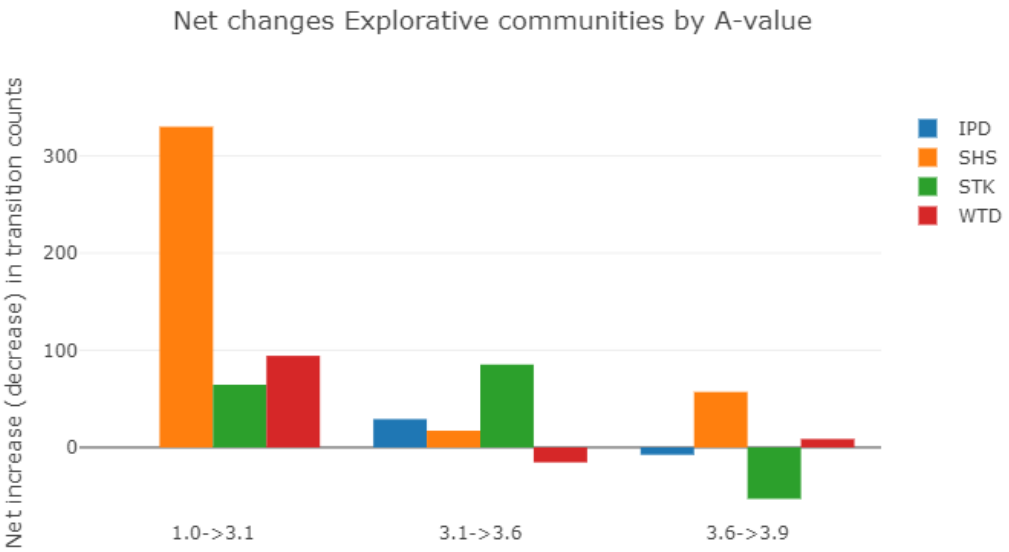


Figure 8-58: Net flow changes for Explorative communities by KD and A

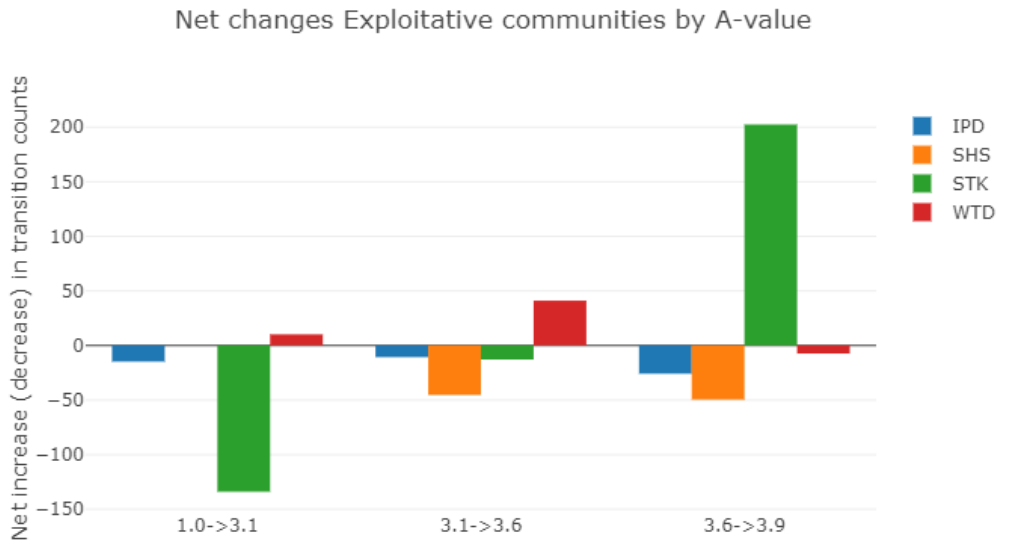


Figure 8-59: Net flow changes for Exploitative communities for KD and A

For SHS, the opposite is true in the case of Exploitation (Figure 8-59 Figure 8-58). SHS consistently diverts resources away from exploitation as A increases, as does IPD. The other distribution mechanisms do not respond consistently.

The balance of flow changes come from the Neutral category (Figure 8-60). The results are mixed for all but the chart shows that SHS diverts resources from Neutral communities for exploration for the 1→3.1 change.

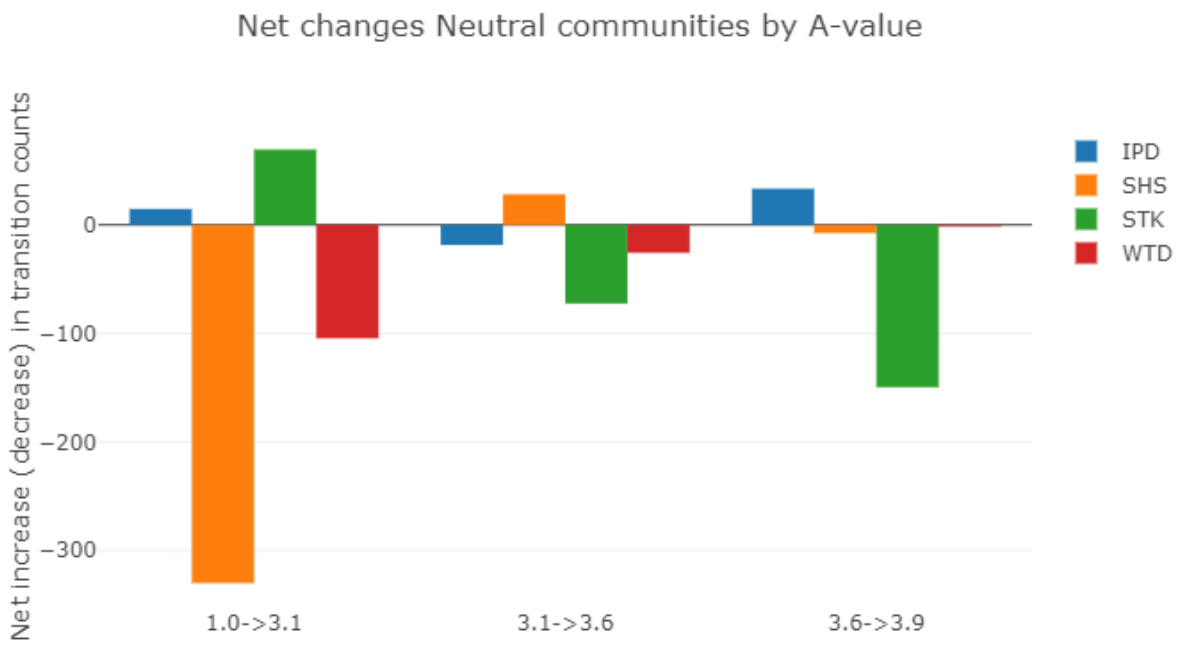


Figure 8-60: Net flow changes for Neutral communities by KD and A

This section focused on the dynamics of knowledge flow grounded on the formation of knowledge communities and community-to-community transitions of the population individuals, across generations. The goal was to shed light on the inner workings of the distribution mechanisms, in relation to their performance and with respect to changes in environmental

dynamic complexity. Through graph-based analytical methods, insights were derived on the allocation of compute resources and the responsiveness of the mechanisms to complexity. The next chapter summarizes the goals and findings of this research and draws conclusions about the hypotheses posited in Chapter 7.

8.5 Summary and Conclusions

Knowledge distribution is a key determiner of CA performance and is an active area of research in Cultural Algorithms (Al-Tirawi & Reynolds, 2018) (Reynolds & Kinnaird-Heether, 2013). The knowledge distribution mechanisms researched thus far have all been competitive mechanisms. The goal of this research is to investigate mechanisms that also span cooperation. The rich field of Game theory is used as the source and inspiration for new distribution mechanisms. Three new game-based distribution mechanisms are devised and tested namely, Iterated Prisoner's Dilemma, Stag-Hunt and Stackelberg.

A new CA software system, CATGame, is constructed for this purpose. The system supports a generic mechanism to inject and use arbitrary games for knowledge distribution. The game distribution mechanisms leverage this framework. The new mechanisms are benchmarked against the default CA mechanism Weighted Majority. CATGame is meant to solve numerical optimization problems. A separate system, CATNeuro, is also constructed to understand the effectiveness of a competitive/cooperative distribution mechanism in the domain of neural architecture search. Chapter 9 and Chapter 10 are dedicated to CATNeuro software system.

Here the performance of the distribution mechanisms in CATGame are tested via a modified Cones World test benchmark that incorporates the logistic equation (Eq **7**-**2**) to create dynamic problem landscapes whose complexity is controlled by the A multiplier of the logistic equation.

The design of experiment and research hypotheses are documented in Chapter 7. Analysis of the data collected from experimental runs is provided in the prior sections of this chapter.

Briefly, section 8.1 analyzes the base performance of the tested mechanism in terms of generations-to-solution. Section 8.2 contrasts the Diffusion metric for the distribution mechanisms. And, 8.3 comprehends the segregation of the population under the different mechanisms and in response to environmental dynamic complexity. Finally, 8.4 looks at the dynamics of knowledge flow in the 'social' network under the various mechanisms to uncover the resource allocation patterns in response to the rate of environmental change.

Each of the hypotheses posed in Chapter 7 are now taken up. Each of these are discussed next and inferences are drawn about whether these holds and what are the caveats, if any.

Hy 7-1	***"Cooperative knowledge distribution is effective for problem solving in dynamically complex environments"***	***{holds with exceptions}***

The performance characteristics of the distribution mechanisms (Table 8-1, Table 8-2, Table 8-3 and Table 8-4) clearly indicate that at least two of the mechanisms – that are inclusive of cooperation – perform well when compared with the default competitive mechanism WTD. Stag-Hunt supports the most cooperation and also performs the best. However, Stackelberg (as interpreted for knowledge distribution) performs well for static environments but cannot keep up with others in dynamic environments. Stackelberg employs a structured model of cooperation that does not incorporate local knowledge and consequently is less efficient in resource allocation. Comparatively speaking, Stackelberg allocates more resources to exploitation (Figure 8-52) than others. By contrast, knowledge flow analysis shows that the best performer

allocates progressively more resources to exploration as complexity increases (Figure 8-58). Thus, the hypothesis holds for the type of cooperation that incorporates local knowledge into decision making. If the decision making is centralized or oblivious of local conditions, it does not seem to hold.

Hy 7-2 ***"Cooperative knowledge distribution exhibits better robustness than competitive distribution"*** ***{holds with exceptions}***

Robustness is about how quickly a system adjusts to change. The basic performance charts (Figure 8-1, Figure 8-2, Figure 8-3, Figure 8-4) show that two of the 3 cooperative mechanisms (IPD and Stag-Hunt) are robust as they settle to a base rate of performance relatively quickly in the face of periodic environment change. WTD, the competitive mechanism, take longer to settle as A values increase. For mild complexity (A=1) WTD is on par with the best after 10 landscapes. However, as complexity increases, WTD takes increasing longer to settle. Stackelberg is the contrarian cooperative mechanism and in fact is not robust at all as its performance becomes progressively worse, at least for the 50-landscape horizon used in the experiment. As the Page Rank derived tree charts show (Figure 8-46, Figure 8-48, Figure 8-49, Figure 8-52) Stackelberg allocates more resources to exploitation where the better performing mechanisms devote more to exploration. Once again it can be surmised that cooperative systems that incorporate local knowledge are more resilient than ones that are centrally planned. Also, competitive mechanisms are less resilient in general but still more than those with oligopolistic cooperation.

Hy 7-3 ***"Diffusion is higher for more dynamically complex environments"*** ***{does not hold}***

The Diffusion trend charts for each KD (Figure 8-9, Figure 8-10, Figure 8-11, Figure 8-12) show that the only mechanisms that shows some sensitivity to A is WTD. The Diffusion for WTD is distinctly lower for A=1; for non-linear A values is no discernable distinction. In general, this hypothesis does not hold but there are exceptions.

Hy **7-4** ***"Segregation is higher for higher complexity environments"*** ***{holds with exceptions}***

The segregation trend-by-landscape charts for each KD (Figure 8-17, Figure 8-18, Figure 8-19, Figure 8-21) show that IPD exhibits some separation in segregation with respect to A whereas SHS, WTD and STK do not. However, at the aggregate level, Figure 8-29, both IPD and SHS show a distinct response to increasing A with increasing segregation levels. It seems that higher performing mechanisms tend to increase segregation in response to complexity. Consequently, this hypothesis partially holds. It holds for 'social rank' based distribution mechanisms (IPD, Stag-Hunt) and not for the oligopolistic one.

Hy **7-5** ***"Stag-Hunt and IPD distributions will produce higher segregation than WTD distributions"*** ***{holds}***

This hypotheses holds and clear evidence exists in Segregation trend charts Figure 8-13, Figure 8-14, Figure 8-15, and Figure 8-16. The Segregation levels for IPD and Stag-Hunt remain consistently higher than those for WTD. The reason for this is that in WTD the Knowledge Sources that are forced out due to the voting mechanisms, are added back to 20% of the randomly selected population that lowers overall Segregation. This is done to protect against the "tyranny of the majority" (Moeckli, 2018). The US electoral college system of voting is a similar measure. Without this

provision, certain types of knowledge could be lost at some point and not regained. The cooperative mechanisms on the other hand are self-adjusting and don't need a similar provision.

Hy 7-6 ***"Stackelberg produces lowest segregation among the mechanisms tested"*** ***{does not hold}***

Given that Stackelberg uses a structured model of cooperation, it was expected that it would maintain low segregation levels as all Knowledge Sources get an equal share, albeit in the order of their relative strength (section 6.3). However, what emerges is a picture of increasing Segregation over the sequence of 50 landscapes (Figure 8-26). For all complexity values, Stackelberg does start with low Segregation levels (at 0.45) but these continue to rise and are seen to reach 0.8 at the end of the landscape sequence. Considering, that Stackelberg does not perform well in dynamic environments (Figure 8-1, Figure 8-2, Figure 8-3, Figure 8-4) its continually changing Segregation levels suggest that it does not find an equilibrium state of knowledge levels, at least within the test limit of 50 landscapes. Thus, it can be concluded that this hypothesis does not hold.

Hy 7-7 ***"The community transition weighted graphs for the tested knowledge distribution mechanisms are visibly distinguishable from each other"*** ***{holds}***

The community-to-community transitions graphs for the different distribution mechanisms are: ***WTD*** Figure 8-30, Figure 8-31, Figure 8-32, & Figure 8-33; ***IPD*** Figure 8-34, Figure 8-35, Figure 8-36 & Figure 8-37; ***Stag-Hun***t Figure 8-38, Figure 8-39, Figure 8-40 & Figure 8-41; and ***Stackelberg*** Figure 8-42, Figure 8-43, Figure 8-44& Figure 8-45. The chord diagrams are a reflection of the inner workings of

the corresponding mechanisms. These diagrams crystalize the dynamics of the knowledge flow for each KD-A combination into a single view. The chord diagrams show that each mechanism operates distinctly. The patterns are very similar within a mechanism across the different A values but quite distinct between the mechanisms. Consequently, it is concluded this hypothesis holds without caveats.

Hy 7-8 ***"The community transition weighted graphs for a tested knowledge distribution mechanism are appreciably different for different A values"*** ***{does not hold}***

Hy **7**-**8** is related to Hy **7**-**7** discussed above. Here the supposition was that the chord charts for the different A values for the same mechanism are also visibly distinct from each other. As mentioned in the analysis for Hy **7**-**7** above, this is not case. There are some differences between the chord diagrams of the same mechanism but in general it does not hold. The premise underlying this hypothesis was that chord diagrams would be sensitive enough to allow one to also distinguish the responsiveness of the mechanisms to environmental complexity. Given that this hypothesis does not hold, additional, more sensitive analytical methods were required to understand the responsiveness (see Figure 8-54, Figure 8-55, Figure 8-56, and Figure 8-57).

Hy **7**-**9** ***"Community importance weights for the tested distribution mechanisms are different from each other, reflecting their different operational characteristics"*** ***{holds}***

The page rank derived tree charts (Figure 8-46, Figure 8-48, Figure 8-49 and Figure 8-52) show that this is indeed the case. It is expected given that Hy 7-7 was found to hold. The Page Rank data is

derived from the underlying data used for the chord diagrams and so the two support each other.

Hy **7-10** ***"Community importance rankings for the tested distribution mechanisms vary by environmental complexity"*** ***{holds with exceptions}***

The parallel coordinates charts (Figure 8-47, Figure 8-50, Figure 8-51 and Figure 8-53) show that this hypothesis mostly holds but not in all cases. Stag-Hunt's responsiveness is not surfaced in the corresponding parallel chords chart. In the strictest sense, this hypothesis does not hold, however, since it is true for 3 out of the 4 cases, one can state it holds but with some exceptions. In fact, the lack of sensitivity of the parallel chords analysis also prompted development of additional methods to measure sensitivity.

Hy **7-11** ***"The tested distribution mechanisms are responsive to changes in environmental complexity"*** ***{holds}***

This hypothesis is a direct statement relating the mechanisms' responsiveness to A. As mentioned above, initial analytical methods were not sensitive enough to uncover the A value relationship. However, the explorative-exploitation community balance analysis clearly indicates that all mechanisms show statistically significant responses to A changes, in terms of compute resource allocations. See Figure 8-54, Figure 8-55, Figure 8-56 and Figure 8-57. This hypothesis holds without caveat.

Hy **7-12** ***"Cooperative distribution mechanisms are more responsive to changes in environmental complexity than competitive mechanism"*** ***{does not hold}***

Going by the analysis presented in Figure 8-54, Figure 8-55, Figure 8-56 and Figure 8-57, this hypothesis does not hold. The most

sensitive mechanism is the competitive mechanism WTD in terms of the number of statistically significant changes to net flow driven by changes in A.

Hy **7-13** ***"Better performing distribution mechanisms will exhibit consistent responses to changes in environmental complexity"*** ***{holds}***

This hypothesis only holds for Stag-Hunt, which is the top performing mechanism and shows consistent allocation changes in response to A as seen in Figure 8-58 and Figure 8-59. The changes are subtle but still statistically significant. While not true for all case, this hypothesis holds for the best mechanism tested and so is considered to hold for the purpose of this analysis.

In summary, CA knowledge distribution mechanisms that span cooperation as well as competition perform better than the default competitive mechanism, Weighted Majority, when faced with dynamic environments of varying complexity. However, this is not true for all such mechanisms. The research concludes that 'social rank' based cooperation (IPD and Stag-Hunt) performs significantly better overall complexity levels except non-linear (A=1). Here SHS performs significantly better but IPD does not. While structured or oligopolistic cooperation, seen in Stackelberg, does not perform better that WTD for all complexity levels.

The Stackelberg model works well for static environments but is not able to track changes in dynamic environments. The rigid or centrally planned method of cooperation does not take local knowledge into account and hence resource allocation is not optimal. Stackelberg disproportionately allocates more resources to exploitation when compared with others.

The Weighted Majority ‘wisdom of the crowd’ model is competitive in low complexity environments (i.e. A=1 / linear) but then is not able to keep up with ‘social rank’ based cooperation under IPD and Stag-Hunt, the non-linear complexities tested.

The best performing mechanism Stag-Hunt is also the most cooperative. Stag-Hunt is a variation of IPD. Stag-Hunt is from Evolutionary Game theory while IPD is from classical Game theory. Stag-Hunt is biased toward cooperation where IPD is biased towards defection.

Once it was clear that cooperation improves CA knowledge distribution, a new challenge was taken up in order to test cooperative knowledge distribution in a completely different domain from numerical optimization. The next two chapters describe the CATNeuro system that evaluates competitive and cooperative mechanisms in relation to optimization of neural network architectures.

CHAPTER 9 CATNEURO – A CA-DRIVEN FRAMEWORK FOR DEEP LEARNING

9.0 Introduction

Experimental results with the Cones World show that Stag-Hunt performs well for numeric optimization problems, especially in dynamic environments (i.e. where the optima may change over time) as compared to the baseline CA knowledge distribution mechanism, Weighted Majority. However, to better understand whether game-based, cooperative knowledge distribution is indeed a robust addition to the CA family, requires additional evidence. Thus, the CATNeuro system was constructed to test how effective cooperative knowledge distribution can be in a domain vastly different from numerical optimization. The CATNeuro system optimizes the structure and parameters of deep learning models using an implementation of Cultural Algorithms adapted for such a task. CATNeuro can be configured to use either Stag-Hunt or Weighted Majority distribution mechanism.

Contemporary deep learning models are multi-layered directed graphs (quite different from traditional multi-layer perceptrons) (Goodfellow, Bengio, & Courville, 2016). Many times, it is not clear what is the best architecture for a given problem. Researchers often spend many months trying to find the optimal architecture. CATNeuro can assist researchers in optimizing deep learning model structures by performing an intelligent search in this space. At the very least CATNeuro can out point out promising candidate architectures that researchers can investigate further.

As noted earlier, Cultural Algorithms are better suited to problem solving in complex environments (Figure 1-1) because the CA stores and uses more information than other evolutionary optimization methods. CA information overhead can be amortized better when

working with problems in complex domains. Deep learning model optimization is a very complex problem. The training time for a single model is in the order of minutes (if not hours) and therefore any CA information overhead is miniscule by comparison.

This chapter describes the CATNeuro system and the experimental framework used to evaluate the effectiveness of CATNeuro with Stag-Hunt and WTD for deep learning model optimization. Stag-Hunt was selected since it was the best performing cooperative mechanism and WTD was to baseline its performance.

The test bed problem selected is the construction of a deep learning model that can be used as a controller to play a fighting video game. Section 9.1 provides an overview of the FightingICE game system used for evaluating CATNeuro. Section 9.2 gives an overview of a neuro-evolution methodology named NEAT that provides the inspiration for the CATNeuro search mechanism. The detailed description of the CATNeuro system is provided in 9.3. Section 9.5 details the mechanism used to train the controller, which relies partly on ideas from the reinforcement learning literature. They are used to generate the training data for the neural network. The training regimen is described in 9.6. Section 9.7 describes the experimental framework used to evaluate the performance of Stag-Hunt and Weighted Majority distribution mechanisms for optimization in the space of neural network architectures. The experimental results are analyzed in Chapter 10.

9.1 ICE Competition Fighting Game

The FightingICE is a research test bed for AI maintained by Intelligent Computer Entertainment (ICE) Lab, Ritsumeikan University, Japan. ICE holds an annual competition for competing AI controllers (Fighting Game AI Competition, 2018). The game is a 2D street fighting game (Figure

9-1). Each player has 56 actions available that it can play from. The actions are a mix of offense, defense and positioning moves.

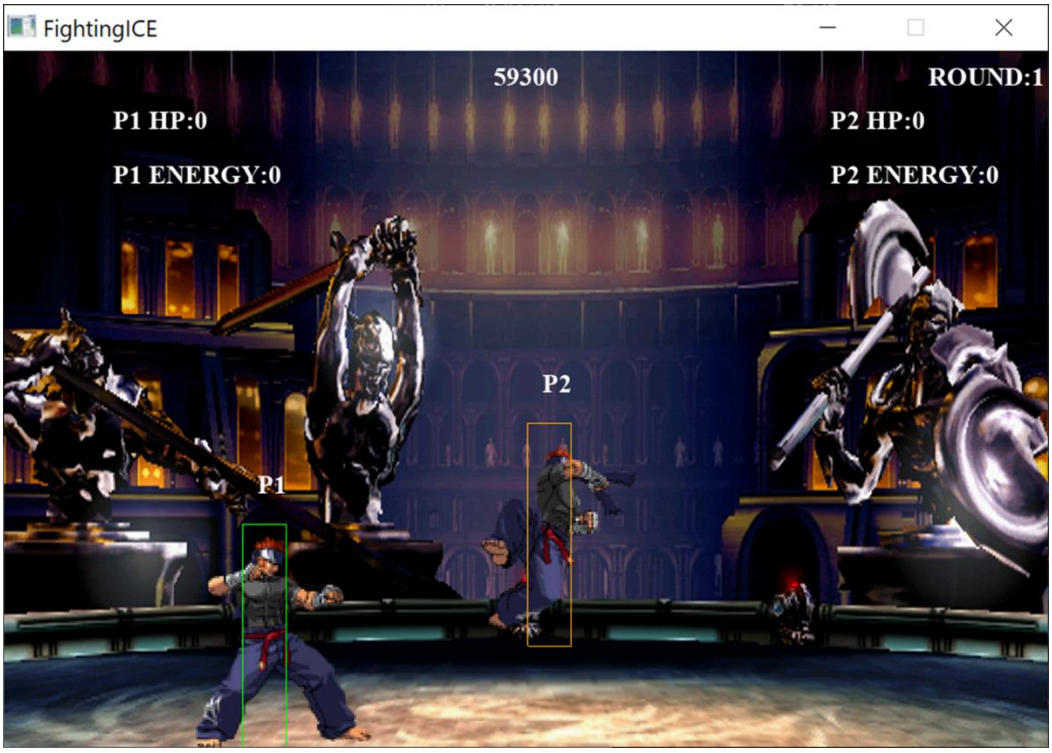


Figure 9-1: ICE competition fighting game screen capture

The game controller must supply one of the 56 actions when requested by the game framework. The game framework provides access to the game state but its delayed by 15 frames. States for both players are provided by the game framework. The AI for the controller is a function that essentially maps the game state (current and historical) to an action.

The game is fast paced so decisions have to be supplied in a timely manner. The rules of the games are somewhat complex. The hits between players transfer energy from one player to another. Stored energy can be used to throw projectiles with greater damage potential. It takes a while for human player to master the game.

The championship winning AIs are coded by human experts who understand the game well. They know how to exploit different situations in the game and make moves that are precisely timed. As per current knowledge no purely machine learned AI has been able to beat a human developed AI.

The controller framework developed to play the game accepts a deep learning model that conforms to certain specifications for input and output. The input to the model is 72x1 vector that represents current and historical game state (Table 9-2). The output is a 56x1 vector that represents a probability distribution over the 56 actions. At each frame, the controller calls the configured model with the state vector and samples from the output distribution to select the action to play.

The CA-driven controller is played against two types of opponents that are described below:

- Jerry Mizuno AI (JM) – is an AI controller supplied with the FightingICE. It is a AI developed by academics at Ritsumeikan University (which is the home of FightingICE). JM uses a combination of K Nearest Neighbors (KNN) and fuzzy logic (Chu & Thawonmas, 2017). It can be considered a benchmark AI where the fighting decisions are mostly made algorithmically.

- 2018 champion called “Thunder”. Thunder is a championship level AI that is very advanced (FightingICE 2018 Championship Results). It was developed by (presumably) an expert human player and programmer Eita Aoki who seemingly is well aware of the game rules. Thunder is fast and seems to exhibit high-level strategies (e.g. has offense and defense modes). It also exploits specific game situations like pinning the opponent in a corner with

repeated, strikes. It was not expected that the controller would be able to beat this champion. However, playing against a strong player is good for differentiating between the performances of the underlying models more precisely. As developed, the reinforcement learning based CATNeuro AI controller learns from observation. Specific game rules were not encoded into the controller (i.e. the controller is not model based). In general, it is difficult for a purely AI-driven approach to infer specific game rules given limited computed resources.

The CATNeuro system is configured to evolve models that conform to the input/output requirements of the controller framework. The system is free to structure the model however as long the input/output constraints are not violated. Also, the graph sizes of the various populations (Blueprint and Modules) are limited so that overly large models are not produced through the process of graph evolution.

To evolve the deep learning models, training data is required. The training data should conform to the input/output specification – i.e. input should be a 72x1 vector and output a 56x1 vector. The construction of the training data is an involved process that takes a day or two to complete and is comprised of several steps. The next section provides a brief overview of Reinforcement Learning that is the basis for the creation of the training data for the controller models. A more detailed description of the process to acquire the training data in provided in section 9.5.

9.2 Neuro Evolution of Augmented Topologies (NEAT)

The NEAT methodology was developed by Stanley and Miikkulainen and is described in detail in the NEAT paper (Stanley & Miikkulainen, 2002). NEAT is a population-based methodology where each individual is a directed graph. The population is evolved via operations on directed

graphs. These are described later in this section. The core idea of NEAT is to start simple and progressively ‘complexify’ the graphs by adding nodes and connections.

The CATNeuro adaption of NEAT applies the same operations as defined by NEAT but there are many differences between the two methodologies. Firstly, NEAT does not have a social network that binds the population and thus has no notion of knowledge distribution or Belief Space, etc. Secondly, the graph operations are applied randomly in NEAT where the operations are organized under the five Knowledge Sources under CA and applied through the workings of the knowledge distribution mechanism (see Table 2-1). Thirdly, to manage complex graph structures CATNeuro also relies on the topological sort of the graphs whereas NEAT only uses a simpler mechanism based on innovation numbers (Stanley, Bryant, & Miikkulainen, 2005). Fourthly, the distance metric used to gauge similarity of any two graphs is materially different between CATNeuro and NEAT; CATNeuro defines a graph distance metric that is finer grained than the innovation number-based method used in NEAT.

9.2.1 Graph Operations under NEAT

The basic graph operations under NEAT are:

- Toggle Connection
- Add Connection
- Add Node
- Crossover
- Mutate Parameter

Each graph connection contains a Boolean switch that can be switched On or Off through mutation. The Toggle Connection operation is demonstrated in Figure 9-2. When the switch is off, the connection is dropped when the graph is translated into a deep learning model (described in the next section).

Figure 9-2 (and other related figures in this section) show before and after-operation views of graphs. The changes are highlighted in Yellow. The number associated with each connection is the innovation number (Stanley, Bryant, & Miikkulainen, 2005). A counter is maintained that is incremented whenever a new connection is added. The innovation number can be used to determine the order of connections; useful for the crossover operation (described later) among others.

Another counter is maintained for internal nodes (i.e. not input or output). This counter is incremented whenever a new node is added. The node numbers are taken from this counter. This is useful for knowing the order of nodes and is used to prevent cycles in the graph (among other uses).

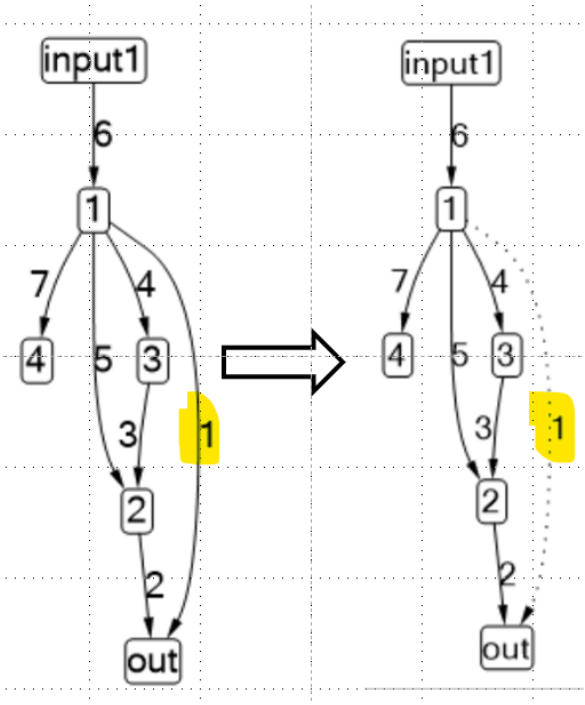


Figure 9-2: Toggle connection operation - connection #1 switched off

The Add Connection operation is shown in Figure 9-3. Under this operation the graph is mutated by adding a new connection between two previously unconnected nodes. In CATNeuro, topological sort is performed in order to ensure that cycles are not introduced when adding a new connection. Connections are only added from nodes earlier in the sort to those that come later.

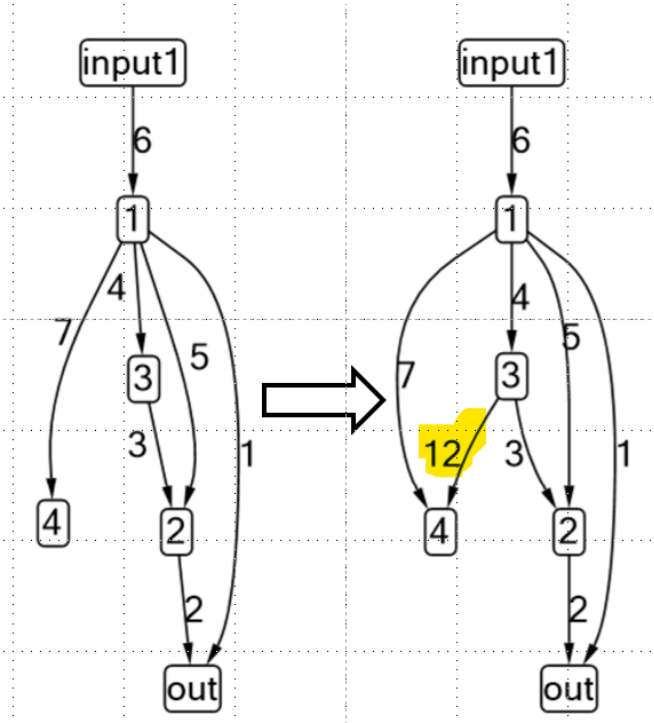


Figure 9-3: Add connection operation - connection #12 add between node '3' and node '4'

The Add Node operation is demonstrated in Figure 9-4. In the Add Node operation, an existing connection is split by adding a new node in between. However, the way this is performed is a little involved. First a connection is chosen and it is disabled. A new node is created. Then a new connection is added from the source of the disabled connection to the newly created node. Finally, a connection is added form the newly created node to the target of the disabled connection. For example, in Figure 9-4, the connection #1 between '1' and 'out' is selected. The #1 connection is disabled and two new connections #8 and #9 are added that connect '1' and 'out' via the new node '5'. Connections are referred to with hash followed by number (e.g. #1) and nodes with number within single quotes (e.g. '3').

Finally, the Crossover operation is demonstrated in Figure 9-5. Here two graphs are merged into a single graph. The merge operation orders the connections of both graphs by innovation

numbers. A merge operation is performed that keeps the common connections from both but adds any differences from either graph. Finally, any cycles that could have been introduced are removed.

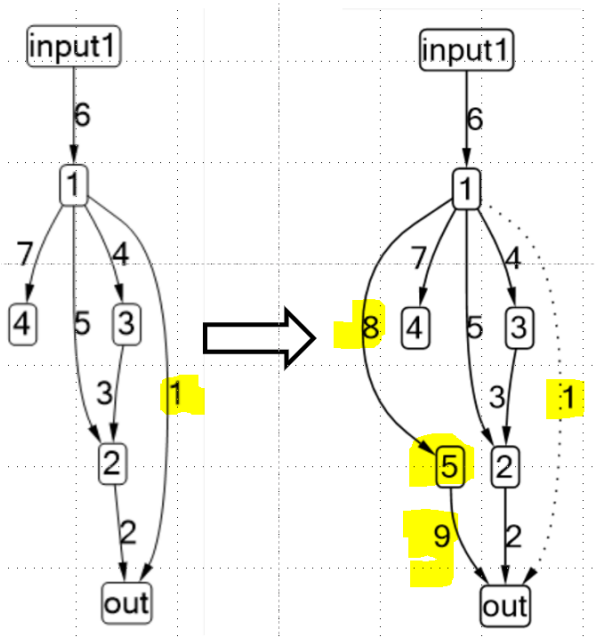


Figure 9-4: Node add operation - node '5' was added between node '1' and node 'out' while the existing connection between '1' and 'out' was toggled off

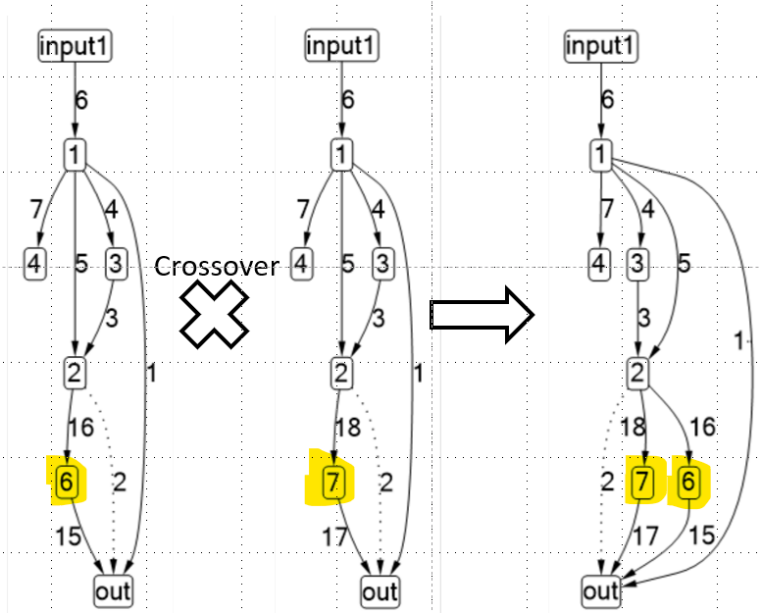


Figure 9-5: Crossover graph operation - merges two graphs

The Mutate Parameter does not modify the structure of the graph but changes some property of one of the non-input nodes. The properties these nodes can hold are discussed in the next section. The NEAT inspired graph operations are the basis of the graph evolution under CATNeuro, however there are many other operational details for CATNeuro that are covered next.

9.3 CATNeuro System

CATNeuro is actually an ensemble of populations. Each individual in every population contains a directed graph. These are evolved via the graph operations described earlier. A population is a species unto itself. Speciation is used to protect and nurture individuals so that they are not eliminated too early from the pool (Howard & Berlocher, 1998). One population is for Blueprint individuals and the rest for Modules - a concept taken from the "CoDeepNEAT" system (Miikkulainen, et al., 2017).

The Blueprint individuals define the macro structure of a deep learning model graph. Each internal node of the Blueprint individual is replaced with the graph of an individual selected from one of the Module species – when the Blueprint is 'assembled' just prior to evaluation. An example of such a structure is given in Figure 9-6. The process of assembly is explained later in this section.

Blueprint internal nodes reference Module species – i.e. they contain the id of one of the available Module species that the CATNeuro is configured to run with. Under the Mutate Parameter operation, this id is modified to point to one of the available Module species.

The internal nodes of Module graphs refer to deep learning operations, e.g. Dense Layer, Normalization layer, etc. (Goodfellow, Bengio, & Courville, 2016). Under the Mutate Parameter operation the parameters of the operation are modified. For example, for the Dense layer, the number of dimensions are evolved; for the Normalization layer the type of normalization is chosen from either Batch Normalization (Ioffe & Szegedy, 2015) or Layer Normalization (He, Zhang, Ren, & Sun, 2015); etc.. A randomly selected deep learning operation is assigned to a new node when it is created for a Module individual. Thereafter only the parameters are evolved.

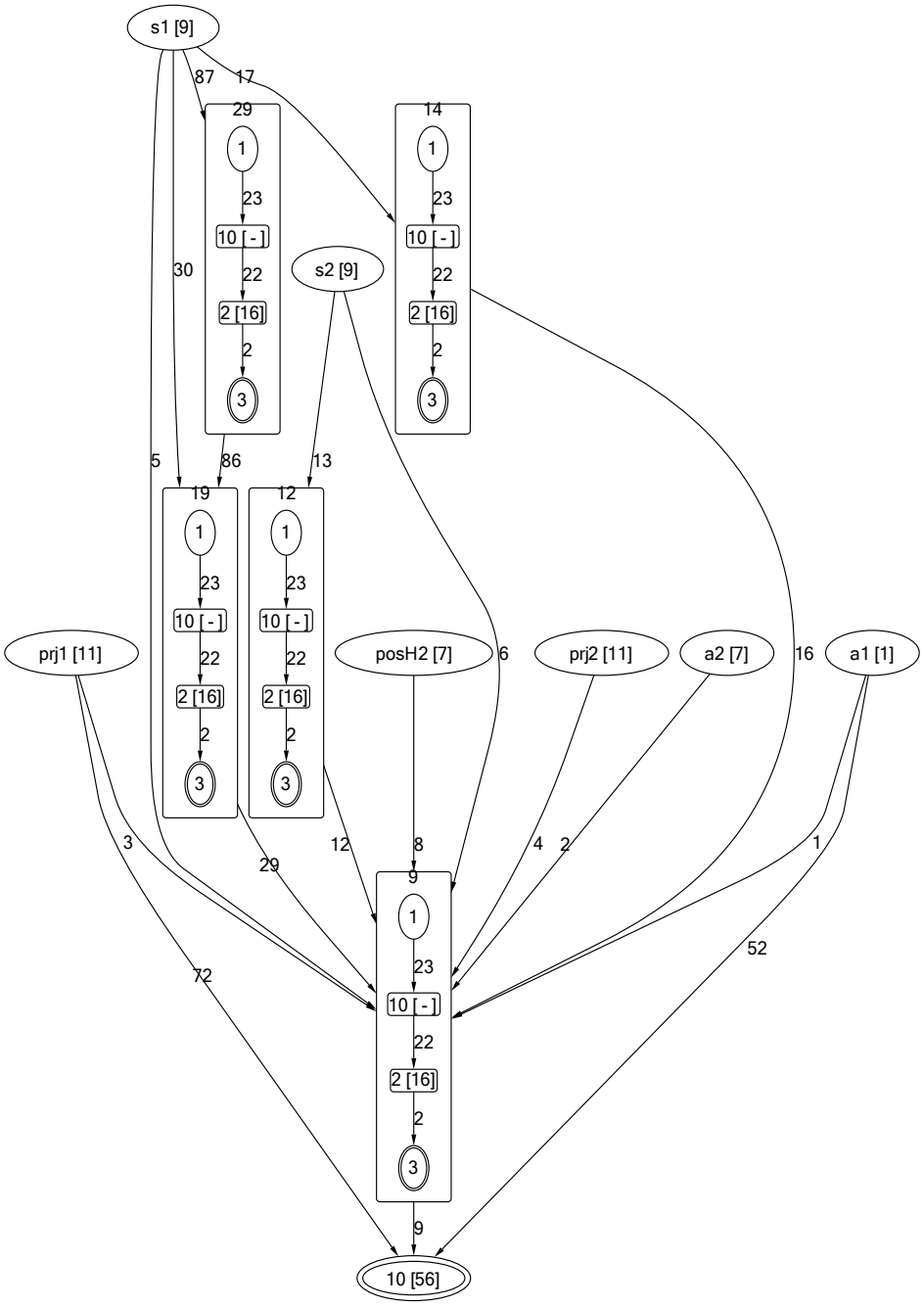


Figure 9-6: Two-level graph – 'Blueprint' outer graph with embedded 'module species' subgraphs – blueprint and species populations are evolved separately

There is no limit to the number of module species. (For the CATNeuro experiment conducted in this research, three module species were used).

The input and output nodes of the Blueprint species individual's graph are configured to be the input and output required for the task at hand. For example, if the task is binary classification of images then the input node would represent the dimensions of the input images, e.g. 224x224x3 for width x height x number of colors. And the output node would be a 2x1 vector (for the two classes). The system allows for multiple input nodes but is restricted to a single output node.

The input and output nodes for a Module individual's graph are just connectors. When a Blueprint is assembled, its each of its internal nodes is replaced by a randomly selected Module individual's graph from the Module species that the Blueprint node points to. The input and output nodes of the selected Module individual's graph respectively connect to all the incoming and outgoing connections of the replaced Blueprint node. In Figure 9-6, input nodes are represented by ellipses with single line borders and output nodes by ellipses with double-lined borders. The Blueprint node '14' has the incoming connection #17 that is stipulated to connect to '1' input node of the embedded Module individual. Similarly, the '3' output node of the embedded Module individual is stipulated to connect to #16 – the outgoing connection for Blueprint node '14'.

Following CoDeepNEAT (Miikkulainen, et al., 2017), all Blueprint nodes that point to the same species are replaced with the same randomly selected individual from that Module species. Many recent advances in deep learning are attributed to cellular or modular structures that are used repeatedly in the network (Szegedy, et al., 2015) (He, Zhang, Ren, & Sun, 2015). The use of Modules is there to help find such modular structures. Such structures control the unconstrained growth of the neural networks. In addition they can be used to grow or shrink the network capacity by adding or removing such units.

Before a Blueprint individual can be evaluated, it has to be assembled. A Blueprint individual's internal nodes are replaced with selected Module individuals to obtain a structure called Network Assembly. The Network Assembly is translated into a deep learning model for a particular library (e.g. Tensorflow, PyTorch, CNTK, etc.). The translation of a Network Assembly to a deep learning model for a specific library is performed by a configurable component called Evaluator.

When CATNeuro is initialized for a particular task it is configured with an Evaluator. A Network Assembly is an abstract representation of a deep learning model. The job of the Evaluator is to convert that into a concrete model; train the model using whatever training data the Evaluator is configured with; and return the training loss and model size (Figure 9-7).

The CATNeuro system is multi-objective since it trades off performance (training loss) against the model size (number of parameters in the model). The pareto ranking function is also a pluggable component. It can be chosen to suite the task at hand. The 'fitness' associated with each individual is a 2x1 vector (for loss and size). By contrast the individual fitness is a single number CATGame.

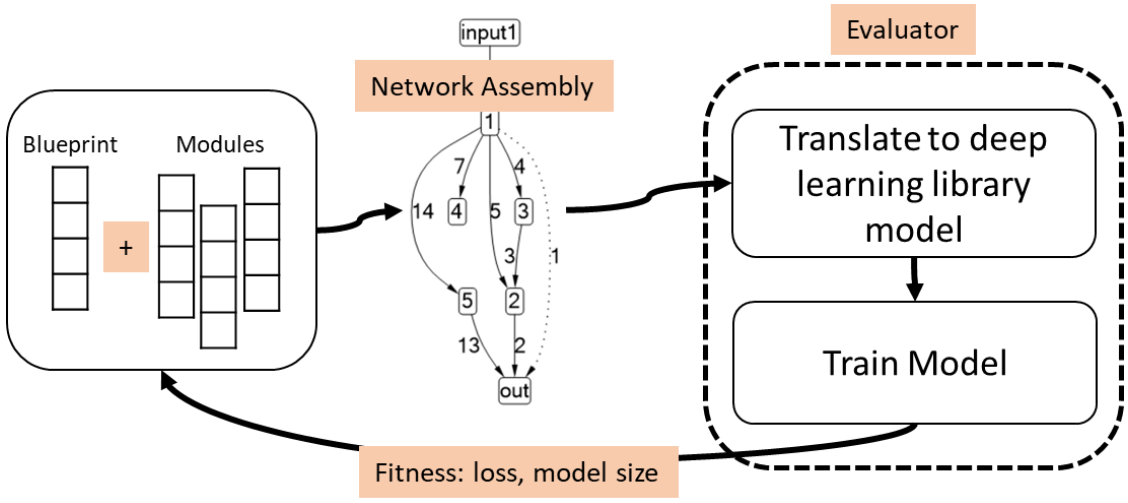


Figure 9-7: Translate Network Assembly to concrete model and train using training data

After all Network Assemblies have been evaluated, the fitness is assigned as follows:

- Blueprint individuals are assigned fitness values (loss and model size) returned from the Evaluator
- Module individuals are assigned the average fitness (loss and model size) of all the Blueprint individuals where they were used

By using average fitness for Module individuals, the individuals are protected and not eliminated too early in the process. After evaluation and fitness assignment each population is evolved following the standard process (Chapter 2):

- Induct top individuals into the Belief Space via the ***Acceptance*** function
- Update the Belief Space with the ***Update*** function
- Distribute knowledge in the population network and evolve each individual via the assigned Knowledge Source – in the ***Influence*** function

The Belief Space and any other internal state needed to evolve populations is maintained separately for each population. In addition, CATNeuro maintains the top n Network Assemblies discovered thus far. The multi-objective ranking of Network Assemblies is done with the configured ranking function. Each Network Assembly contains enough information to be translatable into a deep learning model when required. The output of the CATNeuro systems is the ranked list of best Network Assemblies when it stops. The procedure for running CATNeuro is as follows:

1. Construct Network Assemblies
2. Evaluate Network Assemblies
3. Check for termination condition
4. If terminating then stop else
    a. evolve populations to obtain new generations
    b. go to 1

The termination condition may be a MAX number of generations or it could be an expression such as “stop when no improvement is found in x generations”.

The Knowledge Sources operate somewhat differently in CATNeuro than in CATGame. Each Knowledge Source has two associated functions: ***acceptance*** for the induction of the knowledge from the population space; and ***influence*** for the impartation of knowledge to the next generation. Under CATNeuro, the Knowledge Sources conceptually are the same as for CATGame but the

implementation can be very different, especially for the influence function. The influence and acceptance functions for CATNeuro Knowledge Sources are discussed next.

9.3.1 Influence Functions

In numeric optimization problems there is generally good locality – i.e. small changes in parameter values lead to correspondingly small changes in fitness. The exploration-exploitation balance requires some underlying idea of locality to be meaningful. The cooperative knowledge distribution of Stag-Hunt is quite reliant on balancing exploration with exploitation. Such locality is harder to establish for graph evolution since the changes that are discreet. For example, adding a new connection may potentially make a big difference to the fitness of the model. However, in general one would expect adding a connection to be more disruptive than say toggling a connection; adding a node to be more disruptive than adding a connection; and so on.

Using a probabilistic notion of locality, the available graph operations are associated with Knowledge Sources with weight distributions (see Table 9-1). The Knowledge Sources can thus be ranked on the explorative-exploitative scale. Exploitative Knowledge Sources (e.g. History) are biased towards selecting graph operations that will make relatively less disruptive changes. The Knowledge Source influence function samples from the associated distribution to select an operation to apply when modifying a population individual. The explorative-exploitative ranking in CATNeuro however is different from that in CATGame. For example, Normative knowledge is considered exploitative in CATNeuro – unlike in CATGame – as it does not modify the graph structure; it only modifies the parameters of the graph nodes or some meta parameters associated with Blueprint individuals, such as the learning rate used for neural network training.

Table 9-1: Knowledge Source Mapping to Graph Operations with Associated Weights

KNOWLEDGE SOURCE	MUTATION SELECTION POLICY	
HISTORY	Crossover	0
	AddNode	0
	AddConnection	0.2
	MutateParm	0.2
	ToggleConnection	0.2
SITUATIONAL	Crossover	0
	AddNode	0.1
	AddConnection	0.5
	MutateParm	0.1
	ToggleConnection	0
DOMAIN	Crossover	0.1
	AddNode	0.5
	AddConnection	0.1
	MutateParm	0
	ToggleConnection	0
TOPOGRAPHICAL	Crossover	0.5
	AddNode	0.1
	AddConnection	0.1
	MutateParm	0
	ToggleConnection	0
NORMATIVE	Crossover	0
	AddNode	0
	AddConnection	0
	MutateParm	1
	ToggleConnection	0

The weights associated with each KS are normalized into true probabilities before sampling for a graph operation. The unnormalized weights are easier to modify when tuning them by hand.

The weights were selected with limited empirical testing. Future work will include focus on better turning of the weights by training networks across a variety of tasks.

The CATNeuro populations can be configured with some restrictions. The number of nodes can be restricted so the graph does not grow beyond a certain size. For Module species, restricting graph size is desirable so modular components remain small and reusable and not become too specialized. When a graph reaches the allowed maximum size, the Add Node operation is removed from the probability distribution of actions (Table 9-1) so that a different operation is selected instead. The CATNeuro population also can be restricted to use only a subset of the graph evolution operations. This is done in the case of small Module populations so resources are focused on more fruitful regions of the search space.

9.3.2 Acceptance Functions

The induction of knowledge from the population space into the Belief Space is conceptually similar to that in CATGame. Unlike the influence functions however, the acceptance functions are generally less affected. Note that since there are multiple populations in CATNeuro the internal state needed by Knowledge Sources to operate is separate for each population. For example, Normative knowledge maintains separate parameter densities for each population. The CATNeuro versions of the acceptance functions for the KS are described next with differences from CATGame highlighted.

Topographic: In CATGame, Topographic knowledge clusters individuals into promising regions of the search space using the K-means algorithm and the Euclidean distance metric – inspired by BSO (Shi, 2011). Topographic does the same in CATNeuro except that the distance metric used is a specially constructed graph distance metric.

History: History is very similar in both because it just needs to keep track of the best individuals over time.

Normative: In CATGame Normative maintains promising ranges of numerical values. In CATNeuro, Normative is conceptually similar but the implementation is very different. Normative in CATNeuro needs to handle categorical (non-continuous) values in many cases (e.g. Module species ids). Since there is no natural ordering for ids, the concept of range does not apply. Normative knowledge instead maintains ***probability densities*** for the parameters it tracks. When it needs to evolve parameters, Normative samples from spin wheels for categorical parameters and Kernel Density estimates for continuous parameters (Cosma Shalizi CMU, 2009) (see Figure 9-8).

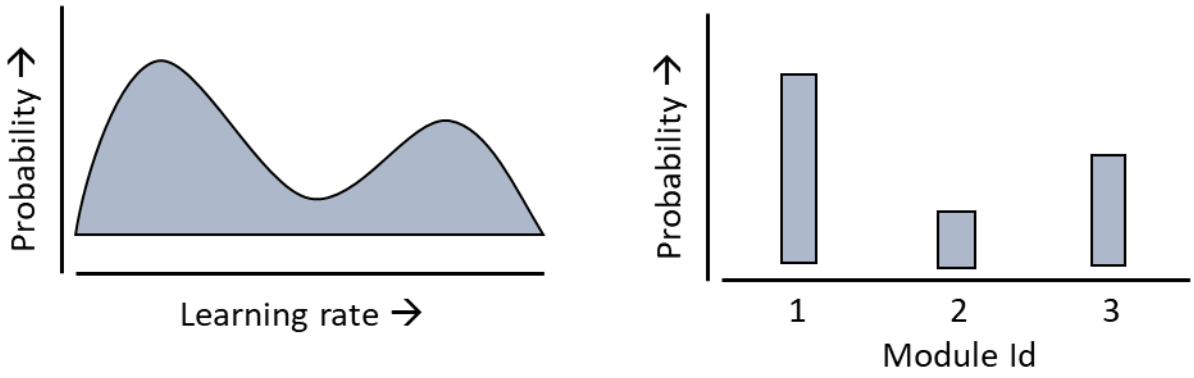


Figure 9-8: Examples of probability densities maintained by Normative knowledge for two types of parameters

The probability values are calculated by incorporating the fitness (training loss only) of the best performing individuals across generations. Normative uses this data for the Mutate Parameter graph operation. However, with some probability, this operation may also be performed by other KS (History and Situational). Normative thus share's its internal state with other KS so that they are able to apply the Mutate Parameter operation as well. The density estimates are maintained

by the aforementioned innovation numbers. Recall that each connection in the graph is assigned an innovation number; it is a global, monotonically increasing value. As a graph evolves the added innovation numbers stay the same and therefore can be used as anchors for pinning density estimates to. Each estimate applies to the target of the corresponding connection. For example, if the target is a node that references modules (in Blueprints) then the density estimates are for module ids. These estimates are only sampled when there are enough samples available. Real valued parameters (e.g. learning rate) are sampled using the Gaussian kernel with bandwidth equal to $\frac{1}{10}th$ the configured range of the parameter. For discrete valued parameters, the system maintains non-zero probabilities for all cases so that there is always some chance of selecting any available case.

Situational: Situational maintains a list of top n exemplars in both CATNeuro and CATGame.

Domain: In CATNeuro, when Domain modifies an individual under its influence, it can either mutate the graph of the individual in question or replace its graph with the mutated graph of a randomly selected top performer. NEAT is greedy so Domain propagates the current generation top performers with a configured probability. Thus, Domain inducts the current generation top performers in CATNeuro. In CATGame, Domain does not maintain any state as it uses parameter slopes for influence that are calculated at the time the influence function is applied.

9.3.3 Knowledge Distribution

Knowledge distribution in CATNeuro is similar to that in CATGame. Weighted Majority uses an algorithm that for all practical purposes is the same as for CATGame (see 3.2). The Stag-Hunt distribution is slightly different. Due to a stricter requirement around ranking of Knowledge Sources from exploitative to explorative. In the CATNeuro adaption of Stag-Hunt there are p

cooperative generations followed by one evaluative one. In a cooperative generation individual i behaves cooperatively. In an evaluative generate i behaves individualistically.

Considering the cooperative case, as before (section 3.2) K is the set of KS; i indexes the population; and j indexes $i's$ neighbors; $F = \{r | r \in \mathbb{R}\}$ is set of real numbers and $F_i \subseteq F = \{fit_{i1}, fit_{i2}, \dots, fit_{ij}, \dots\}$ be the fitness of $i's$ neighbors in the current generation. $SocialRank: \mathbb{R} \times F \rightarrow \mathbb{Z}$ is function that returns 'social' rank of i based on its and neighbors' fitness values; $sr_i = SocialRank(fit_i, F_i)$. Let $P: K \rightarrow \mathbb{Z}$ be a probability distribution where $\sum_{k \in K} P(k) = 1$. Also, $KSforRank: \mathbb{Z} \rightarrow P$ return a probability distribution over K given a rank. In CATNeuro the Knowledge Source assigned is sampled from the returned probability distribution; defined as $Sample: P \rightarrow K$. The probabilities are constructed so that a low rank will return a probability distribution that is biased towards explorative Knowledge Sources. And vice-a-versa for a high rank. Since Knowledge Sources are not deterministically explorative or exploitative due to poor locality of graph operations, the knowledge assignments are done probabilistically to compensate. The new KS assigned to $k_{inew} = Sample(KSForRank(sr_i))$.

In the evaluative case, let $O = < k_0, k_1, \dots, k_{n-1} >$ be an ordering of KS according to each's explorative potential; o_k is offset of k in this ordering and $O[q]$ returns the k at offset q. Let $e_i, e_2, .., e_g, \dots$ index evaluative generations and fit_{ie_g} be $i's$ fitness in generation e_g. Then,

$$k_{inew} = \begin{cases} O[O_{k_i} - 1 \; mod \; n] & fit_{ie_g} \geq fit_{ie_{g-1}} \\ O[O_{k_i} + 1 \; mod \; n] & otherwise \end{cases}.$$

If $i's$ fitness has improved since the previous evaluative generation then it adopts the next most exploitative KS in the ladder (and wraps around if at bottom) and vice-a-versa. Here i acts alone without considering it's neighbors. The mechanism thus associates more explorative

strategies to comparatively underperforming individuals (as compared to its neighbors) and vice-a-versa. The exploration-exploitation balance is performed using the local signal of ‘social’ rank.

In summary, the CATNeuro system optimizes in the space of directed graphs where CATGame searches in a real-valued hyperspace. CATNeuro utilizes speciation through multiple populations. By contrast CATGame uses a single population. There are many other differences between the two systems. However, the knowledge distribution mechanisms are largely similar between the two and operate on the same principles. Stag-Hunt was seen to perform well against a complex, dynamic environment. Now it is put to test for a hierarchically complex optimization problem. The performance of Stag-Hunt in a different domain will provide further insights into whether cooperative mechanisms are indeed beneficial for problem solving in highly complex domains. The performance of Weighted Majority and Stag-Hunt distribution systems are compared on the optimization of deep learning model for a controller to play a fighting game. The fighting game used for the experiment is described in the next section.

9.4 A Brief Overview of Reinforcement Learning

The deep learning model used for the ICE game controller has to be trained to play the game effectively and for this significant amount of training data is required. The most relevant discipline for creating data for such a task is the field of Reinforcement Learning (RL). RL is a form of machine learning that fits between supervised learning and unsupervised learning. In RL an agent interacts with the world or environment to achieve a goal. The agent:

a. Has a capacity to take actions that affect the environment

b. Can observe or receive feedback from the environment

c. Is motivated to achieve a high-level goal through a reward structure

For the ICE game, the agent would be the deep learning model (embedded in the controller application). The model selects an action to take after observing the state of the game at every time step. The goal is to win each round by trying to land the maximum number of hits on the opponent while protecting oneself from being hit. Such a setup is labeled a Markov Decision Process (MDP) (also Markov Reward Process) (Kober, Bagnell, & Peters, 2013) and is commonly used in the field of robotics, multi-agent systems, games and control applications.

An MDP is a 4-tuple (S, A, R, P) where:

- $S = a\ finite\ set\ of\ states$
- $A = a\ finite\ set\ of\ available\ actions$
- $R : S\ A\ x \times S \rightarrow \mathbb{R}\ is\ the\ reward\ function$
- $P: S \times A \times S \rightarrow [0,1] is\ transition\ probability$

A policy $\Pi: \mathrm{S} \times \mathrm{A} \rightarrow \mathbb{R}$ is the probability of tacking action $a \in A$ when in state $s \in S$, i.e. $\Pi(s, a) = P(a|s)$.. In most scenarios, the rewards associated with future time steps is discounted by a discounted by a discount factor $\Upsilon \in [0, 1)$. The ***value*** function V is the expected sum of future rewards given an MDP and policy Π and is defined as:

Value function:

$$\boldsymbol{V^{\pi} = E\left[\sum_{t=0}^{\infty} \Upsilon^{t} r_{t} | s_{0} = s\right]} \qquad \textbf{Eq 9-1}$$

The goal of Reinforcement Learning is to find the optimal policy π^* that maximizes the expected sum of future rewards. There are a wide variety of settings and algorithms that can be use to achieve the optimum policy (Sutton & Barto, 2018).

Reinforcement Learning is a vast subject that has a long history (Sutton & Barto, 2018). It has recently resurged in popularity due to high profile achievements such as beating the world champion Lee Sedol at the game of Go (Silver, et al., 2017).

Figure 9-9 is a partial taxonomy of RL algorithms. In model-based algorithms the agent is either given a model or learns a model of the environment and then operates accordingly to interact with the world. A recent example is the AlphaZero (Schrittwieser, et al., 2019) model from Google Deep Mind that is a successor to AlphaGo noted earlier. AlphaZero is programmed with explicit rules of the game Go.

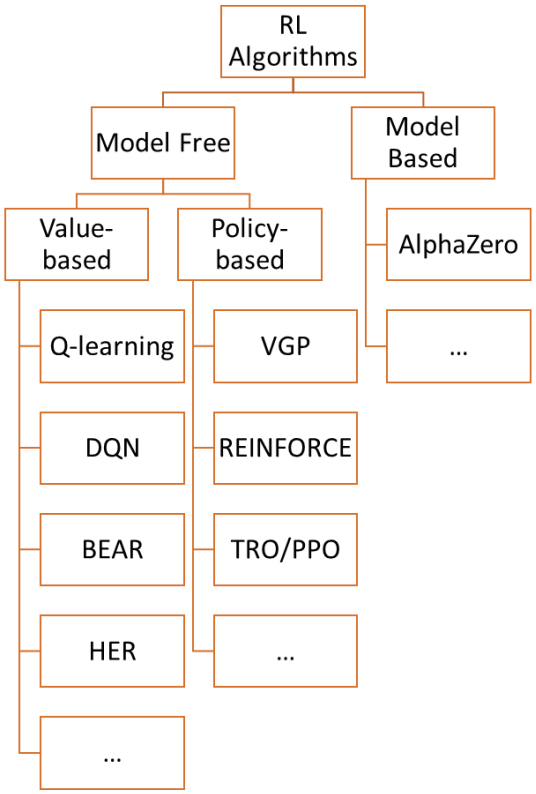


Figure 9-9: A partial taxonomy of Reinforcement Learning algorithms

Model-free algorithms don't have to construct a model of the world but instead focus on what action to take in a given state, to progress towards the goal. Model-free can be divided into value-

based and policy-based. Value-based methods focus on determining the value of states or state-action pairs from which an optimal policy can be derived as in Eq 9-1. Policy-based methods instead learn the policy directly from experience obtained in interacting with the environment. In some situations, the number of states or state-action pairs is prohibitively large and its not feasible use value-based methods. RL is an active area of research. The seminal policy optimization approach named VPG for (vanilla) Policy Gradient, was introduced by Sutton et al. in 1999 (Sutton, McAllester, Singh, & Mansour, 1999). Since then many variations and improvements have been developed such as Trust Region Policy optimization (TRPO) (Schulman, Levine, Moritz, Jordan, & Abbeel, 2015) and Proximal Policy optimization (PPO) (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017).

For completeness, it should be mentioned that the so-called Actor-Critic methods combine features of policy and value based approaches together for Reinforcement Learning (Konda & Tsitsiklis, 2002).

One recent breakthrough was in 2015 when Deep Mind published deep Q-network (DQN) (Mnih, et al., 2015) that achieved human-level performance on Atari games. DQN used a value-based method called Q-learning (Sutton & Barto, 2018) but with a deep neural network to approximate the value function. The term ***Deep Reinforcement Learning*** (DRL) is used when a deep neural network is exploited to approximate the value or policy function. DRL has re-energized the field of Reinforcement Learning. Other recent development in Deep RL algorithms are Hindsight Experience Replay (HER) (Andrychowicz, et al., 2017) and BEAR (Kumar, Fu, Tucker, & Levine, 2019).

The next section describes the Reinforcement Learning based strategy used for creating the training data for the evolution of FightingICE controllers.

9.5 Training Regime for ICE Game Controller

A policy-based RL approach is followed for training data creation. (As an aside, a value-based approached based on Q-learning was also tried but it did not work well due to the large state space involved).

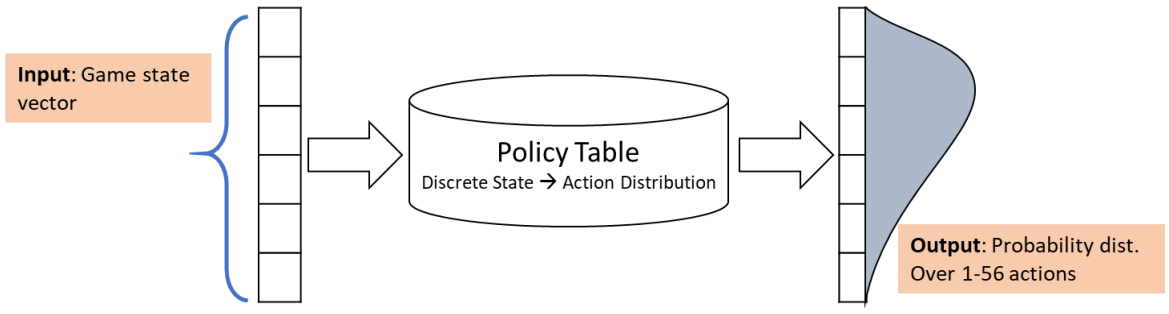


Figure 9-10: Policy table constructed with reinforcement learning provides a mapping from game state to an action policy

Conceptually the process to acquire training data is as follows:

1. Create an empty dictionary structure (Policy Table) that maps a 72x1 vector (key) to a 56x1 vector (value) – see Figure 9-10. This table maps game state to action distribution and will be populated as described in subsequent steps

2. Play 100 or so games using a controller that makes random moves against Thunder AI and record raw game frames (non-pixel data only) to obtain a large collection of realistic game states. Here P1 (player 1) is designated as the 'controller' player and P2 as the opponent. Thunder AI was used as the opponent in all subsequent steps

3. Use the recorded frames to play different scenarios using the ICE provided simulator. (The ICE game package comes with a simulator that can be used to simulate game play from

any recorded frame; the starting frame and a sequence of actions for each of the players is required to make use of the simulator)

4. For each ***scenario*** (recorded frame), extract the corresponding 72x1 state vector and save that into the Policy Table mapped to a uniform 56x1 vector (i.e. any action is equally likely).
5. Play each scenario 10 times with different randomly chosen sequence of actions for P1 and the actual recorded sequence of actions for P2
6. For each simulated play, note the score and if P1 won, update the Policy Table to increase the probabilities of the actions taken using methods from Reinforcement Learning (Sutton & Barto, 2018) and vice versa
7. Iterate over all of the recorded frames repeatedly until the Policy Table converges
8. The Policy Table is the training data for the neural network models
9. Train a neural network model using the previous iteration of the Policy Table. Use this intermediate model in the controller to play and record ***new*** game frames. Now instead of taking random actions, the actions are chosen from a model trained with the previous iteration of the Policy Table.
10. Again, use the simulator and the newly recorded frames to update the Policy Table till it converges. Now, instead of random actions, the actions for P1 are chosen as follows:
    a. If a state has been played before, then choose actions from the current policy table in an ***epsilon-greedy*** way. With epsilon probability (e.g. 95%) sample the current policy associated with the state and with 1 – epsilon probability (e.g. 5% in this

case) choose a random action. (Epsilon-greedy approach is a way to balance exploration with exploitation).

b. Otherwise choose a random sequence of actions

11. Repeat the process starting from step #9 an additional n times to get the final Policy Table that will be used for evaluation of the distribution mechanisms – see Figure 9-11. The number n is chosen till there is no appreciable improvement in game play. Here it was 3 iterations of the process.

About 1GB worth of raw game frames were recorded that resulted in a Policy Table of about 1M rows – i.e. distinct states. It takes about 24 hours for the Policy Table to converge. The Policy Table is a discrete mapping from game state to action distribution. The deep learning model trained on the Policy Table is a continuous mapping - i.e. it can provide an action distribution even if the input state vector does not exist in the Policy Table. The deep learning model is a highly compressed representation of the Policy Table – it is usually less than 100KB in size. Thus, the model is better suited for use in the controller (than the underlying table).

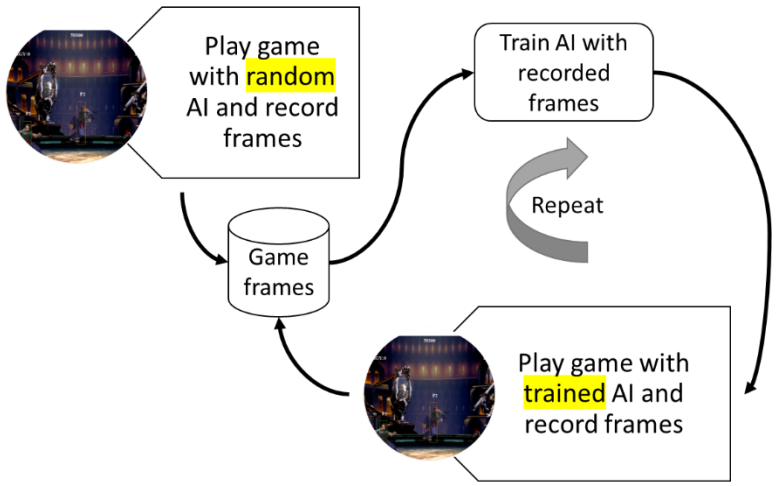


Figure 9-11: AI controller is trained on saved game frames; the process is boot strapped with a random AI and iteratively improved with better trained AIs

Table 9-2 provides the layout of the 72x1 input state vector. The information in the input vector is the current position and speed of the two players (P1 & P2); players' scores and energies; whether or any attacks or projectiles are active; P2 action history; and P1 and P2 position history. Player P1 is being controlled by the CA controller and P2 by "Thunder" AI.

The 72x1 input vector is sliced into 8 'semantic' units labelled by the "Semantic unit" column in Table 9-2. The 8 semantic units are carved up from the 72x1 vector so as to provide more context for neural network architecture selection. In deep learning models, one type of learning is finding the right representations of the input data that help in solving the problem at hand. Usually the lower layers of the network process raw input and find embeddings (Roweis & Saul, 2000) that capture the semantics of the input in some abstract manner. Dividing up the input into logical sections should help this process. A minimal neural network architecture structure with the 8 semantic inputs is shown in Figure 9-12.

Table 9-2: Policy Table Input State Layout Structure

Offset	Description	Semantic unit
0,1	P1 x y position	s1
2,3	P1 speedX speedY	s1
4	P1 energy level	s1
5,6,7,8	P1 state: air | stand | crouch | down	s1
9,10	P1 attack: speedX speedX	prj1
11,12	P1 attack: settingSpeedX settingSpeedY	prj1
13	P1 attack: is_projectile	prj1
14	P1 attack: is_active	prj1
15	P1 attack: is_downProp	prj1
16,17,18,19	P1 attack: 4 attack types (THROW_A, THROW_B, THROW_HIT, THROW_SUFFER)	prj1
20	P1 action: 1 of 56 actions	a1
21-41	P2 state – repetition of P1 state	s2; prj2; a2
42-47	P2 6 historical actions	a2
48-59	P1 last 6 x,y positions	posH1
60-71	P2 last 6 x,y positions	posH2

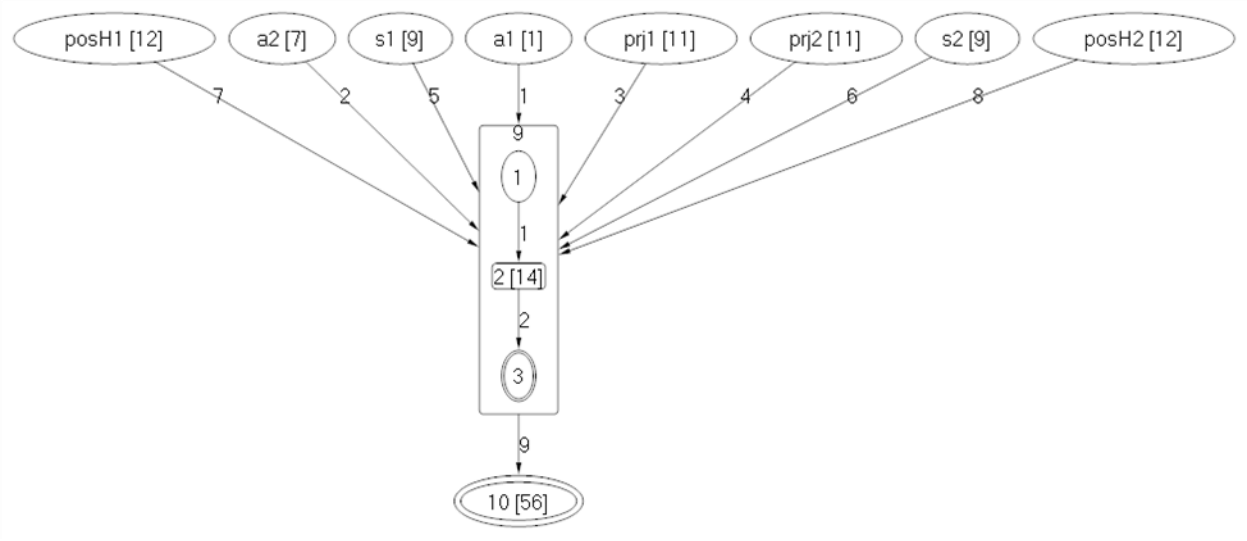


Figure 9-12: Graphical structure of a minimal model – 8 'semantic' inputs, 1 intermediate node and 1 output node

In order to construct the Policy Table, the raw game frames are read in sequence. A sliding window of 10 frames is used so that an input state vector with the required history of P2 actions can be constructed Figure 9-13.

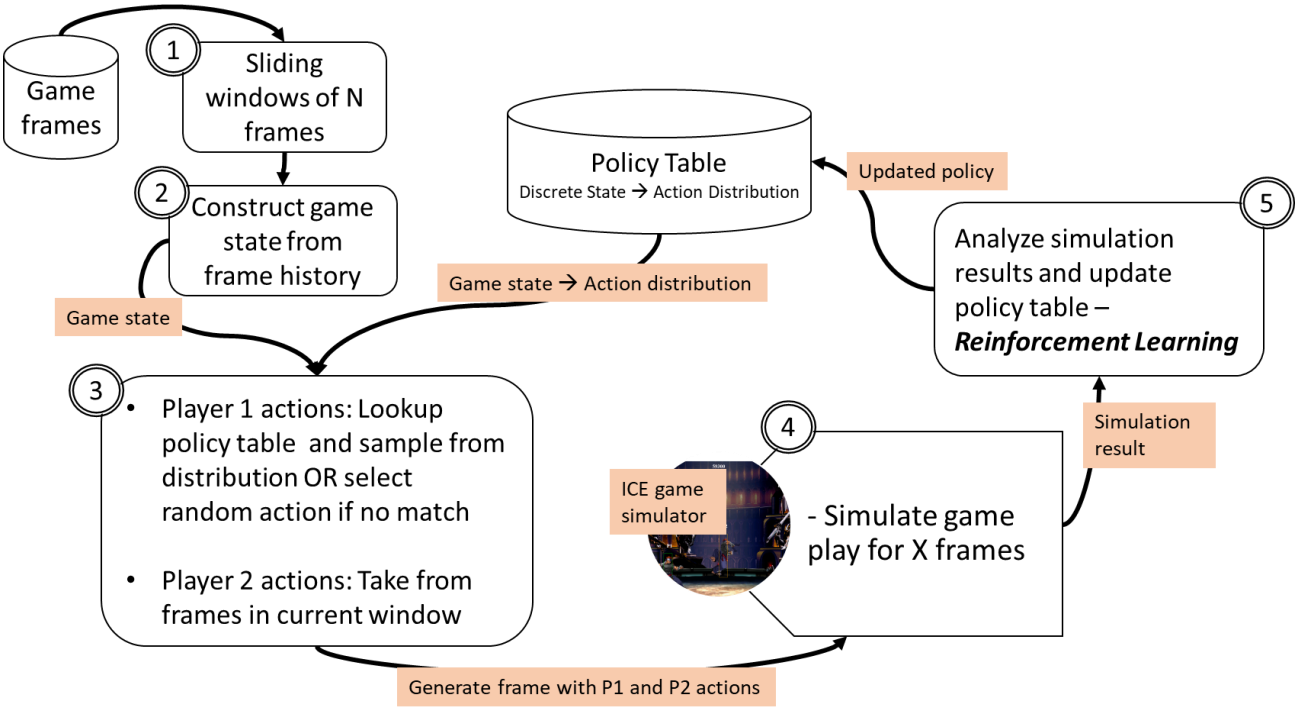


Figure 9-13: Reinforcement learning process to learn policy for discrete states

The simulator can be configured to simulate a given number of frames starting from the provided frame and a sequence of actions for both players. A sequence of 3 actions for each player are simulated for 100 frames. The time horizon for a single action is up to 20 frames so 100 frames are sufficient to complete all 3 actions. The game is fast paced so a sequence of the next 3 actions is sufficient time-horizon for action planning. The simulator completes one action before choosing the next action in the given sequence. The action sequence for P2 is taken from the raw game frames. The action sequence for P1 are either randomly generated or based on the currently available Policy Table.

The final Policy Table is the data for neural network training. A series of candidate models are generated with CATNeuro – configured to run with each of the distribution mechanisms – and tested in the controller. This process is explained next.

9.6 Neural Architecture Search with CATNeuro

A deep learning model trained from the Policy Table is a compact representation of the Policy Table. It is faster to use; much more compact; and supports mapping from any game state even if the state does not exist in the Policy Table. Therefore, using a derived model is better for game play than using the underlying table directly. If the state space of a game is small or can be discretized effectively than a Policy Table would be a good choice but here that is not the case. With most deep learning models, it is not clear what is the right architecture at the outset. Discovering an architecture is a time-consuming process that involves trial-and-error. A neural architecture search tool like CATNeuro can assist by shortening the search time or freeing up human time by substituting it with machine time.

The Evaluator component used for this task (see section 9.3) is configured to train models with a sample of the Policy Table. This was done to shorten the training time for CATNeuro. A sample of 130K rows was randomly selected from the full 1M row Policy Table. With the full Policy Table this would have been very time consuming. Even with the small sample, the time to complete 6 CATNeuro runs was is in the range of 24 to 36 hours on single GPU box. A population size of 36 is used so each generation (or time-step) requires 36 graph evolutions and subsequent translations and training. The vast majority of the time is consumed by model training. The models have to be trained on a Graphical Processing Unit (GPU), which is a limited resource. The current hardware configuration is limited to a single GPU and therefore it takes 20-30 minutes per generation. However, the training is parallelizable so each model can be concurrently trained on a separate GPU; such a configuration, if available, would drastically reduce the time required to evolve each generation.

The sample Policy Table is further split 70/30 into training and evaluation sets (Figure 9-14). This is done to prevent overfitting. The candidate models generated by CATNeuro are trained on the training set and tested on the evaluation set. The training is done in iterations called epochs. Each epoch is a full sweep of the training set. Each model is tested on the evaluation set after each epoch. If the evaluation loss is higher than after a previous epoch, the training is stopped. Training further would risk overfitting.

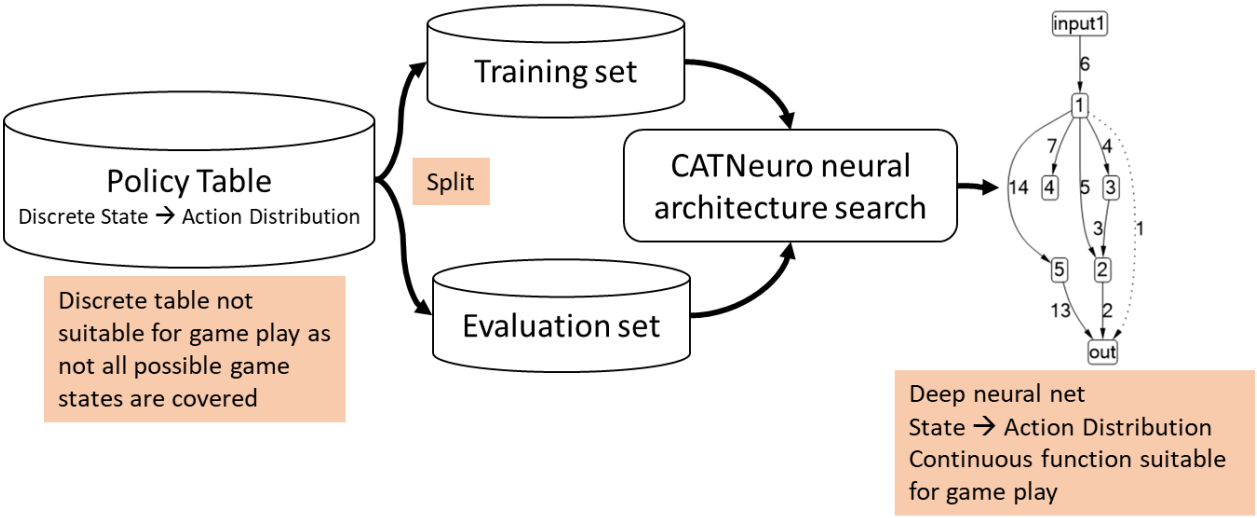


Figure 9-14: CATNeuro neural architecture search process

The termination condition expression is “stop if no improvement is seen in 10 generations”. CATNeuro usually runs for 20 to 60 generations before termination.

CATNeuro is configured to output the 20 best models per run. The top 10% of the models from each run are trained on the full Policy Table (again with 70/30 split to prevent overfitting). The fully trained models are then tested in the controller and game statistics are recorded for assessment and comparison. The next section describes the experimental setup in more detail.

9.7 Experimental Setup and Evaluation Methodology for Game Controller Models

CATNeuro is run 6 times for each KD mechanism in order to evolve deep learning models using a sample of the Policy Table for training. This process takes 2 or 3 days and therefore the sample size was kept quite small. The training is done on an Acer Predator laptop with a mobile version of the NVidia GTX 1080 GPU. A GPU is required for training models. The experimental parameter settings are given in Table 9-3.

Top 10% of the models are taken from each run and trained on the full Policy Table and then run in the controller to play against the Jerry Mizuno AI and the 2018 champion Thunder. Jerry Mizuno is supplied by Ritsumeikan University, the maintainers of FightingICE. It is considered a benchmark AI controller where the game decisions are made algorithmically using KNN and fuzzy logic (Chu & Thawonmas, 2017). Pertinent data from all played games are recorded in log files. The format of the log files is in Table 9-4. Beyond the scores for P1 (CATNeuro controller) and P2 (Jerry Mizuno and Thunder) and the number of hits to each, the log file data is also used to extract the action distribution for P1 and P2. There are 56 actions available to the player. A more versatile or general model will have a wider repertoire of responses and therefore should have relatively more balanced distribution over actions. A weaker model will tend to overuse certain actions. Therefore, the breadth of responses is an important basis of comparison for the models produced under the two distribution mechanisms.

Table 9-3: Parameters Settings for CATNeuro Experiment

Parameter	Value	Comment

Knowledge Distribution Mechanisms	Stag-Hunt Weighted Majority (WTD)	
Population size	36	For all populations Blueprint and Modules
Network topology	Hexagonal	
Number of Module species	3	
Blueprint limits	20 total nodes	New nodes are not added after this limit is reached
Module 1 limits	3 nodes Only Mutate Parameter operation allowed	This species is meant to support embedding functionality (Roweis & Saul, 2000)
Module 2 and 3 limits	4 nodes	
Termination condition	Stop if no improvement seen in 10 generations	
Sample size	6 per KD mechanism	6 runs of CATNeuro for each KD mechanism can take 24-36 hours

Models evaluated with game controller	12 per KD mechanism	The top 10% of the models produced (i.e. 2 out of 20 * 6) from each run were trained with the full Policy Table and used for testing in the game controller
Deep learning library	Microsoft CNTK	The current Evaluator only supports CNTK (Seide & Agarwal, 2016). Future versions are planned to include Tensorflow and PyTorch
Game character	Zen	For both P1 and P2
Number of games played per model	10	Each game has 3 rounds

Table 9-4: CATNeuro Log File Format for Game Statistics

Column	Description

P1 Action	Action taken by player 1. P1 is played by CATNeuro controller
P1 Last Hit Frame	Frame number of when P1 was last hit
P1 Score	P1 score (also called HP)
P1 Eng	P1 energy level
P2 Action	Action taken by P2. P2 is played by 2018 champion "Thunder" AI or "Jerry Mizuno"
P2 Last Hit Frame	Frame number of when P2 was last hit
P2 Score	P2 score
P2 Eng	P2 energy level

9.7.1 Basic Performance Analysis

The log file data is aggregated to measure the basic game performance of the models produced by the two KD mechanisms against each of the two opponents, Jerry Mizuno and Thunder. The performance comparison is on the basis of:

- Hits landed on opponent aggregated across all models and by best model by KD
- Hits received from opponent aggregated across all models and by best model by KD
- Relative score (player score – opponent score)

Statistical tests for difference in means are performed by calculating the mean standard deviations of the values on a per round basis. Each round is an independent game segment. The number of rounds per each KD-opponent combination is 360, sufficient for statistical testing.

9.7.2 Relative Action Distribution Analysis

Figure 9-15 is an example of a 100% stacked column chart that puts the relative action distribution of two hypothetical distribution mechanisms head-to-head. In this example, KD1 has higher penetration in 3 of the 4 actions shown and therefore is the mechanism that produces more versatile models. The relative action distribution charts provide a view into how different the strategies of the two players are, on a relative basis.

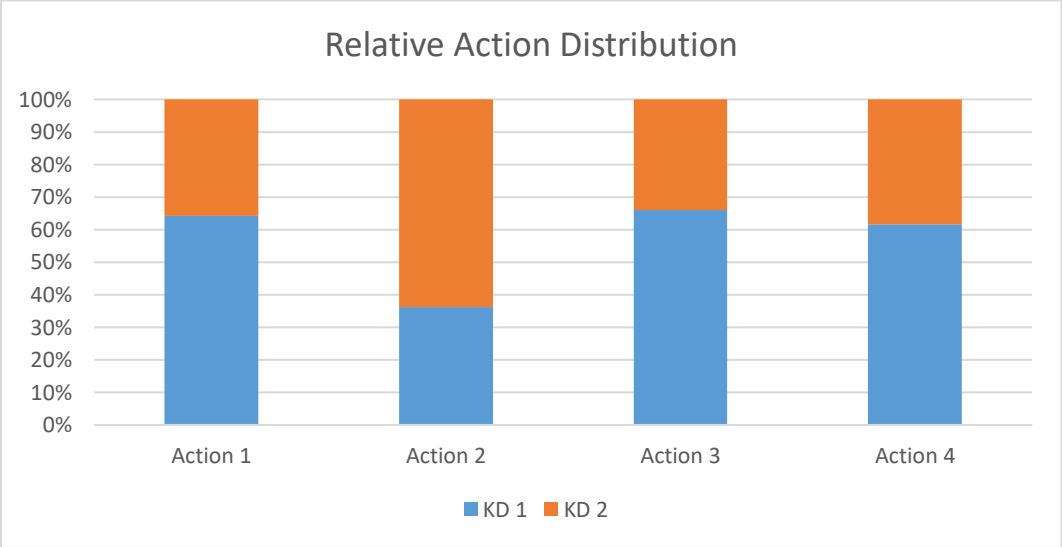


Figure 9-15: Hypothetical relative action distribution between two models of two distribution mechanisms

The relative action distribution analysis is a view into the high-level strategies adopted by the opposing players. It does not convey the repeated sequence of actions or 'combos' that may be prevalent in the attendant strategies. For this, combo analysis was performed which is now explained.

9.7.3 Combo Analysis

The main idea is to find sequences of actions that match the pattern described next. Find combinations of 3 distinct actions done in a sequence where the number of frames does not exceed 30 between the 1st and 2nd actions & 2nd and 3rd actions. Such sequences are classified as combos. The limit of 30 frames is taken from the FightingICE game documentation. Hits landed with 30 frames of each other are considered together and can boost the score beyond the sum of the individual hit scores. An example is AIR_A → AIR → AIR_DA. It is two air attacks interspaced with the AIR action.

The individual combos are then aggregated into higher level patterns to reduce the data complexity and ease analysis. Each action is either offensive (O), defensive (D) or tactical (T). (Tactical actions are positioning actions, such as jump, forward walk, etc.). Each combo is binned into a category defined by the permutation of the letters from the set {'O', 'D', 'T'}. Hence the OTD category will contain all combos that have the [offense] → [defense] → [tactic] pattern. Permutations where all categories are the same (e.g. OOO) are excluded from consideration.

For each category the counts are determined by round. The data is then aggregated to find mean and standard deviation for each category and KD mechanism over the 360 rounds. This data then is used to compare the two KD mechanisms using statistical testing and data visualization.

9.7.4 Model Properties

In addition to the log file data, the ***properties*** of the models produced by each of the mechanism are also analyzed. The CATNeuro system is configured to output the best 20 models found during a run. Only the best 2 are used in the controller for game play. However, all 20 are analyzed with respect to structural and other properties to gain further insight into the operation

of the tested distribution mechanisms. With 6 runs, there are 6*20=120 models available per mechanism. The aggregated properties of each mechanism's models are compared and contrasted. These properties are described in Table 9-5.

Table 9-5: Model Properties Analyzed

Model metric	Comments
Training Loss	The training loss returned by the Evaluator after training the model. For the task at hand, loss is the mean squared error between the model output and ground truth from the training data. It is the 1st value in the fitness vector
Number of parameter weights	The total number of weights in the deep learning model that are optimized in the training. This a measure of model size. It is the 2nd value in the fitness vector
Number of nodes	The total number of nodes in the model produced. It includes the input and output nodes of the Blueprint individual and all of the embedded Module individuals. It is another measure of model size.
Maximum path length	The length of the maximum path from the input to the output. The maximum lengths of the module subgraphs in

	the path are included in the path length. It is another measure of model size – the depth of the model.
Number of edges	The total number of edges in the graph including those in embedded Module subgraphs. It is another measure of model size
Generations to discovery	The number of generations the after which a 'best' model was discovered. As noted earlier, a CATNeuro run stops when no improvement is discovered for 10 generations. A mechanism that is continually able to find frequent improvements will have 'best' models that are found in later generations as it will tend to run for longer. This is an metric is an important measure of KD performance.

Due to the number of samples possible, statistical testing is used to make judgements about the differences in parameters listed in Table 9-5. However, an alternative approach is to visually compare the probability densities instead (Cosma Shalizi CMU, 2009).

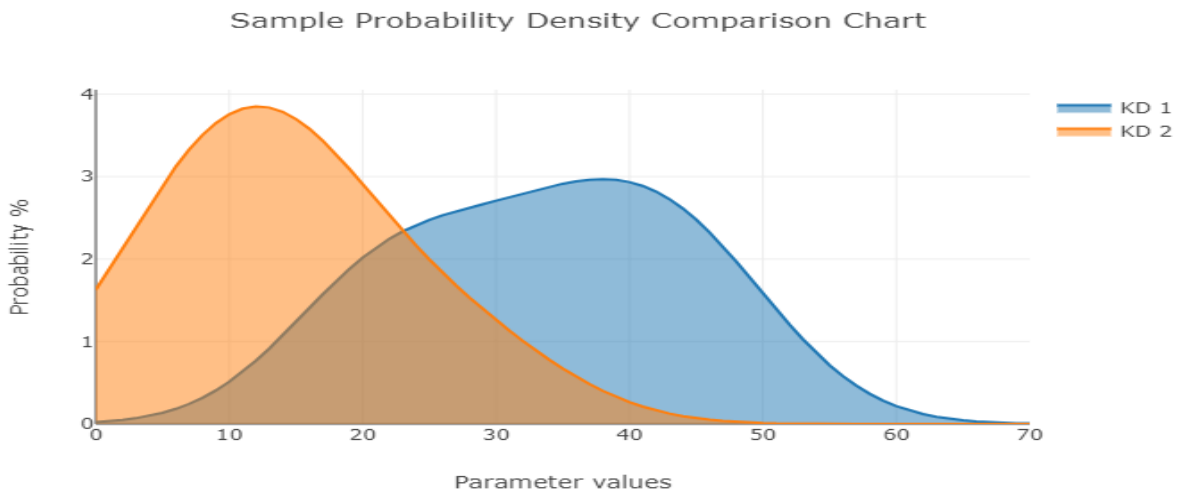


Figure 9-16: Sample probability density comparison chart

A hypothetical example of such a chart is shown in Figure 9 14. This chart can be used to compare the distributions of the same parameter type for two different groups. Each density curve can be thought of as a smoothed version of a histogram. In the example chart, the KD1 distribution (Blue) is clearly shifted to the right and seems to be flatter. And KD2's distribution is peakier and shifted to the left. X-axis has the values of the parameter type that is being compared. Y-axis is the probability. The area of each curve will sum to 1.0. This chart is a useful way of understanding how the 'mass' of a group of values is distributed.

The parameters listed in Table 9-5 are compared with density charts similar to the example in Figure 9-16. Conclusions about parameter differences are drawn on the basis of the density curves associated with Stag-Hunt and WTD mechanisms.

Structural and other numeric properties of the graphical models are another quantitative way of comparing models produced under Stag-Hunt and WTD. Since the models are directed graphs that have a certain structure, the models can also be assessed qualitatively. The best 6 models

associated with each mechanism are drawn with the help of the Microsoft Automated Graph Layout Library (Microsoft Research). This tool produces graph layouts that are human readable, if the number of nodes is less than 100 or so. The models produced from the two distribution mechanisms are compared and assessed with respect to their visual structure.

The experimental framework discussed thus far provides a language or basis for postulating formal hypotheses about the expected outcomes. These are posited and discussed next.

Experimental results from CATGame show that cooperative distribution allocates compute resources more efficiently when faced with complex, dynamic environments, than the standard CA distribution mechanism Weighted Majority. Correspondingly is it expected that cooperative distribution will perform better in the hierarchically complex domain of neural architecture search (Hy **9**-**1**).

Cooperative knowledge distribution will yield more versatile models	**Hy** 9-1

The termination condition used does not the cap the number of generations to a fixed number. Instead optimization is allowed to run till no improvement is detected in 10 generations. Given that Stag-Hunt balances resources well between exploration and exploitation, it should be able to find small improvements more frequently and should sustain search for longer (Hy **9**-**2**).

Cooperative knowledge distribution will sustain longer search runs	**Hy** 9-2

Since NEAT (section 9.1) is largely an additive process it follows that the longer the search process continues the larger the models are likely to be in terms of the number of nodes, edges,

etc. (Hy **9**-**3**). Many of the properties listed in Table 9-5 relate to model size – e.g. number of nodes; number of edges; number of tunable parameters; etc. All are expected to be consistent with respect to each other.

Longer search runs produce larger models	**Hy** 9-3

Given better search performance of Stag-Hunt in CATGame, it is postulated that this mechanism will find better models in CATNeuro as well. The prima facia performance measure under CATNeuro is the training loss. Training loss is the error between the ground truth of the training data the output produced by the model. The game performance is a secondary measure because the search process cannot directly optimize that. As mentioned earlier, loss is measured as mean square error. A lower value indicates that the model is more faithfully able to match the training data (Hy **9**-**4**) and this is a desirable goal. One issue with neural network (and other machine learning models) is that they can overfit the training data. To prevent overfitting, the models are trained on the training set and then periodically evaluated on the test set. Initially loss on the training set and the test set decrease but after a point the training loss continues to decrease but the test loss (i.e. loss on the test set) may start to rise. Beyond that point the model is in danger of being overfit. In CATNeuro the training is stopped when test loss starts to increase and therefore the chance of overfit is low.

The training loss is lower for cooperative distribution mechanism	**Hy** 9-4

In CATGame, the performance differential between WTD and cooperative mechanisms widened with increasing environmental complexity (i.e. higher A-value). In CATNeuro there are two levels of environmental complexity faced by the models – the mid-level benchmark opponent Jerry Mizuno and the 2018 champion Thunder. It is postulated (Hy 9-5) that the cooperative mechanism Stag-Hunt should perform better than the competitive WTD, when faced with the more challenging opponent than when facing the benchmark AI.

Cooperative distribution performs better than competitive distribution under more complex environmental conditions	**Hy 9-5**

9.8 Chapter Summary

This chapter described the CATNeuro system designed to optimize deep learning model architectures. Partial inspiration for CATNeuro comes from the NEAT methodology for neuro-evolution but the two are quite different in many important ways. For example, CATNeuro uses a population bound with a 'social' network where NEAT does not. Also, being a Cultural Algorithms system CATNeuro has a Belief Space component comprised of Knowledge Sources. The Influence function distributes knowledge among the population. The population individuals are evolved under the influence of the associated Knowledge Sources. By contrast, NEAT uses a randomized greedy evolutionary strategy. The particular operations of the Knowledge Source to perform in the space of directed graphs are also described in detail.

CATNeuro has multiple populations to support speciation. The Blueprint population is there to evolve macro structures of the deep learning models. Modules are for the evolution of reusable, modular or cellular components. Modules are mixed into Blueprints in order to create Network

Assemblies that are abstract representation of models. These are then translated into concreted models for a particular deep learning library (e.g. Tensorflow) and trained with the training data for the optimization task. The training results provide a way of assigning fitness values to all population individuals. The fitness values are required by CATNeuro to guide the optimization process.

The implementations of the Stag-Hunt and Weighted Majority distribution mechanisms are both described here for CATNeuro. In order to evaluate the performance of Stag-Hunt with respect to WTD, the FightingICE game test bed is employed. A reinforcement learning based method is used to create the training data required for CATNeuro optimization. The model training regimen used is also documented. The models produced via CATNeuro runs are then used in a game controller to play multiple games against a benchmark AI and a top-level AI that was the 2018 ICE champion. Finally, the experimental setup to compare the performances of Stag-Hunt and WTD based models is presented to support the testing of specific hypotheses. The next chapter analyzes the results of running CATNeuro as per the experimental framework to evolve controller models and then using the models to play games against the selected opponent AI. Also analyzed are the features and properties of the models produced by the two distribution mechanisms.

CHAPTER 10 CATNEURO KNOWLEDGE DISTRIBUTION PERFORMANCE ANALYSIS

10.0 Introduction

As per the experimental framework defined in the previous chapter, CATNeuro was run 6 times each for Stag-Hunt and WTD distribution mechanisms. The top 2 (10%) models from each run were played against the benchmark "Jerry Mizuno" AI and the 2018 champion "Thunder" AI for 10 games each. Section 10.1 compares and analyses the performance of the Stag-Hunt and WTD models used in game play. The data for the analysis comes from about 100mb of logged data collected during game play. The aggregate performance over all models is discussed as well that for best models by different metrics. Section 10.2 compares the action distribution of the Stag Hunt and WTD players and those of the opponents when playing against the CATNeuro players. Section 10.4 compares the properties of the models produced under the Stag-Hunt and WTD mechanisms. Model properties are aggregated from 120 models for each mechanism (see 9.7). Finally, section 10.5 summarizes the results of the analyses done in the prior sections of this chapter. It also draws conclusions about the hypotheses postulated in 9.7.

Also, for demonstration purposes, the video samples of game play are available here:

- Stag-Hunt vs Thunder: https://www.youtube.com/watch?v=pc4ls8MzOV4
- Stag-Hunt vs Jerry Mizuno: https://www.youtube.com/watch?v=ciKTgyMKvG0

10.1 Game Performance Comparison

The aggregate game performance of the models produced by Stag-Hunt and WTD is discussed first. Followed by performance for best models when playing Jerry Mizuno and then best models when playing Thunder.

10.1.1 Aggregate Model Performance Results

Aggregate results for Stag-Hunt and WTD models against Jerry Mizuno are given in Table 10-1 for hits landed on opponent and Table 10-2 for hits received from the opponent. Both Stag-Hunt and WTD generated models perform well against the benchmark AI, Jerry Mizuno. The hits landed by CATNeuro models is much higher than hits received. However, the models produced form both KD mechanisms perform equally well against the opponent. There are no statistically significant differences between the hits landed and hits received values between Stag-Hunt and WTD when playing against Jerry Mizuno.

Table 10-1: Performance Summary - CATNeuro vs. Jerry Mizuno - Hits to Opp.

Performance Summary - CATNeuro vs. Jerry Mizuno Opp.		
Average hits to opp. per round		
	Stag-Hunt	WTD
Avg. hits / round	25.63	25.64
Standard Deviation	4.88	4.46
Two-sample t-test p value	0.987 (samples=360)	

Table 10-2: Performance Summary - CATNeuro vs. Jerry Mizuno – Hits received from Opp.

Performance Summary - CATNeuro vs. Jerry Mizuno Opp.
Average hits received from opp. per round

	Stag-Hunt	WTD
Avg hits received / round	18.57	18.51
Standard Deviation	3.52	3.56
Two-sample t-test p value	0.809 (samples=360)	

The corresponding results against Thunder are given in Table 10-3 (hits-to-opponent) and Table 10-4 (hits received from opponent). Thunder is a much stronger opponent than Jerry Mizuno. Both Stag-Hunt and WTD produced models cannot compete against Thunder. However, here Stag-Hunt produced models perform significantly better when than the WTD produced models in terms of the hits-to-opponent metric. The difference is statistically significant in favor of Stag-Hunt.

Table 10-3: Performance Summary - CATNeuro vs. Thunder - Hits to Opp.

Performance Summary - CATNeuro vs. Thunder Opp.		
Average hits to opp. per round		
	Stag-Hunt	WTD
Avg hits to opp. / round	10.48	9.38
Standard Deviation	4.25	3.91
Two-sample t-test p value	**0.0003** (samples=360)	

This is not case for hits received. There is no significant difference between Stag-Hunt and WTD models for hits received from opponent Thunder.

Thunder was programmed by a human expert. It contains explicit knowledge of game rules and can exploit specification situations in the game. Conversely, Jerry Mizuno is mostly algorithmically driven with a combination of Fuzzy Logic and K-nearest neighbors (KNN) methods (Chu & Thawonmas, 2017). It would be very hard for a purely AI driven approach to master the game unless much more compute resources are deployed. As a reference, Deep Mind's, Alpha Go used 40 days of training time to achieve a critical performance breakthrough (Deep Mind, 2017). Even then, the system was given the rules of Go in the form of code – the AI did not learn the rules by itself.

Table 10-4: Performance Summary - CATNeuro vs. Thunder - Hits received from Opp.

Performance Summary - CATNeuro vs. Thunder Opp.		
Average hits received from opp. per round		
	Stag-Hunt	WTD
Avg hits recv. from opp. / round	36.74	36.78
Standard Deviation	6.28	7.18
Two-sample t-test p value	0.9384 (samples=360)	

Each game has three rounds. The hits to/received-from results on a per round basis are given in Table 10-5 for Jerry Mizuno and Table 10-6 for Thunder.

Table 10-5: Hits by Round – Jerry Mizuno Opp.

Hits by Round – Jerry Mizuno Opp. Avg (Min, Max)			
Type	1	2	3
Stag-Hunt hits to opp.	26 (17,43)	26 (16,36)	25 (14,36)
WTD hits to opp.	26 (14,36)	26 (15,41)	25 (15,36)
Hits t-test p-values	0.7146	0.5241	0.2969
Stag-Hunt hits recv. from opp.	19 (12,28)	18 (9,26)	19 (6,27)
WTD hits recv. from opp.	18 (9,27)	19 (12,29)	19 (9,34)
Hits recv. t-test p-values	0.5474	0.6624	0.8028

Table 10-6: Hits by Round – Thunder Opp.

Hits by Round – Thunder Opp. Avg (Min, Max)			
Type	1	2	3

Stag-Hunt hits to opp.	11 (4,25)	11 (2,25)	10 (1,21)
WTD hits to opp.	9 (2,22)	9 (2,19)	9 (2,21)
Hits to opp. t-test p-values	0.0195	0.0157	0.1266
Stag-Hunt hits recv. from opp.	38 (22,53)	36 (23,59)	37 (23,53)
WTD hits recv. from opp.	37 (18,56)	37 (21,56)	37 (23,54)
Hits recv. t-test p-values	0.1948	0.1403	0.9357

The statistical test results for differences in the means between Stag-Hunt and WTD for each round are also included in the tables. Here again Stag-Hunt performs statistically better (on a per round basis) but only for the hits-to-opponent measure.

Another measure is the relative score – it's the score achieved by the CATNeuro player minus that achieved by the opponent. The relative-score distributions for Stag-Hunt and WTD are shown in Figure 10-1 for Jerry Mizuno and Figure 10-2 for Thunder. Y-axis is probability and x-axis is the relative-score. The distributions represent the relative-scores achieved by the Stag-Hunt and WTD models, respectively, in the games played. The title of the chart also shows the means and the t-test for the differences between the means. For Thunder, at 17% probability of the means being the same, it is not as significant as the p-value for hits-to-opponent metric (Table 10-3) but it is still meaningful. With additional samples, the p-value should indicate better significance. As-is, the relative-score is additional confirmation that when facing the stronger opponent, the Stag-Hunt

derived models perform better than WTD derived ones. For reference, equivalent data for Jerry Mizuno does not show statistical significance (Figure 10-1). Both approaches did well against the AI model.

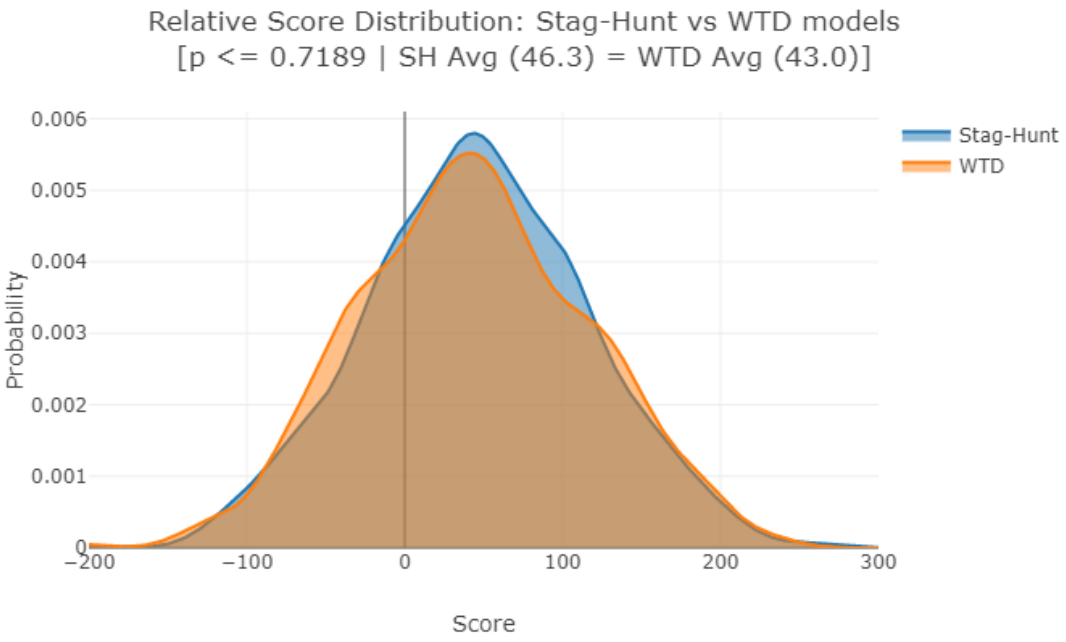


Figure 10-1: Relative-score density plot for Stag-Hunt and WTD when playing Jerry Mizuno

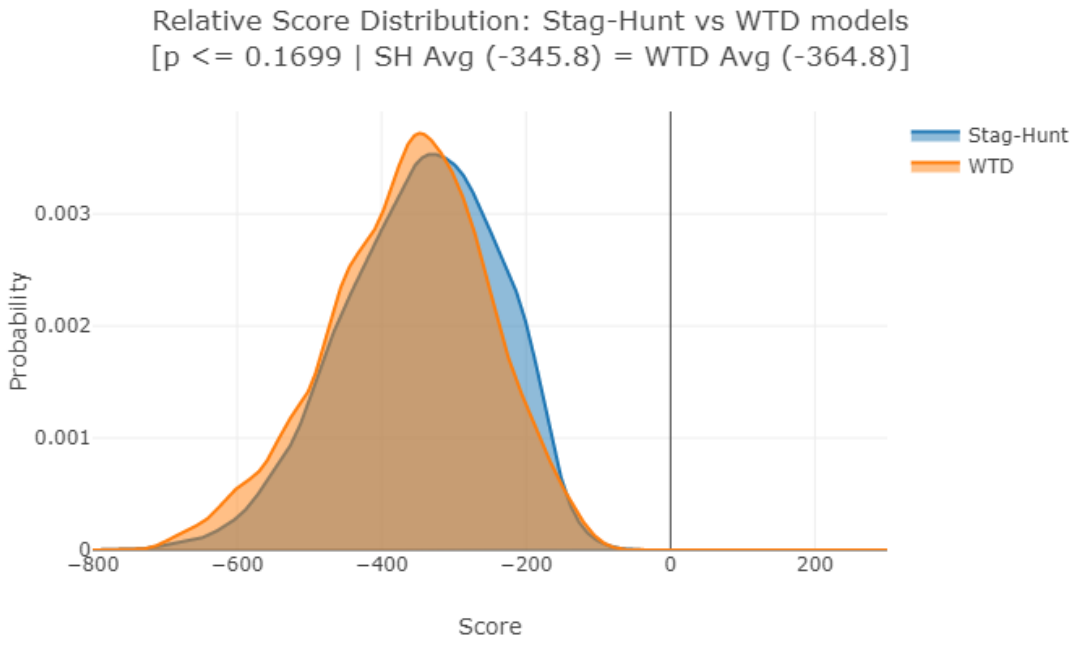


Figure 10-2: Relative-score density plot for Stag-Hunt and WTD when playing Thunder

10.1.2 Best Model Performance Results for Jerry Mizuno

Best models for Stag-Hunt and WTD when playing Jerry Mizuno AI are shown in tables Table 10-7 and Table 10-8. The best models are described in terms of:

a. The number of hits to opponent

b. The relative score (player score – opp. score)

The associated graphical depictions of the neural network models for (a) are given in Figure 10-3 for WTD and Figure 10-4 for Stag-Hunt. And for (b) the corresponding models are in Figure 10-5 for WTD and Figure 10-6 for Stag-Hunt.

The relative score is the difference between the score achieved by CATNeuro model minus that for the opponent (Jerry Mizuno in this section). The scores are on a per round basis. As there are three rounds per game and the ten games are played by each model, the statistics are based on a sample size of 30.

Table 10-7: Best Model (hits to opp.) - CATNeuro vs. Jerry Mizuno Opp.

Best Model (hits to opp.) - CATNeuro vs. Jerry Mizuno Opp.		
Average hits to opp. per round		
	Stag-Hunt	WTD
Avg hits to opp. / round	26.93	27.4
Standard Deviation	4.71	4.1
Two-sample t-test p value	0.6837 (samples=30)	

Table 10-8: Best Model (relative-score) - CATNeuro vs. Jerry Mizuno Opp.

Best Model (relative-score) - CATNeuro vs. Jerry Mizuno Opp.		
Relative Score = player score – opp. score		
	Stag-Hunt	WTD
Relative score	25.5	10
Standard Deviation	72.57	89.3

Two-sample t-test p value	0.4636 (samples =30)

The results for the best models, against Jerry Mizuno, do not show significant differences between Stag-Hunt and WTD. They both do equally well. For the relative-score model, (Table 10-8) seemingly Stag-Hunt average is quite a bit better than WTD but the difference is not statistically significant.

In terms of the graphical models that achieve the best score by hits-to-opponent (Figure 10-3 for WTD and Figure 10-4 Stag-Hunt), the WTD model is much larger and deeper whereas the Stag-Hunt is smaller and shallower. This is consistent with the aggregate statistics for model sizes and depth (discussed later in section 10.4); Stag-Hunt models tend to be smaller.

The case is different for best models in terms of relative-score (Figure 10-5 for WTD and Figure 10-6 for Stag-Hunt). Here the Stag-Hunt model is larger and deeper than the equivalent WTD model. Hits-to-opponent measures the offensive posture of the models but relative-score measures the balance between offence and defense. A larger model make sense in that it will retain more of the information available in the training data – especially if the training loss is also lower, which is the case here.

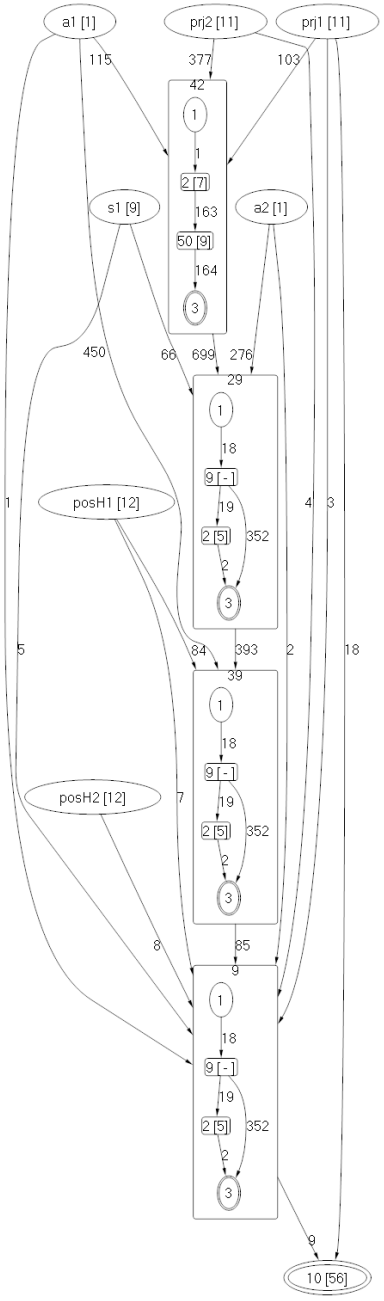


Figure 10-3: WTD vs. Jerry Mizuno best model by hits to opp. [Training Loss:1.86, Tunable Parms:8620, Nodes:24, Conns=34, Depth=17]

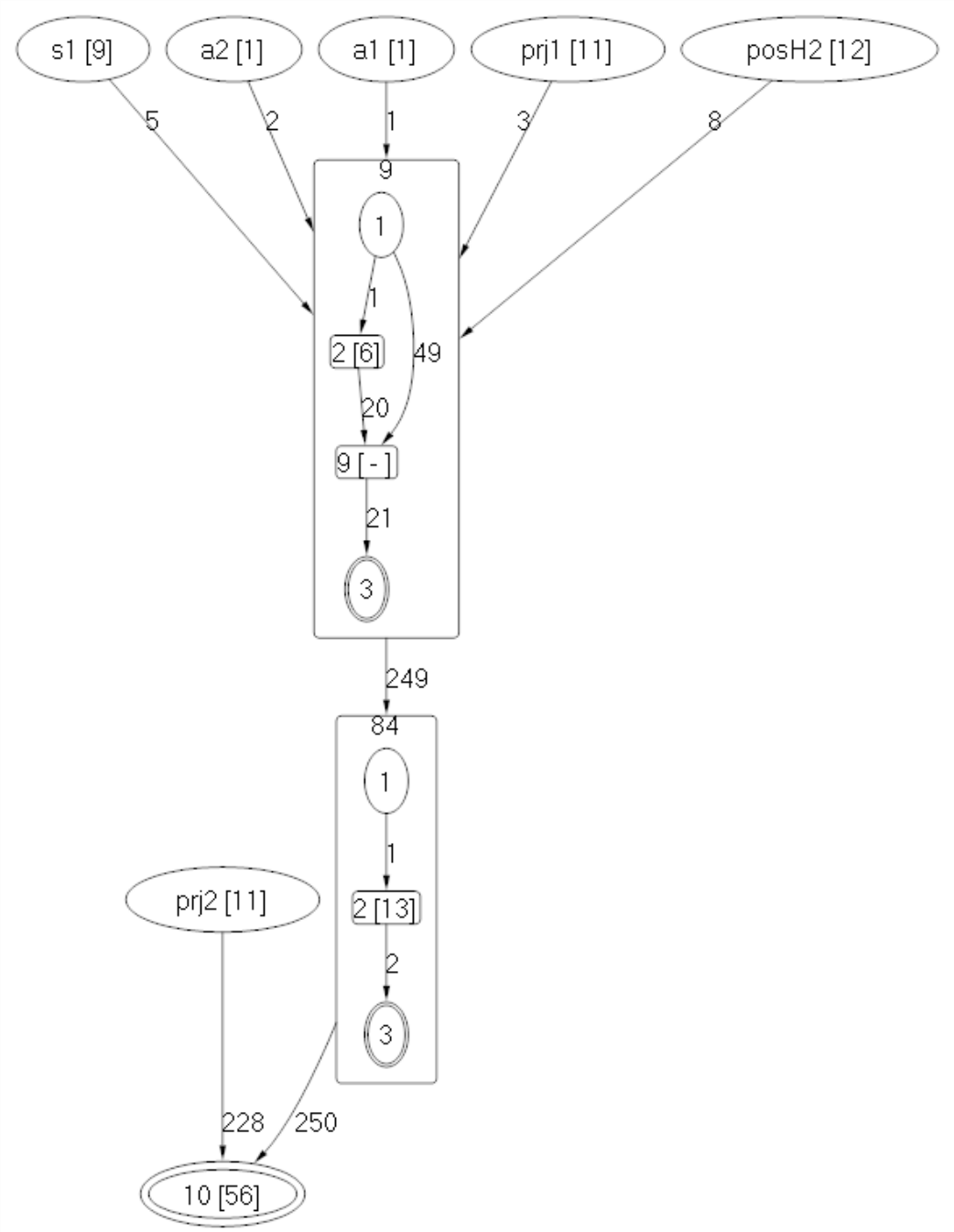


Figure 10-4: Stag-Hunt vs. Jerry Mizuno best model by hits to opp. [Training Loss:1.88, Tunable Parms:2293, Nodes:14, Conns=14, Depth=8]

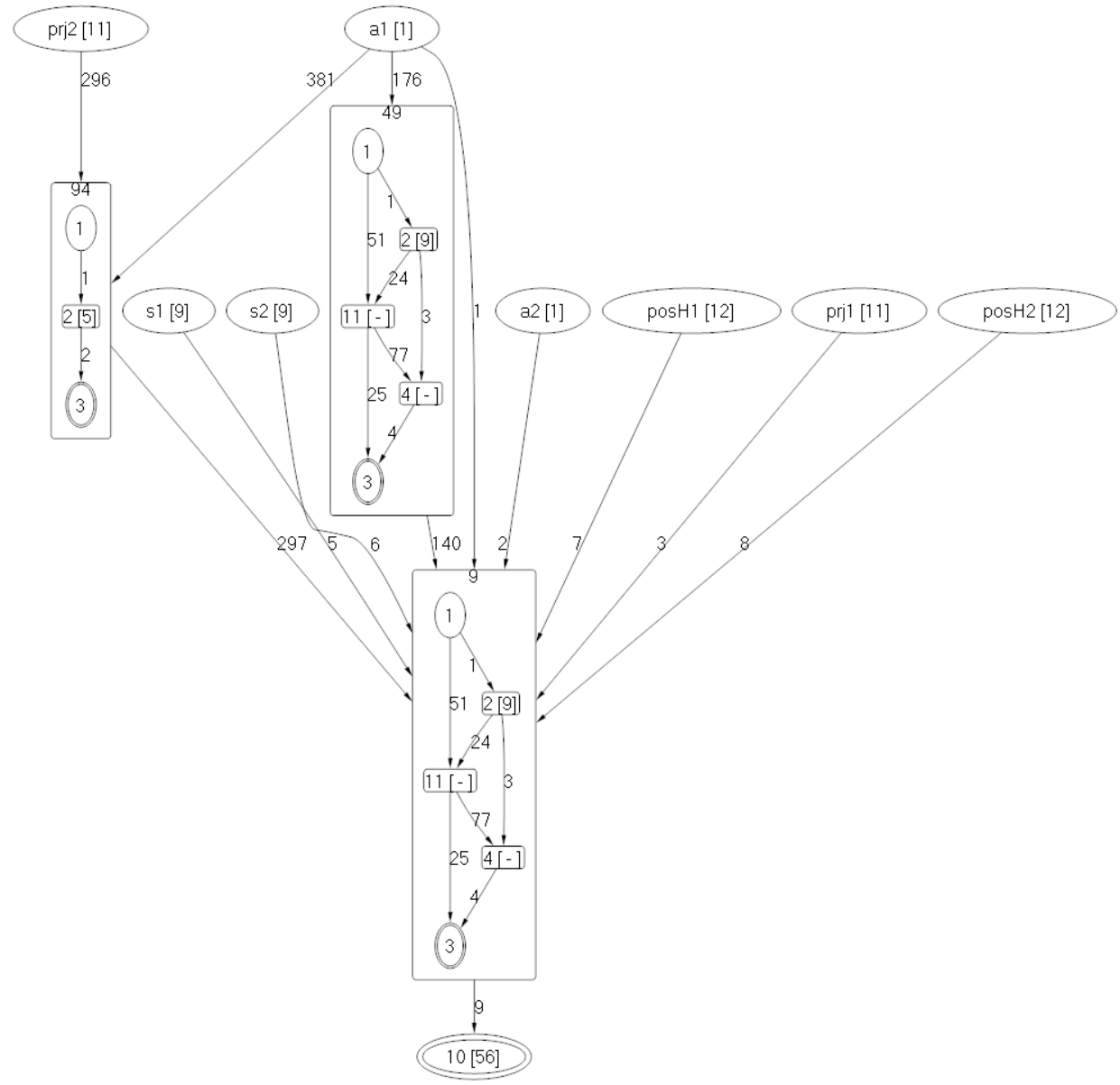


Figure 10-5: WTD vs. Jerry Mizuno best model by relative-score [Training Loss:1.85, Tunable Parms:7744, Nodes:22, Conns=29, Depth=11]

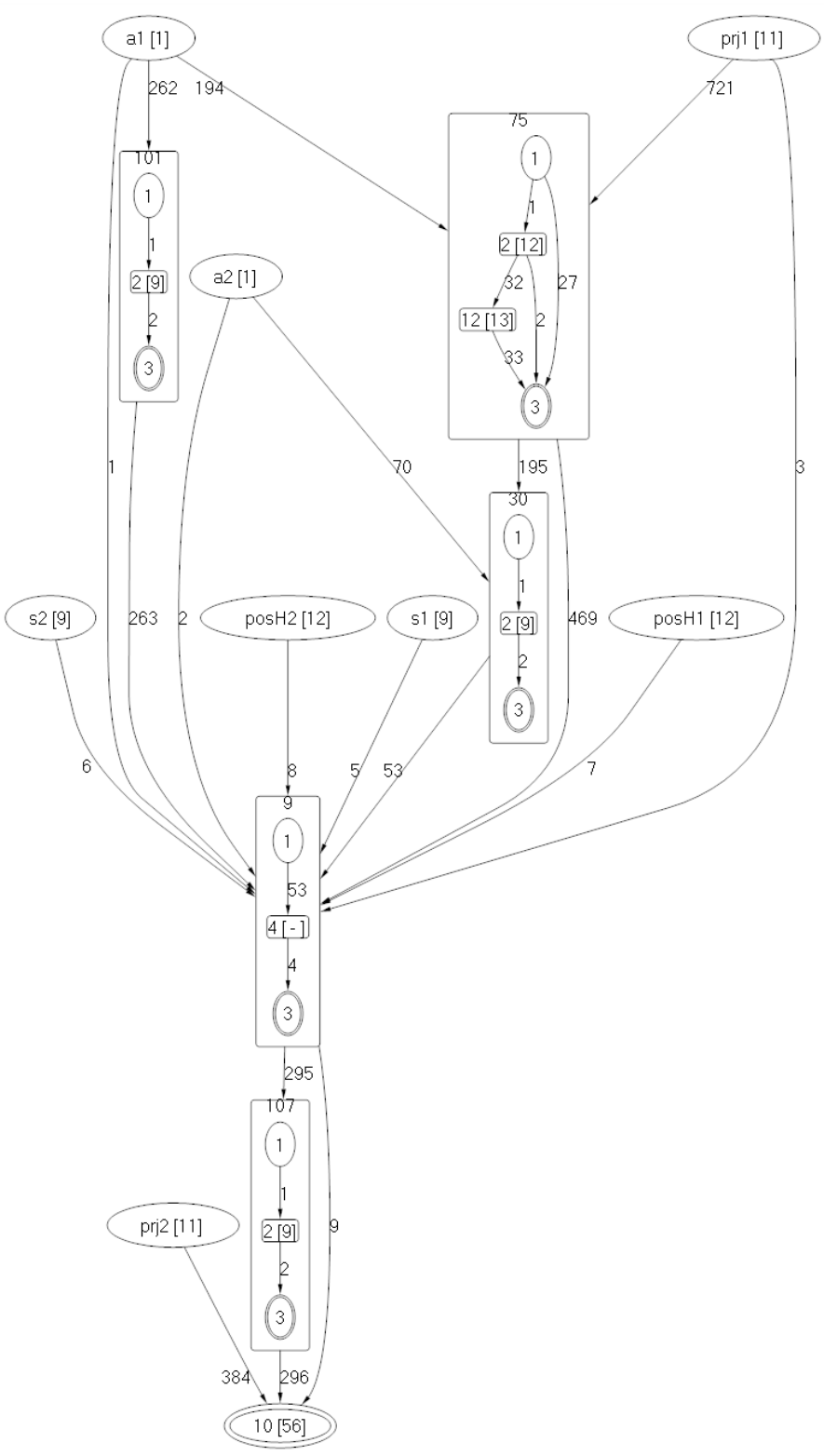


Figure 10-6: Stag-Hunt vs. Jerry Mizuno best model by relative-score [Training Loss:1.84, Tunable Parms:9636, Nodes:25, Conns=32, Depth=14]

10.1.3 Best Model Performance Results for Thunder

The previous section compared the best model performance results and graphical model structures for Jerry Mizuno a benchmark AI supplied with FightingICE. This section discusses equivalent results models for Thunder, the 2018 champion.

Results for the best models by hits-to-opponent are given in Table 10-9 and those for relative-score are in Table 10-10.

Table 10-9: Best Model (hits to opp.) - CATNeuro vs. Thunder Opp.

Best Model (hits to opp.) - CATNeuro vs. Thunder Opp.		
Average hits to opp. per round		
	Stag-Hunt	WTD
Avg hits to opp. / round	11.6	11.17
Standard Deviation	3.52	3.49
Two-sample t-test p value	0.6339 (samples=30)	

Table 10-10: Best Model (relative-score) - CATNeuro vs. Thunder Opp.

Best Model (relative-score) - CATNeuro vs. Thunder Opp.		
Relative Score = player score – opp. score		
	Stag-Hunt	WTD

Relative score	-395.07	-394.37
Standard Deviation	126.03	101.87
Two-sample t-test p value	0.9812 (samples=30)	

The tables show that for the best models produced by Stag-Hunt and WTD, there are no significant differences between the equivalent models. The best models perform equally well. This shows that WTD is able to produce good models but its less consistent than Stat-Hunt as borne out by the aggregate results discussed earlier (Figure 10-2).

The corresponding graphical models are in Figure 10-7 (WTD for hits to opp.); Figure 10-8 (Stag-Hunt hits to opp.); Figure 10-9 (WTD for relative-score); and Figure 10-10 (Stag-Hunt for relative-score).

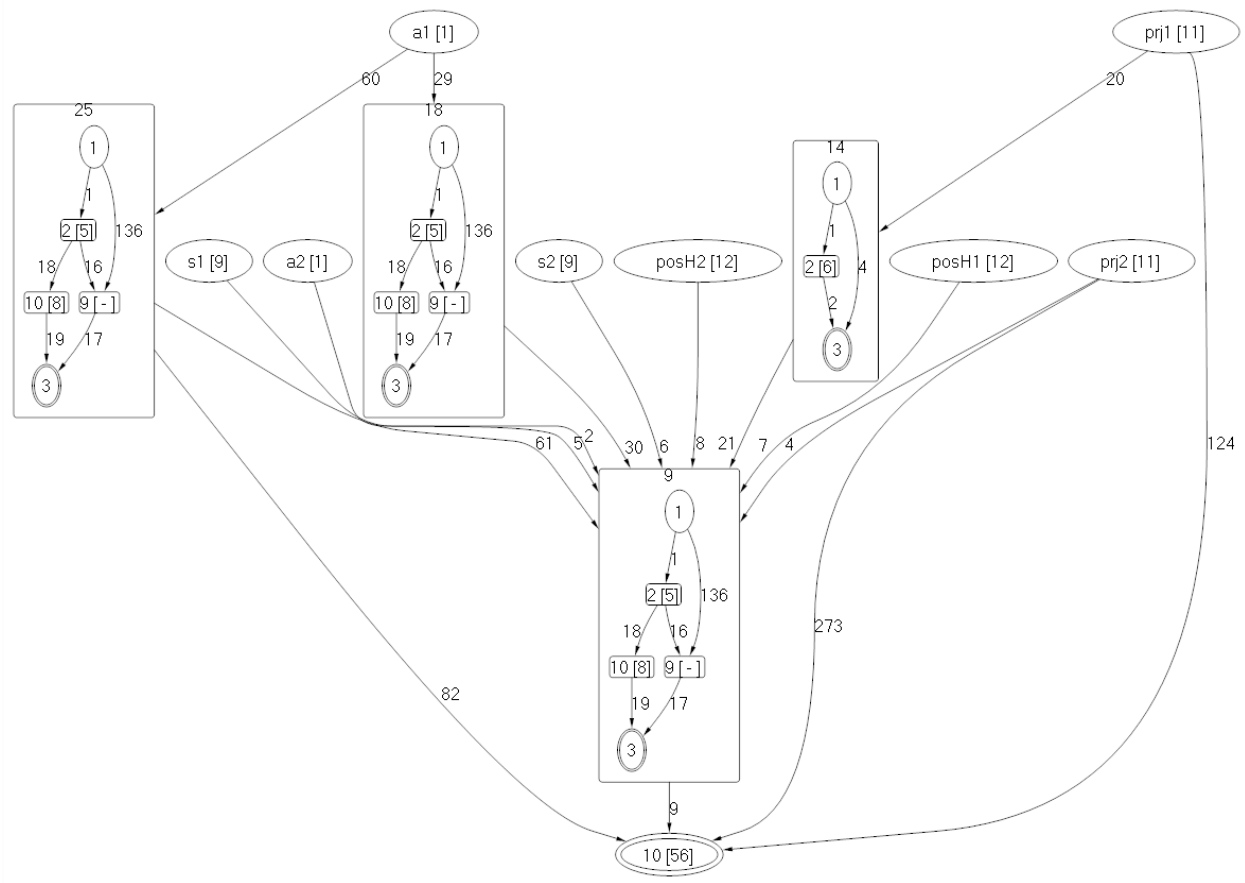


Figure 10-7: WTD vs. Thunder best model by hits to opp. [Training Loss:1.91, Tunable Parms:9619, Nodes:27, Conns=37, Depth=9]

The equivalent graphical models for Stag-Hunt are smaller in terms of the number of tunable parameters or weights and deeper than those for WTD. For the hits-to-opponent best models, WTD has 9619 tunable weights and a depth of 9. By contrast, the Stag-Hunt has 9556 weights and depth of 16. These patterns hold true for the relative-score best models. This shows that Stag-Hunt is able to produce smaller (but deeper models) than WTD, that perform equally well against the opponent.

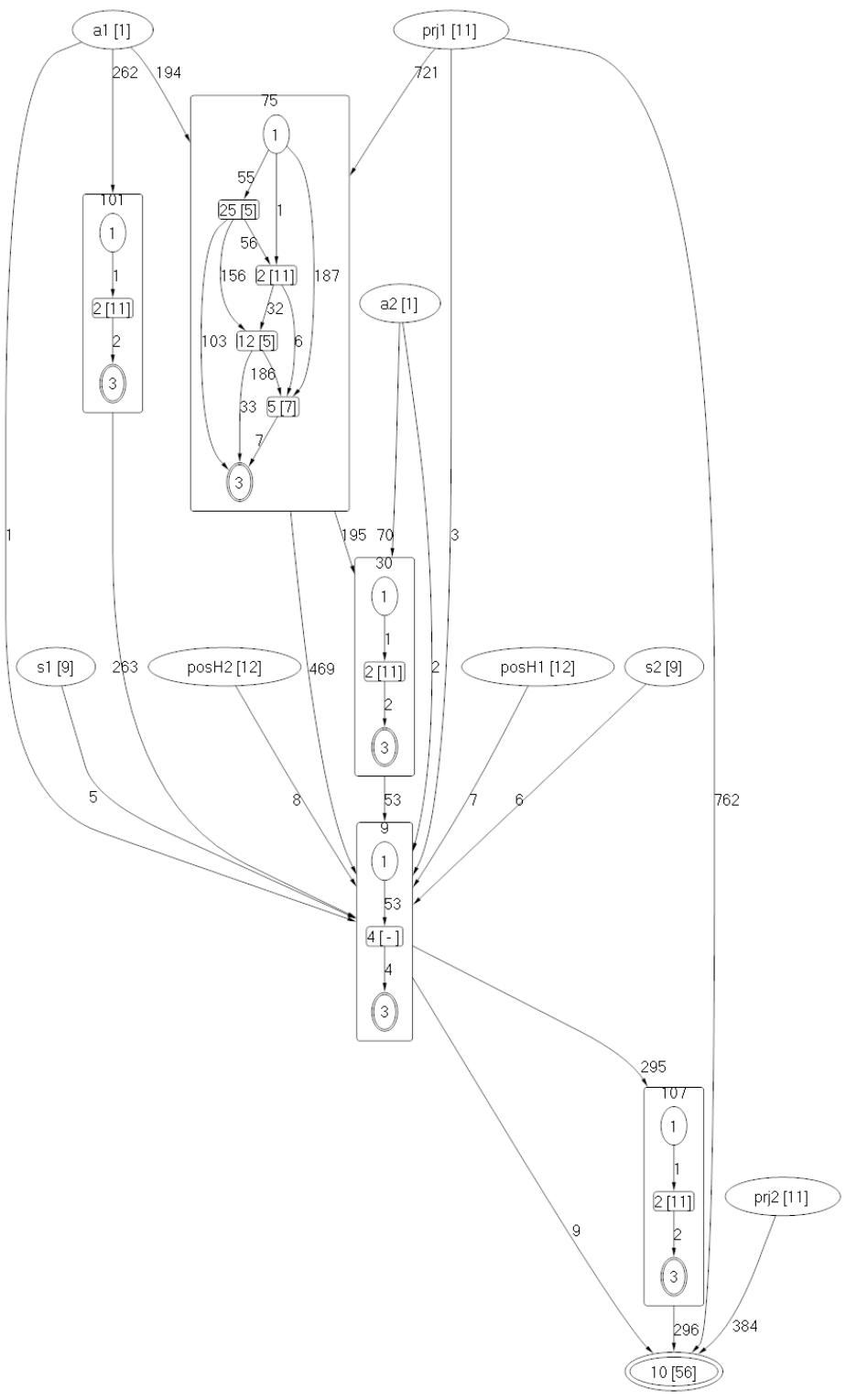


Figure 10-8: Stag-Hunt vs. Thunder best model by hits to opp. [Training Loss:1.85, Tunable Parms:9556, Nodes:27, Conns=39, Depth=16]

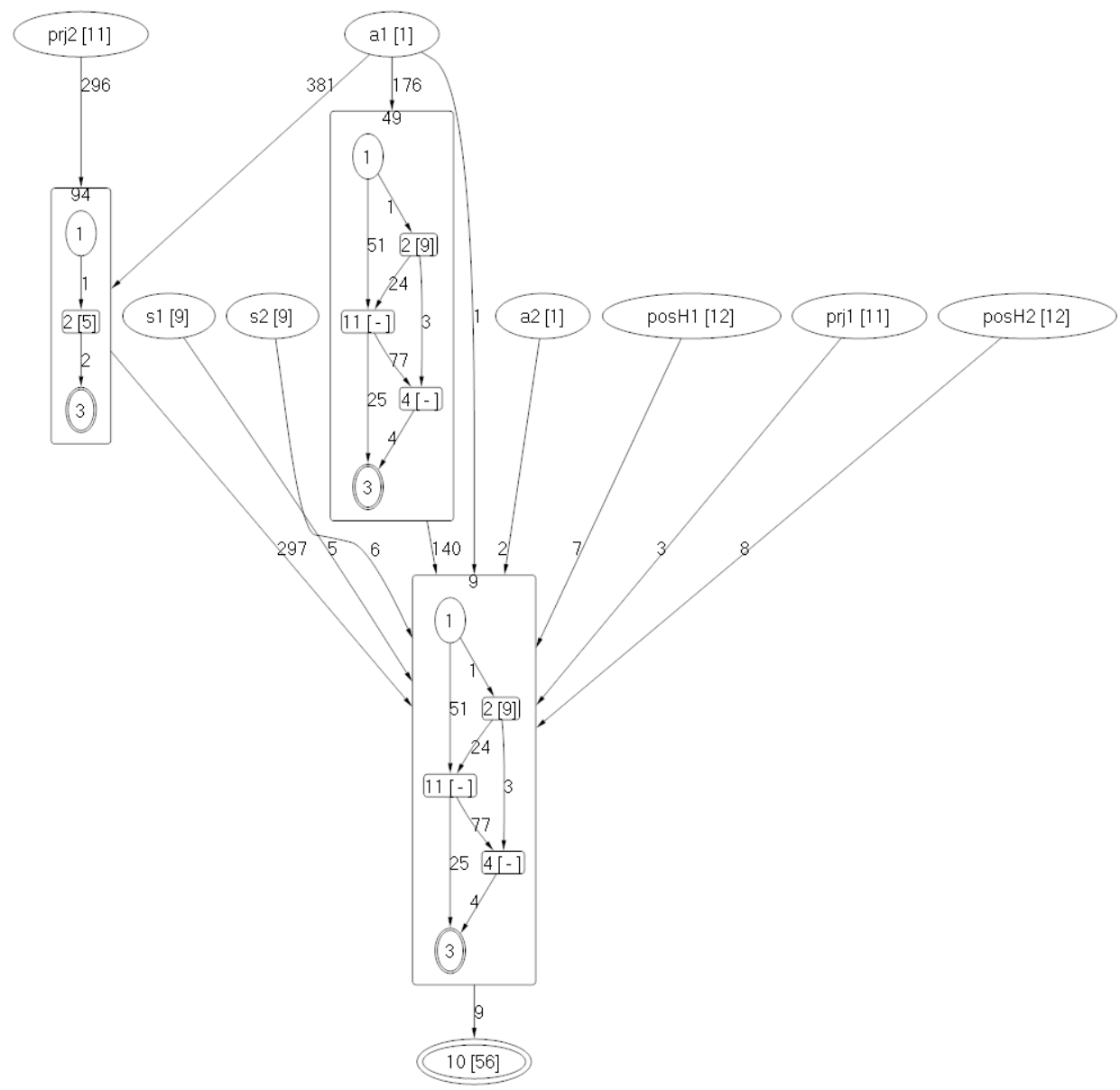


Figure 10-9: WTD vs. Thunder best model by relative-score [[Training Loss:1.85, Tunable Parms:7744, Nodes:22, Conns=29, Depth=11]

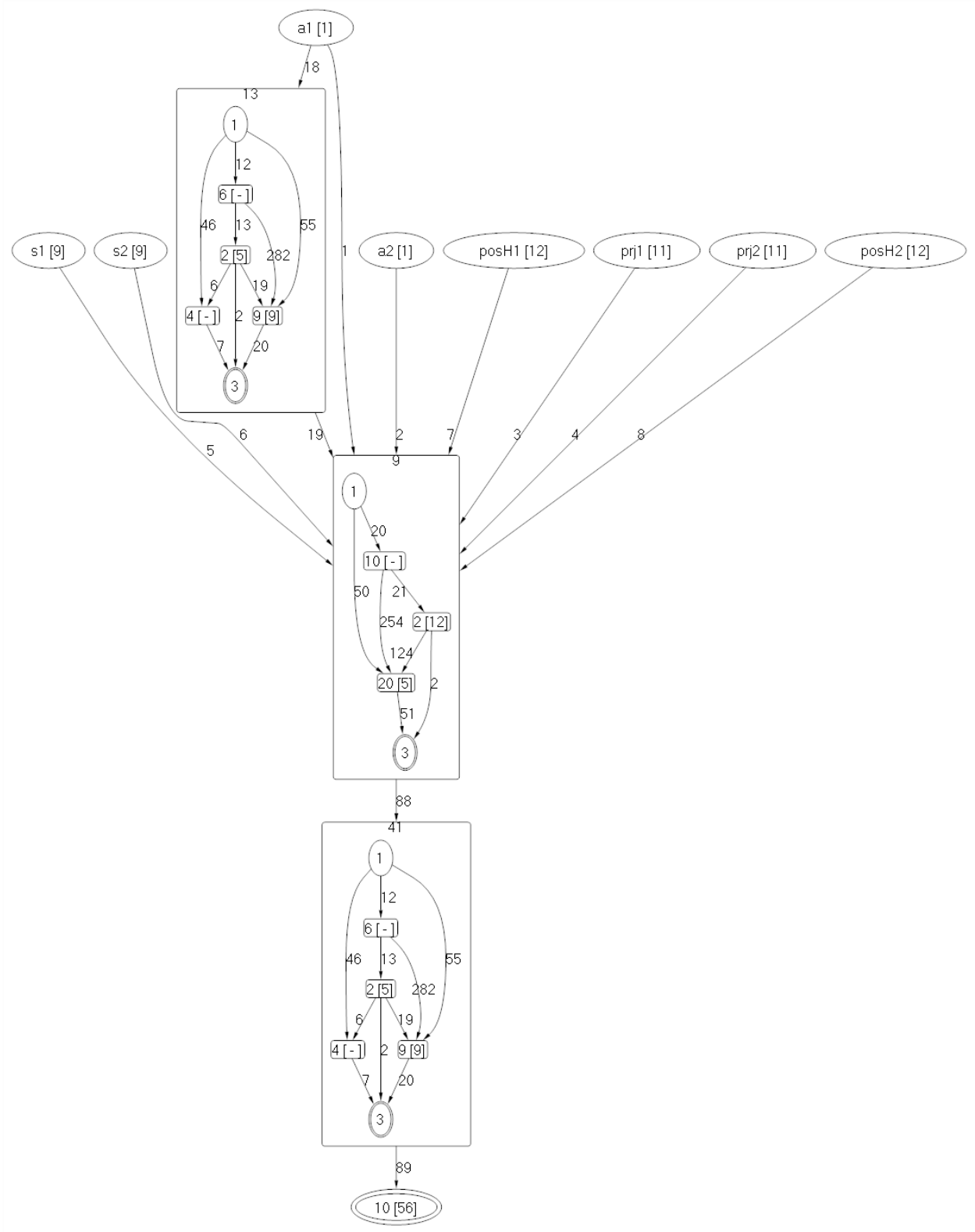


Figure 10-10: Stag-Hunt vs. Thunder best model by relative-score [Training Loss:1.81, Tunable Parms:4925, Nodes:26, Conns=39, Depth=16]

This section compared the aggregate and best model performances of the models produced by the two KD mechanisms. Both the game scores and the graphical model were compared. The next section compares the action distributions of the Stag-Hunt and WTD models used for game play.

10.2 Action Distribution Comparison

As described in section 9.1, the controller in the FightingICE game can chose from of one of 56 actions. The actions can be classified as offense, defense or positioning (tactical movement). Not all actions are necessarily played by all controllers. This section compares the action distributions in a "head-to-head" manner for the Stag-Hunt and WTD models used in game play. Two perspectives can be taken a) the action distribution of the CATNeuro players when they are playing against the same opponent and b) the action distribution of the opponent when playing Stag-Hunt vs when playing WTD.

For both mechanisms, the distribution over actions is quite uneven (i.e. the frequencies of the actions vary considerably) and raw counts or histograms are not very informative. The projection of the same data on a head-to-head basis (see Figure 10-11 for an example) is easier to use for analysis and insights. The action distributions add to the understanding of players' skill in the sense that a more skillful player will display a greater variety of actions. The presentations of the data in this way is labelled "relative action distribution". Note that the data is just what side (top or bottom) plays the action higher number of times. If the corresponding colored bar crosses over the green line in the middle, it means that the count is higher for that side. The charts are relative, i.e. they are do not denote absolute counts; an action may only be played a few times by each player or 100's of times during the course of a round in the game. This limits the usefulness of the

charts but even so some insights can be obtained. Such charts make the comparison easier as the frequency of actions between the two players can be directly compared.

The relative lengths of the colored bars can be used to visually judge the relative difference between the counts of the corresponding action. However, a simpler metric is the count of bars of one color that are longer than opposing bars of the other color. Such counts are given in the title area of each relative action distribution chart. Again, referring to Figure 10-11, the number of actions where Stag-Hunt dominates is 29. The corresponding value for WTD is 24. This metric will be referred to several times in the following analysis so its best to give a name. Let it be HAC for "higher action count" so HAC for Stag-Hunt is 29 and that for WTD is 24 in Figure 10-11. The relative action distributions for the Jerry Mizuno and Thunder are discussed separately next.

10.2.1 Action Distributions – Jerry Mizuno

This section discusses the relative action distributions for the CATNeuro players vs. Jerry Mizuno for both a) aggregated across all models; b) for best models for Stag-Hunt and WTD with respect to hits-to-opponent; and c) for best models with respect to relative-score.

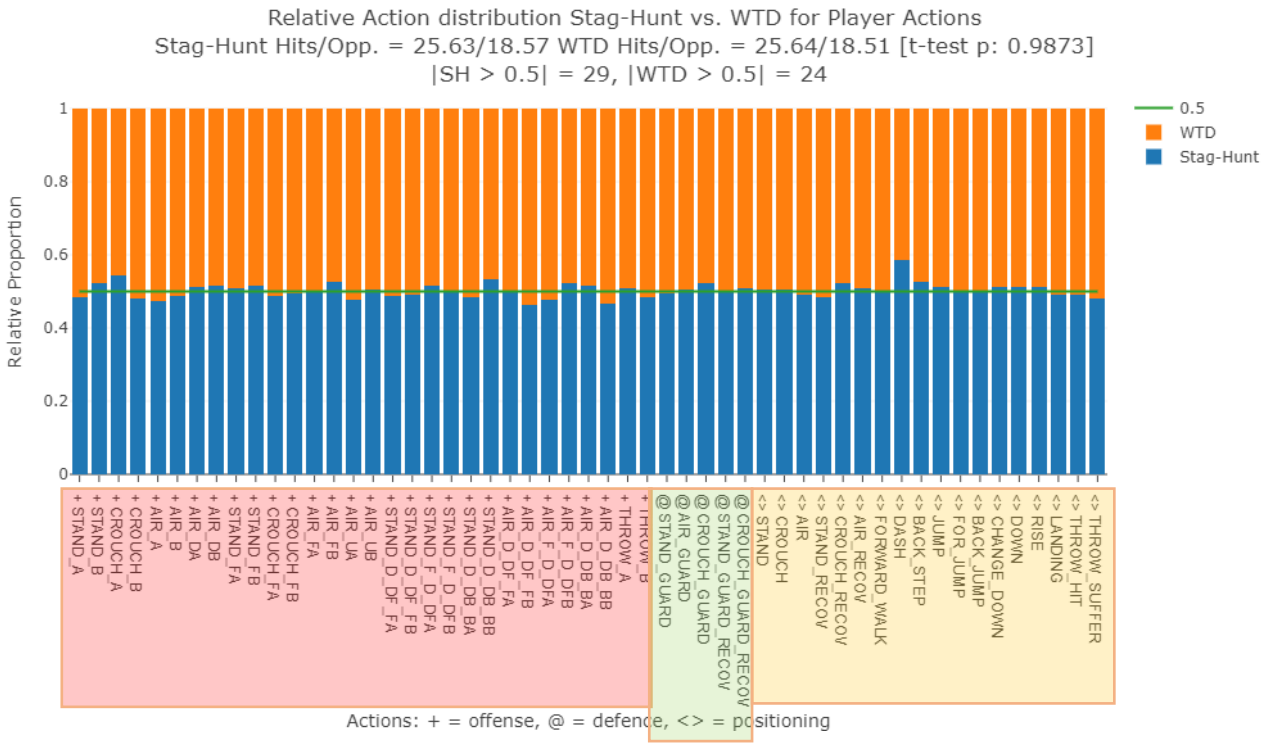


Figure 10-11: Jerry Mizuno - Relative Action distribution Stag-Hunt vs. WTD for Player Actions

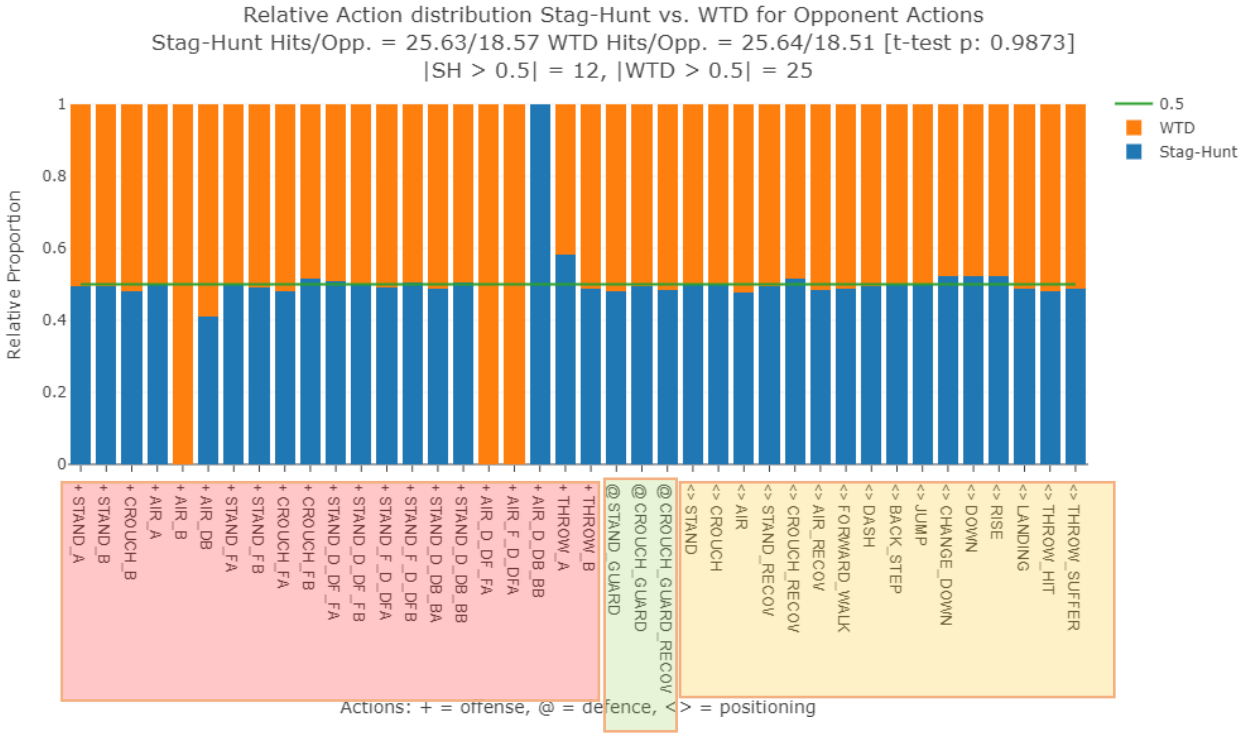


Figure 10-12: Jerry Mizuno - Relative Action distribution Stag-Hunt vs. WTD for Opponent Actions

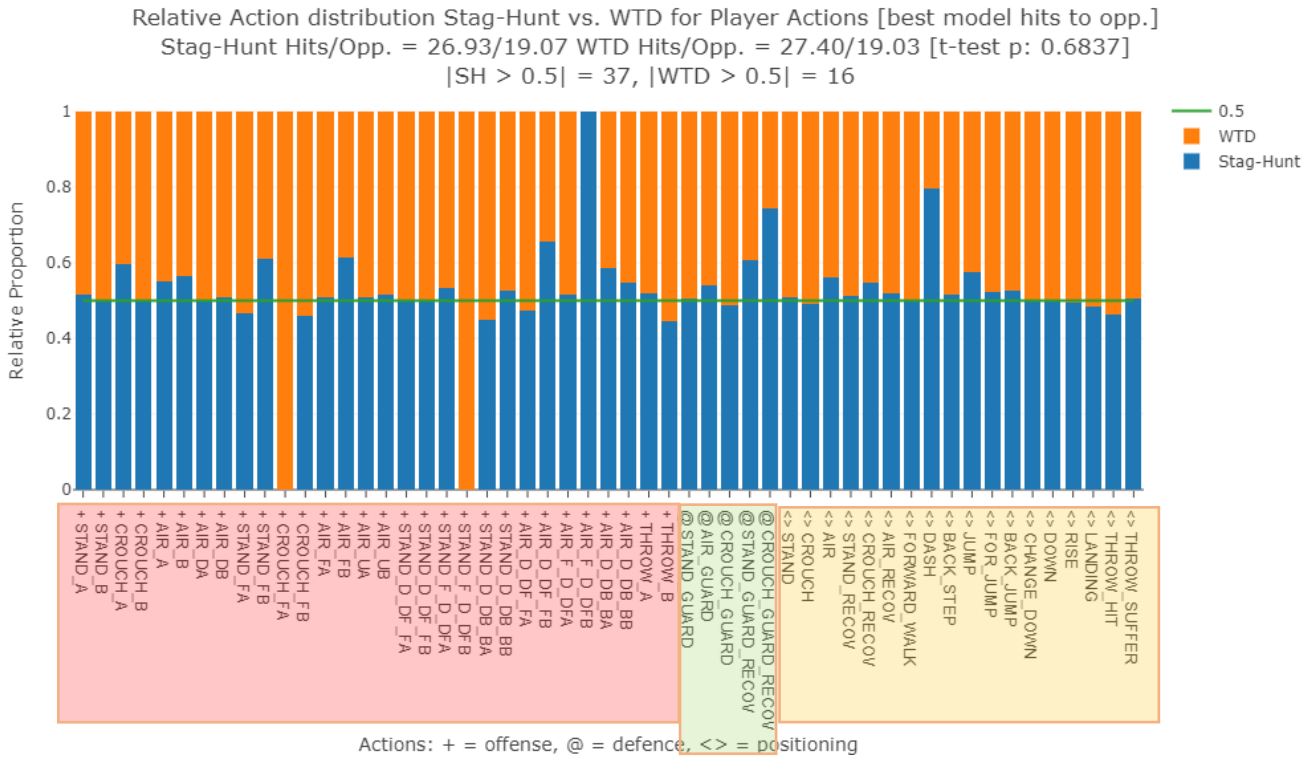


Figure 10-13: Jerry Mizuno - Relative Action distribution Stag-Hunt vs. WTD for Player Actions [best model hits to opp.]

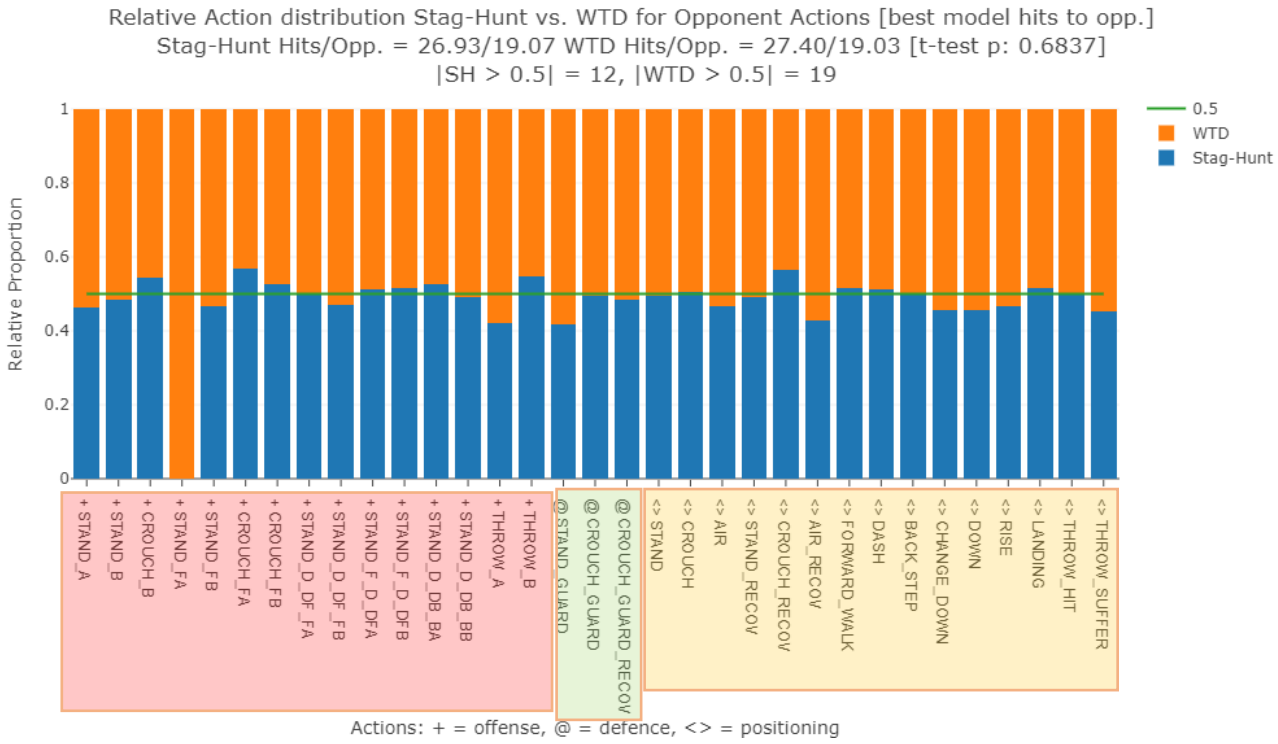


Figure 10-14: Jerry Mizuno - Relative Action distribution Stag-Hunt vs. WTD for Opponent Actions [best model hits to opp.]

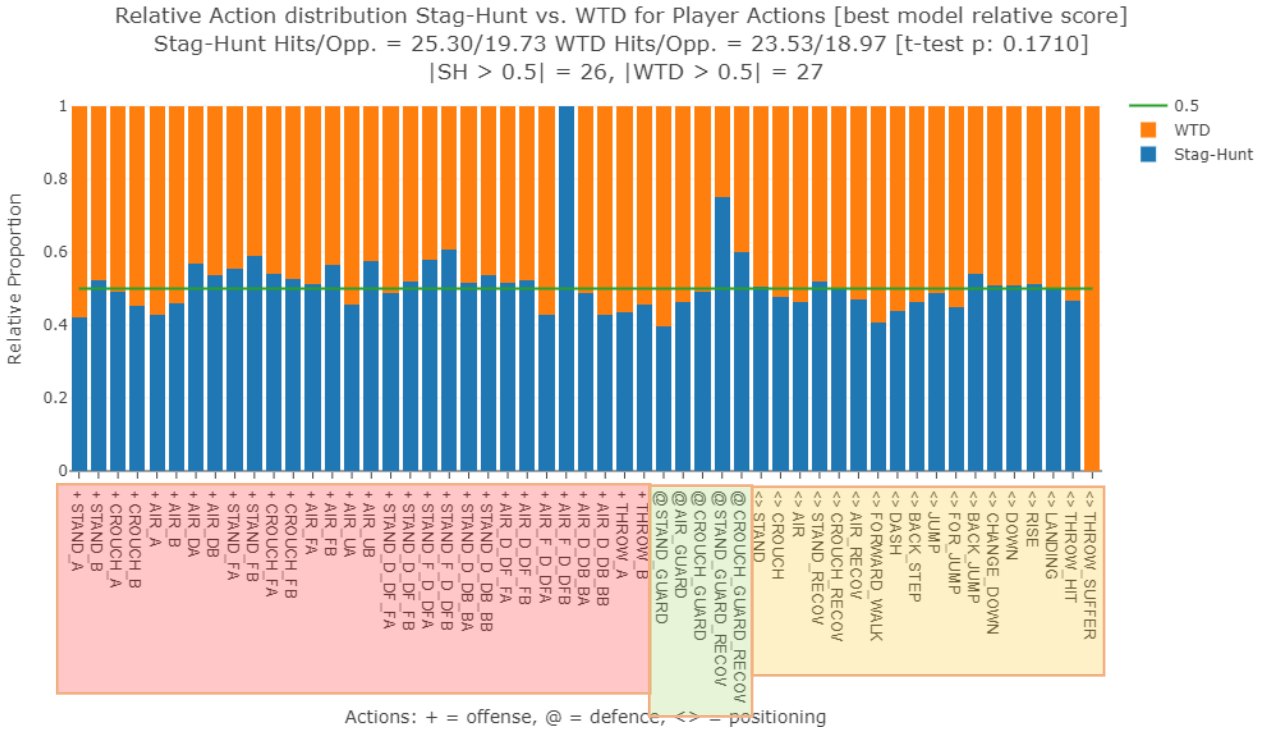


Figure 10-15: Jerry Mizuno - Relative Action distribution Stag-Hunt vs. WTD for Player Actions [best model relative-score]

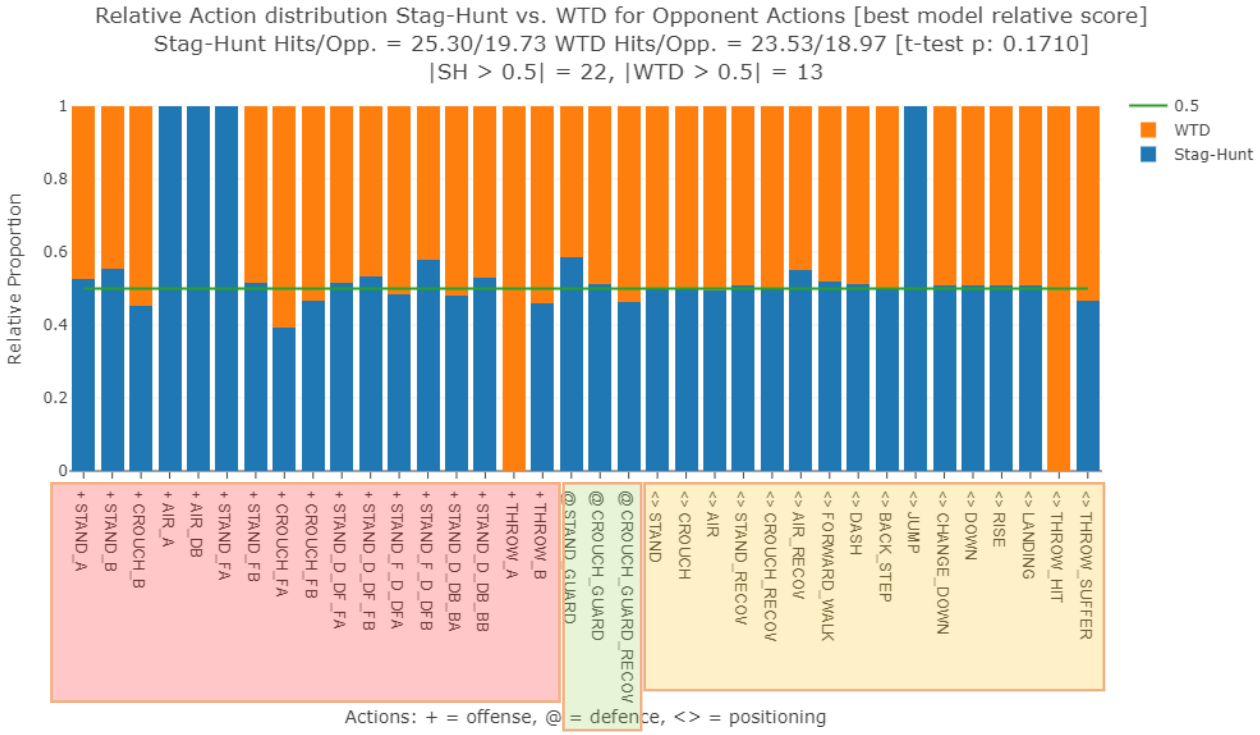


Figure 10-16: Jerry Mizuno - Relative Action distribution Stag-Hunt vs. WTD for Opponent Actions [best model relative-score]

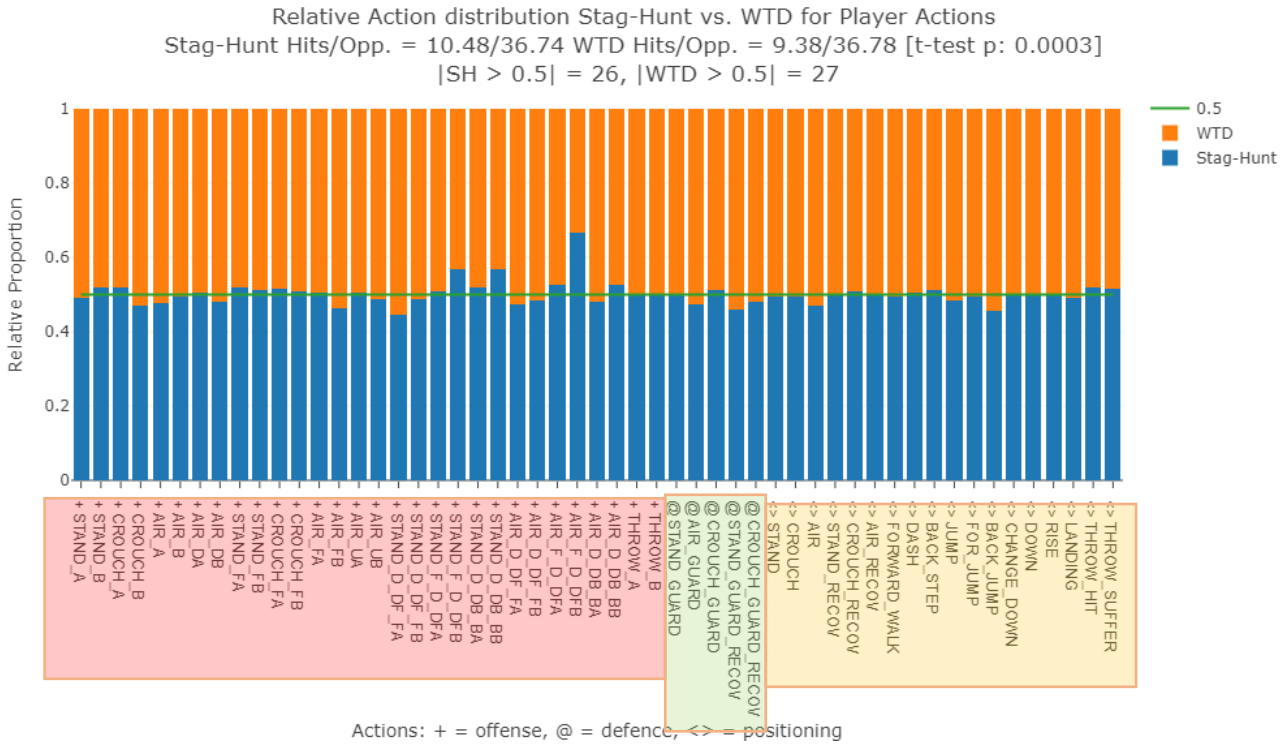


Figure 10-17: Thunder - Relative Action distribution Stag-Hunt vs. WTD for Player Actions

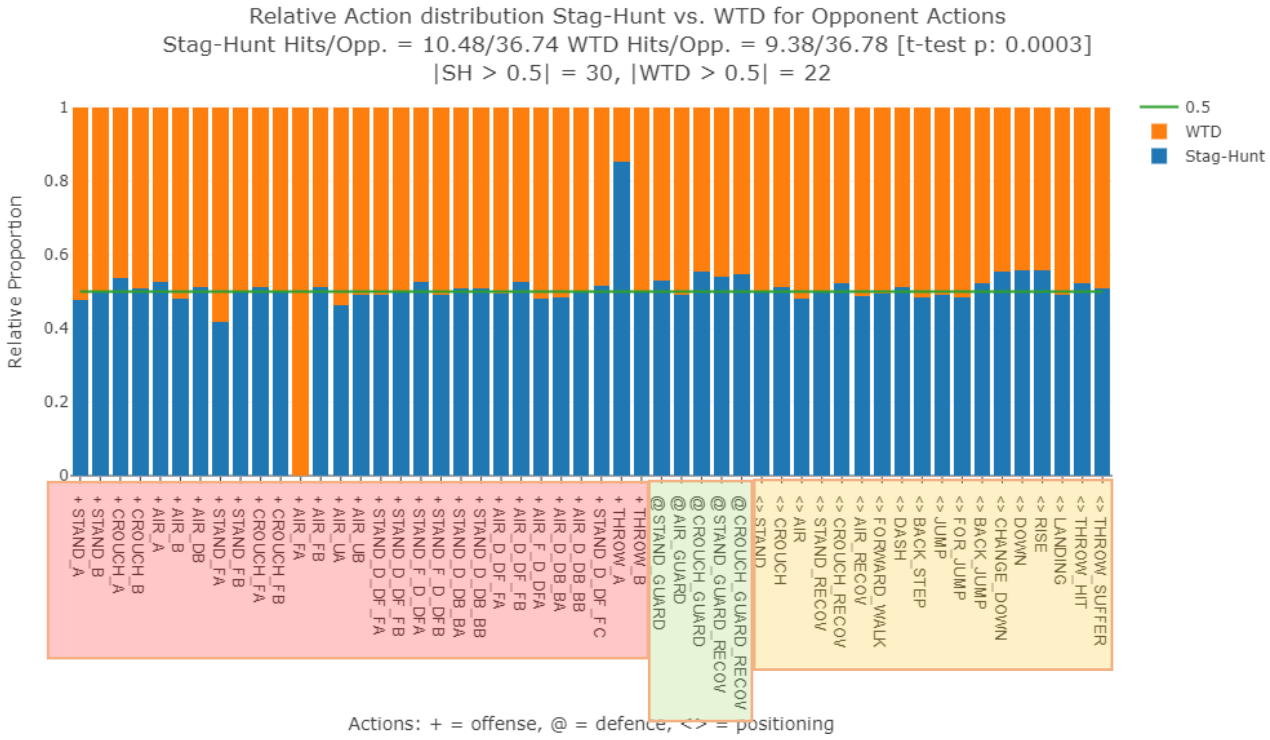


Figure 10-18: Thunder - Relative Action distribution Stag-Hunt vs. WTD for Opponent Actions

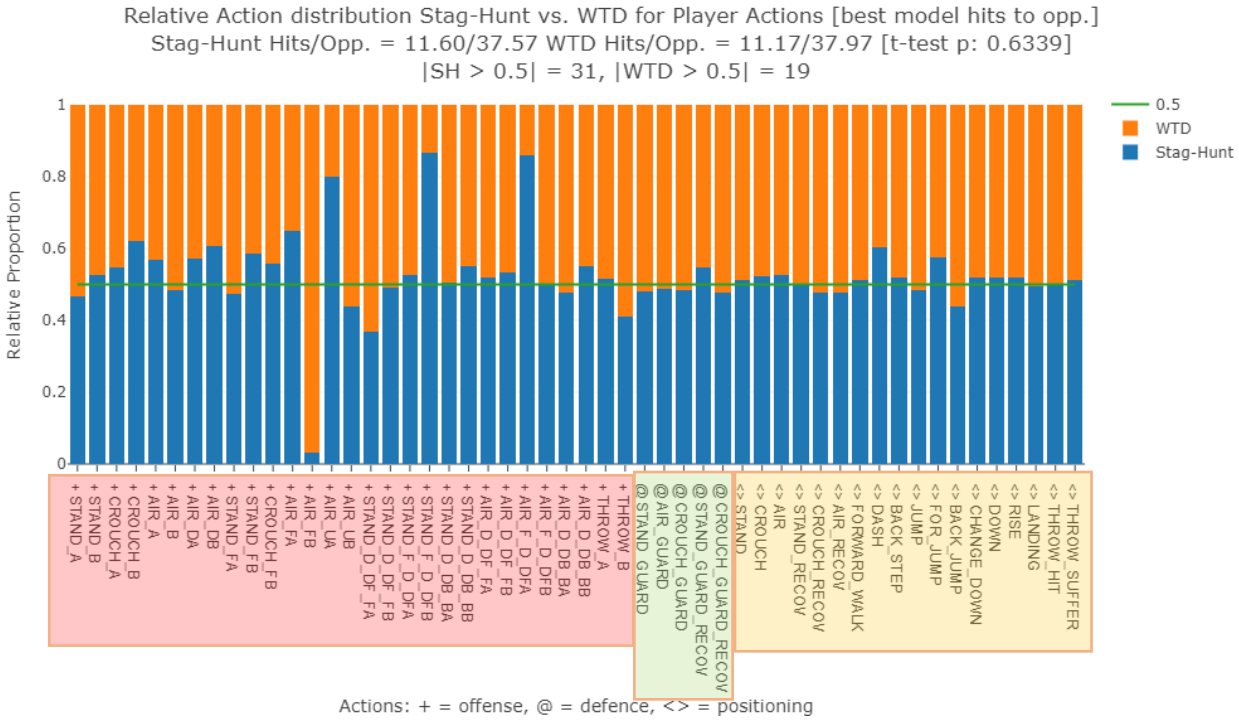


Figure 10-19: Thunder - Relative Action distribution Stag-Hunt vs. WTD for Player Actions [best model hits to opp.]

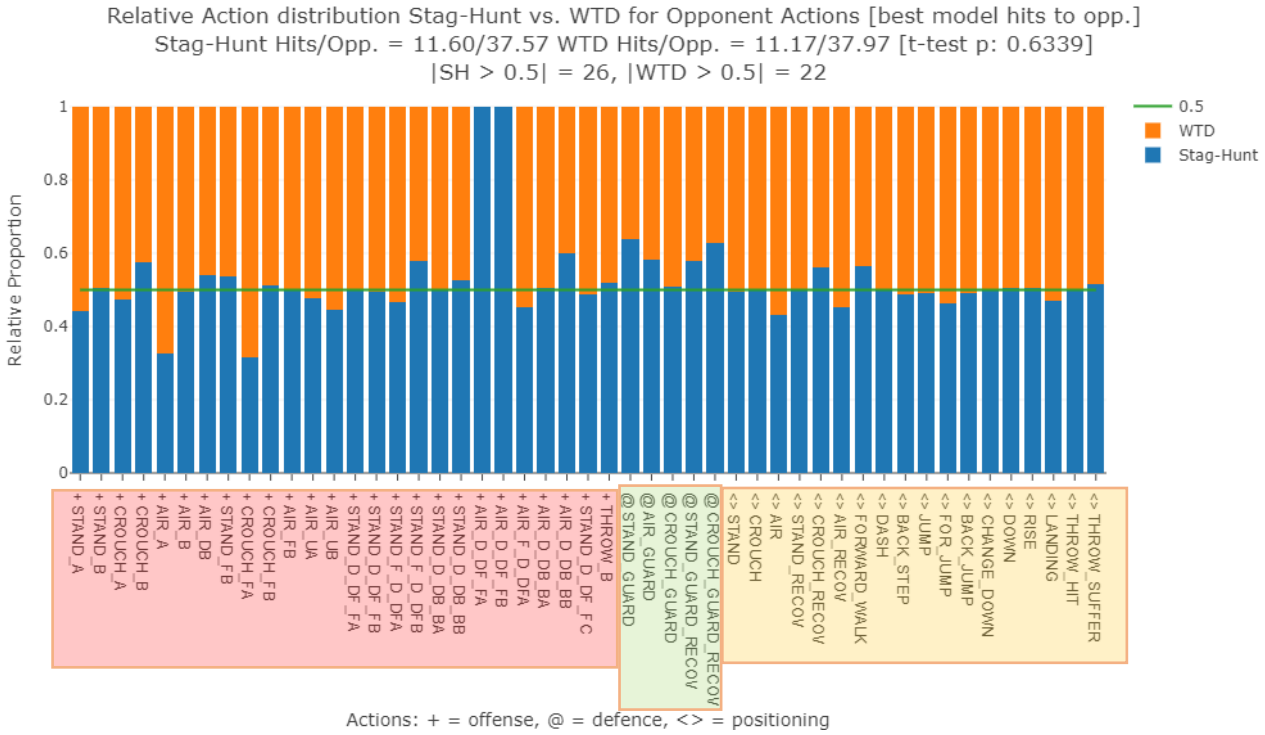


Figure 10-20: Thunder - Relative Action distribution Stag-Hunt vs. WTD for Opponent Actions [best model hits to opp.]

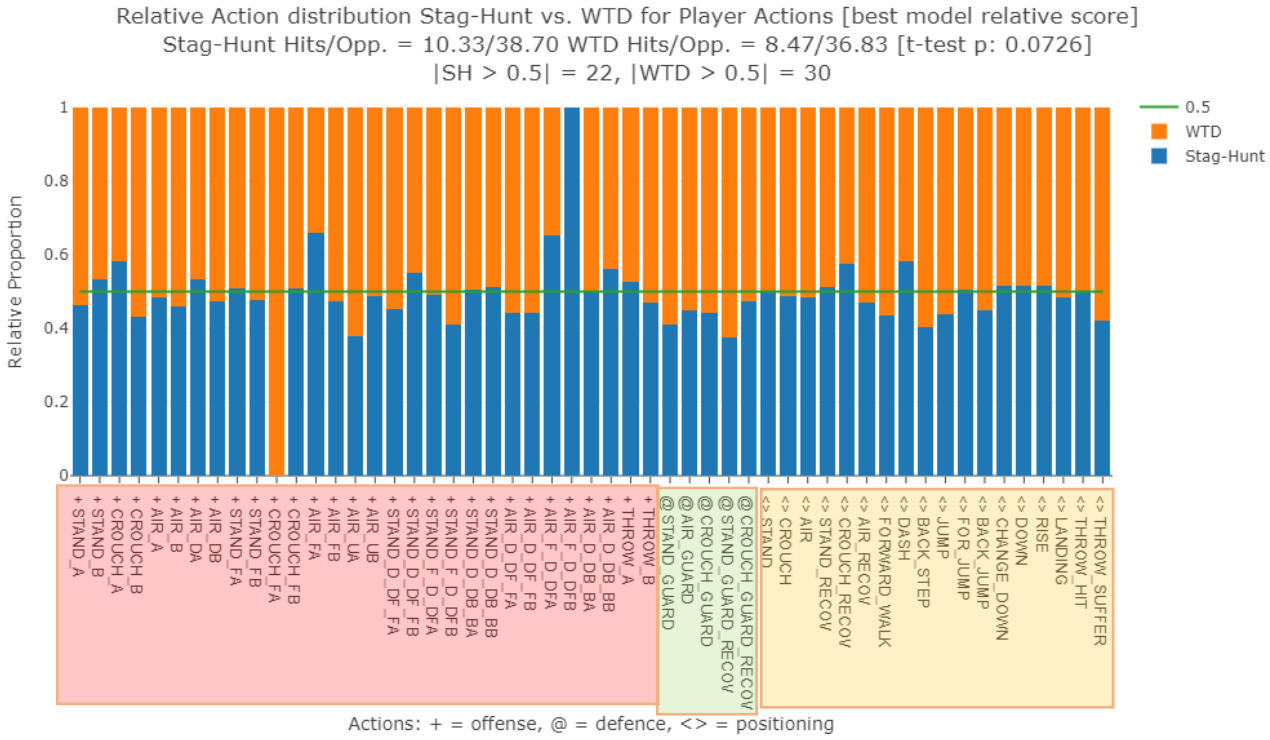


Figure 10-21: Thunder - Relative Action distribution Stag-Hunt vs. WTD for Player Actions [best model relative-score]

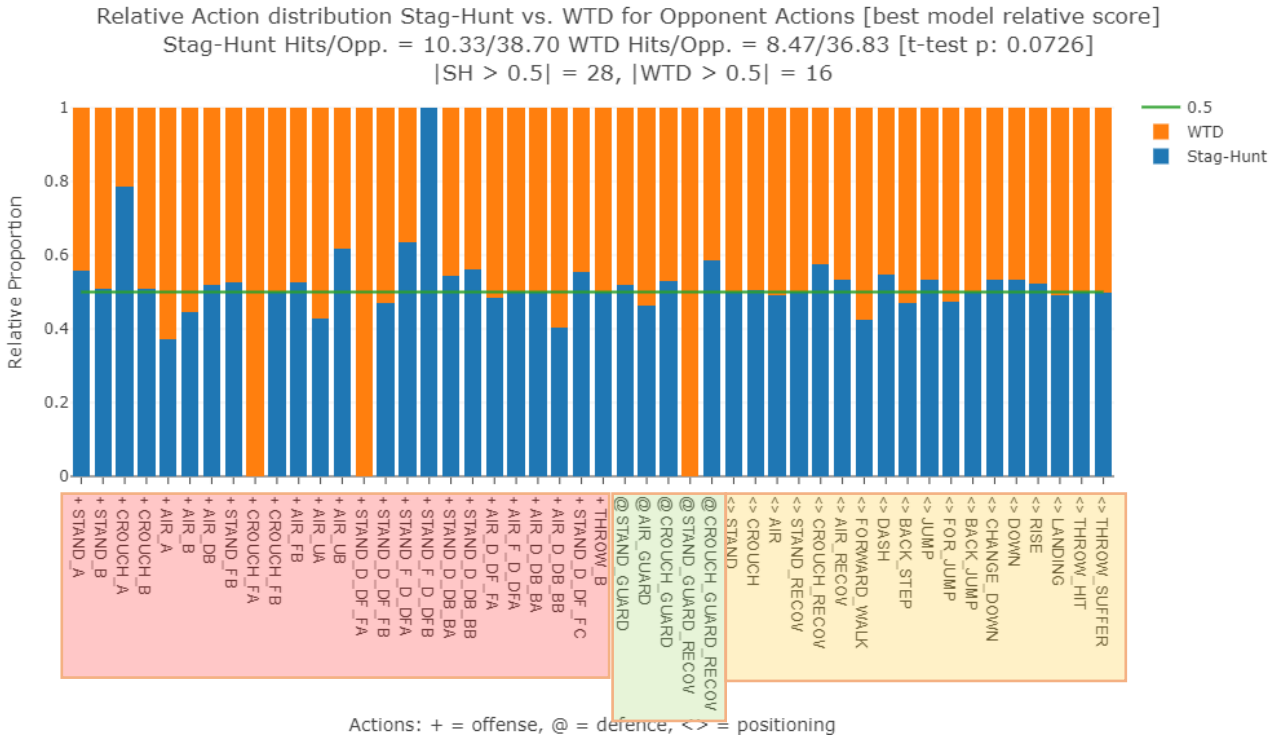


Figure 10-22: Thunder - Relative Action distribution Stag-Hunt vs. WTD for Opponent Actions [best model relative-score]

Figure 10-11 is relative action distribution of Stag-Hunt players vs WTD players, aggregated across all models. By contrast Figure 10-12 shows the relative action distribution of Jerry Mizuno AI when playing Stag-Hunt players vs. when playing WTD players. The titles of the graphs show the number of actions for which the count of actions is higher by KD mechanism, i.e. the HAC values, introduced earlier. For example, in Figure 10-11, the Stag-Hunt and WTD HAC values are 29 and 24, respectively. Together, both players used 29 + 24 = 53 actions (out of the available 56) at least at one point in the games.

It is interesting to note that the opponent, Jerry Mizuno has a HAC (25) that is twice as higher when playing against WTD players than against Stag-Hunt players (12). This indicates that on average Jerry Mizuno uses a greater variety of actions against WTD. In other words, on the whole, WTD forces Jerry Mizuno to be more versatile. Also, the total actions played by Jerry Mizuno is 25+12 = 37. This is quite a bit less than those played by CATNeuro players (53). There are 3 'air' actions that Jerry Mizuno used when playing against WTD that it did not use against Stag-Hunt. The converse is true for only one action.

Looking at Figure 10-11, there seems to be no major differences between WTD and Stag-Hunt however Figure 10-12 shows that, from the opponent's perspective there are discernable differences. Jerry Mizuno is using different strategies (distribution over actions) when playing against Stag-Hunt vs WTD. The dynamics are not captured when looking at just the CATNeuro player distributions but a more complete picture emerges with the combined view of both the players' and opponent's distributions. The overall result for Jerry Mizuno surmises that the models produced by the two KD mechanism tend to learn different ways of playing the game – i.e. the mapping from game-state to action distribution is different.

Figure 10-13 and Figure 10-14 are relative action distributions for the best models in terms of hits-to-opponent. As before, Figure 10-13 is from the CATNeuro players' perspective and Figure 10-14 from the opponent's. The 'players action distributions clearly show that two models are using different strategies – defense and positioning actions are higher for Stag-Hunt and the mixes are different for offense between the two. The HAC for Stag-Hunt (37) is also much higher than that for WTD (16). If anything, the WTD model is stresses offense over defense as it uses two attack types that Stag-Hunt never uses. The slightly higher 'hits score for WTD is maybe an indication of that. Figure 10-13 is Jerry Mizuno's actions when playing against the two KD mechanisms. The most interesting aspect is that the combined HAC is 12+19=21. This means that the, against the best models, Jerry Mizuno only uses 21 out of the 56 available actions. Since these are the most aggressive models it is quite likely the models are pinning the opponent down and therefore the opponent can respond with a limit set of actions.

Figure 10-15 and Figure 10-16 are the corresponding charts for the best models by relative-score. Relative-score is the difference between a CATNeuro player's and the opponent's scores. First off, the score difference between Stag-Hunt and WTD is approaching statistical significance in favor of Stag-Hunt. The p-value is 17% - with more samples it could be reduced further. Then, visually, it can be seen in both charts that strategies followed are all different. Stag-Hunt is more aggressive as the HAC for just the offensive actions is higher. Jerry Mizuno (Figure 10-16) is also more aggressive against Stag-Hunt as there are 3 offense actions that it uses against Stag-Hunt but not against WTD (these are "AIR_A", "AIR_DB", "STAND_FA"). Also, Jerry Mizuno uses the "JUMP" action against only Stag-Hunt – most likely to avoid getting hit when on the ground. This indicates that Stag-Hunt is more engaged with the opponent as compared to WTD.

Note that “AIR_” prefix (for offense actions) is for attacks done when the character is in the air. These could be different kinds of kicks, or punches. The exact sequence of moves varies by the character type; of which there are two - Zen and Garnet. All games were played with Zen on both sides. Similarly, “STAND_” and “CROUCH_” prefixes related to actions while in the standing and crouching states, respectively.

10.2.2 Action Distributions – Thunder

This section performs similar analysis for the Thunder opponent as was done for Jerry Mizuno in the previous section. Figure 10-17 and Figure 10-18 are aggregate-level charts from the players’ and the opponent’s perspectives, respectively. Here Stag-Hunt does statistically better than WTD but that is not really apparent from the action distributions in Figure 10-17; except that Stag-Hunt uses more of the “AIR_F_D_DFB”, “STAND_D_DB_BB” and “STAND_F_D_DFA” attacks (the differences are visually discernable). Form the opponent’s perspective (Figure 10-18), at the aggregate level, the action distributions of Thunder when playing Stag-Hunt vs. when playing WTD are about the same except for two actions. Against Stag-Hunt, Thunder uses “THROW_A” attack much more often and does not seem to use the “AIR_FA” attack. On the whole the HAC value for Stag-Hunt (30) is higher than for WTD (22). This implies that Thunder is forced to display more versatility against Stag-Hunt. In contrast, it was noted earlier that Jerry Mizuno (the weaker opponent) displayed more versatility against WTD (Figure 10-12). Also Thunder uses more types of actions (30+22=52) than Jerry Mizuno (12+25=37) at the aggregate level. This also shows that Thunder is the stronger player.

Figure 10-19 and Figure 10-20 are relative distributions from the players’ and the opponent Thunder’s perspectives, for the best models by hits-to-opponent. The overall performance of Stag-

Hunt and WTD models (Figure 10-19) is not statistically different but the strategies followed are quite different. The HAC score for the Stag-Hunt model (31) is much higher than for WTD (19). This shows that the Stag-Hunt model is relatively more versatile that the WTD one. Comparatively, Stag-Hunt is more offensive, and WTD is more defensive. The HAC value for the just the ***offense*** actions is 17 for Stag-Hunt vs. 9 for WTD. The defense HAC is 1 for Stag-Hunt and 4 for WTD.

From Thunder's perspective (Figure 10-20) it is also apparent that Stag-Hunt is the more aggressive model. The HAC of Thunder's defense actions is 5 to 0 for Stag-Hunt opponent vs WTD; i.e. Thunder is forced to be more defensive when playing against the Stag-Hunt model. The offense HAC for Thunder is about the same against both, overall, although there is a marked difference in some specific actions. This shows that models from both KDs are engaging well with Thunder but Stag-Hunt is trying to land more hits, which Thunder is defending well against.

Figure 10-21 and Figure 10-22 are relative distributions for the best model by relative-score. Here the results are statistically significant in favor of Stag-Hunt. It is somewhat surprising that WTD is dominates in terms of HAC, across the board. This implies that Stag-Hunt is able to win with an overreliance on a few specific, well-timed moves or 'combos'. The aspect of timeliness is not captured in this view of the data. However, a peek into timeliness can be obtained by observing what the opponent is doing.

Figure 10-22 shows that Thunder's HAC against Stag-Hunt (28) is distinctly higher than against WTD (16). Thus, Thunder needs to be more versatile when playing against Stag-Hunt. This indicates that Stag-Hunt is making more timely moves (or using combos) that are forcing Thunder to respond accordingly.

This section showed the relative action distributions for the CATNeuro players vs. Thunder as a) aggregated across all models and b) for the best models by hits-to-opponent, and relative-score. This view of the data covers the overall stance or action strategies of the players and opponents but not the prevalent sequence of actions used by the different players/opponents. The next section analyses sequences of actions or combos.

10.3 Combo Analysis

As described in section 9.7.3 the combo sequence patterns are aggregated into categories such as OTD, OTO, etc. on a per round basis. This data is used to test for statistical differences between the two KD mechanisms, against each of the two opponents.

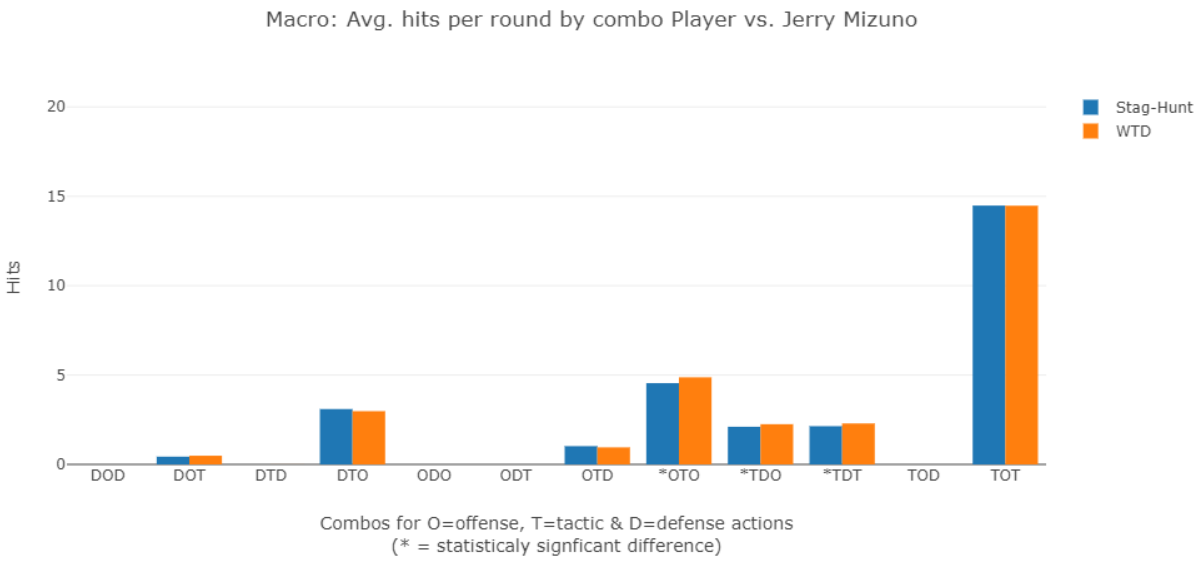


Figure 10-23: Hits by combo type - player vs. Jerry Mizuno

The hits landed against opponent Jerry Mizuno by the models from both mechanisms, under the different combo categories or types, are shown in Figure 10-23. Statistically significant differences are marked with an '*'. Models from both mechanisms land the majority of the hits

under the TOT (tactic-offence-tactic) combo category however there is no statistical difference between the hits landed by the two types of CATNeuro players. The TOT category is seen to represent agility with offense. WTD does statistically better in OTO, TDO and TDT categories but the differences are very small.

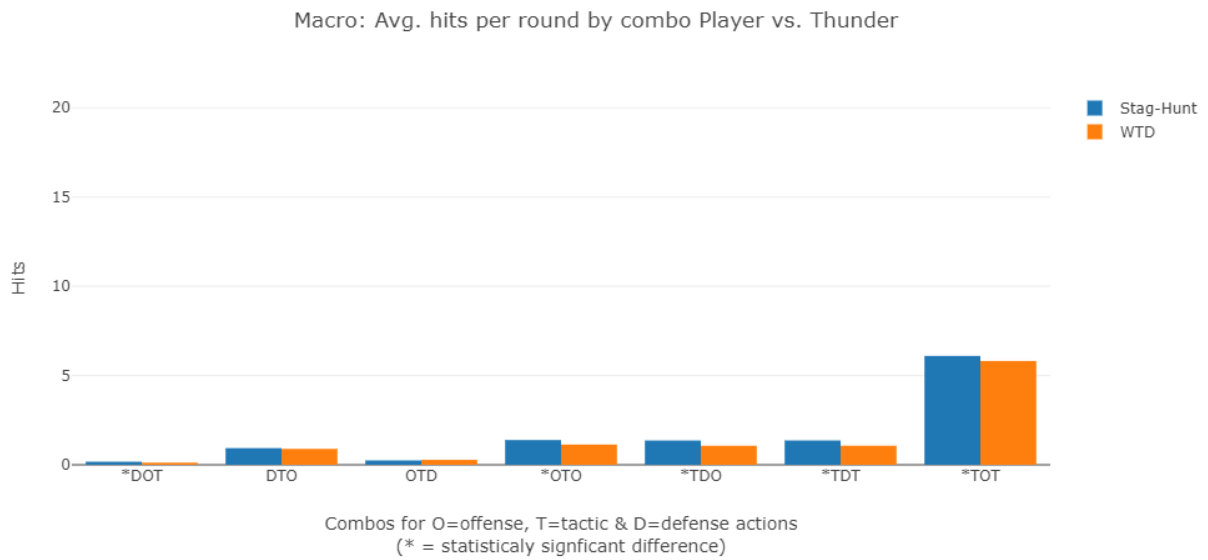


Figure 10-24: Hits by combo type - player vs. Thunder

Similar data against Thunder is shown in Figure 10-24. Against the stronger player, Stag-Hunt performs statistically more hits under several of the categories namely, DOT, OTO, TDO, TDT, and TOT. Categories with zero counts are not shown. Although the per round differences are minor, Stag-Hunt is consistently higher than WTD across all the categories. As with Jerry Mizuno, most hits are still landed with the TOT type combos. This data also shows that Thunder is the stronger opponent as the hit rates for both CATNeuro players are much lower than those against Jerry Mizuno.

From the above analysis it is evident that Stag-Hunt is able to produce more versatile models than WTD does. This difference is only evident when playing against the stronger player, Thunder.

Stag-Hunt models are able to land statistically more hits that WTD produced ones across a wide spectrum of combo categories. The models considered here and in the prior sections of this chapter are the top 10% models used for game play. The properties of the full set of generated models are discussed next.

10.4 Model Properties Comparison

This section compares the properties of the models generated by WTD and Stag-Hunt mechanisms. Out of all the models produced (120), only the top 10% (in terms of lowest training loss) were used to play the games with Jerry Mizuno and Thunder.

Even though the majority of the models were not used to play games, they can still serve as a basis of comparison between Stag-Hunt and WTD mechanisms and provide further insight. Each CATNeuro run returns the top 20 models discovered in the run. As there are 6 runs per mechanism, there are 6 x 20 = 120 models per mechanism used for comparison in this section. Note that the models produced are orthogonal with respect to the opponents played against (i.e. Jerry Mizuno and Thunder); i.e. the same models were played against both opponents.

The model properties analyzed were first given in Table 9-5. Each of these will be discussed next. The analysis presented for each property includes density plots (as explained earlier in section 9.7) as well the means and p-values for statistical significance.

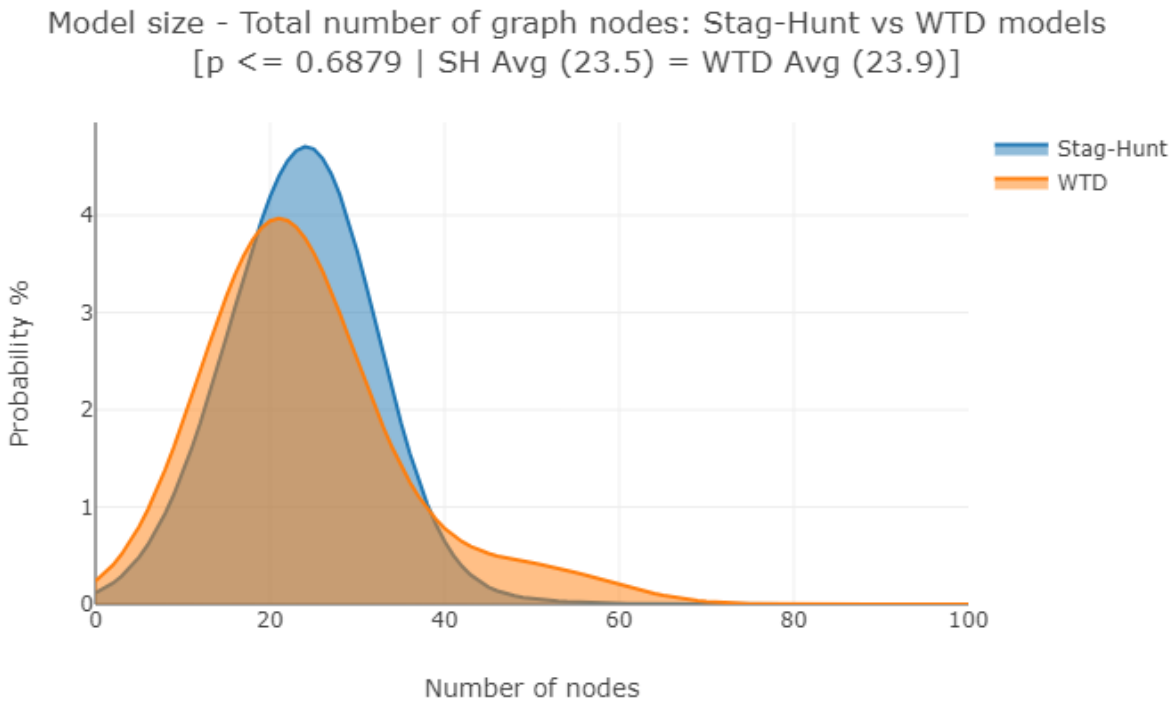


Figure 10-25: Stag-Hunt and WTD distributions for the number of nodes contained in the models produced

Figure 10-25 shows the density and of the number of nodes in the models produced. Statistically and visually there is not much difference between the two mechanisms. Models with about 24 nodes are produced on average by both mechanisms. Note the count reflects the lowest level nodes in each model, including input and output nodes. The count excludes the Blueprint nodes that are replaced by module species subgraphs at assembly time. The consistency of Stag-Hunt is slightly better in Figure 10-25 as mass is more narrowly distributed. And the mode for Stag-Hunt is higher.

Figure 10-26 show the density for the total number of edges in the models. From the statistical perspective, Stag-Hunt models have a smaller number of edges on the whole. The difference is statistically significant. However, looking at the density plots, Stag-Hunt has higher mode and its

mass is also more compactly distributed. The greater consistency of Stag-Hunt seems to be a persistent theme.

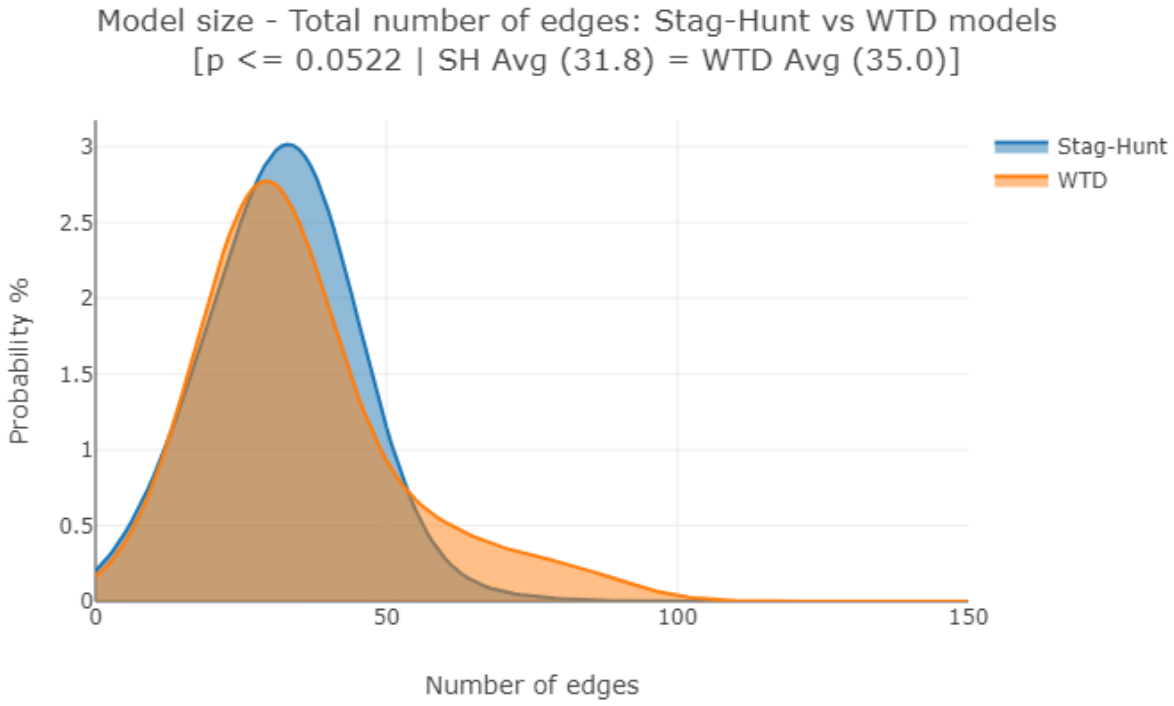


Figure 10-26: Stag-Hunt and WTD distributions for the number of edges in the models produced

Figure 10-27 is for the maximum path length (or depth) of the models. It is the length of the longest path from top to bottom. Statistically, there is no significant difference between the models produced by the two KD mechanisms. However, visually the mode for Stag-Hunt is higher and as before the probability mass distribution is narrower.

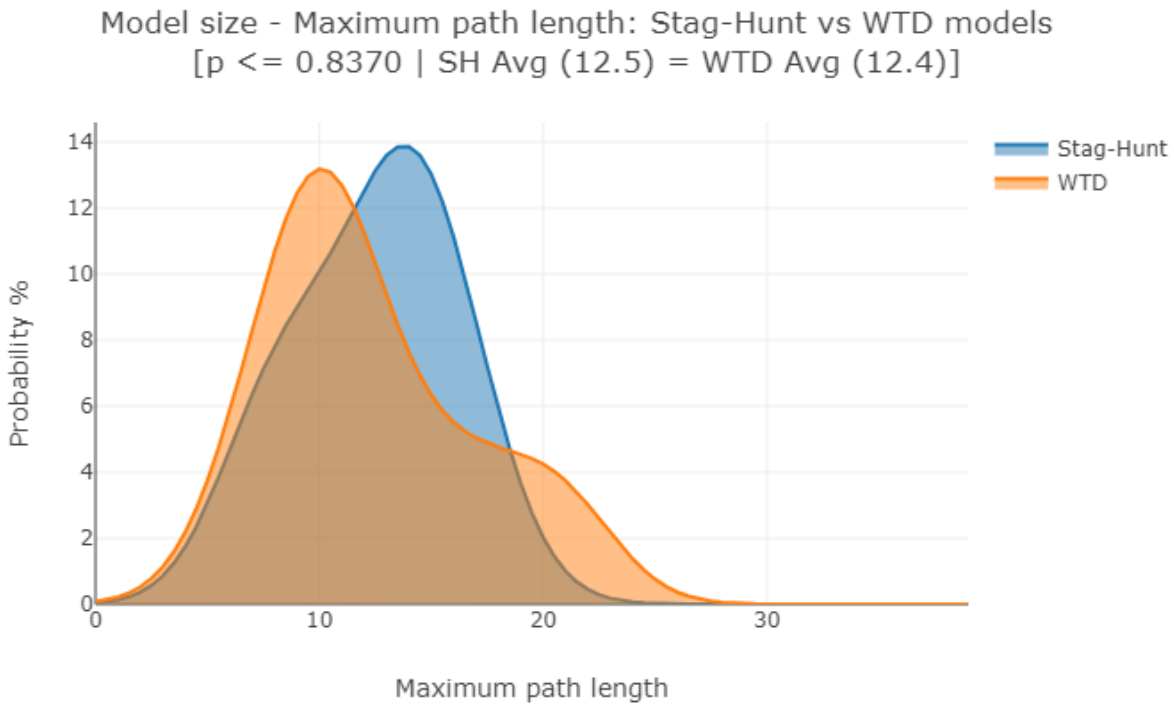


Figure 10-27: Stag-Hunt and WTD distributions for the maximum path lengths of the models produced

The distributions over the number of tunable weights or parameters are shown in Figure 10-28. These values are calculated by the deep learning framework used for model training in this experiment – CNTK (Seide & Agarwal, 2016). CNTK calculates this value after a Blueprint and selected modules have been assembled into a complete deep learning model. It reflects the real-world size of the model. In general, it is desirable to have smaller models for a variety of reasons, provided the model accuracy is acceptable. Smaller models are easier to train and faster to use at runtime. This is particularly important for game play because near real-time response is required to play the game effectively. Also, smaller models tend to overfit less. The CATNeuro system has a pluggable ranking mechanism (multi-objective support) that balances model size with training loss to give preference to smaller models given the similar or same loss (accuracy).

Here Stag-Hunt is statistically better in that it produces smaller models that have about 1300 less weights to train, on average (7625 – 6291 = 1334). Figure 10-28 shows that there is more to the story than just the statistics. Unlike the other density plots discussed thus far, the mass for Stag-Hunt is more widely distributed. It is bi-modal where the higher mode matches that for WTD at 9K and the other peaks at around 5.5K. All-in-all the chart shows that Stag-Hunt can find smaller models that are also good performers. The performance aspect – in terms of training loss – will be discussed next.

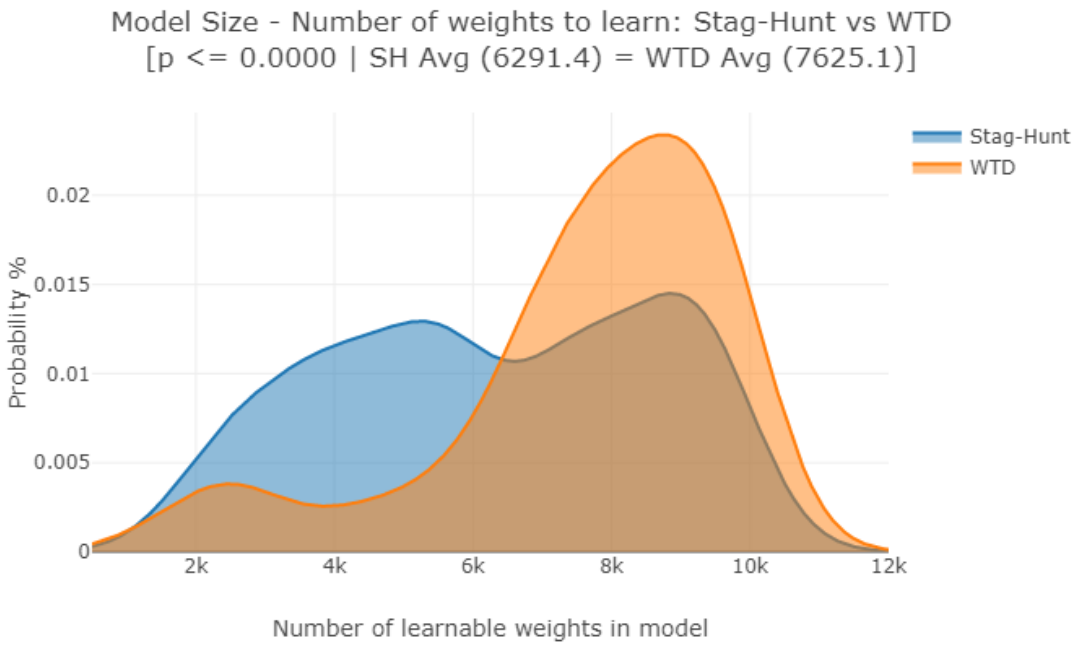


Figure 10-28: Stag-Hunt and WTD distributions for the number of parameter weights for the models produced

Figure 10-29 shows the training loss density. It is the mean squared error loss between the training data and the model output. Here again Stag-Hunt does better with statistical significance. Moreover, the mass distribution is narrower indicating that Stag-Hunt is more consistently able to find good solutions.

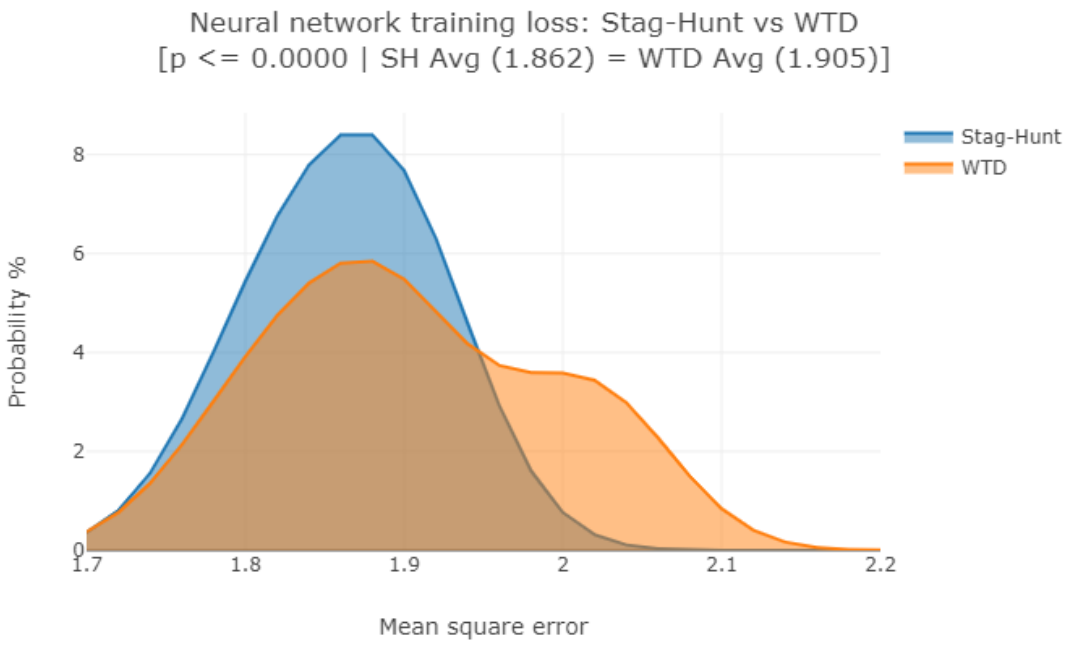


Figure 10-29: Stag-Hunt and WTD distributions for training loss

The final chart in this series is the number of generations at which the best model was found, shown in Figure 10-30. There are no meaningful differences between the number of generations taken by the two KD mechanisms to find the best model, either statistically or visually. Also, the range is quite large. The termination condition used was "terminate if no improvement in 10 generations". In the majority of the cases the best models were found at close to 30 generations however some were found after 70 generations. As mentioned before the training time is quite long and a single run can take a day or so on a single GPU machine.

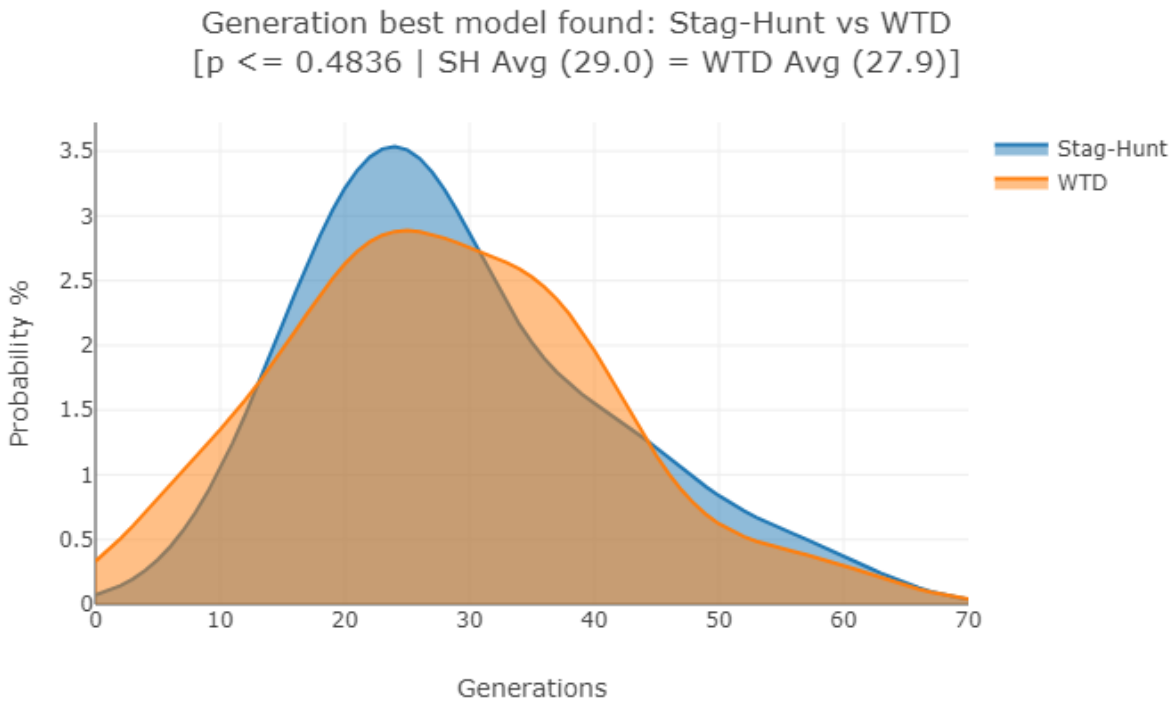


Figure 10-30: Stag-Hunt and WTD distributions for the generations at which the best models were discovered

10.5 Summary and Conclusions

For the CATNeuro system, 6 samples were taken with each KD mechanisms. A total of 120 models were produced for each and the top 10% played against both Jerry Mizuno and Thunder. The prior sections discussed the performance of the game models (10.1); the strategic variation in the models as reflected in the relative action distributions (10.2); and the overall properties of the models produced (10.4).

For the game performance, Stag-Hunt performs statistically better in:

a. The hits-to-opponent metric, at the aggregate level, against Thunder (Table 10-1)

b. Relative-score aggregated across all models, again when playing Thunder (Figure 10-2)

c. Best model by relative-score against Jerry Mizuno (Figure 10-16)

d. Best model by relative-score against Thunder (Figure 10-21)

More differences are apparent between two KD mechanism when playing the stronger opponent Thunder. For the benchmark AI Jerry Mizuno, both mechanisms perform equally well, except for c) above.

The relative action distributions reveal the strategies followed by each of the players – be it CATNeuro or opponents. As expected, the weaker opponent Jerry Mizuno uses a considerably smaller number of actions than Thunder does. At the aggregate level, how the opponents respond is more informative than the action distributions of the CATNeuro players themselves. This is apparent in Figure 10-12 for Jerry Mizuno where the HAC for Stag-Hunt is much lower than that for WTD. These figures are flipped in Figure 10-18 for Thunder where the HAC of Stag-Hunt is higher than that for WTD. The stronger opponent uses a greater variety of actions against Stag-Hunt models. In general, Stag-Hunt produced models that are more offense oriented that WTD ones. This is apparent in Figure 10-18 where Thunder uses more defense actions when playing against Stag-Hunt models.

Considering model properties, it is noted that Stag-Hunt:

a. produces smaller models than WTD does (Figure 10-28)

b. it is consistently better in terms of training performance (Figure 10-29)

c. produces models with a smaller number of edges than WTD does (Figure 10-26)

However, beyond purely statistical measures, most density plots show that Stag-Hunt is more consistent (has narrower distributions) than WTD, except for when it comes to model size for the number of tunable parameter weights.

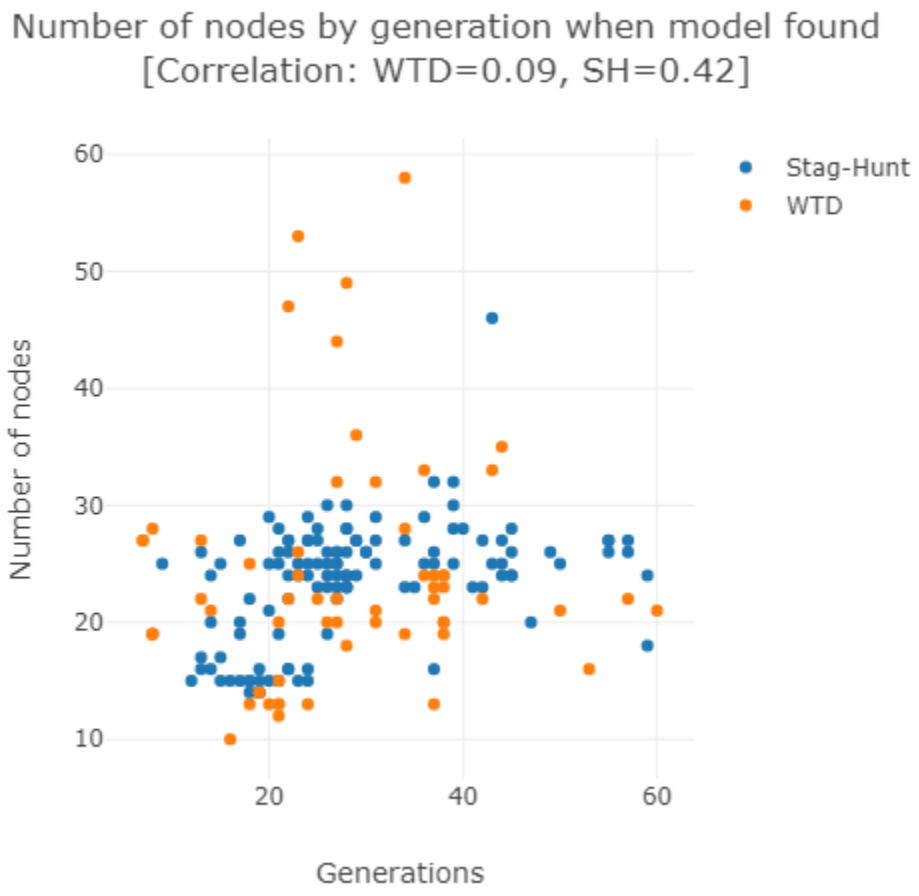


Figure 10-31: Correlation between generations and model size

10.6 Testing the Hypotheses

After the experimental results, the hypotheses postulated in 9.7 are discussed and conclusions drawn next.

Hy 9-1 ***"Cooperative knowledge distribution will yield more versatile models"*** ***holds***

The data to answer this question comes from the game performance measures, relative action distributions and combo categories. From a performance perspective, Stag-Hunt performs better when faced with Thunder for the measures listed earlier. However, against the benchmark AI Jerry Mizuno there are no significant performance differences between Stag-Hunt and WTD (except for one case) since both did well against Jerry Mizuno bot for different reasons. Looking at the relative action distributions there is also no clear and consistent pattern. Firstly, the HAC values for the players are not very telling. There is greater differential between the HAC values of the opponent when playing CATNeuro. But here again there is lack of consistency as HAC values for Jerry Mizuno show that WTD is more versatile but those for Thunder show that Stag-Hunt is more versatile. However, the definitive evidence that Stag-Hunt produces more versatile models comes from the combo analysis performed in section 10.3. Stag-Hunt is able to land significantly more hits than WTD, under a variety of combo categories, when playing against the stronger player Thunder and therefore Hy 9-1 is accepted.

Hy 9-2 ***"Cooperative knowledge in CATNeuro distribution will sustain longer search runs"*** ***{does not hold}***

This hypothesis is answerable from Figure 10-30 that shows the density of the number of generations to find the top models for both mechanisms. Both statistically and visually there is nothing to choose between Stag-Hunt and WTD and therefore this hypothesis is rejected.

Hy 9-3 ***"Longer search runs produce larger models"*** ***{holds with exceptions}***

Here a relationship between the search time and model size is being postulated. Figure 10-31 shows this relationship graphically along with the correlation measures for Stag-Hunt and WTD. Stag-Hunt (0.42) exhibits good correlation between search time and model size whereas WTD (0.09) does not. Stag-Hunt behaves as postulated; it seems to be more disciplined. The higher correlation for Stag-Hunt is also visually apparent (blue dots in the chart). Initially the correlation is linear but then number of nodes seem to level off as generations increase. This is primarily due to the limits imposed on the models in terms of sizes of the population individuals in the CATNeuro configuration used for the runs. Hy 9-3, thus partially holds. It holds for Stag-Hunt but not for WTD.

Hy 9-4 ***"The training loss is lower for cooperative distribution mechanism"*** ***{holds}***

Hy 9-4 is relatively easy to determine. Figure 10-29 shows the training loss distributions for Stag-Hunt and WTD. Average Stag-Hunt loss at 1.862 mean square error is lower than 1.905 for WTD and difference is statistically significant. Visually, the distribution of loss for Stag-Hunt is also narrower and peakier. This shows that Stag-Hunt is able to more consistently produce models with lower training loss and therefore Hy 9-4 holds.

Hy 9-5 ***"Cooperative distribution performs better than competitive distribution under more complex environmental conditions"*** ***{holds}***

As with CATGame, CATNeuro was tested with multiple levels of environmental complexity. Here complexity is in the form of the strength of the opponents played against - Jerry Mizuno the benchmark AI included with FightingICE; and Thunder the 2018 champion. Considering the aggregate performance results for both opponents, Table 10-1 shows the hits-to-opponent metric for Jerry

Mizuno and Table 10-3 for Thunder. Against Jerry Mizuno both KD mechanisms perform equally well are able to beat the opponent. However, against Thunder (representing higher complexity) Stag-Hunt performs significantly better. This is somewhat consistent with the relative-score measures shown in Figure 10-1 for Jerry Mizuno and Figure 10-2 for Thunder. For Jerry Mizuno there is no statistical difference between the relative-scores of the two KD mechanisms. However, against Thunder, Stag-Hunt performs better than WTD and with a p-value of 17%. All told there is strong evidence that cooperative distribution performs relatively better when environmental complexity is higher and therefore Hy 9-5 is established.

The experimental results for the framework developed in section 9.7 were analyzed and discussed in this section. Also, the hypotheses postulated in 9.7 were discussed in light of the obtained results and conclusion drawn as to each's validity. The next chapter summarizes the research effort and outlines the possible future works emanating from this exploration.

CHAPTER 11 CONCLUSIONS AND FUTURE WORK

11.0 Introduction

Cultural Algorithms are knowledge-driven stochastic optimization methods meant for problem solving in complex systems. Inspired by anthropological processes, the CA brings much machinery to bear on such tasks (Chapter 2). From the Belief Space which is a persistent and responsive store of knowledge; a socially networked population space; to intelligent knowledge distribution mechanisms; it is aptly equipped to tackle the behavior of a complex system.

The role of the knowledge distribution mechanisms is germane as they are the key allocators of computational resources in a CA system; even more so today when CA research focus has shifted to solve dynamically and hierarchically complex multi-objective problems. CA knowledge distribution mechanisms have steadily grown in their level of information processing capability (entropy) to tackle increasingly complex problems (Chapter 3). Earliest system used random distribution of knowledge then competitive mechanisms were developed, specifically majority weighted. The focus of this research is on using games for knowledge distribution, particularly cooperation-inclusive games since all prior mechanisms have been competitive.

Game theory is a deep and vast subject area (Chapter 4) but provides a fertile source of ideas for knowledge distribution mechanisms. Three distribution mechanisms were studied, inspired by several games in classical and evolutionary game theory, namely: Iterated Prisoner's Dilemma, Stag-Hunt and Stackelberg; all of which span both cooperation and competition. IPD and Stag-Hunt are related in that Stag-Hunt is an evolutionary game theory variant of Prisoner's Dilemma from classical game theory. Stag-Hunt involves the notion of time in a sense missing from IPD which is a single-shot game, played repeatedly.

CATGame, a new software system, was constructed in order to facilitate this research. It contains a generic mechanism that can be used for injecting arbitrary games into the **Influence** function of the CA for knowledge distribution (Chapter 5). This mechanism is used by concrete adaptions of the aforementioned three games (Chapter 6). In this research, the three cooperative/competitive mechanisms are contrasted against the default Weighted Majority mechanism (3.2) which is purely competitive.

IPD and Stag-Hunt are played from the perspective of the players in the population space. Each individual plays the game with all of its network neighbors. Due to the structure of the population space, complete symmetry and reciprocity is not possible. Each individual is playing against players who in turn are playing against a slightly different set of players (their respective neighbors). Thus, the games cannot be solved in a classical sense of finding the Nash equilibrium – apart from the fact that it would be computationally infeasible to do so. Instead, the players make cooperative/competitive decisions based on the best available information. Knowledge distribution in each is a two-step process where the players first are classified as Cooperator / Defector and then based on that, the knowledge assignments are performed. As a Cooperator an individual forgoes egoistic behavior and instead behaves according its rank in society (i.e. social rank - Listing 6-2, Listing 6-3). A relatively low-ranking individual accepts a relatively explorative Knowledge Source and high ranker, a relatively exploitative one. As a Defector the individual keeps its current assignment or accepts the locally dominant KS, depending on factors.

Stackelberg is played from the perspective of the Knowledge Sources that reside in the Belief Space. In microeconomics Stackelberg players make production and (implicitly pricing) decisions based on their relative strengths and potential first-mover advantage. Inspired by this, Stackelberg

KD allocates the best individuals to the strongest KS but in a way that leaves room for the less strong KS. The decision making in Stackelberg is more centralized and structured, as in a centrally planned economic system. By contrast IPD and Stag-Hunt are more dynamic and utilize more local (i.e. neighborhood) information.

The performance of the new KD cooperative/competitive mechanisms is compared with Weighted Majority, a purely competitive mechanism, with a dynamic landscape generator (7.1); Cones World. The dynamic complexity is controlled by the setting the A multiplier of the logistic equation. Values of A=1.0, 3.1, 3.6 and 3.9 are used. A=1.0 induces linear changes; at A > 3.0 the changes become non-linear; and at 3.9 the chaotic values are produced. The optimization landscapes are changed after 2500 generations while the optimization run is still underway. A total of 50 landscapes are generated in sequence for a single run. Each KD-A combination is run 200 times to obtain statistically significant results (7.6).

Resilience is measured by how quickly the system is able to find the new optimum after the proverbial rug is pulled from under it. The main performance metric is the generations-to-solution or G2S (7.2). G2S is tracked by landscape change. A new landscape in the sequence is created by changing the heights of the cones in the previous landscape using the values obtained from the logistic sequence generator.

CA is a 'social' system and hence the behavior of the system can be tracked with several social metrics. Diffusion (7.3) and Segregation Index (7.4) are social metrics that measure static aspects of the system. However, the CA is also a dynamical system and so to understand the dynamic aspects Markovian analysis is performed that involves several approaches strung together (7.5).

Community formations are detected using the Frequent Pattern Growth algorithm. Community-to-community transitions are analyzed with Google Page Rank and other graph-based approaches.

11.1 The CATGame Results

Experimental data was collected using the Wayne State grid computing facility. Over 1 terabyte of detailed log data was collected from the 200 sample runs for each KD-A combination. The experimental results are tabulated and presented in Chapter 8. Inferences about the hypotheses postulated in section 7.6, are drawn in section 8.5.

CATGame is a numerical optimization system meant for use in static and dynamic environments. The cooperative, game-based knowledge distribution achieved varied results under the different levels of complexity. It was found that IPD and Stag-Hunt generally performed the best from linear to chaotic; both were the most resilient to environmental changes (8.1). Stackelberg on the other hand was not able to track the changes as well. It performed well initially (i.e. in the first few landscapes of the sequence) but then its G2S performance became progressively worse over the progression of the landscapes. Also, Stackelberg performs progressively worse with increasing non-linear complexity. Weighted Majority shows robust behavior in the face of complexity. It is quite robust at A=1 but still lags behind IPD and Stag-Hunt in the earlier landscapes but catches up to them later in the sequence (Figure 8-1). With higher A values, IPD and Stag-Hunt start to distance themselves from the rest (Figure 8-2, Figure 8-3 & Figure 8-4). Overall Stag-Hunt is the most resilient of all the mechanisms tested. It quickly adapts to environment change levels and tracks the changes well over time.

If Weighted Majority is the “wisdom of the crowd” then IPD and Stag-Hunt represent “cooperation in the context of social rank”. Social rank directed cooperation does indeed seem

to work well in the face of environmental uncertainty. However, the structured cooperation model of Stackelberg is not seen as being as effective. One difference between the two types of cooperation is that Stackelberg does not take into account local information. It is akin to a centrally planned economic system; i.e. where the resource allocation decisions are centralized. The other mechanisms (including WTD) have a 'market' economy aspect where allocation decisions are decentralized and take into account local conditions. The collapse of socialism in recent geo-political history is perhaps a reminder that excessive centralization is not effective when the pace of change is high.

The static and dynamic social analyses provide further insight into the operations of the different KD mechanisms. The most telling is Schelling's Segregation Index. Higher performing mechanisms have consistent response in terms of exhibiting higher average segregation as environmental complexity changes from linear to non-linear to chaotic (Figure 8-20). To wit, WTD, IPD and Stag-Hunt all exhibit an increase in average segregation in the population as complexity changes form linear to chaotic. Further, IPD and Stag-Hunt show a higher degree of change in population segregation than WTD. This indicates that IPD and Stag-Hunt are more sensitive to environmental changes than WTD. Stackelberg on the other hand is not consistent it its responses. Here the segregation first increases and then decreases as change tends to chaotic. Segregation index is an emergent phenomenon. It can be seen as response to the degree of stress placed on the system. More consistent response means that the underlying mechanisms withstand and don't break down under varying degree of duress.

The dynamic analysis (8.4) shows that both cooperative mechanisms and the competitive mechanism work differently. This is quite evident when observing Page Rank derived tree charts

(Figure 8-46, Figure 8-48, Figure 8-50 and Figure 8-52). All mechanisms have different 'signature' in terms of the community rankings produced by the dynamics. The signatures remain somewhat consistent even across A-values. When one compares the chart for Stackelberg with those of the other mechanisms, it can be seen that Stackelberg allocates comparatively more resources to exploitation. This partly explains the lack of Stackelberg performance in a dynamic environment that seems to require higher degree of exploration especially as the environment becomes more chaotic.

The community-to-community transition data is projected into another view. The communities are categorized as Explorative, Neutral or Exploitative, depending on each's explorative index. Then statistically significant changes in net flow are measured and plotted by each A transition (e.g. 1.0 → 3.1, 3.1 → 3.6, etc.). Net flow here means net change (increase – decrease) into a particular category. Take the Explorative category and 1.0 → 3.1 (linear → non-linear) transition. The statistically significant inflow ([Neutral; Exploitative] → Explorative) and outflow (Explorative → [Neutral; Exploitative]) are measured and the differences taken. This value is the resource increase / decrease into the Explorative category due to the change in A 1.0→3.1. This is done for all transitions and all categories. Figure 8-58 and Figure 8-59 show the net flow for Explorative and Exploitative categories, respectively. The interesting result is that the best performing mechanism – Stag-Hunt – is very consistent in allocating progressively more resources to exploration and progressively less resources to exploitation, with each increment in A. None of the other mechanisms are completely consistent. This is a strong indication that the underlying mechanism of resource allocation in Stag-Hunt is very robust in the face of environmental complexity. It also shows if the environment rate of change is higher, comparatively more resources need to be

directed to exploration. The observed segregation behavior can also be explained from this result; increasing exploration (and decreasing exploitation) changes the mix of the communities in the population and therefore increases segregation as explorative KS will tend to dominate. Considering that the KD mechanism is the primary factor in the distribution of knowledge (i.e. allocation of compute resources) in the CA, it can be concluded that Stag-Hunt is the most consistent in making allocation decisions under varying levels of complexity and therefore shows as being the most robust in the face of it.

CATGame was meant to test the behavior of cooperative knowledge distribution under dynamic complexity. Another notion of complexity is hierarchical complexity. How well does cooperation work to solve hierarchically complex problems?

To answer this question the CATNeuro system was constructed to find optimal model structures for deep learning models. CATNeuro uses CA for Neural Architecture Search (NAS) – an emerging field that is currently drawing considerable research interest. The top evolutionary computation conference “IEEE World Congress on Computational Intelligence” (WCCI) 2020 has an entire track dedicated to Neural Architecture Search.

CATNeuro uses speciation with multiple populations to evolve optimal models (inspired by NEAT and derivative works) (9.1). This is a hierarchical optimization problem (Figure 1-4). Tier one is the overall graph structure (blueprint); tier two is the selection of module species that are assembled into a particular blueprint; and tier three is the optimization of parameters such as e.g. learning rate, dimensions of dense nodes, activation types, etc.

11.2 CATNeuro Results

CATGame is for numerical optimization in dynamic environments and CATNeuro is finding optimal graph structures – two very different domains and two different notions of complexity. The best performing cooperative mechanism from CATGame – Stag-Hunt – is implemented and its performance compared with the default competitive mechanism – Weighed Majority (9.3).

The test problem is to evolve an optimal model to play the FightingICE game against the two selected opponents – the benchmark AI Jerry Mizuno and the 2018 champion Thunder (9.1). The test problem favors small and fast models that can play the game at the required frame rate. The deep learning model training process requires specialized infrastructure and can take a long time (e.g. days in some cases). Keeping the test problem manageably small is helpful but even here it can take about a week to complete the training and test cycles (including playing the games with the selected opponents). For the particular test case, the train-test cycle time can be minimized if access is available to a 'farm' of 150 GPUs. The sample size for each KD-A mechanism is only 6 vs. 200 for the CATGame experiment. Additionally, each sample run for CATGame was over 2500*50=125000 generations vs. only 30-70 generations for CATNeuro. CATGame produces much more data. The kind of statistical analysis done for CATGame is not feasible for CATNeuro. Instead the focus is on comparing the performance of the models generated by the two KD mechanisms against the two players and the comparison of the structural properties and other aspects of the generated models (9.7).

Data required to train the models was obtained through the application of Reinforcement Learning using a policy-based approach (9.4). This process is explained in section 9.5; it takes about 24 hours to complete and results in a large policy table of about 1M rows and 1G size on disk.

CATNeuro runs are performed with a sample of the training data. Top models are then trained on the full dataset and played against the opponents. Under the experimental setup (9.7), each model is played 10 times against each opponent and the game statistic recorded.

The experimental results for CATNeuro are organized by:

a. Game performance (10.1)

b. Action strategies (10.2)

c. Combo analysis (10.3)

d. Model properties (10.4)

The game performance results show that against the benchmark AI, Jerry Mizuno, models from both KD mechanism performed well and won all the games. Both approaches were able to use the same basic techniques to defeat the opponent.

Against the 2018 champion Thunder all games are lost by each mechanism (which is more of a function of the available learnings from RL derive training data). However here, Stag-Hunt derived models do better than WTD models, with statistical significance, for *hits-to-opponent* and *relative-score* metrics. This shows that Stag-Hunt can extract relatively more information from the training data.

Considering the *relative action distributions*, no consistent patterns emerge between the two types of CATNeuro players, in head-to-head comparison (Figure 10-11, Figure 10-17). However, observing the opponent strategies is more telling - i.e. actions distributions of Jerry Mizuno and Thunder playing WTD vs. when playing Stag-Hunt, respectively (Figure 10-12, Figure 10-18). One

can also gauge a player's performance by looking at how the opponent chooses to respond. Jerry Mizuno responds with greater versatility when playing WTD models (HAC: WTD = 25, Stag-Hunt=12). Conversely, Thunder shows greater versatility when playing Stag-Hunt (HAC: WTD=22, Stag-Hunt=30). The results show that opponents respond differently and hence the models learnt by the two KD mechanism behave differently, at the aggregate level. Greater versatility of the opponent against a certain player indicates that the player is forcing the opponent to respond with greater variety, by making more timely moves. From this perspective, Stag-Hunt is making the stronger player Thunder work harder than WTD does.

The combo analysis provides clear evidence that Stag-Hunt does indeed create more versatile models than WTD does. This is evidenced by the fact that Stag-Hunt lands significantly more hits on the stronger opponent Thunder under a variety of combo types.

The model properties comparison shows that Stag-Hunt produces significantly smaller models with respect to number of edges (Figure 10-26) and number of learnable weights (Figure 10-28). Also, importantly, Stag-Hunt models have lower training loss on average (Figure 10-29). In general, smaller (more parsimonious) models are desired, provided accuracy (training loss) is not compromised. Stag-Hunt seems better able to balance these conflicting goals.

Next, several hypotheses postulated in section 9.7 related to NAS and CATNeuro are analyzed and addressed in 10.5. The primary question this research set out to answer is whether cooperative knowledge distribution improves CA performance in complex environments. Drawing much from Game Theory this proposition is studied with respect to dynamic and hierarchical notions of complexity. The results show that, for the numerical optimization domain (dynamic complexity), cooperation in the context of social rank makes the system more robust and performs

better than the default competitive mechanism WTD. WTD is also robust but less so as complexity increases. In the neural architecture search domain (hierarchical complexity) the results are mixed with a slight edge for cooperation. In vivo (training loss), cooperation performs significantly better but in vitro (game play) it is only marginally better. The signal is weaker in the NAS problems and therefore “wisdom of the crowd” is about as effective as social rank centered cooperation.

11.3 Future Direction

The CATGame system is configured with a generic game mechanism that can be exploited for analyzing other cooperative and competitive game mechanisms. Game Theory – both classical and evolutionary – have deep reserves to draw from. Exploring other games or evolutionary strategies for knowledge distribution will extend the understand for building robust systems in the face of complexity.

Deep learning is a prime area for further exploration. Optimal model topology and hyperparameter tuning is an active research area. Model tuning is a time-consuming task that still requires much human input and therefore automation to free up human capital is much desired. This research shows that evolutionary algorithms are effective means of addressing the NAS challenge. By design CA is well suited to solving problems in this domain.

However, CATNeuro is a new system with many missing features such the ability to construct models with convolutions and recurrence. It needs to be extended to provide better coverage of the available functionality in deep learning toolkits. The current translation mechanism is for the CNTK toolkit only. Translations for other popular toolkits such as Tensorflow, PyTorch and support for ONNX (open neural network exchange) formats are fruitful avenues of future work.

Training deep learning models is already a very time consuming. Adding stochastic search on top greatly extends the time required to find optimal models. If history is an indicator, hardware to train deep learning models should become, faster, cheaper and more plentiful. CATNeuro is built with parallel model training support but it needs to be developed further to seamless access vast arrays of training hardware to reduce search time. Many completing NAS approaches are being developed and CATNeuro should be benchmarked against the top contenders to derive additional insights for improvements.

APPENDIX A CATGAME FUNCTIONAL INTERFACE

CATGame is written in a strongly-typed functional programming language F#. The equivalent to UML diagrams in functional programming is the functional interface; it shows the high-level structure in terms of the top data structures and function types.

```
///type definitions for the CA 'interface'
//defined in a functional programming way
module rec CA

///CA structure - instance of CA that can be stepped through for optimization
type CA<'k> =
    {
        BeliefSpace             : BeliefSpace<'k>
        Acceptance              : Acceptance<'k>
        Update                  : Update<'k>
        Influence               : Influence<'k>
        Population              : Population<'k>
        Network                 : Network<'k>
        Fitness                 : Fitness
        Optimization            : OptimizationKind
        EnvChngSensitivity      : EnvChngSensitivity
    }

///how should we respond to environmental changes
//CA may or may not reset set internal state based on this setting
type EnvChngSensitivity =

    ///CA does not adjust internal state if environment changes
    | Insensintive

    ///After how many environmental changes to re-adjust.
    ///A value of 1 means re-adjust to every environment change
    | Every of int

///Instructs CA how to respond to environment change
type EnvChngeType =
    | NoChange    //environment did not change
    | Adjust      //environment changed - adjust internal state accordingly
    | Track       //environment changed but only note the changes - do not adjust
internal state

type OptimizationKind = Minimize | Maximize  //minimization or maximization problem

///tree structure of the belief space knowledge source
type BeliefSpace<'k> = KnowledgeSource<'k> Tree

///knowledge source type
type KnowledgeSource<'k> =
    {
```

```
        ///Knowledge type identifier (Domain, Normative, etc.)
        Type        : Knowledge

        ///Acceptance function type of a knowledge source
        Accept      : EnvChngeType -> Individual<'k> array -> Individual<'k> array
* KnowledgeSource<'k>

        ///Influence function type of a knowledge source
        Influence   : Population<'k> -> Temperature -> Individual<'k> ->
Individual<'k>
    }

type Tree<'a>        = Leaf of 'a | Node of 'a * Tree<'a> list | Roots of Tree<'a>
list

type Knowledge       = Situational | Historical | Normative | Topgraphical | Domain
| Other of string

///CA acceptance function type
type Acceptance<'k>  = BeliefSpace<'k> -> Population<'k> -> Individual<'k> array

///CA update function type
type Update<'k>      = EnvChngeType -> BeliefSpace<'k> -> Individual<'k> array ->
BeliefSpace<'k>

///CA influence function type
type Influence<'k>   = Influence of (
                            EnvChngeType
//environment change signal
                                -> Population<'k>
                                -> BeliefSpace<'k>
                                -> Network<'k>
                                -> Fitness
                                -> OptimizationKind
                                -> (Population<'k>*BeliefSpace<'k>*Influence<'k>))
//returns updated population, beliefSpace and influence function

///Population individual (parameterized by KS type)
type Individual<'k>  = {Id:Id; Parms:float array; Fitness:float; KS:'k}

///Population is an array of indviduals
type Population<'k>  = Individual<'k> array

///Network function type
type Network<'k>     = Population<'k> -> Id -> Individual<'k> array

///Fitness function type
type Fitness         = (float array -> float) ref

///Id of the population individual (alias to int)
type Id = int

///The level of influence to apply (alias to float)
type Temperature = float
```

```
///Parameters and fitness values extracted from 'best' individuals
type Marker = {MParms:float[]; MFitness:float}

///Structure to hold single step in a CA run
type TimeStep<'k> = {CA:CA<'k> ; Best:Marker list; Progress:float list; Count:int;
EnvChngCount:int}

///function type to specify the termination of a CA run
type TerminationCondition<'k> = TimeStep<'k> -> bool

///Parameter types and ranges for the fitness problem
//TODO: make this part of the CA structure
type Parm =
    /// float parameter type
    | F of      v:float     * min:float     * max:float

    ///integer parameter type (stepped through as whole integers by optimiztion
engine)
    | I of      v:int       * min:int       * max:int
```

APPENDIX B DIFFUSION STATISTICAL TESTS

Statistical tests for diffusion by A value and landscape sequence number

B.1 A=1

A=1.0											
Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD != WTD		SHS Mean	SHS Stdv.	TTest SHS != WTD	STK Mean	STK Stdv.	TTest STK != WTD	
1	0.684882	0.121618	0.780535	0.089758	✓	1	0.746273	0.108345	✓ 1	0.679806	0.114296	✗	-1
2	0.687477	0.119064	0.787176	0.083064	✓	1	0.74995	0.102994	✓ 1	0.676499	0.126035	✗	-1
3	0.684755	0.12406	0.789956	0.082407	✓	1	0.755716	0.107955	✓ 1	0.688059	0.136849	✗	-1
4	0.700683	0.11311	0.78068	0.090117	✓	1	0.744355	0.1055	✓ 1	0.694031	0.13191	✗	-1
5	0.698064	0.111194	0.788226	0.083669	✓	1	0.748019	0.104404	✓ 1	0.683351	0.143686	✗	-1
6	0.693739	0.119736	0.793242	0.088193	✓	1	0.757102	0.102204	✓ 1	0.670536	0.147563	✗	-1
7	0.709505	0.105973	0.781428	0.084126	✓	1	0.751023	0.105409	✓ 1	0.684314	0.141452	✓	1
8	0.702985	0.104832	0.789006	0.086827	✓	1	0.739933	0.107892	✓ 1	0.678651	0.151576	✗	-1
9	0.695203	0.105259	0.789141	0.088857	✓	1	0.761557	0.103981	✓ 1	0.672343	0.152721	✗	-1
10	0.712062	0.108715	0.794573	0.083074	✓	1	0.746811	0.114625	✓ 1	0.677003	0.146087	✓	1
11	0.69707	0.103368	0.7906	0.081256	✓	1	0.742922	0.101499	✓ 1	0.675945	0.147589	✗	-1
12	0.697958	0.112889	0.798081	0.08386	✓	1	0.757478	0.09598	✓ 1	0.675252	0.156115	✗	-1
13	0.716086	0.104594	0.783515	0.089342	✓	1	0.740356	0.10938	✓ 1	0.675	0.152749	✓	1
14	0.707458	0.104027	0.784248	0.09071	✓	1	0.746873	0.099454	✓ 1	0.682042	0.156038	✗	-1
15	0.701595	0.103989	0.78366	0.083372	✓	1	0.761845	0.113859	✓ 1	0.663623	0.152164	✓	1
16	0.715643	0.100254	0.780222	0.085561	✓	1	0.737937	0.112452	✓ 1	0.665498	0.157514	✓	1
17	0.701386	0.10257	0.783793	0.07836	✓	1	0.748862	0.099211	✓ 1	0.66809	0.162928	✓	1
18	0.699386	0.111457	0.782913	0.084018	✓	1	0.769469	0.103063	✓ 1	0.661528	0.164893	✓	1
19	0.710602	0.101651	0.791706	0.077268	✓	1	0.75059	0.10499	✓ 1	0.668954	0.154445	✓	1
20	0.703459	0.107197	0.776848	0.084197	✓	1	0.742592	0.11575	✓ 1	0.672912	0.16166	✓	1
21	0.704564	0.105387	0.783802	0.087268	✓	1	0.757151	0.104141	✓ 1	0.673362	0.156358	✓	1
22	0.719116	0.103226	0.788442	0.08394	✓	1	0.751202	0.103949	✓ 1	0.670067	0.170649	✓	1
23	0.707719	0.102246	0.791199	0.084894	✓	1	0.750697	0.101771	✓ 1	0.669311	0.164176	✓	1
24	0.705619	0.103606	0.78995	0.086316	✓	1	0.756306	0.111141	✓ 1	0.66111	0.163299	✓	1
25	0.721959	0.1029	0.796866	0.084014	✓	1	0.751971	0.102576	✓ 1	0.663513	0.174028	✓	1
26	0.713581	0.092427	0.790068	0.092361	✓	1	0.741577	0.108924	✓ 1	0.66527	0.16638	✓	1
27	0.702832	0.099305	0.796962	0.076186	✓	1	0.74966	0.102619	✓ 1	0.66401	0.171893	✓	1
28	0.719331	0.094545	0.791589	0.091976	✓	1	0.750976	0.104326	✓ 1	0.682449	0.17483	✓	1
29	0.710846	0.097283	0.786303	0.087269	✓	1	0.752458	0.10127	✓ 1	0.669808	0.173113	✓	1
30	0.710787	0.101626	0.782875	0.083857	✓	1	0.75978	0.109431	✓ 1	0.661848	0.174714	✓	1
31	0.722009	0.09758	0.791256	0.079366	✓	1	0.750603	0.096286	✓ 1	0.657426	0.174425	✓	1
32	0.711615	0.093664	0.786411	0.080015	✓	1	0.744159	0.10177	✓ 1	0.667024	0.164655	✓	1
33	0.718205	0.094413	0.791132	0.081753	✓	1	0.774511	0.097148	✓ 1	0.6599	0.17148	✓	1
34	0.721808	0.093274	0.779819	0.092479	✓	1	0.743855	0.113972	✓ 1	0.665646	0.166079	✓	1
35	0.712471	0.106477	0.791853	0.079962	✓	1	0.762173	0.104071	✓ 1	0.670907	0.170523	✓	1
36	0.703958	0.098777	0.790397	0.080349	✓	1	0.75941	0.111587	✓ 1	0.667412	0.179751	✓	1
37	0.728623	0.092716	0.786076	0.085904	✓	1	0.751829	0.106928	✓ 1	0.662247	0.170191	✓	1
38	0.718543	0.090921	0.788111	0.085554	✓	1	0.748928	0.110549	✓ 1	0.671081	0.174132	✓	1
39	0.70511	0.093918	0.792339	0.09461	✓	1	0.752441	0.108106	✓ 1	0.653107	0.174451	✓	1
40	0.728001	0.093181	0.789072	0.082896	✓	1	0.73907	0.115497	✗ -1	0.669324	0.178122	✓	1
41	0.712711	0.09836	0.796738	0.078806	✓	1	0.739345	0.111957	✓ 1	0.669396	0.167688	✓	1
42	0.706687	0.100932	0.789975	0.080265	✓	1	0.764759	0.102756	✓ 1	0.679659	0.165671	✓	1
43	0.73096	0.100518	0.785917	0.092041	✓	1	0.743227	0.114726	✗ -1	0.691699	0.178781	✓	1
44	0.703552	0.097078	0.788628	0.094385	✓	1	0.752098	0.106679	✓ 1	0.667964	0.176942	✓	1
45	0.70899	0.096284	0.788668	0.080364	✓	1	0.761593	0.101796	✓ 1	0.663816	0.174319	✓	1
46	0.708277	0.090335	0.783386	0.094583	✓	1	0.748228	0.105595	✓ 1	0.664753	0.168172	✓	1
47	0.712029	0.091367	0.777213	0.085377	✓	1	0.752178	0.106934	✓ 1	0.67132	0.171637	✓	1
48	0.71104	0.104562	0.793231	0.078602	✓	1	0.755518	0.099145	✓ 1	0.663813	0.175721	✓	1
49	0.719724	0.096669	0.789087	0.080813	✓	1	0.741099	0.114254	✓ 1	0.677531	0.165102	✓	1
50	0.711279	0.097382	0.780448	0.081187	✓	1	0.740595	0.10316	✓ 1	0.671032	0.174517	✓	1

B.2 A=3.1

A=3.1											
Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD != WTD		SHS Mean	SHS Stdv.	TTest SHS != WTD	STK Mean	STK Stdv.	TTest STK != WTD
1	0.640029	0.143723	0.764436	0.098668	✓	1	0.721924	0.115861	✓ 1	0.656812	0.119541	✗ -1
2	0.685225	0.13055	0.799963	0.081801	✓	1	0.769206	0.093999	✓ 1	0.674923	0.131545	✗ -1
3	0.683283	0.124613	0.783895	0.089099	✓	1	0.762084	0.105927	✓ 1	0.663291	0.125099	✗ -1
4	0.714847	0.103599	0.802095	0.080014	✓	1	0.762165	0.111891	✓ 1	0.673696	0.129572	✓ 1
5	0.711711	0.096364	0.780635	0.093108	✓	1	0.733923	0.116358	✓ 1	0.655334	0.13441	✓ 1
6	0.724081	0.092492	0.803524	0.086127	✓	1	0.760155	0.105041	✓ 1	0.6708	0.131886	✓ 1
7	0.717974	0.083031	0.785738	0.095373	✓	1	0.756088	0.096878	✓ 1	0.687961	0.133108	✓ 1
8	0.730318	0.080306	0.789068	0.091583	✓	1	0.751888	0.10091	✓ 1	0.673366	0.139718	✓ 1
9	0.712636	0.084233	0.794065	0.087378	✓	1	0.76917	0.098363	✓ 1	0.673451	0.143484	✓ 1
10	0.726044	0.085414	0.803852	0.082916	✓	1	0.754314	0.104357	✓ 1	0.686108	0.135219	✓ 1
11	0.721038	0.092796	0.787316	0.081242	✓	1	0.755526	0.09081	✓ 1	0.666785	0.144173	✓ 1
12	0.722351	0.074053	0.796904	0.08035	✓	1	0.772667	0.095524	✓ 1	0.684457	0.141601	✓ 1
13	0.721093	0.080374	0.782677	0.091541	✓	1	0.748807	0.095225	✓ 1	0.664748	0.148915	✓ 1
14	0.7251	0.072713	0.798391	0.084792	✓	1	0.750393	0.106569	✓ 1	0.678805	0.147387	✓ 1
15	0.72498	0.080153	0.793335	0.078345	✓	1	0.756981	0.106209	✓ 1	0.669704	0.142574	✓ 1
16	0.736115	0.074458	0.784844	0.089105	✓	1	0.763388	0.094896	✓ 1	0.669369	0.132514	✓ 1
17	0.723098	0.076929	0.793427	0.090979	✓	1	0.740861	0.107278	✗ -1	0.665234	0.141655	✓ 1
18	0.732422	0.078128	0.780392	0.091964	✓	1	0.770722	0.093025	✓ 1	0.672724	0.151893	✓ 1
19	0.725352	0.072269	0.78741	0.091281	✓	1	0.748151	0.10648	✓ 1	0.654295	0.149652	✓ 1
20	0.731414	0.074799	0.792497	0.087691	✓	1	0.752793	0.096005	✓ 1	0.672606	0.147933	✓ 1
21	0.725409	0.078614	0.786072	0.089564	✓	1	0.746331	0.107384	✓ 1	0.680718	0.151935	✓ 1
22	0.731046	0.074876	0.785711	0.0861	✓	1	0.753566	0.102636	✓ 1	0.677939	0.162433	✓ 1
23	0.721559	0.07301	0.778284	0.08907	✓	1	0.747842	0.106418	✓ 1	0.659995	0.155695	✓ 1
24	0.720577	0.075168	0.789821	0.09096	✓	1	0.783275	0.101918	✓ 1	0.667256	0.157433	✓ 1
25	0.735565	0.073085	0.786401	0.095907	✓	1	0.738662	0.096162	✗ -1	0.658306	0.149637	✓ 1
26	0.727977	0.078886	0.796061	0.081296	✓	1	0.753197	0.107757	✓ 1	0.672634	0.159869	✓ 1
27	0.720186	0.075493	0.789071	0.086872	✓	1	0.774862	0.092815	✓ 1	0.655878	0.156608	✓ 1
28	0.726096	0.073314	0.790239	0.083061	✓	1	0.75629	0.111004	✓ 1	0.668583	0.161129	✓ 1
29	0.730294	0.074579	0.791204	0.090015	✓	1	0.749052	0.106103	✓ 1	0.667836	0.157742	✓ 1
30	0.72804	0.078514	0.781514	0.087664	✓	1	0.76823	0.100888	✓ 1	0.664247	0.163508	✓ 1
31	0.71999	0.073832	0.798185	0.088888	✓	1	0.75978	0.099862	✓ 1	0.666975	0.162373	✓ 1
32	0.73116	0.075047	0.78651	0.088097	✓	1	0.760271	0.109458	✓ 1	0.658983	0.166843	✓ 1
33	0.718915	0.079285	0.793528	0.083208	✓	1	0.759914	0.105721	✓ 1	0.661698	0.165156	✓ 1
34	0.7308	0.073688	0.795791	0.084319	✓	1	0.75658	0.109117	✓ 1	0.665189	0.163884	✓ 1
35	0.723712	0.073613	0.779061	0.090597	✓	1	0.750205	0.096656	✓ 1	0.665621	0.16101	✓ 1
36	0.718211	0.077439	0.788314	0.086373	✓	1	0.76424	0.099372	✓ 1	0.665518	0.159768	✓ 1
37	0.732189	0.072761	0.787864	0.088878	✓	1	0.74154	0.099722	✗ -1	0.655424	0.157312	✓ 1
38	0.710864	0.081912	0.790344	0.095999	✓	1	0.749943	0.105313	✓ 1	0.665226	0.163143	✓ 1
39	0.727764	0.078916	0.783193	0.087169	✓	1	0.756376	0.106139	✓ 1	0.65655	0.161139	✓ 1
40	0.733495	0.077549	0.785753	0.087966	✓	1	0.750581	0.106174	✗ -1	0.663015	0.161677	✓ 1
41	0.720976	0.083972	0.794646	0.086623	✓	1	0.749928	0.101026	✓ 1	0.661919	0.164582	✓ 1
42	0.716796	0.07727	0.774104	0.083681	✓	1	0.756598	0.118814	✓ 1	0.664869	0.167053	✓ 1
43	0.721736	0.073172	0.782348	0.089875	✓	1	0.748222	0.093302	✓ 1	0.657532	0.162838	✓ 1
44	0.725452	0.073637	0.781935	0.09295	✓	1	0.74193	0.110352	✗ -1	0.666983	0.160909	✓ 1
45	0.718843	0.073004	0.783571	0.088782	✓	1	0.769112	0.100217	✓ 1	0.665862	0.166137	✓ 1
46	0.729812	0.074789	0.779821	0.090501	✓	1	0.741503	0.101051	✗ -1	0.667026	0.174438	✓ 1
47	0.718129	0.0775	0.785467	0.086418	✓	1	0.745561	0.110843	✓ 1	0.652376	0.153896	✓ 1
48	0.729141	0.071806	0.785786	0.083805	✓	1	0.772399	0.090297	✓ 1	0.663055	0.164173	✓ 1
49	0.721022	0.073804	0.785075	0.082912	✓	1	0.75974	0.094685	✓ 1	0.674093	0.166156	✓ 1
50	0.724685	0.077806	0.788945	0.084914	✓	1	0.753353	0.09282	✓ 1	0.659333	0.163059	✓ 1

B.3 A=3.6

A=3.6											
Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD != WTD		SHS Mean	SHS Stdv.	TTest SHS != WTD	STK Mean	STK Stdv.	TTest STK != WTD
1	0.668878	0.13145	0.763533	0.098507	✓	1	0.7125	0.125102	✓ 1	0.680615	0.111592	✗ -1
2	0.69923	0.109417	0.797914	0.079529	✓	1	0.75893	0.102214	✓ 1	0.704211	0.116839	✗ -1
3	0.682715	0.121451	0.779505	0.090236	✓	1	0.769016	0.101092	✓ 1	0.682207	0.125979	✗ -1
4	0.720614	0.098751	0.805165	0.076782	✓	1	0.776427	0.094234	✓ 1	0.687063	0.130997	✓ 1
5	0.716605	0.100754	0.800766	0.094028	✓	1	0.751315	0.103421	✓ 1	0.685213	0.129883	✓ 1
6	0.720026	0.086444	0.789614	0.078831	✓	1	0.781486	0.093825	✓ 1	0.681972	0.145622	✓ 1
7	0.726222	0.083856	0.793882	0.091536	✓	1	0.752682	0.106525	✓ 1	0.682594	0.147737	✓ 1
8	0.722952	0.077389	0.785611	0.082339	✓	1	0.759433	0.10509	✓ 1	0.681783	0.146078	✓ 1
9	0.71899	0.074593	0.779688	0.092464	✓	1	0.779647	0.103883	✓ 1	0.659122	0.146411	✓ 1
10	0.726248	0.07685	0.79658	0.081647	✓	1	0.749604	0.105818	✓ 1	0.678787	0.158509	✓ 1
11	0.719231	0.082477	0.799868	0.08376	✓	1	0.764084	0.097194	✓ 1	0.665261	0.156945	✓ 1
12	0.72905	0.075827	0.785415	0.08507	✓	1	0.762474	0.093482	✓ 1	0.662998	0.156761	✓ 1
13	0.734102	0.077008	0.792802	0.086302	✓	1	0.758482	0.104564	✓ 1	0.660134	0.15266	✓ 1
14	0.733144	0.078487	0.801119	0.090777	✓	1	0.767554	0.092116	✓ 1	0.687883	0.148392	✓ 1
15	0.728979	0.084105	0.784964	0.082442	✓	1	0.764152	0.091576	✓ 1	0.670123	0.160007	✓ 1
16	0.746481	0.074972	0.803082	0.079695	✓	1	0.759688	0.104178	✗ -1	0.673365	0.150819	✓ 1
17	0.729853	0.075054	0.801995	0.084263	✓	1	0.745276	0.105926	✗ -1	0.667587	0.15174	✓ 1
18	0.726967	0.080003	0.796617	0.0764	✓	1	0.766075	0.105665	✓ 1	0.681281	0.14859	✓ 1
19	0.730034	0.07687	0.792975	0.09214	✓	1	0.750862	0.110545	✓ 1	0.65109	0.154331	✓ 1
20	0.720034	0.079476	0.788928	0.08132	✓	1	0.751571	0.110594	✓ 1	0.656377	0.158173	✓ 1
21	0.72309	0.076538	0.789304	0.093153	✓	1	0.765281	0.101714	✓ 1	0.686208	0.157719	✓ 1
22	0.733205	0.082338	0.797059	0.089218	✓	1	0.758964	0.097323	✓ 1	0.670408	0.156377	✓ 1
23	0.726312	0.080203	0.780024	0.096248	✓	1	0.747153	0.107601	✓ 1	0.668365	0.155455	✓ 1
24	0.712519	0.079068	0.790932	0.082215	✓	1	0.768789	0.09338	✓ 1	0.668094	0.158192	✓ 1
25	0.72718	0.078241	0.786702	0.091944	✓	1	0.741371	0.106283	✗ -1	0.674948	0.160903	✓ 1
26	0.736376	0.074539	0.79211	0.085851	✓	1	0.745115	0.102921	✗ -1	0.662977	0.162435	✓ 1
27	0.715576	0.078802	0.787526	0.076645	✓	1	0.780559	0.092619	✓ 1	0.668478	0.162839	✓ 1
28	0.733959	0.074111	0.795819	0.089085	✓	1	0.759414	0.103957	✓ 1	0.681778	0.158328	✓ 1
29	0.721935	0.077864	0.80022	0.085335	✓	1	0.761987	0.095198	✓ 1	0.6622	0.152122	✓ 1
30	0.715585	0.073342	0.800879	0.077758	✓	1	0.775621	0.091291	✓ 1	0.66182	0.159562	✓ 1
31	0.732161	0.080065	0.784392	0.08836	✓	1	0.747576	0.10697	✗ -1	0.661214	0.152593	✓ 1
32	0.733156	0.070388	0.804051	0.081748	✓	1	0.762911	0.097161	✓ 1	0.665658	0.164584	✓ 1
33	0.722094	0.077498	0.790896	0.079396	✓	1	0.76592	0.099701	✓ 1	0.664095	0.16708	✓ 1
34	0.731382	0.070486	0.784667	0.08952	✓	1	0.736405	0.110344	✗ -1	0.648224	0.162502	✓ 1
35	0.728637	0.083968	0.78639	0.091304	✓	1	0.744117	0.099425	✗ -1	0.675761	0.159443	✓ 1
36	0.730155	0.07743	0.795696	0.091542	✓	1	0.76835	0.091927	✓ 1	0.654642	0.159995	✓ 1
37	0.726527	0.084567	0.773479	0.096967	✓	1	0.732473	0.1001	✗ -1	0.652884	0.163596	✓ 1
38	0.717458	0.077274	0.787801	0.078085	✓	1	0.743877	0.099966	✓ 1	0.657378	0.154513	✓ 1
39	0.724391	0.078745	0.777496	0.095509	✓	1	0.772155	0.107539	✓ 1	0.653097	0.161985	✓ 1
40	0.73382	0.077212	0.801576	0.079337	✓	1	0.751696	0.10223	✓ 1	0.660219	0.157632	✓ 1
41	0.722331	0.080576	0.781573	0.093193	✓	1	0.742363	0.103614	✓ 1	0.647158	0.157647	✓ 1
42	0.721915	0.075704	0.787132	0.091055	✓	1	0.756974	0.113852	✓ 1	0.675585	0.163129	✓ 1
43	0.732566	0.07046	0.795581	0.08174	✓	1	0.750698	0.09528	✓ 1	0.658839	0.158314	✓ 1
44	0.719467	0.074721	0.781077	0.087796	✓	1	0.754931	0.096722	✓ 1	0.661647	0.16387	✓ 1
45	0.720397	0.07695	0.779817	0.08779	✓	1	0.776477	0.099117	✓ 1	0.656375	0.160781	✓ 1
46	0.720961	0.079163	0.786697	0.08847	✓	1	0.740605	0.103713	✓ 1	0.662654	0.156163	✓ 1
47	0.71442	0.07884	0.777531	0.090077	✓	1	0.724583	0.110352	✗ -1	0.658697	0.168265	✓ 1
48	0.721363	0.086555	0.795436	0.086234	✓	1	0.756702	0.092034	✓ 1	0.655932	0.156241	✓ 1
49	0.736041	0.073314	0.787628	0.088092	✓	1	0.754363	0.100687	✓ 1	0.676173	0.168686	✓ 1
50	0.723235	0.080563	0.79105	0.092173	✓	1	0.733807	0.108851	✗ -1	0.654879	0.16468	✓ 1

B.4 A=3.9

A=3.9											
Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD != WTD		SHS Mean	SHS Stdv.	TTest SHS != WTD	STK Mean	STK Stdv.	TTest STK != WTD	
1	0.647377	0.132309	0.758473	0.111472	✓	1	0.715501	0.126944	✓	1	0.666351	0.118174	✗ -1
2	0.692919	0.123214	0.802024	0.084764	✓	1	0.760189	0.102682	✓	1	0.682295	0.111077	✗ -1
3	0.712467	0.119656	0.794511	0.09151	✓	1	0.776745	0.090508	✓	1	0.691184	0.124392	✗ -1
4	0.719282	0.094679	0.797936	0.078415	✓	1	0.749713	0.102672	✓	1	0.694242	0.118179	✓ 1
5	0.709817	0.102726	0.803479	0.086105	✓	1	0.766034	0.102456	✓	1	0.689432	0.138005	✗ -1
6	0.717085	0.088422	0.784258	0.081711	✓	1	0.777986	0.089115	✓	1	0.67041	0.127288	✓ 1
7	0.723892	0.086572	0.789487	0.089729	✓	1	0.756211	0.095856	✓	1	0.687574	0.133442	✓ 1
8	0.723927	0.088887	0.799105	0.078707	✓	1	0.749124	0.105527	✓	1	0.692066	0.129757	✓ 1
9	0.723587	0.080256	0.794831	0.078514	✓	1	0.771053	0.101499	✓	1	0.676903	0.1399	✓ 1
10	0.729508	0.089599	0.793407	0.093236	✓	1	0.755782	0.109587	✓	1	0.683213	0.145637	✓ 1
11	0.719125	0.086582	0.783523	0.083925	✓	1	0.770313	0.093323	✓	1	0.677577	0.149602	✓ 1
12	0.719902	0.088754	0.784217	0.085268	✓	1	0.771481	0.112852	✓	1	0.676572	0.146461	✓ 1
13	0.728424	0.084107	0.801672	0.080373	✓	1	0.759693	0.105316	✓	1	0.674676	0.145969	✓ 1
14	0.716043	0.081707	0.781108	0.090979	✓	1	0.752135	0.114345	✓	1	0.68807	0.145561	✓ 1
15	0.724507	0.089817	0.79603	0.085583	✓	1	0.76189	0.103848	✓	1	0.686272	0.146015	✓ 1
16	0.725345	0.077977	0.784451	0.090715	✓	1	0.751209	0.098799	✓	1	0.681791	0.147936	✓ 1
17	0.723456	0.076541	0.788922	0.089849	✓	1	0.734239	0.108621	✗	-1	0.677839	0.153144	✓ 1
18	0.730167	0.077148	0.791992	0.087739	✓	1	0.766139	0.094315	✓	1	0.683357	0.148107	✓ 1
19	0.735179	0.085961	0.78896	0.090273	✓	1	0.745887	0.09929	✗	-1	0.674582	0.157524	✓ 1
20	0.720593	0.072217	0.79154	0.086798	✓	1	0.738366	0.104301	✓	1	0.672317	0.159693	✓ 1
21	0.728283	0.078811	0.779139	0.099411	✓	1	0.763183	0.103354	✓	1	0.689735	0.164542	✓ 1
22	0.73087	0.07268	0.784727	0.089168	✓	1	0.743498	0.113604	✗	-1	0.675136	0.162858	✓ 1
23	0.715477	0.079458	0.769302	0.093825	✓	1	0.746963	0.100304	✓	1	0.669485	0.159632	✓ 1
24	0.710512	0.089004	0.79096	0.087041	✓	1	0.757596	0.103981	✓	1	0.669522	0.15366	✓ 1
25	0.735105	0.076479	0.778676	0.092547	✓	1	0.744274	0.113941	✗	-1	0.674619	0.161664	✓ 1
26	0.713214	0.077349	0.784718	0.095099	✓	1	0.754779	0.101443	✓	1	0.674483	0.162672	✓ 1
27	0.726492	0.078083	0.785922	0.088911	✓	1	0.758709	0.110053	✓	1	0.673468	0.157697	✓ 1
28	0.723772	0.07856	0.772227	0.095649	✓	1	0.746524	0.108861	✓	1	0.687248	0.159854	✓ 1
29	0.72905	0.079058	0.779767	0.082885	✓	1	0.746246	0.106951	✗	-1	0.674479	0.165261	✓ 1
30	0.729755	0.069832	0.78369	0.101444	✓	1	0.769389	0.101044	✓	1	0.678183	0.163468	✓ 1
31	0.72867	0.076166	0.790242	0.081362	✓	1	0.745159	0.100941	✗	-1	0.671654	0.170776	✓ 1
32	0.720831	0.081247	0.786019	0.088805	✓	1	0.754837	0.097807	✓	1	0.665635	0.160779	✓ 1
33	0.722202	0.080084	0.785555	0.088346	✓	1	0.764966	0.107621	✓	1	0.671769	0.163808	✓ 1
34	0.732189	0.073135	0.782788	0.092587	✓	1	0.739029	0.111584	✗	-1	0.673126	0.163386	✓ 1
35	0.72656	0.078442	0.782034	0.09799	✓	1	0.746476	0.099067	✓	1	0.680827	0.158739	✓ 1
36	0.724838	0.070115	0.787916	0.090292	✓	1	0.762407	0.095375	✓	1	0.677847	0.166027	✓ 1
37	0.731951	0.082573	0.789357	0.084464	✓	1	0.74708	0.100921	✗	-1	0.671943	0.163686	✓ 1
38	0.728584	0.076383	0.784363	0.088525	✓	1	0.748721	0.099262	✓	1	0.677334	0.160149	✓ 1
39	0.721464	0.083333	0.7824	0.086981	✓	1	0.747547	0.102063	✓	1	0.673895	0.167231	✓ 1
40	0.733598	0.077429	0.766508	0.089595	✓	1	0.739231	0.101197	✗	-1	0.66792	0.167591	✓ 1
41	0.725232	0.084096	0.773144	0.095703	✓	1	0.74088	0.10167	✗	-1	0.662986	0.175984	✓ 1
42	0.715111	0.081098	0.792077	0.089144	✓	1	0.75855	0.10464	✓	1	0.679893	0.170663	✓ 1
43	0.729054	0.078401	0.782653	0.092113	✓	1	0.722308	0.119755	✗	-1	0.666884	0.164483	✓ 1
44	0.72388	0.07848	0.775902	0.092421	✓	1	0.750739	0.094044	✓	1	0.659704	0.169651	✓ 1
45	0.72605	0.07691	0.777653	0.093473	✓	1	0.771232	0.097839	✓	1	0.66985	0.1638	✓ 1
46	0.730582	0.076754	0.784668	0.087933	✓	1	0.753956	0.097206	✓	1	0.672133	0.174793	✓ 1
47	0.730839	0.076673	0.776105	0.097796	✓	1	0.735928	0.113929	✗	-1	0.663752	0.164391	✓ 1
48	0.721349	0.084437	0.786796	0.082893	✓	1	0.759983	0.107073	✓	1	0.667948	0.170563	✓ 1
49	0.729122	0.085545	0.787653	0.08769	✓	1	0.742273	0.102861	✗	-1	0.679144	0.177623	✓ 1
50	0.719218	0.077729	0.789326	0.09337	✓	1	0.737229	0.104699	✗	-1	0.670384	0.165425	✓ 1

B.5 T-Tests by Landscape and Successive A Values for WTD

KD=WTD											
Landscape	μ A=1	σ A=1	μ A=3.1	σ A=3.1	μ A=3.6	σ A=3.6	μ A=3.9	σ A=3.9	μ 1!=3.1	μ 3.1!=3.6	μ 3.6!=3.9
1	0.684882	0.121618	0.640029	0.143723	0.668878	0.13145	0.647377	0.132309	✓ 1	✓ 1	✗ -1
2	0.687477	0.119064	0.685225	0.13055	0.69923	0.109417	0.692919	0.123214	✗ -1	✗ -1	✗ -1
3	0.684755	0.12406	0.683283	0.124613	0.682715	0.121451	0.712467	0.119656	✗ -1	✗ -1	✓ 1
4	0.700683	0.11311	0.714847	0.103599	0.720614	0.098751	0.719282	0.094679	✗ -1	✗ -1	✗ -1
5	0.698064	0.111194	0.711711	0.096364	0.716605	0.100754	0.709817	0.102726	✗ -1	✗ -1	✗ -1
6	0.693739	0.119736	0.724081	0.092492	0.720026	0.086444	0.717085	0.088422	✓ 1	✗ -1	✗ -1
7	0.709505	0.105973	0.717974	0.083031	0.726222	0.083856	0.723892	0.086572	✗ -1	✗ -1	✗ -1
8	0.702985	0.104832	0.730318	0.080306	0.722952	0.077389	0.723927	0.088887	✓ 1	✗ -1	✗ -1
9	0.695203	0.105259	0.712636	0.084233	0.71899	0.074593	0.723587	0.080256	✗ -1	✗ -1	✗ -1
10	0.712062	0.108715	0.726044	0.085414	0.726248	0.07685	0.729508	0.089599	✗ -1	✗ -1	✗ -1
11	0.69707	0.103368	0.721038	0.092796	0.719231	0.082477	0.719125	0.086582	✓ 1	✗ -1	✗ -1
12	0.697958	0.112889	0.722351	0.074053	0.72905	0.075827	0.719902	0.088754	✓ 1	✗ -1	✗ -1
13	0.716086	0.104594	0.721093	0.080374	0.734102	0.077008	0.728424	0.084107	✗ -1	✗ -1	✗ -1
14	0.707458	0.104027	0.7251	0.072713	0.733144	0.078487	0.716043	0.081707	✗ -1	✗ -1	✓ 1
15	0.701595	0.103989	0.72498	0.080153	0.728979	0.084105	0.724507	0.089817	✓ 1	✗ -1	✗ -1
16	0.715643	0.100254	0.736115	0.074458	0.746481	0.074972	0.725345	0.077977	✓ 1	✗ -1	✓ 1
17	0.701386	0.10257	0.723098	0.076929	0.729853	0.075054	0.723456	0.076541	✓ 1	✗ -1	✗ -1
18	0.699386	0.111457	0.732422	0.078128	0.726967	0.080003	0.730167	0.077148	✓ 1	✗ -1	✗ -1
19	0.710602	0.101651	0.725352	0.072269	0.730034	0.07687	0.735179	0.085961	✗ -1	✗ -1	✗ -1
20	0.703459	0.107197	0.731414	0.074799	0.720034	0.079476	0.720593	0.072217	✓ 1	✗ -1	✗ -1
21	0.704564	0.105387	0.725409	0.078614	0.72309	0.076538	0.728283	0.078811	✓ 1	✗ -1	✗ -1
22	0.719116	0.103226	0.731046	0.074876	0.733205	0.082338	0.73087	0.07268	✗ -1	✗ -1	✗ -1
23	0.707719	0.102246	0.721559	0.07301	0.726312	0.080203	0.715477	0.079458	✗ -1	✗ -1	✗ -1
24	0.705619	0.103606	0.720577	0.075168	0.712519	0.079068	0.710512	0.089004	✗ -1	✗ -1	✗ -1
25	0.721959	0.1029	0.735565	0.073085	0.72718	0.078241	0.735105	0.076479	✗ -1	✗ -1	✗ -1
26	0.713581	0.092427	0.727977	0.078886	0.736376	0.074539	0.713214	0.077349	✗ -1	✗ -1	✓ 1
27	0.702832	0.099305	0.720186	0.075493	0.715576	0.078802	0.726492	0.078083	✓ 1	✗ -1	✗ -1
28	0.719331	0.094545	0.726096	0.073314	0.733959	0.074111	0.723772	0.07856	✗ -1	✗ -1	✗ -1
29	0.710846	0.097283	0.730294	0.074579	0.721935	0.077864	0.72905	0.079058	✓ 1	✗ -1	✗ -1
30	0.710787	0.101626	0.72804	0.078514	0.715585	0.073342	0.729755	0.069832	✗ -1	✗ -1	✓ 1
31	0.722009	0.09758	0.71999	0.073832	0.732161	0.080065	0.72867	0.076166	✗ -1	✗ -1	✗ -1
32	0.711615	0.093664	0.73116	0.075047	0.733156	0.070388	0.720831	0.081247	✓ 1	✗ -1	✗ -1
33	0.718205	0.094413	0.718915	0.079285	0.722094	0.077498	0.722202	0.080084	✗ -1	✗ -1	✗ -1
34	0.721808	0.093274	0.7308	0.073688	0.731382	0.070486	0.732189	0.073135	✗ -1	✗ -1	✗ -1
35	0.712471	0.106477	0.723712	0.073613	0.728637	0.083968	0.72656	0.078442	✗ -1	✗ -1	✗ -1
36	0.703958	0.098777	0.718211	0.077439	0.730155	0.07743	0.724838	0.070115	✗ -1	✗ -1	✗ -1
37	0.728623	0.092716	0.732189	0.072761	0.726527	0.084567	0.731951	0.082573	✗ -1	✗ -1	✗ -1
38	0.718543	0.090921	0.710864	0.081912	0.717458	0.077274	0.728584	0.076383	✗ -1	✗ -1	✗ -1
39	0.70511	0.093918	0.727764	0.078916	0.724391	0.078745	0.721464	0.083333	✓ 1	✗ -1	✗ -1
40	0.728001	0.093181	0.733495	0.077549	0.73382	0.077212	0.733598	0.077429	✗ -1	✗ -1	✗ -1
41	0.712711	0.09836	0.720976	0.083972	0.722331	0.080576	0.725232	0.084096	✗ -1	✗ -1	✗ -1
42	0.706687	0.100932	0.716796	0.07727	0.721915	0.075704	0.715111	0.081098	✗ -1	✗ -1	✗ -1
43	0.73096	0.100518	0.721736	0.073172	0.732566	0.07046	0.729054	0.078401	✗ -1	✗ -1	✗ -1
44	0.703552	0.097078	0.725452	0.073637	0.719467	0.074721	0.72388	0.07848	✓ 1	✗ -1	✗ -1
45	0.70899	0.096284	0.718843	0.073004	0.720397	0.07695	0.72605	0.07691	✗ -1	✗ -1	✗ -1
46	0.708277	0.090335	0.729812	0.074789	0.720961	0.079163	0.730582	0.076754	✓ 1	✗ -1	✗ -1
47	0.712029	0.091367	0.718129	0.0775	0.71442	0.07884	0.730839	0.076673	✗ -1	✗ -1	✓ 1
48	0.71104	0.104562	0.729141	0.071806	0.721363	0.086555	0.721349	0.084437	✓ 1	✗ -1	✗ -1
49	0.719724	0.096669	0.721022	0.073804	0.736041	0.073314	0.729122	0.085545	✗ -1	✓ 1	✗ -1
50	0.711279	0.097382	0.724685	0.077806	0.723235	0.080563	0.719218	0.077729	✗ -1	✗ -1	✗ -1

APPENDIX C SEGREGATION STATISTICAL TESTS

C.1 A=1

A=1.0											
Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD != WTD		SHS Mean	SHS Stdv.	TTest SHS != WTD	STK Mean	STK Stdv.	TTest STK != WTD	
1	0.656009	0.136437	0.67567	0.121409	✗	-1	0.692123	0.139062	✓	1	0.44514	0.124905	✓ 1
2	0.638874	0.136619	0.664599	0.126218	✗	-1	0.66543	0.125283	✓	1	0.505266	0.11745	✓ 1
3	0.654591	0.141854	0.667263	0.129044	✗	-1	0.848041	0.149751	✓	1	0.536298	0.125071	✓ 1
4	0.641322	0.119186	0.666199	0.115674	✓	1	0.688047	0.134422	✓	1	0.562801	0.119929	✓ 1
5	0.636094	0.130697	0.667111	0.115109	✓	1	0.679541	0.130347	✓	1	0.583421	0.123865	✓ 1
6	0.643561	0.138008	0.675576	0.125579	✓	1	0.854851	0.162788	✓	1	0.602909	0.127653	✓ 1
7	0.636459	0.130996	0.673673	0.126863	✓	1	0.709468	0.140292	✓	1	0.610515	0.133936	✗ -1
8	0.626766	0.14422	0.673845	0.124654	✓	1	0.66974	0.124014	✓	1	0.610553	0.134897	✗ -1
9	0.634161	0.131637	0.676348	0.125458	✓	1	0.847058	0.154191	✓	1	0.633336	0.139142	✗ -1
10	0.64476	0.13729	0.670632	0.129605	✗	-1	0.706942	0.125864	✓	1	0.639254	0.143766	✗ -1
11	0.619073	0.123088	0.678854	0.134538	✓	1	0.687	0.125808	✓	1	0.652594	0.136707	✓ 1
12	0.644863	0.135426	0.674953	0.126388	✓	1	0.844728	0.144053	✓	1	0.654982	0.141537	✗ -1
13	0.640202	0.136998	0.687193	0.132185	✓	1	0.672053	0.132856	✓	1	0.670696	0.145464	✓ 1
14	0.636877	0.131576	0.680512	0.131511	✓	1	0.675766	0.136117	✓	1	0.673243	0.14964	✓ 1
15	0.623646	0.138009	0.668596	0.132834	✓	1	0.849597	0.153379	✓	1	0.687678	0.145196	✓ 1
16	0.643199	0.142802	0.67024	0.129318	✓	1	0.708558	0.143448	✓	1	0.685939	0.146465	✓ 1
17	0.635658	0.132495	0.657842	0.120724	✗	-1	0.673573	0.130142	✓	1	0.695564	0.149432	✓ 1
18	0.642053	0.152564	0.676547	0.125693	✓	1	0.85981	0.145856	✓	1	0.700866	0.146453	✓ 1
19	0.63136	0.130191	0.671497	0.139465	✓	1	0.700184	0.140617	✓	1	0.706295	0.142609	✓ 1
20	0.633246	0.144173	0.672439	0.130313	✓	1	0.679161	0.140376	✓	1	0.708866	0.147154	✓ 1
21	0.635164	0.123132	0.685936	0.127308	✓	1	0.852243	0.143217	✓	1	0.714509	0.141059	✓ 1
22	0.631354	0.11741	0.671132	0.119526	✓	1	0.692854	0.123002	✓	1	0.722863	0.145148	✓ 1
23	0.633868	0.130251	0.681792	0.110303	✓	1	0.689757	0.1336	✓	1	0.729939	0.146401	✓ 1
24	0.637781	0.147404	0.692912	0.121459	✓	1	0.861827	0.146302	✓	1	0.733558	0.13774	✓ 1
25	0.616026	0.136763	0.669064	0.12732	✓	1	0.679795	0.130855	✓	1	0.742	0.136625	✓ 1
26	0.636287	0.131389	0.672421	0.127897	✓	1	0.672871	0.136328	✓	1	0.741421	0.136547	✓ 1
27	0.622646	0.126157	0.690678	0.116903	✓	1	0.838389	0.142126	✓	1	0.741015	0.136805	✓ 1
28	0.662067	0.136176	0.67398	0.122793	✗	-1	0.686123	0.132086	✗	-1	0.750892	0.133885	✓ 1
29	0.622462	0.130645	0.675883	0.134923	✓	1	0.66095	0.128726	✓	1	0.74555	0.136237	✓ 1
30	0.627836	0.122537	0.68119	0.131892	✓	1	0.839009	0.138432	✓	1	0.748895	0.131581	✓ 1
31	0.632123	0.132183	0.677959	0.1179	✓	1	0.707006	0.139969	✓	1	0.750947	0.124059	✓ 1
32	0.626711	0.132731	0.660892	0.1254	✓	1	0.666231	0.146479	✓	1	0.757035	0.133546	✓ 1
33	0.638056	0.139891	0.686889	0.125119	✓	1	0.82312	0.152193	✓	1	0.760939	0.130962	✓ 1
34	0.633088	0.149756	0.673804	0.124547	✓	1	0.674702	0.139819	✓	1	0.762737	0.129547	✓ 1
35	0.634684	0.122919	0.669433	0.124572	✓	1	0.666196	0.135756	✓	1	0.773137	0.129876	✓ 1
36	0.634281	0.131362	0.68688	0.133558	✓	1	0.836921	0.148813	✓	1	0.769868	0.12287	✓ 1
37	0.625813	0.137706	0.66081	0.114204	✓	1	0.688342	0.139338	✓	1	0.773219	0.127779	✓ 1
38	0.62505	0.127227	0.687208	0.128933	✓	1	0.668503	0.127502	✓	1	0.774898	0.111673	✓ 1
39	0.620857	0.12466	0.674564	0.134551	✓	1	0.84393	0.152479	✓	1	0.779602	0.115594	✓ 1
40	0.652333	0.138223	0.67017	0.123908	✗	-1	0.681909	0.136132	✓	1	0.77795	0.119601	✓ 1
41	0.641649	0.142385	0.684611	0.127441	✓	1	0.68538	0.130309	✓	1	0.781798	0.113262	✓ 1
42	0.640953	0.13569	0.668246	0.13934	✓	1	0.853368	0.125599	✓	1	0.789386	0.111522	✓ 1
43	0.635482	0.132007	0.664997	0.112136	✓	1	0.691149	0.136708	✓	1	0.787637	0.11272	✓ 1
44	0.62462	0.136778	0.678471	0.11528	✓	1	0.66793	0.126683	✓	1	0.791041	0.114233	✓ 1
45	0.61324	0.118724	0.667506	0.120855	✓	1	0.875637	0.13579	✓	1	0.789763	0.101044	✓ 1
46	0.658114	0.14228	0.672371	0.13285	✗	-1	0.702202	0.135755	✓	1	0.798298	0.103728	✓ 1
47	0.613678	0.123653	0.672518	0.130153	✓	1	0.675667	0.126501	✓	1	0.802275	0.104072	✓ 1
48	0.62483	0.120949	0.675392	0.121676	✓	1	0.826231	0.142186	✓	1	0.798632	0.106578	✓ 1
49	0.608591	0.121599	0.678851	0.121515	✓	1	0.688327	0.127545	✓	1	0.799459	0.103244	✓ 1
50	0.64145	0.13814	0.652801	0.12188	✗	-1	0.655389	0.126442	✗	-1	0.805412	0.102423	✓ 1

C.2 A=3.1

A=3.1											
Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD != WTD		SHS Mean	SHS Stdv.	TTest SHS != WTD	STK Mean	STK Stdv.	TTest STK != WTD	
1	0.647681	0.140271	0.709345	0.125705	✓	1	0.698588	0.15233	✓	1	0.457725	0.113279	✓ 1
2	0.647789	0.133511	0.684403	0.124466	✓	1	0.649866	0.118928	✗	-1	0.50833	0.12283	✓ 1
3	0.646842	0.130996	0.710637	0.130471	✓	1	0.858471	0.142166	✓	1	0.531228	0.114797	✓ 1
4	0.633409	0.135589	0.687401	0.133161	✓	1	0.672713	0.141095	✓	1	0.565012	0.112784	✓ 1
5	0.636512	0.126892	0.716781	0.138378	✓	1	0.680263	0.136461	✓	1	0.578807	0.127397	✓ 1
6	0.627161	0.132222	0.677591	0.134418	✓	1	0.818789	0.144154	✓	1	0.592023	0.129019	✓ 1
7	0.632328	0.129962	0.709015	0.133078	✓	1	0.703	0.143914	✓	1	0.606073	0.128884	✓ 1
8	0.628863	0.120496	0.683287	0.121645	✓	1	0.670237	0.143362	✓	1	0.631942	0.133339	✗ -1
9	0.634579	0.14501	0.710573	0.13421	✓	1	0.846871	0.135376	✓	1	0.63376	0.135995	✗ -1
10	0.633845	0.141458	0.70148	0.127556	✓	1	0.695848	0.131935	✓	1	0.643699	0.139852	✗ -1
11	0.635184	0.132911	0.726933	0.124523	✓	1	0.697597	0.13421	✓	1	0.648325	0.14501	✗ -1
12	0.639705	0.136362	0.698646	0.133305	✓	1	0.825909	0.149226	✓	1	0.653851	0.141909	✗ -1
13	0.645728	0.134265	0.732439	0.12731	✓	1	0.715784	0.119442	✓	1	0.668351	0.146301	✗ -1
14	0.646991	0.152022	0.707909	0.124563	✓	1	0.674266	0.133293	✗	-1	0.667368	0.143413	✗ -1
15	0.631813	0.126376	0.741436	0.132434	✓	1	0.858851	0.160038	✓	1	0.678442	0.149978	✓ 1
16	0.634392	0.124688	0.705038	0.128	✓	1	0.694515	0.135595	✓	1	0.680494	0.149213	✓ 1
17	0.6305	0.132131	0.733424	0.124327	✓	1	0.694819	0.136602	✓	1	0.694526	0.151547	✓ 1
18	0.62317	0.129693	0.690509	0.128832	✓	1	0.832465	0.148547	✓	1	0.703424	0.151211	✓ 1
19	0.635237	0.1245	0.747325	0.115171	✓	1	0.726561	0.139552	✓	1	0.706535	0.149343	✓ 1
20	0.634383	0.129773	0.721974	0.125134	✓	1	0.674939	0.142502	✓	1	0.7095	0.152416	✓ 1
21	0.625377	0.135309	0.754348	0.139893	✓	1	0.862594	0.15376	✓	1	0.714635	0.146668	✓ 1
22	0.651307	0.150332	0.701588	0.118489	✓	1	0.691643	0.131773	✓	1	0.725816	0.143536	✓ 1
23	0.623915	0.127753	0.760915	0.128057	✓	1	0.720784	0.133024	✓	1	0.73283	0.141353	✓ 1
24	0.644901	0.154638	0.740681	0.130283	✓	1	0.85764	0.141648	✓	1	0.730673	0.144472	✓ 1
25	0.625678	0.118351	0.763716	0.126283	✓	1	0.718073	0.153763	✓	1	0.736298	0.140041	✓ 1
26	0.638102	0.126363	0.696108	0.130516	✓	1	0.691895	0.139347	✓	1	0.739597	0.141109	✓ 1
27	0.651833	0.132312	0.755789	0.136411	✓	1	0.880772	0.131621	✓	1	0.743032	0.141956	✓ 1
28	0.641152	0.14269	0.719547	0.13093	✓	1	0.703272	0.139935	✓	1	0.740681	0.137979	✓ 1
29	0.631728	0.13025	0.785453	0.132228	✓	1	0.726506	0.139247	✓	1	0.753725	0.134568	✓ 1
30	0.627974	0.131852	0.716474	0.120736	✓	1	0.858942	0.153625	✓	1	0.754895	0.130255	✓ 1
31	0.646035	0.131268	0.766573	0.125808	✓	1	0.751316	0.143964	✓	1	0.761661	0.133414	✓ 1
32	0.648845	0.138756	0.734287	0.128069	✓	1	0.683406	0.126948	✓	1	0.765406	0.126889	✓ 1
33	0.628064	0.132513	0.764699	0.132699	✓	1	0.886047	0.140957	✓	1	0.767956	0.129388	✓ 1
34	0.641436	0.140306	0.726401	0.117542	✓	1	0.710313	0.148594	✓	1	0.776129	0.127813	✓ 1
35	0.638924	0.144202	0.750497	0.131895	✓	1	0.728877	0.142913	✓	1	0.777801	0.113921	✓ 1
36	0.629325	0.125598	0.737553	0.133395	✓	1	0.840187	0.145161	✓	1	0.775079	0.115128	✓ 1
37	0.636591	0.123861	0.772591	0.128425	✓	1	0.748368	0.14915	✓	1	0.777319	0.120177	✓ 1
38	0.64169	0.140562	0.723099	0.128892	✓	1	0.67698	0.140232	✓	1	0.781661	0.119686	✓ 1
39	0.641243	0.133088	0.755266	0.136505	✓	1	0.857751	0.141333	✓	1	0.778643	0.119479	✓ 1
40	0.64993	0.133084	0.734161	0.132626	✓	1	0.703529	0.136925	✓	1	0.782263	0.11707	✓ 1
41	0.624184	0.121199	0.758985	0.130282	✓	1	0.734029	0.145727	✓	1	0.779477	0.117634	✓ 1
42	0.643515	0.140682	0.722064	0.127043	✓	1	0.851506	0.14682	✓	1	0.786608	0.108503	✓ 1
43	0.629883	0.138547	0.775155	0.132903	✓	1	0.739597	0.132863	✓	1	0.790149	0.108879	✓ 1
44	0.62957	0.141989	0.724436	0.122998	✓	1	0.679556	0.131505	✓	1	0.788746	0.110972	✓ 1
45	0.634468	0.116721	0.751956	0.131649	✓	1	0.888965	0.14713	✓	1	0.792047	0.107412	✓ 1
46	0.647839	0.13963	0.724687	0.137709	✓	1	0.702553	0.145569	✓	1	0.799377	0.10886	✓ 1
47	0.645126	0.141288	0.773623	0.138391	✓	1	0.73483	0.150952	✓	1	0.798266	0.094651	✓ 1
48	0.633263	0.131451	0.744102	0.123969	✓	1	0.867848	0.147188	✓	1	0.803114	0.099166	✓ 1
49	0.640006	0.125628	0.771298	0.129474	✓	1	0.758994	0.149068	✓	1	0.802538	0.09877	✓ 1
50	0.638044	0.136152	0.727082	0.130196	✓	1	0.698915	0.139234	✓	1	0.805275	0.086303	✓ 1

C.3 A=3.6

A=3.6											
Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD != WTD		SHS Mean	SHS Stdv.	TTest SHS != WTD	STK Mean	STK Stdv.	TTest STK != WTD	
1	0.655667	0.151484	0.724535	0.119526	✓	1	0.689953	0.148734	✓	1	0.447368	0.1173	✓	1
2	0.645094	0.140195	0.66981	0.135624	✗	-1	0.6685	0.135904	✗	-1	0.506237	0.110489	✓	1
3	0.643289	0.140562	0.727254	0.131196	✓	1	0.861029	0.138316	✓	1	0.541173	0.107896	✓	1
4	0.64152	0.131204	0.697301	0.136779	✓	1	0.702295	0.134713	✓	1	0.558994	0.116876	✓	1
5	0.647591	0.150035	0.722328	0.126756	✓	1	0.694465	0.149966	✓	1	0.590029	0.121803	✓	1
6	0.648213	0.12848	0.694705	0.129997	✓	1	0.860319	0.145436	✓	1	0.590345	0.120307	✓	1
7	0.627158	0.127031	0.726096	0.12711	✓	1	0.71648	0.15058	✓	1	0.607895	0.12719	✗	-1
8	0.641044	0.14239	0.704901	0.124561	✓	1	0.675924	0.136442	✓	1	0.612067	0.132629	✓	1
9	0.642588	0.130163	0.716412	0.137526	✓	1	0.839135	0.137768	✓	1	0.628778	0.13936	✗	-1
10	0.614988	0.131697	0.707579	0.140312	✓	1	0.715734	0.133236	✓	1	0.63583	0.141858	✗	-1
11	0.648114	0.146554	0.727705	0.128822	✓	1	0.700398	0.142862	✓	1	0.642526	0.143025	✗	-1
12	0.637494	0.142876	0.713292	0.127362	✓	1	0.837807	0.142253	✓	1	0.650301	0.134095	✗	-1
13	0.641348	0.132858	0.714368	0.121898	✓	1	0.701529	0.14895	✓	1	0.653494	0.142413	✗	-1
14	0.628617	0.126544	0.698719	0.132197	✓	1	0.671743	0.133466	✓	1	0.651772	0.146861	✗	-1
15	0.637009	0.127456	0.726664	0.137875	✓	1	0.853547	0.132406	✓	1	0.664602	0.148796	✓	1
16	0.641503	0.13312	0.704942	0.125013	✓	1	0.68336	0.138514	✓	1	0.677646	0.145907	✓	1
17	0.633775	0.147505	0.737412	0.125464	✓	1	0.702149	0.145774	✓	1	0.680749	0.144303	✓	1
18	0.644096	0.14362	0.714933	0.133119	✓	1	0.844567	0.143271	✓	1	0.687675	0.149486	✓	1
19	0.637713	0.125544	0.709643	0.131004	✓	1	0.711503	0.140899	✓	1	0.702634	0.145306	✓	1
20	0.636058	0.140047	0.710026	0.117437	✓	1	0.696412	0.132283	✓	1	0.70555	0.141636	✓	1
21	0.637012	0.127238	0.726292	0.137704	✓	1	0.867857	0.160006	✓	1	0.713594	0.139561	✓	1
22	0.634716	0.129373	0.717678	0.119314	✓	1	0.709977	0.146148	✓	1	0.714702	0.142387	✓	1
23	0.656757	0.143165	0.731456	0.128993	✓	1	0.697319	0.141206	✓	1	0.71817	0.142663	✓	1
24	0.652275	0.134741	0.727237	0.11623	✓	1	0.828763	0.140008	✓	1	0.721345	0.14548	✓	1
25	0.632632	0.134916	0.755064	0.132013	✓	1	0.713415	0.140779	✓	1	0.721079	0.142959	✓	1
26	0.651225	0.141297	0.727281	0.130707	✓	1	0.677009	0.135412	✗	-1	0.730026	0.138365	✓	1
27	0.653982	0.135458	0.74214	0.136828	✓	1	0.865099	0.142261	✓	1	0.74169	0.13581	✓	1
28	0.639784	0.147064	0.725228	0.137462	✓	1	0.701594	0.133285	✓	1	0.73717	0.13764	✓	1
29	0.638599	0.135714	0.746436	0.136862	✓	1	0.725465	0.129508	✓	1	0.742787	0.133187	✓	1
30	0.647254	0.139665	0.728211	0.125232	✓	1	0.858532	0.146698	✓	1	0.745029	0.136325	✓	1
31	0.636865	0.131955	0.74424	0.131292	✓	1	0.736243	0.138835	✓	1	0.757234	0.129922	✓	1
32	0.630728	0.13319	0.715485	0.131356	✓	1	0.704009	0.11935	✓	1	0.761298	0.125454	✓	1
33	0.628784	0.128018	0.746632	0.133497	✓	1	0.868099	0.147668	✓	1	0.759418	0.133276	✓	1
34	0.645433	0.134677	0.71883	0.125946	✓	1	0.698526	0.142892	✓	1	0.768947	0.122284	✓	1
35	0.636781	0.149681	0.753175	0.127781	✓	1	0.705029	0.15057	✓	1	0.769541	0.123858	✓	1
36	0.637485	0.140099	0.723816	0.123486	✓	1	0.858871	0.13946	✓	1	0.76981	0.118423	✓	1
37	0.627097	0.121075	0.739263	0.125049	✓	1	0.715172	0.144379	✓	1	0.772994	0.130535	✓	1
38	0.631854	0.139699	0.737965	0.139505	✓	1	0.698553	0.13342	✓	1	0.772737	0.121966	✓	1
39	0.64355	0.131015	0.755468	0.136928	✓	1	0.873447	0.153665	✓	1	0.772646	0.118566	✓	1
40	0.638942	0.132156	0.736289	0.126969	✓	1	0.718137	0.14837	✓	1	0.777409	0.133906	✓	1
41	0.61738	0.123165	0.748421	0.123691	✓	1	0.70086	0.134111	✓	1	0.782702	0.128306	✓	1
42	0.63776	0.128712	0.729167	0.131102	✓	1	0.850018	0.158043	✓	1	0.784161	0.12143	✓	1
43	0.631497	0.13536	0.76305	0.1296	✓	1	0.728561	0.139606	✓	1	0.788763	0.1168	✓	1
44	0.651146	0.144302	0.732965	0.122073	✓	1	0.694143	0.134992	✓	1	0.790374	0.118115	✓	1
45	0.642044	0.127894	0.735187	0.12516	✓	1	0.878556	0.155636	✓	1	0.789053	0.113427	✓	1
46	0.637538	0.14258	0.726313	0.123242	✓	1	0.70338	0.143663	✓	1	0.794219	0.109978	✓	1
47	0.639442	0.134631	0.759687	0.124761	✓	1	0.705281	0.15431	✓	1	0.804175	0.116035	✓	1
48	0.637655	0.118077	0.724702	0.126926	✓	1	0.837632	0.145927	✓	1	0.80195	0.113346	✓	1
49	0.653307	0.148119	0.762737	0.124956	✓	1	0.738561	0.145235	✓	1	0.80131	0.109727	✓	1
50	0.631076	0.127168	0.734047	0.133882	✓	1	0.682447	0.138742	✓	1	0.801743	0.113241	✓	1

C.4 A=3.9

A=3.9											
Landscape	WTD Mean	WTD Stdv.	IPD Mean	IPD Stdv.	TTest IPD != WTD		SHS Mean	SHS Stdv.	TTest SHS != WTD	STK Mean	STK Stdv.	TTest STK != WTD	
1	0.66036	0.144093	0.689222	0.128802	✓	1	0.686289	0.146705	✗	-1	0.459649	0.11334	✓ 1
2	0.650231	0.136214	0.703722	0.118223	✓	1	0.692287	0.133915	✓	1	0.498444	0.117565	✓ 1
3	0.637848	0.136573	0.705538	0.138933	✓	1	0.847433	0.139673	✓	1	0.532664	0.110527	✓ 1
4	0.628164	0.131724	0.712225	0.119928	✓	1	0.720854	0.119217	✓	1	0.550067	0.114919	✓ 1
5	0.638096	0.125167	0.733728	0.139951	✓	1	0.712456	0.136809	✓	1	0.571687	0.110093	✓ 1
6	0.639605	0.138085	0.717994	0.121871	✓	1	0.872012	0.140268	✓	1	0.591605	0.117615	✓ 1
7	0.63783	0.133941	0.738687	0.136943	✓	1	0.713851	0.140829	✓	1	0.594594	0.120028	✓ 1
8	0.633746	0.137585	0.716035	0.130804	✓	1	0.710155	0.145369	✓	1	0.618766	0.125653	✗ -1
9	0.63688	0.127563	0.736298	0.132499	✓	1	0.888497	0.144633	✓	1	0.619222	0.135733	✗ -1
10	0.655854	0.136926	0.722459	0.132836	✓	1	0.723035	0.141196	✓	1	0.620485	0.136108	✓ 1
11	0.62664	0.139273	0.727135	0.132636	✓	1	0.740211	0.132577	✓	1	0.632851	0.136777	✗ -1
12	0.623643	0.129757	0.74183	0.127577	✓	1	0.864292	0.148277	✓	1	0.648155	0.142604	✗ -1
13	0.639105	0.140843	0.732085	0.129384	✓	1	0.71445	0.135483	✓	1	0.653307	0.145912	✗ -1
14	0.653646	0.147537	0.734944	0.140172	✓	1	0.705743	0.138985	✓	1	0.659588	0.144988	✗ -1
15	0.629509	0.126444	0.736403	0.125422	✓	1	0.855307	0.133567	✓	1	0.667213	0.145253	✓ 1
16	0.614193	0.127388	0.745439	0.129792	✓	1	0.724804	0.145888	✓	1	0.674146	0.142089	✓ 1
17	0.621465	0.122097	0.737462	0.134359	✓	1	0.697687	0.148715	✓	1	0.675883	0.14822	✓ 1
18	0.639161	0.136993	0.733114	0.131628	✓	1	0.874249	0.152737	✓	1	0.683135	0.152513	✓ 1
19	0.645661	0.134454	0.748597	0.136143	✓	1	0.722319	0.146275	✓	1	0.695275	0.146043	✓ 1
20	0.628199	0.134644	0.748737	0.132605	✓	1	0.710436	0.145848	✓	1	0.699114	0.145099	✓ 1
21	0.629722	0.121136	0.735903	0.125448	✓	1	0.852801	0.145082	✓	1	0.695915	0.15112	✓ 1
22	0.641316	0.151657	0.758345	0.125373	✓	1	0.734415	0.131267	✓	1	0.702544	0.144426	✓ 1
23	0.630149	0.139877	0.724515	0.131749	✓	1	0.709465	0.13344	✓	1	0.710822	0.145224	✓ 1
24	0.646997	0.145682	0.75074	0.13296	✓	1	0.876225	0.157825	✓	1	0.72019	0.144962	✓ 1
25	0.62036	0.127126	0.742345	0.137319	✓	1	0.719015	0.143619	✓	1	0.713184	0.144016	✓ 1
26	0.629158	0.135415	0.737322	0.134123	✓	1	0.706401	0.144472	✓	1	0.72698	0.14527	✓ 1
27	0.635152	0.124598	0.752866	0.151428	✓	1	0.850781	0.138903	✓	1	0.731874	0.138742	✓ 1
28	0.654939	0.130814	0.737009	0.128719	✓	1	0.733716	0.136062	✓	1	0.744155	0.142876	✓ 1
29	0.639401	0.134815	0.755579	0.131193	✓	1	0.723541	0.150423	✓	1	0.740307	0.143694	✓ 1
30	0.644266	0.121065	0.756728	0.136024	✓	1	0.880629	0.135542	✓	1	0.744231	0.145263	✓ 1
31	0.626956	0.130116	0.753681	0.12624	✓	1	0.707889	0.15004	✓	1	0.74357	0.14263	✓ 1
32	0.629857	0.130235	0.740807	0.132637	✓	1	0.705219	0.136395	✓	1	0.74114	0.150376	✓ 1
33	0.624813	0.136597	0.731114	0.13499	✓	1	0.879749	0.148144	✓	1	0.74695	0.140229	✓ 1
34	0.646673	0.136139	0.737056	0.128886	✓	1	0.707325	0.140159	✓	1	0.753518	0.144717	✓ 1
35	0.631278	0.134846	0.754193	0.136364	✓	1	0.702559	0.136785	✓	1	0.760246	0.140384	✓ 1
36	0.619205	0.128103	0.76064	0.130624	✓	1	0.879202	0.150863	✓	1	0.765018	0.128682	✓ 1
37	0.633123	0.131253	0.74083	0.134462	✓	1	0.746076	0.147355	✓	1	0.768845	0.130372	✓ 1
38	0.626886	0.129688	0.737652	0.128404	✓	1	0.725234	0.134035	✓	1	0.772632	0.132174	✓ 1
39	0.630854	0.131861	0.741661	0.12614	✓	1	0.877058	0.150453	✓	1	0.779725	0.127346	✓ 1
40	0.626436	0.11901	0.741471	0.124093	✓	1	0.720526	0.138623	✓	1	0.779415	0.126355	✓ 1
41	0.648301	0.161078	0.743927	0.129685	✓	1	0.691839	0.137332	✓	1	0.784699	0.123716	✓ 1
42	0.628901	0.135313	0.768474	0.127686	✓	1	0.867526	0.148416	✓	1	0.788965	0.125869	✓ 1
43	0.623857	0.135046	0.755751	0.136731	✓	1	0.711614	0.155123	✓	1	0.792801	0.125322	✓ 1
44	0.622096	0.134688	0.759348	0.131621	✓	1	0.716029	0.147172	✓	1	0.789333	0.124772	✓ 1
45	0.634994	0.131117	0.755316	0.12447	✓	1	0.878918	0.129254	✓	1	0.789196	0.123433	✓ 1
46	0.636471	0.134768	0.748556	0.133895	✓	1	0.72293	0.152885	✓	1	0.796801	0.11648	✓ 1
47	0.636851	0.154412	0.755696	0.135818	✓	1	0.707184	0.142987	✓	1	0.797006	0.117929	✓ 1
48	0.64383	0.129904	0.750012	0.135068	✓	1	0.877383	0.144543	✓	1	0.797366	0.122332	✓ 1
49	0.65293	0.146496	0.752687	0.127591	✓	1	0.724728	0.134173	✓	1	0.803845	0.117631	✓ 1
50	0.643155	0.138792	0.75843	0.122282	✓	1	0.716772	0.135188	✓	1	0.803588	0.117175	✓ 1

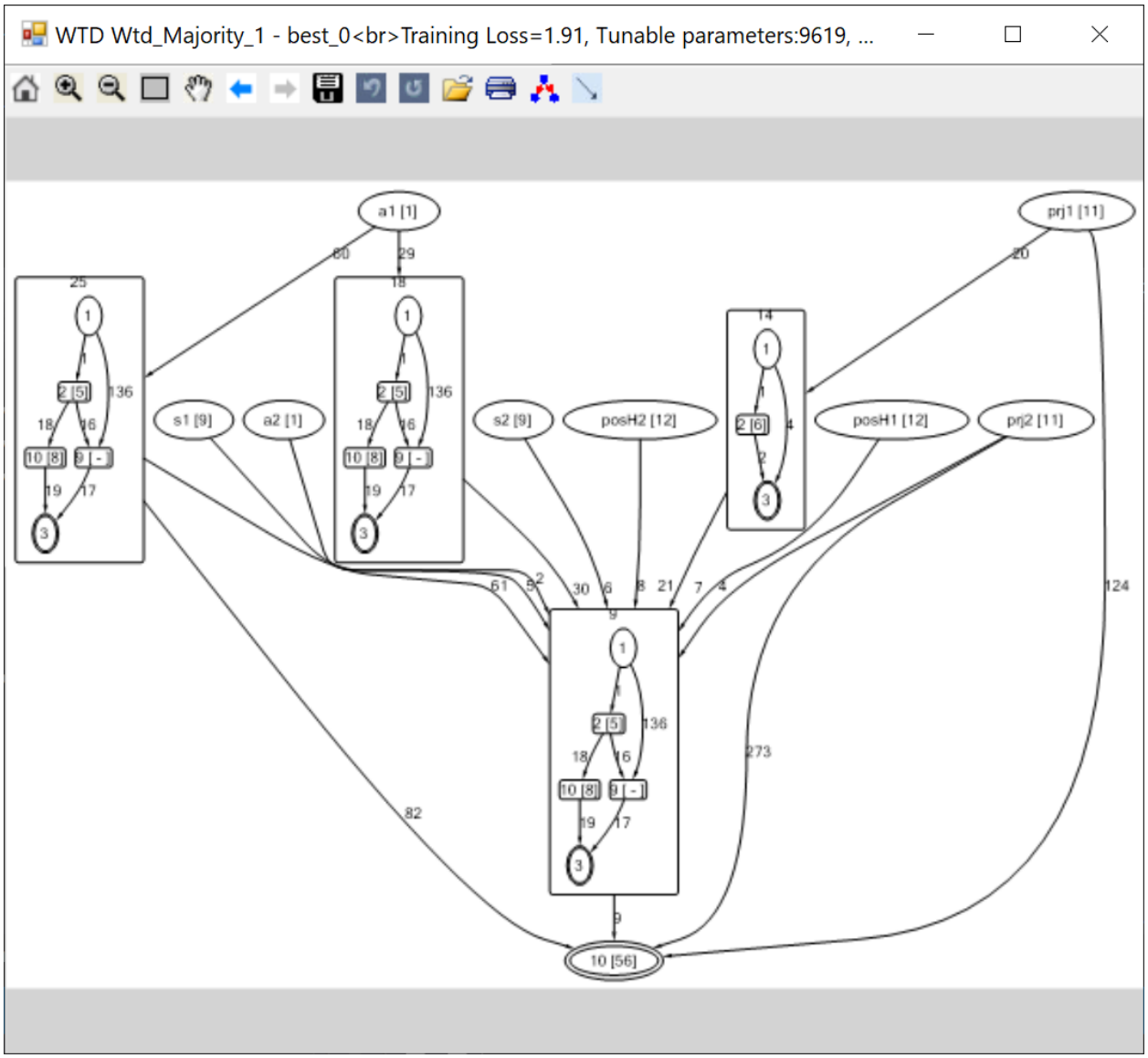
C.5 IPD: T-Tests by Landscape and Successive A Values

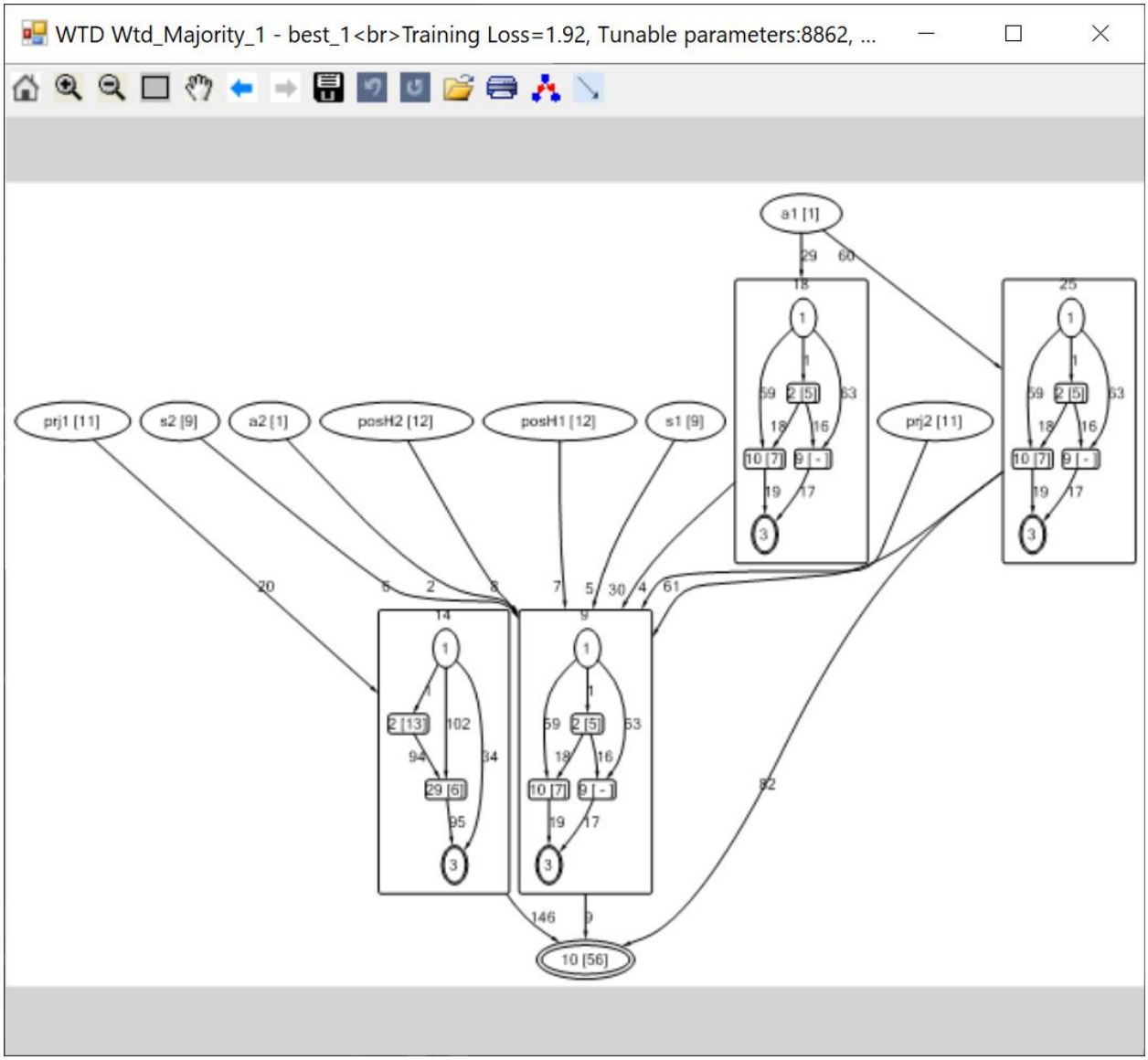
KD=IPD											
Landscape	μ A=1	σ A=1	μ A=3.1	σ A=3.1	μ A=3.6	σ A=3.6	μ A=3.9	σ A=3.9	μ 1!=3.1	μ 3.1!=3.6	μ 3.6!=3.9
1	0.67567	0.121409	0.709345	0.125705	0.724535	0.119526	0.689222	0.128802	✓ 1	✗ -1	✓ 1
2	0.664599	0.126218	0.684403	0.124466	0.66981	0.135624	0.703722	0.118223	✗ -1	✗ -1	✓ 1
3	0.667263	0.129044	0.710637	0.130471	0.727254	0.131196	0.705538	0.138933	✓ 1	✗ -1	✗ -1
4	0.666199	0.115674	0.687401	0.133161	0.697301	0.136779	0.712225	0.119928	✗ -1	✗ -1	✗ -1
5	0.667111	0.115109	0.716781	0.138378	0.722328	0.126756	0.733728	0.139951	✓ 1	✗ -1	✗ -1
6	0.675576	0.125579	0.677591	0.134418	0.694705	0.129997	0.717994	0.121871	✗ -1	✗ -1	✗ -1
7	0.673673	0.126863	0.709015	0.133078	0.726096	0.12711	0.738687	0.136943	✓ 1	✗ -1	✗ -1
8	0.673845	0.124654	0.683287	0.121645	0.704901	0.124561	0.716035	0.130804	✗ -1	✗ -1	✗ -1
9	0.676348	0.125458	0.710573	0.13421	0.716412	0.137526	0.736298	0.132499	✓ 1	✗ -1	✗ -1
10	0.670632	0.129605	0.70148	0.127556	0.707579	0.140312	0.722459	0.132836	✓ 1	✗ -1	✗ -1
11	0.678854	0.134538	0.726933	0.124523	0.727705	0.128822	0.727135	0.132636	✓ 1	✗ -1	✗ -1
12	0.674953	0.126388	0.698646	0.133305	0.713292	0.127362	0.74183	0.127577	✗ -1	✗ -1	✓ 1
13	0.687193	0.132185	0.732439	0.12731	0.714368	0.121898	0.732085	0.129384	✓ 1	✗ -1	✗ -1
14	0.680512	0.131511	0.707909	0.124563	0.698719	0.132197	0.734944	0.140172	✓ 1	✗ -1	✓ 1
15	0.668596	0.132834	0.741436	0.132434	0.726664	0.137875	0.736403	0.125422	✓ 1	✗ -1	✗ -1
16	0.67024	0.129318	0.705038	0.128	0.704942	0.125013	0.745439	0.129792	✓ 1	✗ -1	✓ 1
17	0.657842	0.120724	0.733424	0.124327	0.737412	0.125464	0.737462	0.134359	✓ 1	✗ -1	✗ -1
18	0.676547	0.125693	0.690509	0.128832	0.714933	0.133119	0.733114	0.131628	✗ -1	✗ -1	✗ -1
19	0.671497	0.139465	0.747325	0.115171	0.709643	0.131004	0.748597	0.136143	✓ 1	✓ 1	✓ 1
20	0.672439	0.130313	0.721974	0.125134	0.710026	0.117437	0.748737	0.132605	✓ 1	✗ -1	✓ 1
21	0.685936	0.127308	0.754348	0.139893	0.726292	0.137704	0.735903	0.125448	✓ 1	✓ 1	✗ -1
22	0.671132	0.119526	0.701588	0.118489	0.717678	0.119314	0.758345	0.125373	✓ 1	✗ -1	✓ 1
23	0.681792	0.110303	0.760915	0.128057	0.731456	0.128993	0.724515	0.131749	✓ 1	✓ 1	✗ -1
24	0.692912	0.121459	0.740681	0.130283	0.727237	0.11623	0.75074	0.13296	✓ 1	✗ -1	✗ -1
25	0.669064	0.12732	0.763716	0.126283	0.755064	0.132013	0.742345	0.137319	✓ 1	✗ -1	✗ -1
26	0.672421	0.127897	0.696108	0.130516	0.727281	0.130707	0.737322	0.134123	✗ -1	✓ 1	✗ -1
27	0.690678	0.116903	0.755789	0.136411	0.74214	0.136828	0.752866	0.151428	✓ 1	✗ -1	✗ -1
28	0.67398	0.122793	0.719547	0.13093	0.725228	0.137462	0.737009	0.128719	✓ 1	✗ -1	✗ -1
29	0.675883	0.134923	0.785453	0.132228	0.746436	0.136862	0.755579	0.131193	✓ 1	✓ 1	✗ -1
30	0.68119	0.131892	0.716474	0.120736	0.728211	0.125232	0.756728	0.136024	✓ 1	✗ -1	✓ 1
31	0.677959	0.1179	0.766573	0.125808	0.74424	0.131292	0.753681	0.12624	✓ 1	✗ -1	✗ -1
32	0.660892	0.1254	0.734287	0.128069	0.715485	0.131356	0.740807	0.132637	✓ 1	✗ -1	✗ -1
33	0.686889	0.125119	0.764699	0.132699	0.746632	0.133497	0.731114	0.13499	✓ 1	✗ -1	✗ -1
34	0.673804	0.124547	0.726401	0.117542	0.71883	0.125946	0.737056	0.128886	✓ 1	✗ -1	✗ -1
35	0.669433	0.124572	0.750497	0.131895	0.753175	0.127781	0.754193	0.136364	✓ 1	✗ -1	✗ -1
36	0.68688	0.133558	0.737553	0.133395	0.723816	0.123486	0.76064	0.130624	✓ 1	✗ -1	✓ 1
37	0.66081	0.114204	0.772591	0.128425	0.739263	0.125049	0.74083	0.134462	✓ 1	✓ 1	✗ -1
38	0.687208	0.128933	0.723099	0.128892	0.737965	0.139505	0.737652	0.128404	✓ 1	✗ -1	✗ -1
39	0.674564	0.134551	0.755266	0.136505	0.755468	0.136928	0.741661	0.12614	✓ 1	✗ -1	✗ -1
40	0.67017	0.123908	0.734161	0.132626	0.736289	0.126969	0.741471	0.124093	✓ 1	✗ -1	✗ -1
41	0.684611	0.127441	0.758985	0.130282	0.748421	0.123691	0.743927	0.129685	✓ 1	✗ -1	✗ -1
42	0.668246	0.13934	0.722064	0.127043	0.729167	0.131102	0.768474	0.127686	✓ 1	✗ -1	✓ 1
43	0.664997	0.112136	0.775155	0.132903	0.76305	0.1296	0.755751	0.136731	✓ 1	✗ -1	✗ -1
44	0.678471	0.11528	0.724436	0.122998	0.732965	0.122073	0.759348	0.131621	✓ 1	✗ -1	✓ 1
45	0.667506	0.120855	0.751956	0.131649	0.735187	0.12516	0.755316	0.12447	✓ 1	✗ -1	✗ -1
46	0.672371	0.13285	0.724687	0.137709	0.726313	0.123242	0.748556	0.133895	✓ 1	✗ -1	✗ -1
47	0.672518	0.130153	0.773623	0.138391	0.759687	0.124761	0.755696	0.135818	✓ 1	✗ -1	✗ -1
48	0.675392	0.121676	0.744102	0.123969	0.724702	0.126926	0.750012	0.135068	✓ 1	✗ -1	✗ -1
49	0.678851	0.121515	0.771298	0.129474	0.762737	0.124956	0.752687	0.127591	✓ 1	✗ -1	✗ -1
50	0.652801	0.12188	0.727082	0.130196	0.734047	0.133882	0.75843	0.122282	✓ 1	✗ -1	✗ -1

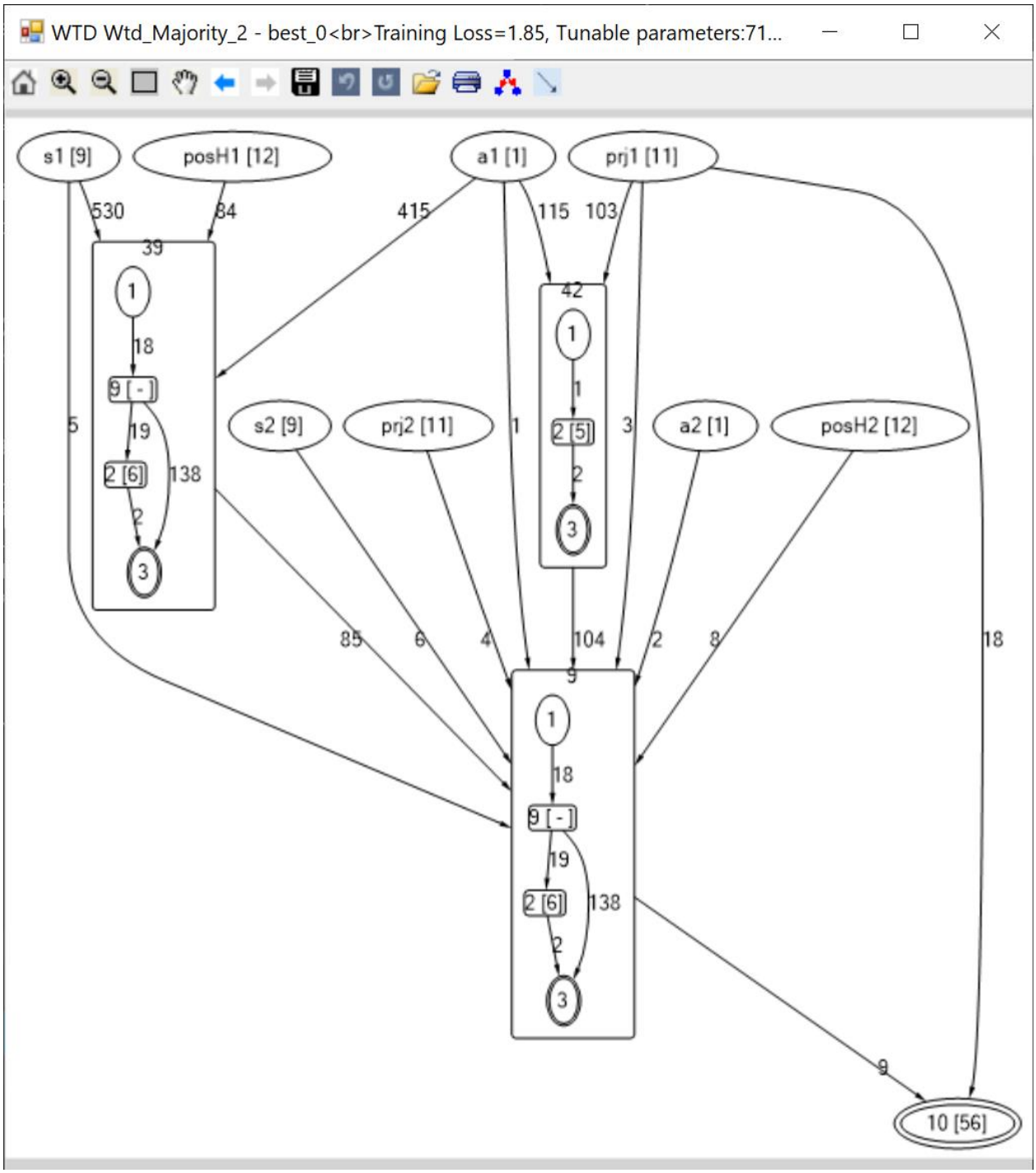
APPENDIX D CATNEURO BEST MODELS

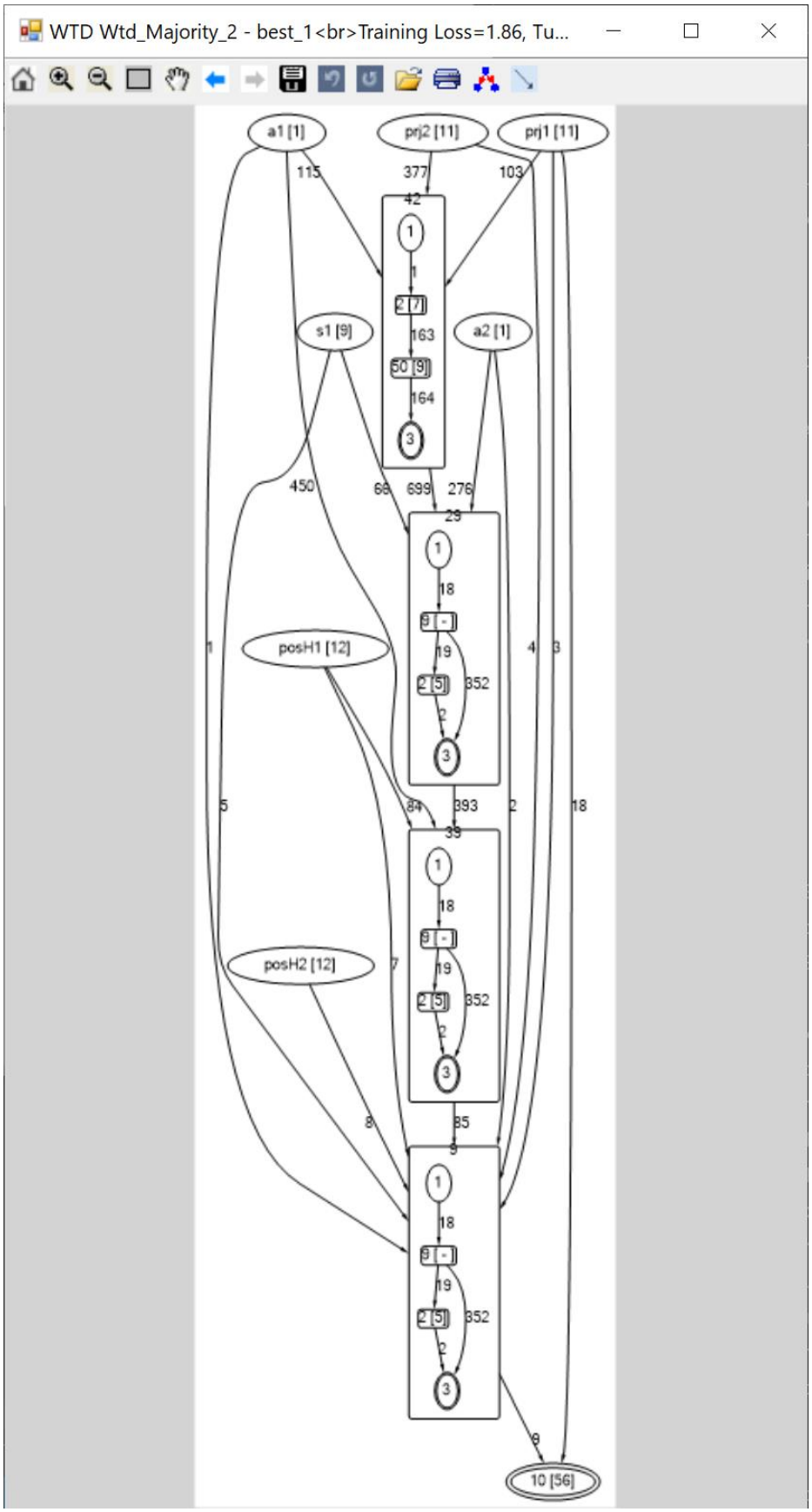
The top 12 models produced from the 6 CATNeuro sample runs are shown below – WTD followed by Stag-Hunt.

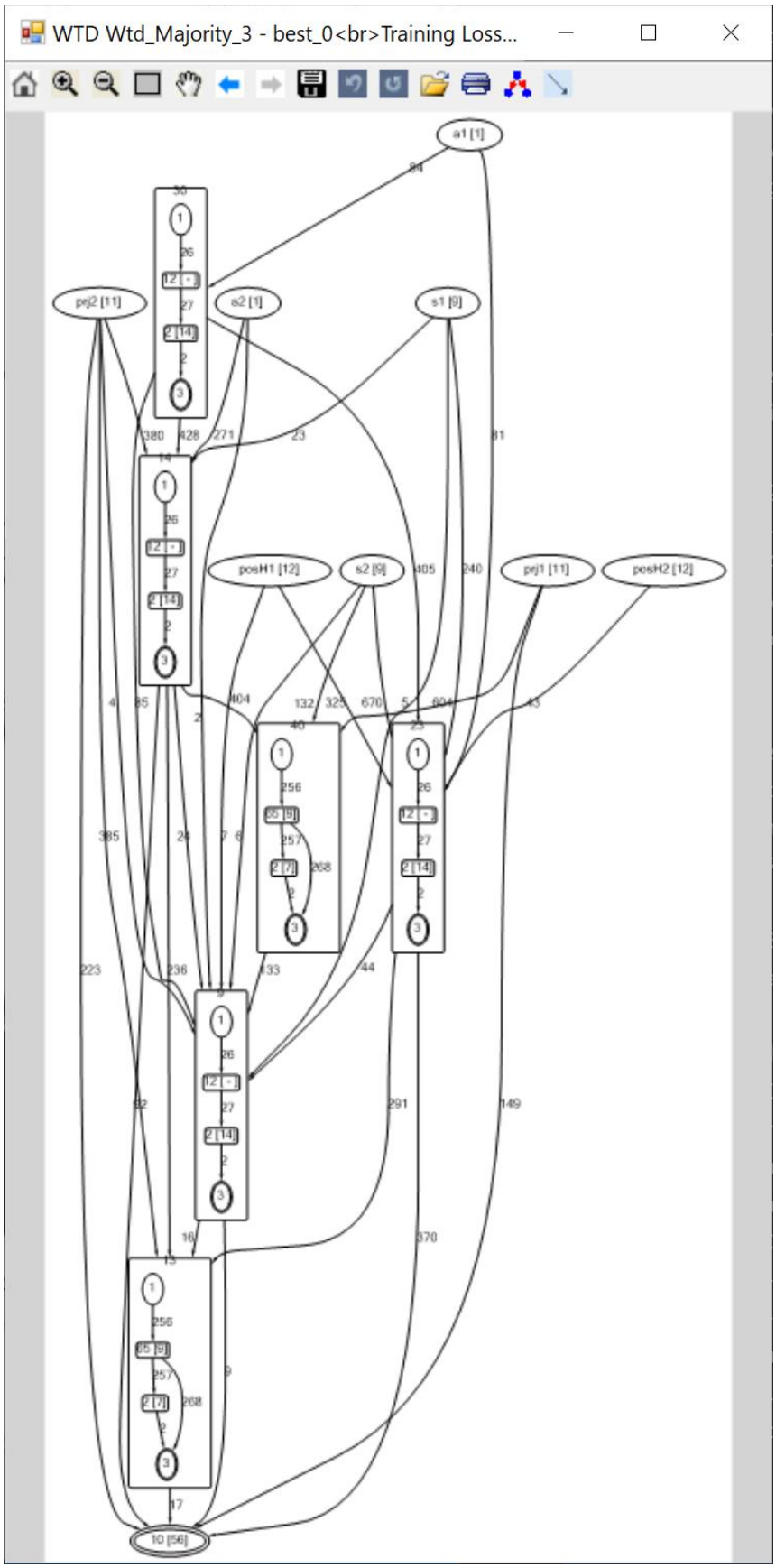
D.1 WEIGHTED MAJORITY MODELS

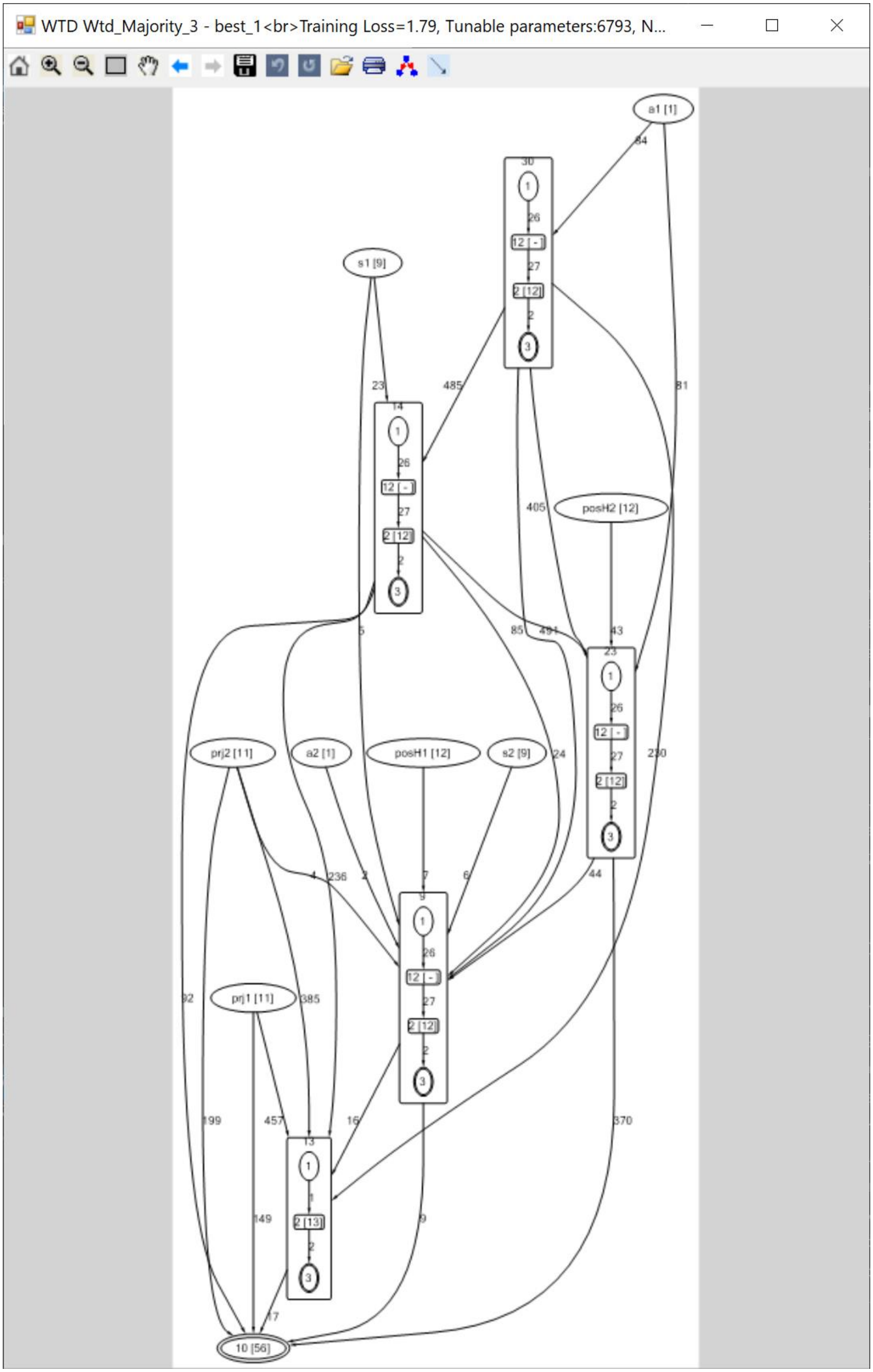


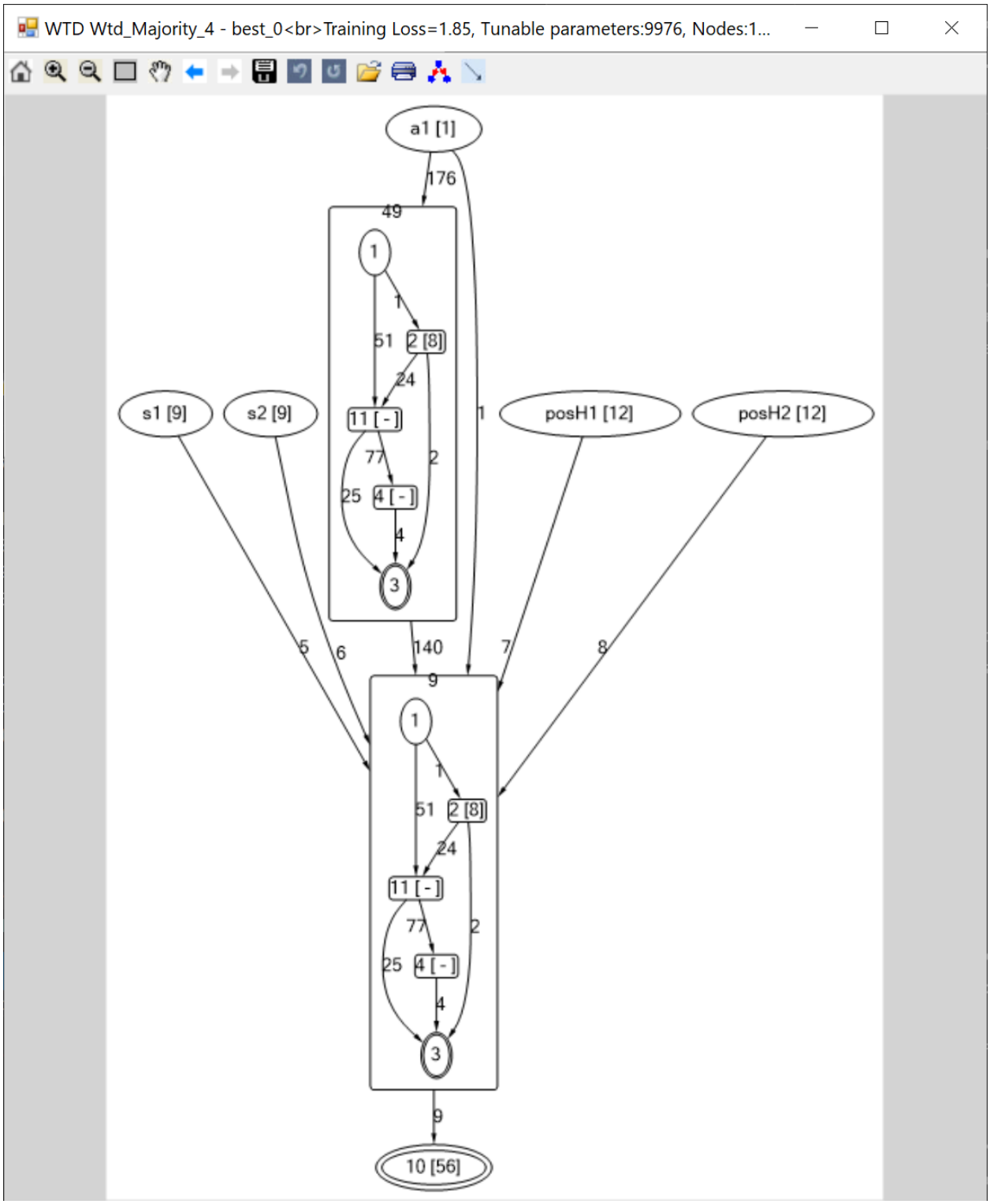

WTD Wtd_Majority_1 - best_1
Training Loss=1.92, Tunable parameters:8862, ...
a1 [1]
prj1 [11]
s2 [9]
a2 [1]
posH2 [12]
posH1 [12]
s1 [9]
prj2 [11]
10 [56]

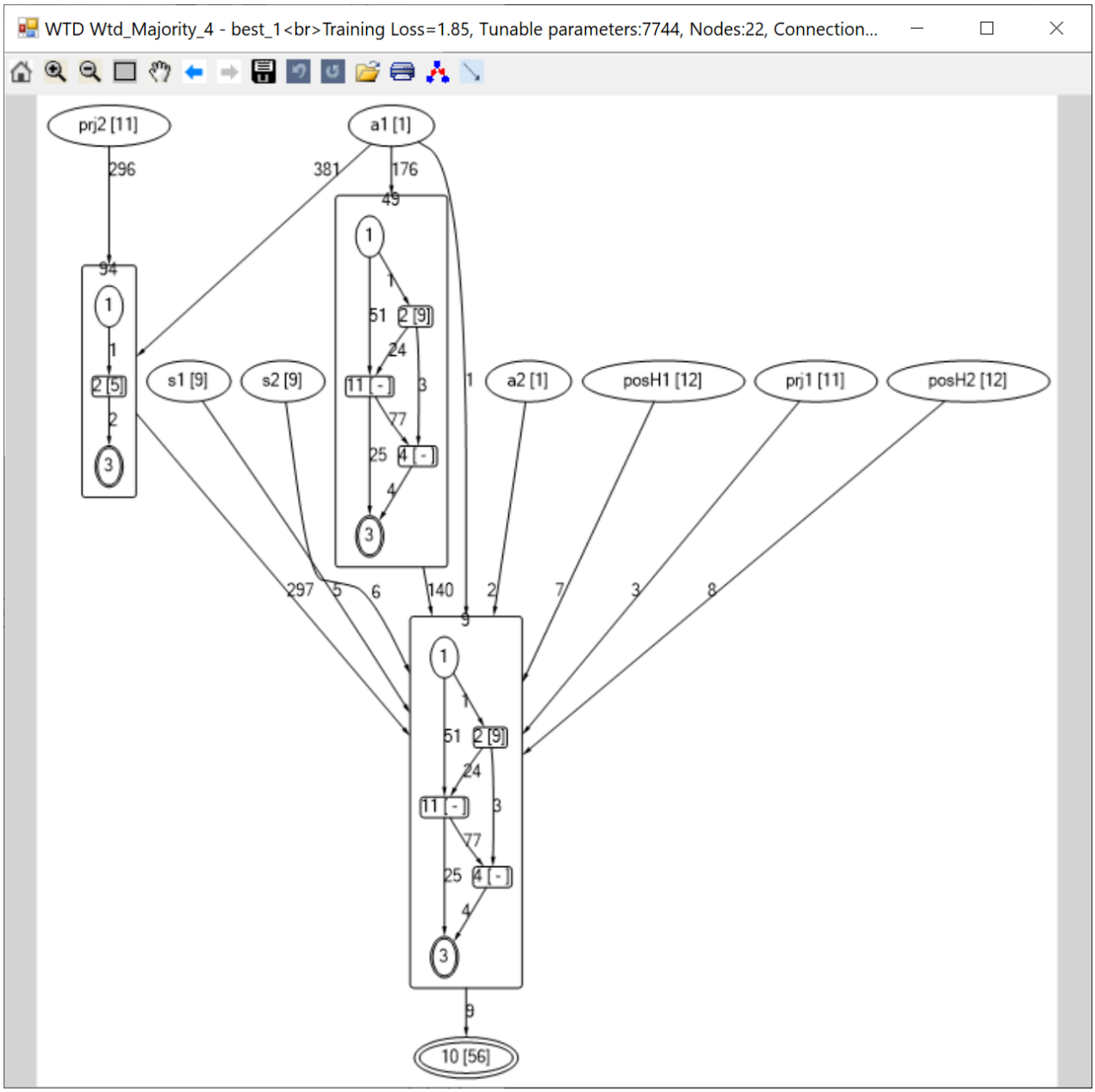

WTD Wtd_Majority_2 - best_0
Training Loss=1.85, Tunable parameters:71...
s1 [9]
posH1 [12]
a1 [1]
prj1 [11]
530
84
415
115
103
39
42
1
18
9 [-]
19
2 [6]
138
2
3
5
s2 [9]
prj2 [11]
1
2 [5]
3
a2 [1]
posH2 [12]
85
6
4
104
2
8
18
9
9
10 [56]

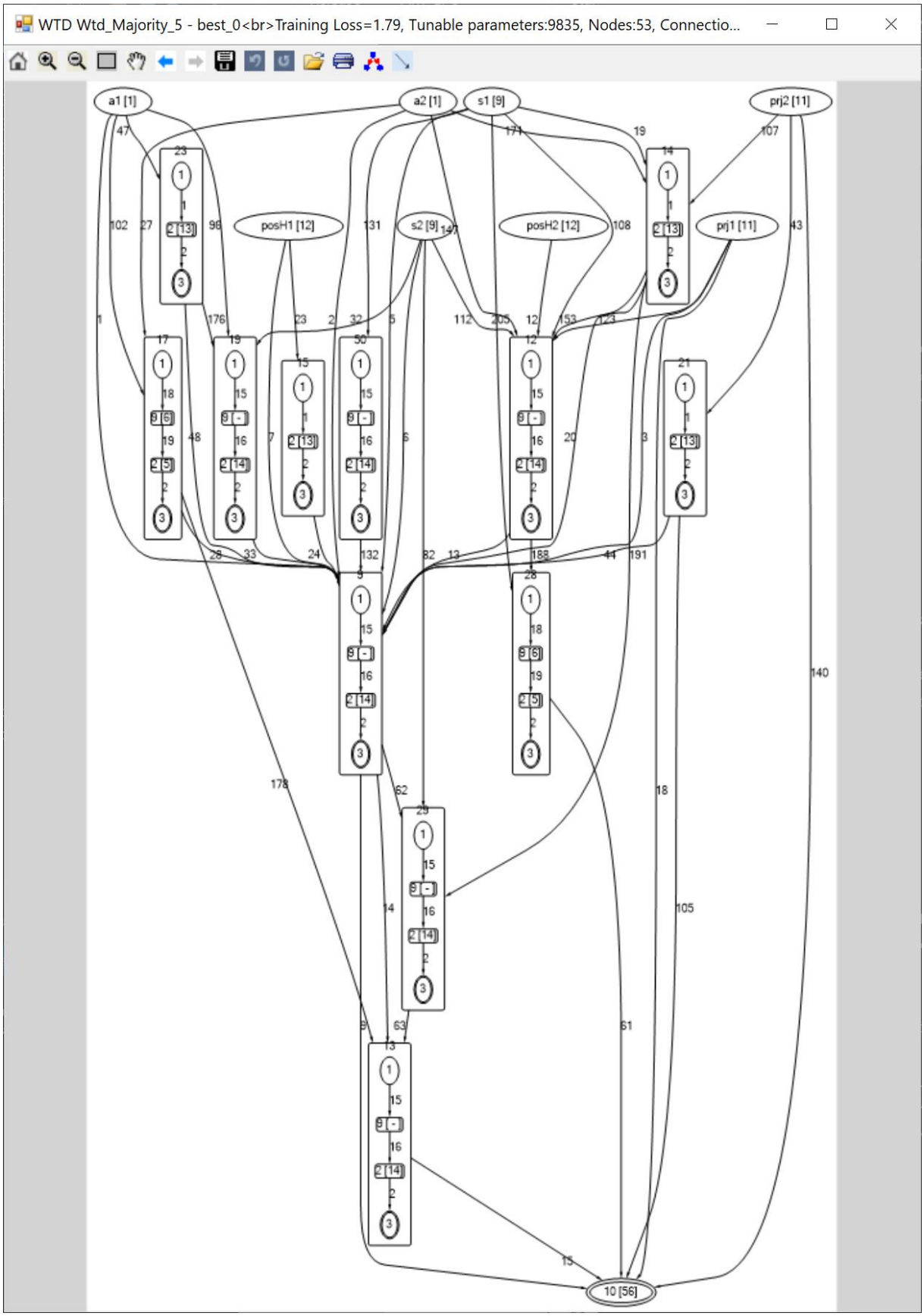

WTD Wtd_Majority_2 - best_1
Training Loss=1.86, Tu...
a1 [1]
prj2 [11]
prj1 [11]
115
377
103
42
1
1
2 [7]
163
50 [9]
164
3
s1 [9]
a2 [1]
450
68
699
276
29
1
18
9 [-]
19
2 [5]
352
2
3
posH1 [12]
1
4
3
5
84
393
2
18
39
1
18
9 [-]
19
2 [5]
352
2
3
posH2 [12]
7
8
85
9
1
18
9 [-]
19
2 [5]
352
2
3
9
10 [56]

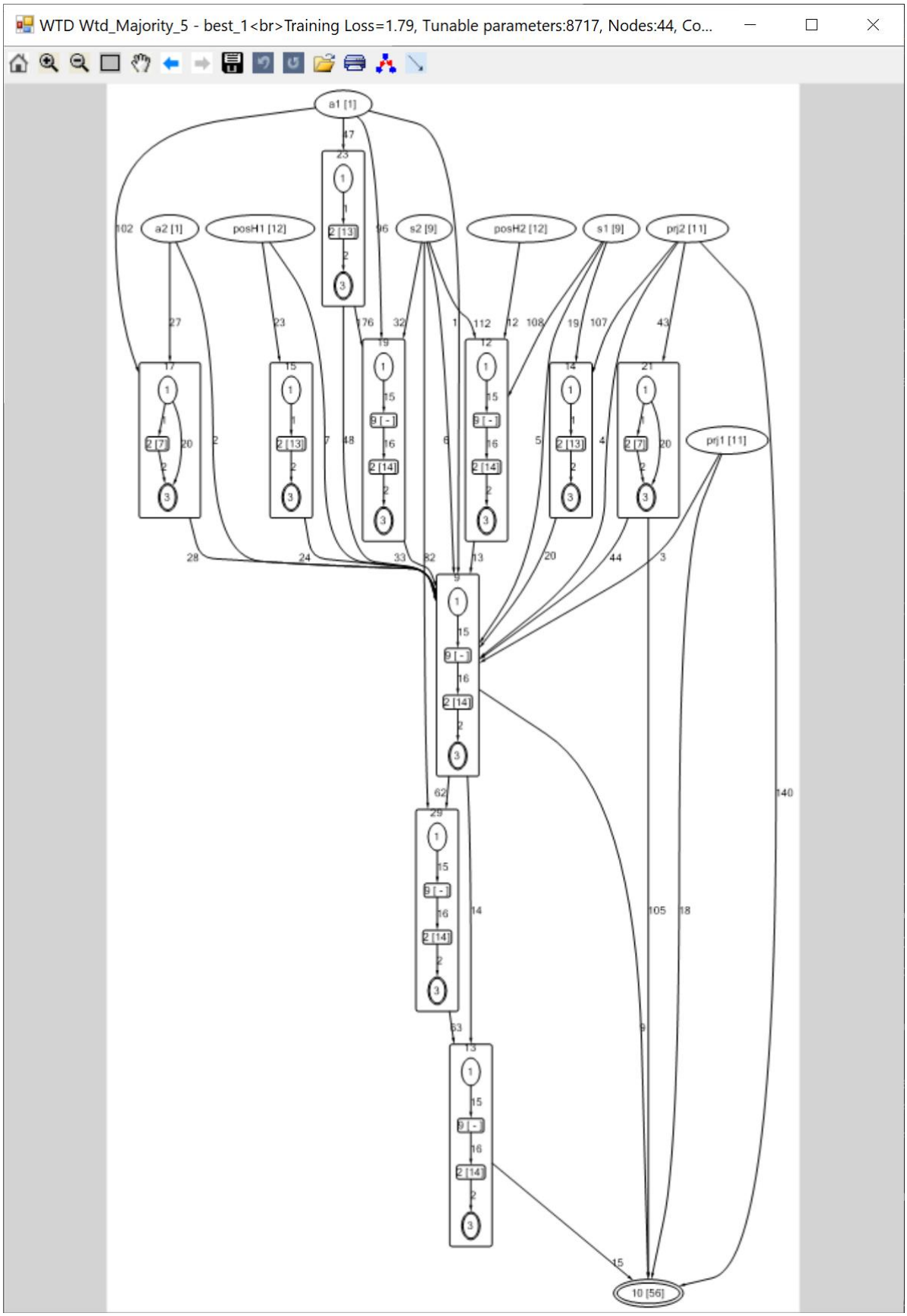

WTD Wtd_Majority_3 - best_0
Training Loss...
a1 [1]
pej2 [11]
a2 [1]
s1 [9]
posH1 [12]
s2 [9]
pej1 [11]
posH2 [12]
10 [56]

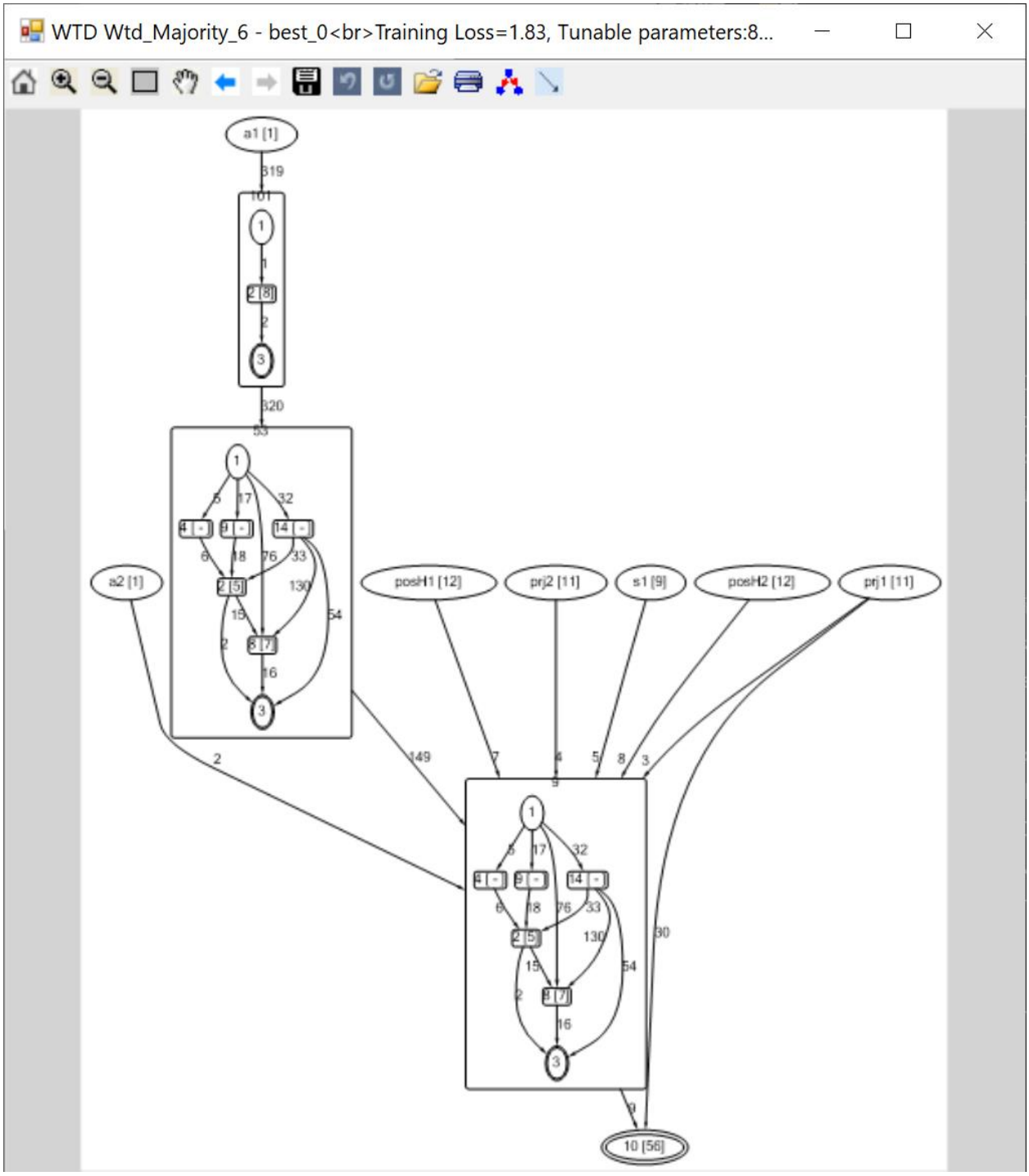

WTD Wtd_Majority_3 - best_1
Training Loss=1.79, Tunable parameters:6793, N...
a1 [1]
s1 [9]
posH2 [12]
prj2 [11]
a2 [1]
posH1 [12]
s2 [9]
prj1 [11]
10 [56]

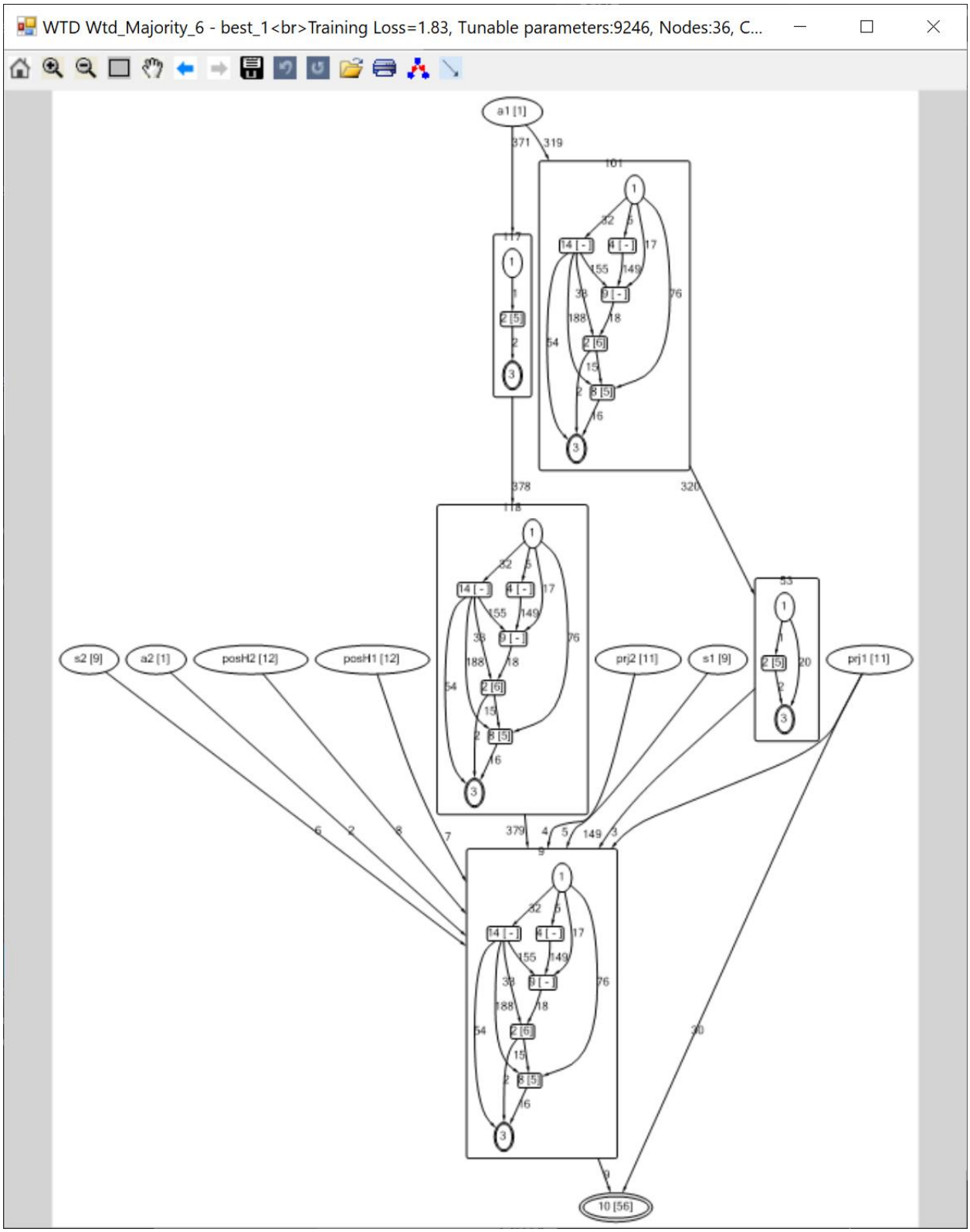

WTD Wtd_Majority_4 - best_0
Training Loss=1.85, Tunable parameters:9976, Nodes:1...
a1 [1]
176
49
1
1
51
2 [8]
24
11 [-]
77
2
25
4 [-]
4
3
s1 [9]
s2 [9]
1
posH1 [12]
posH2 [12]
5
6
140
7
8
9
1
1
51
2 [8]
24
11 [-]
77
2
25
4 [-]
4
3
9
10 [56]


WTD Wtd_Majority_4 - best_1
Training Loss=1.85, Tunable parameters:7744, Nodes:22, Connection...
prj2 [11]
a1 [1]
296
381
176
49
94
1
51
2 [9]
24
2 [5]
11
-
3
77
25
4
-
4
s1 [9]
s2 [9]
a2 [1]
posH1 [12]
prj1 [11]
posH2 [12]
297
5
6
140
2
7
3
8
9
10 [56]

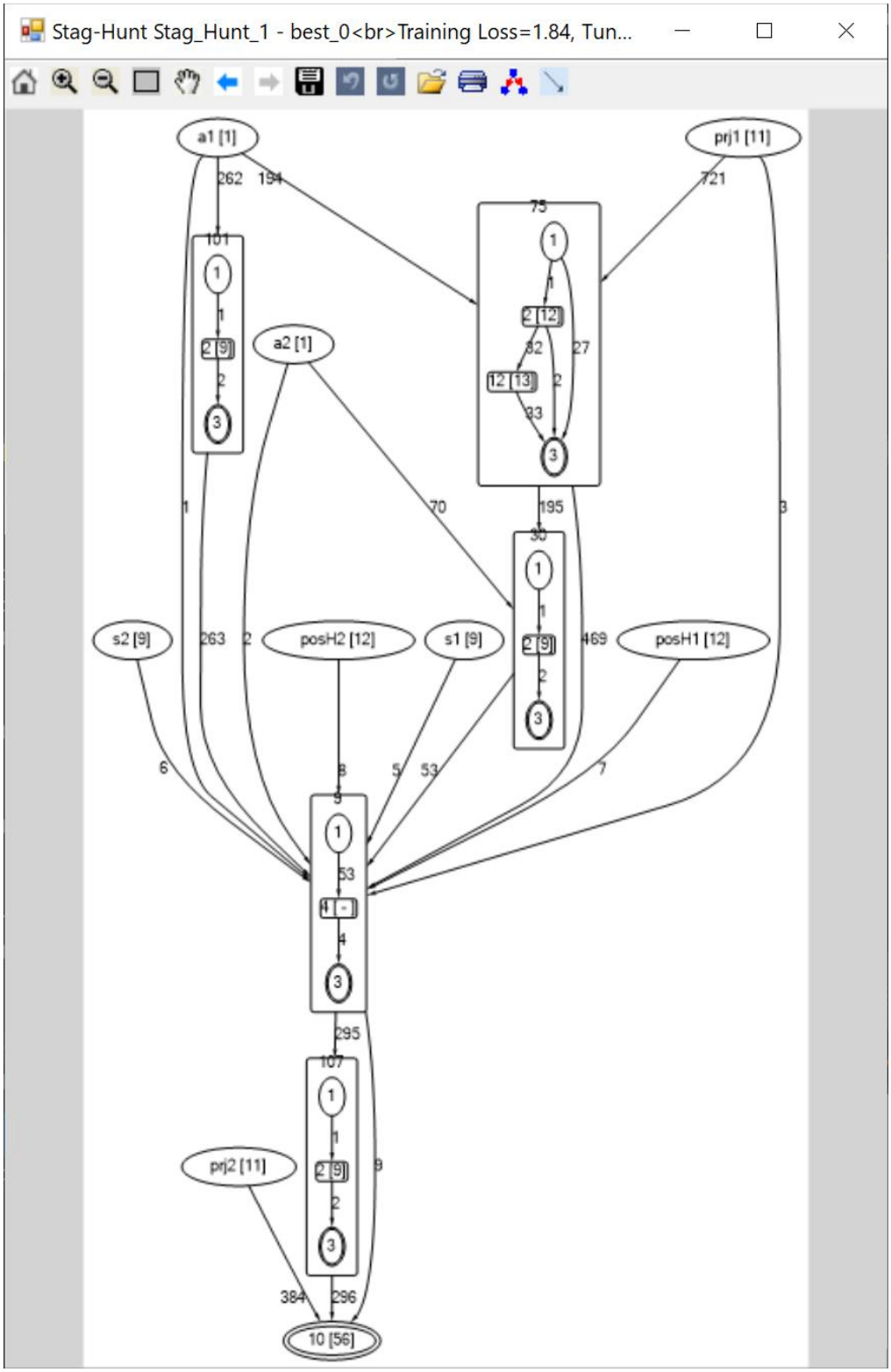

WTD Wtd_Majority_5 - best_0
Training Loss=1.79, Tunable parameters:9835, Nodes:53, Connectio...
a1 [1]
a2 [1]
s1 [9]
prj2 [11]
posH1 [12]
s2 [9]
posH2 [12]
prj1 [11]
10 [56]

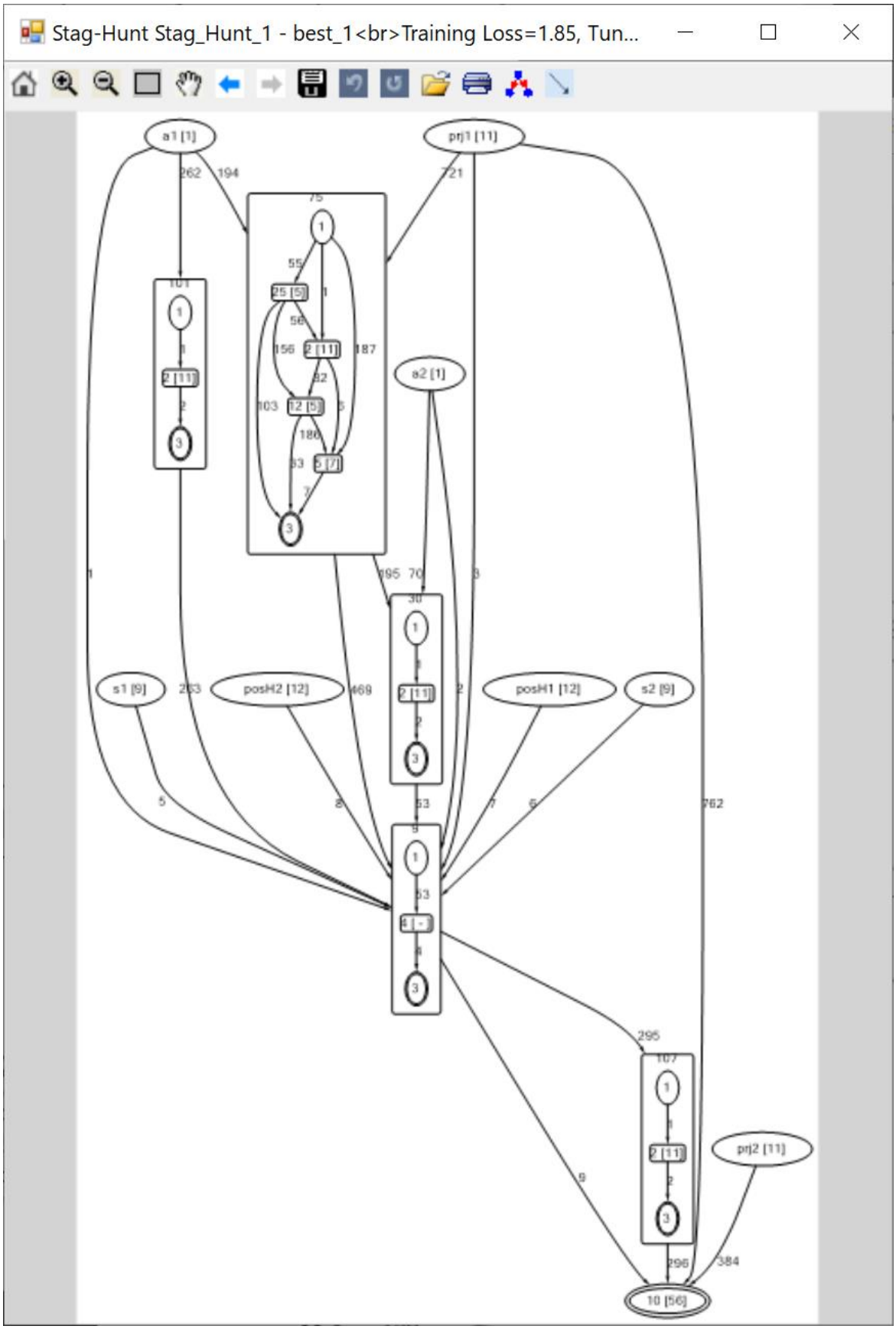

WTD Wtd_Majority_5 - best_1
Training Loss=1.79, Tunable parameters:8717, Nodes:44, Co...
a1 [1]
a2 [1]
posH1 [12]
s2 [9]
posH2 [12]
s1 [9]
prj2 [11]
prj1 [11]
10 [56]

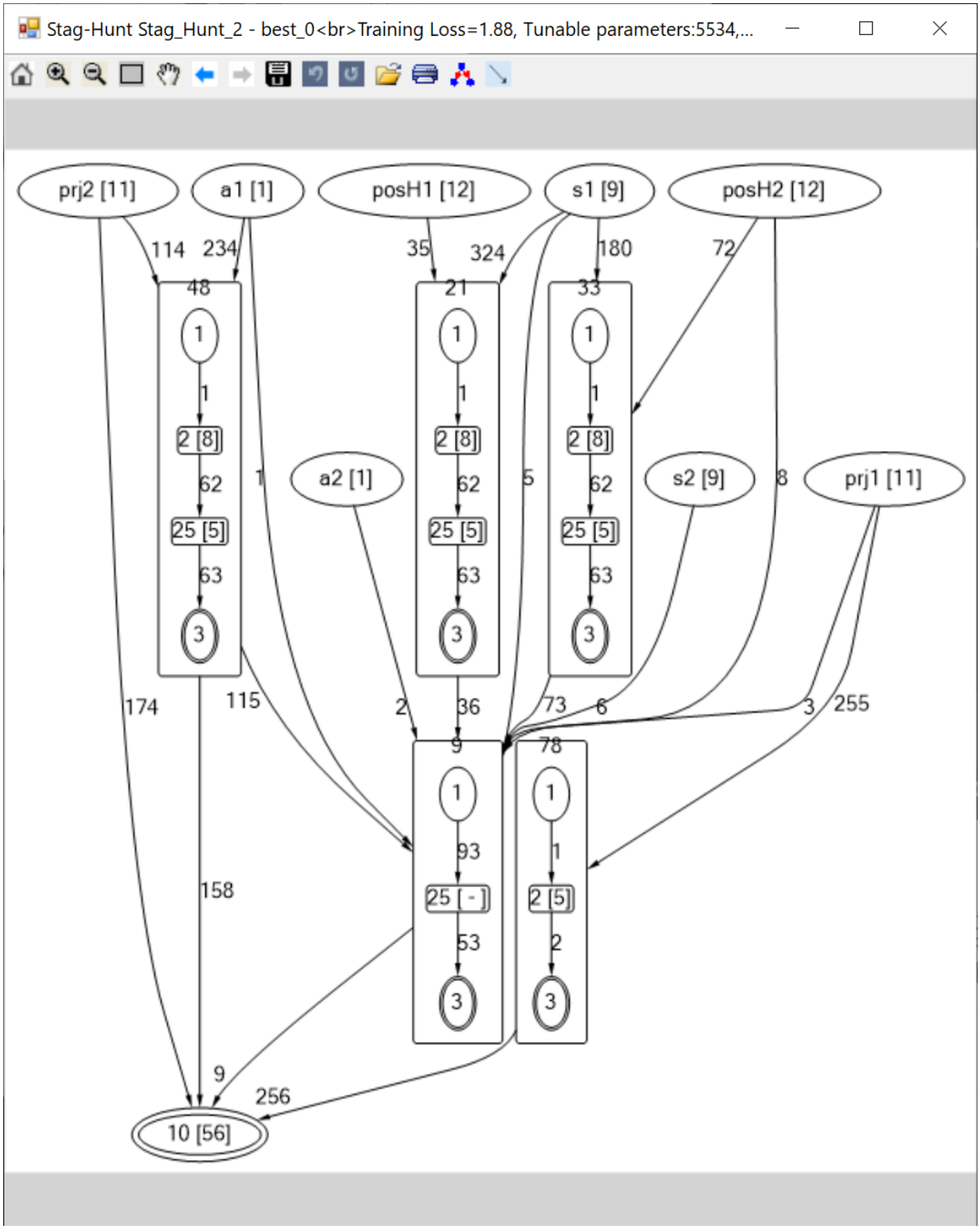

WTD Wtd_Majority_6 - best_0
Training Loss=1.83, Tunable parameters:8...
a1 [1]
319
101
1
2 [8]
2
3
320
53
1
5
17
32
4
9
14
6
18
76
33
2 [5]
130
15
54
2
8 [7]
16
3
a2 [1]
posH1 [12]
prj2 [11]
s1 [9]
posH2 [12]
prj1 [11]
2
149
7
4
5
8
3
9
1
5
17
32
4
9
14
6
18
76
33
2 [5]
130
15
54
2
8 [7]
16
3
30
9
10 [56]

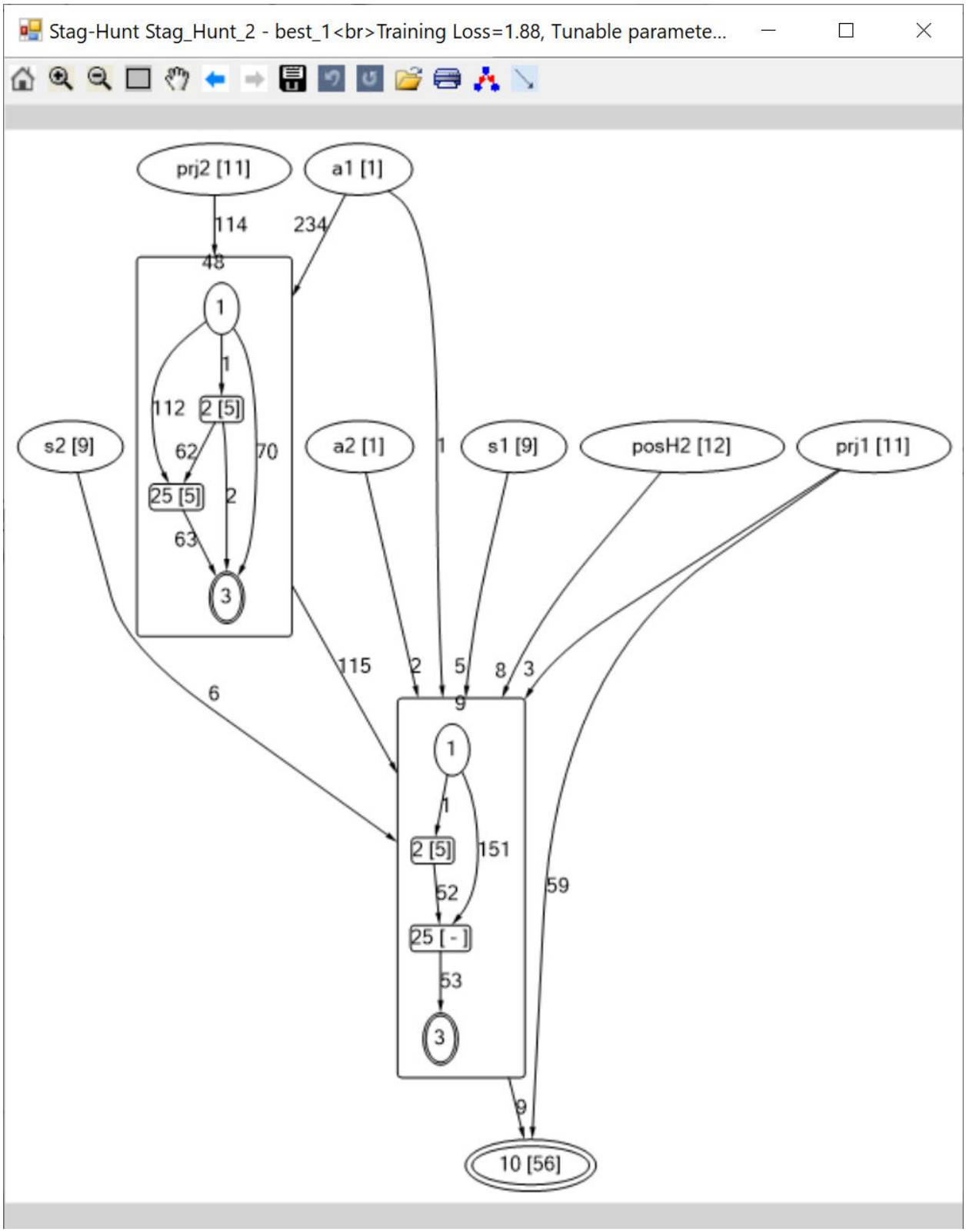

WTD Wtd_Majority_6 - best_1
Training Loss=1.83, Tunable parameters:9246, Nodes:36, C...
a1 [1]
s2 [9]
a2 [1]
posH2 [12]
posH1 [12]
prj2 [11]
s1 [9]
prj1 [11]
10 [56]

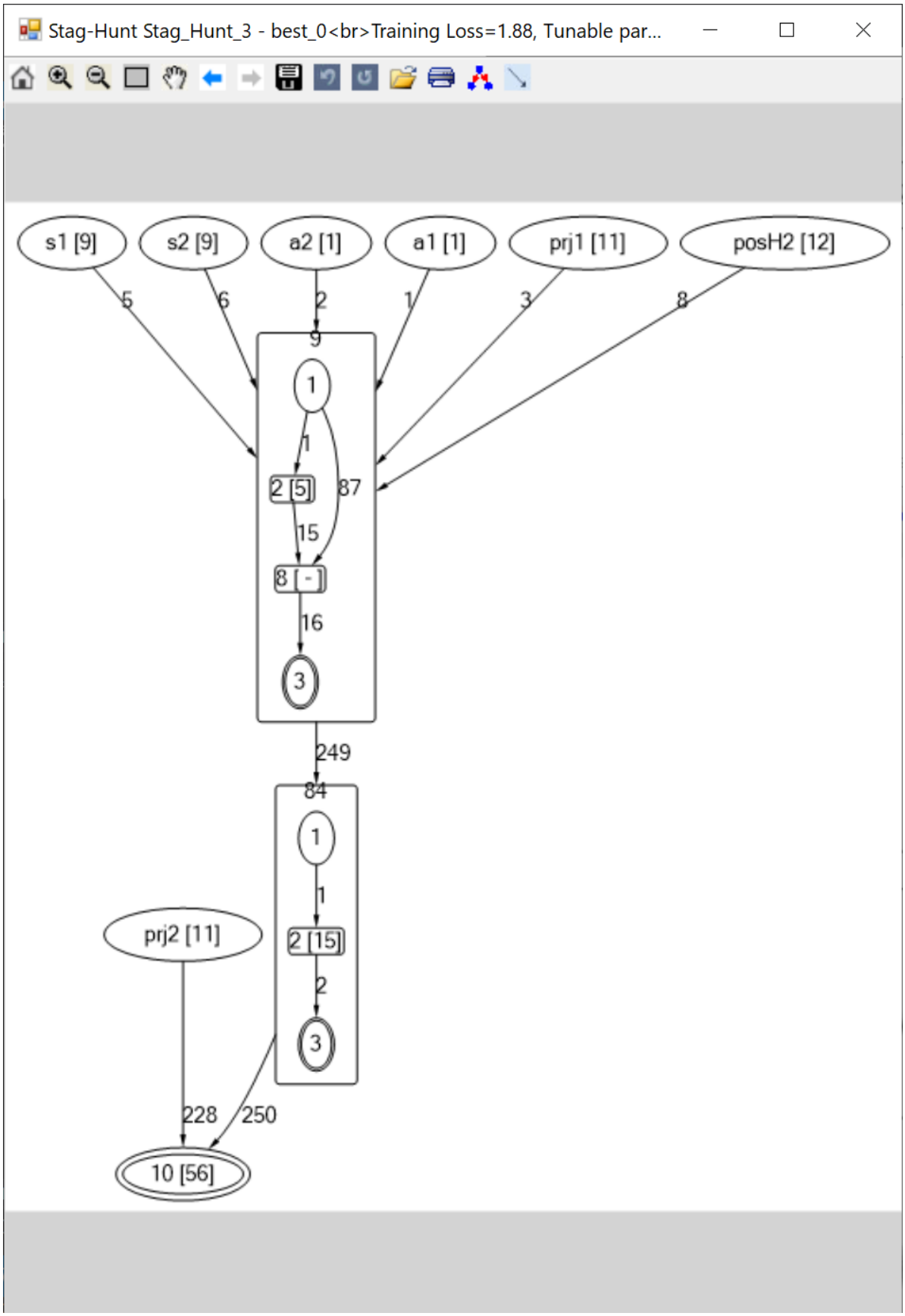
D.2 STAG-HUNT MODELS

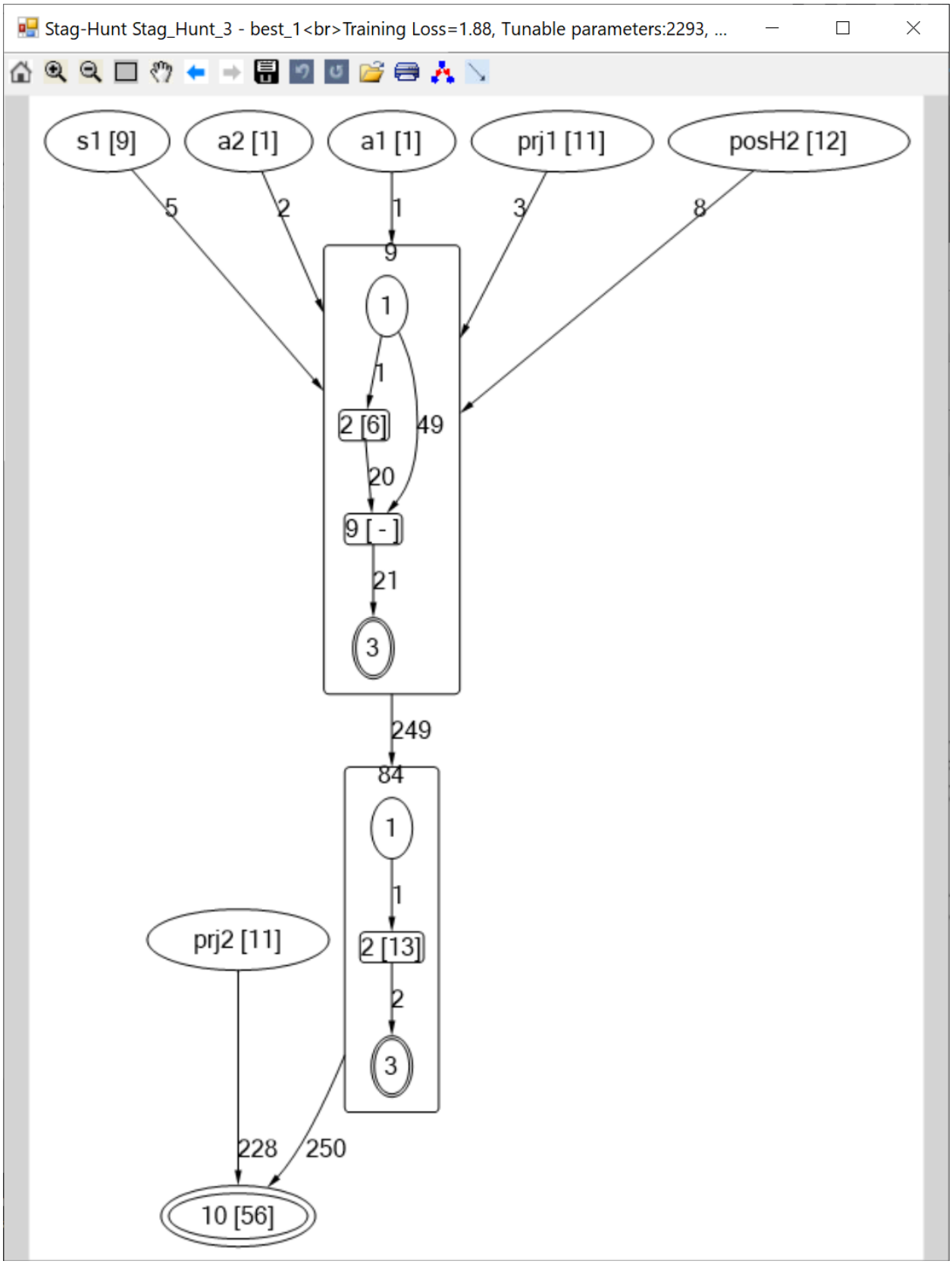


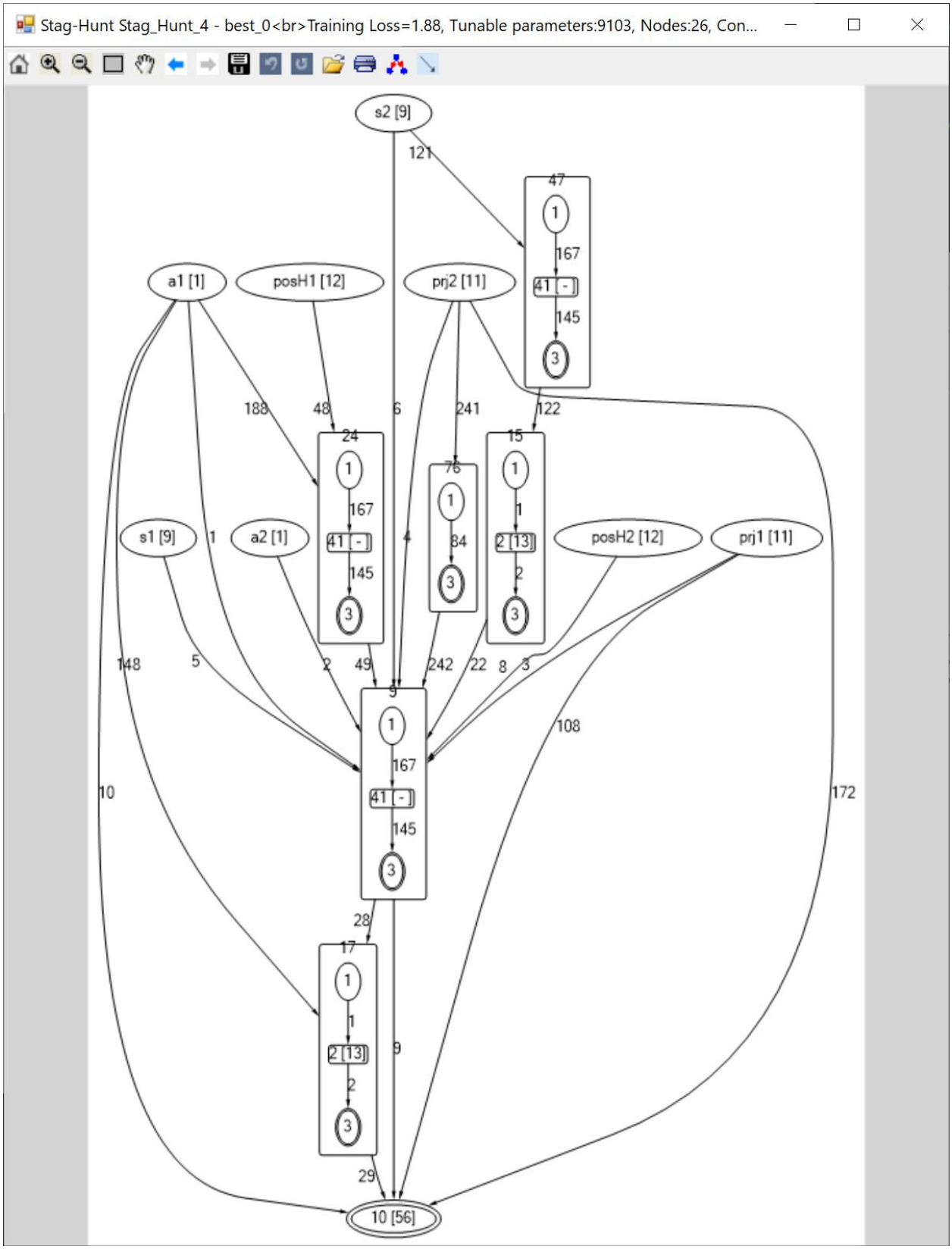

Stag-Hunt Stag_Hunt_1 - best_1
Training Loss=1.85, Tun...
a1 [1]
prj1 [11]
a2 [1]
s1 [9]
posH2 [12]
posH1 [12]
s2 [9]
prj2 [11]
10 [56]

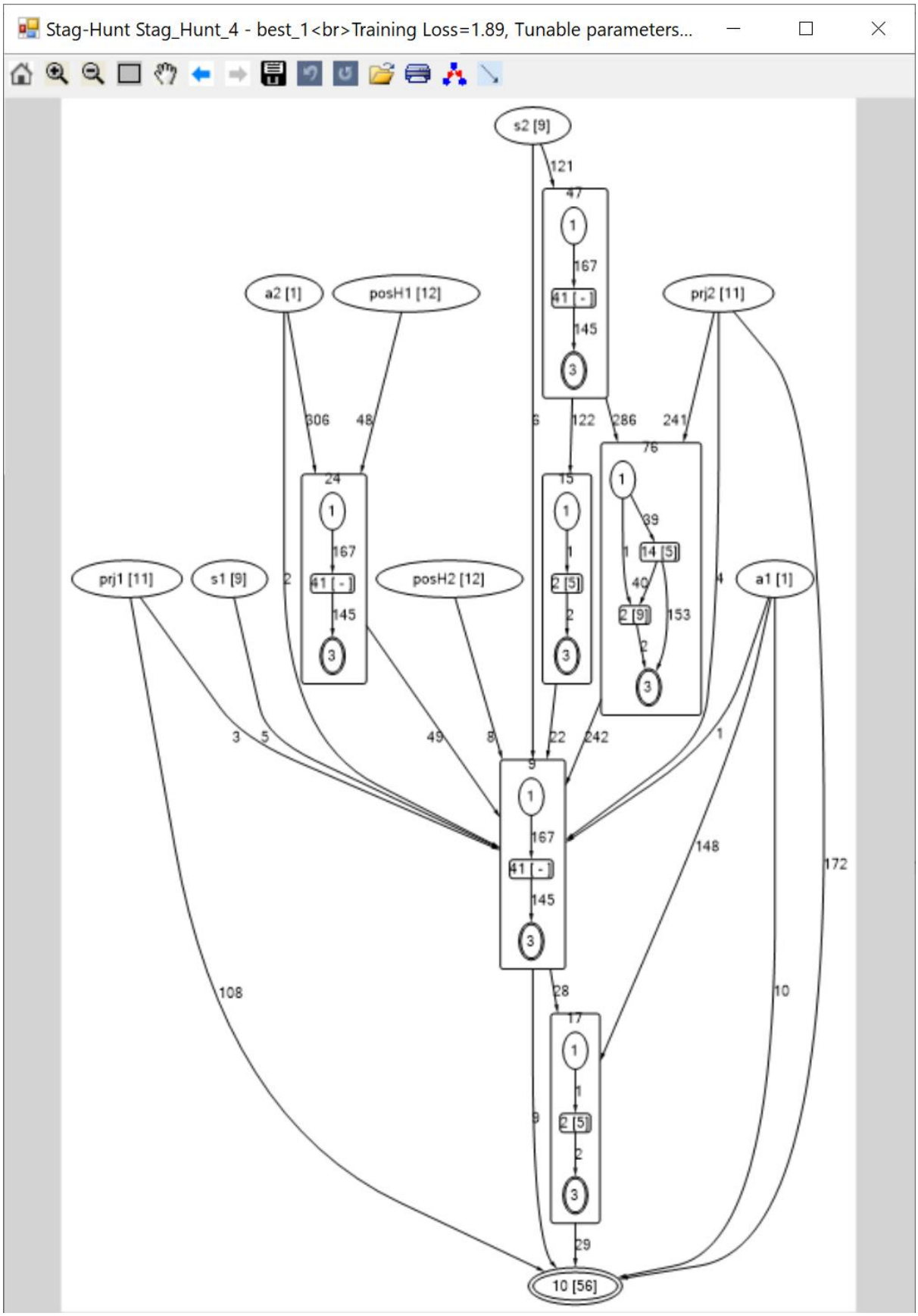

Stag-Hunt Stag_Hunt_2 - best_0
Training Loss=1.88, Tunable parameters:5534,...
prj2 [11]
a1 [1]
posH1 [12]
s1 [9]
posH2 [12]
a2 [1]
s2 [9]
prj1 [11]
10 [56]
2 [8]
25 [5]
25 [-]
2 [5]

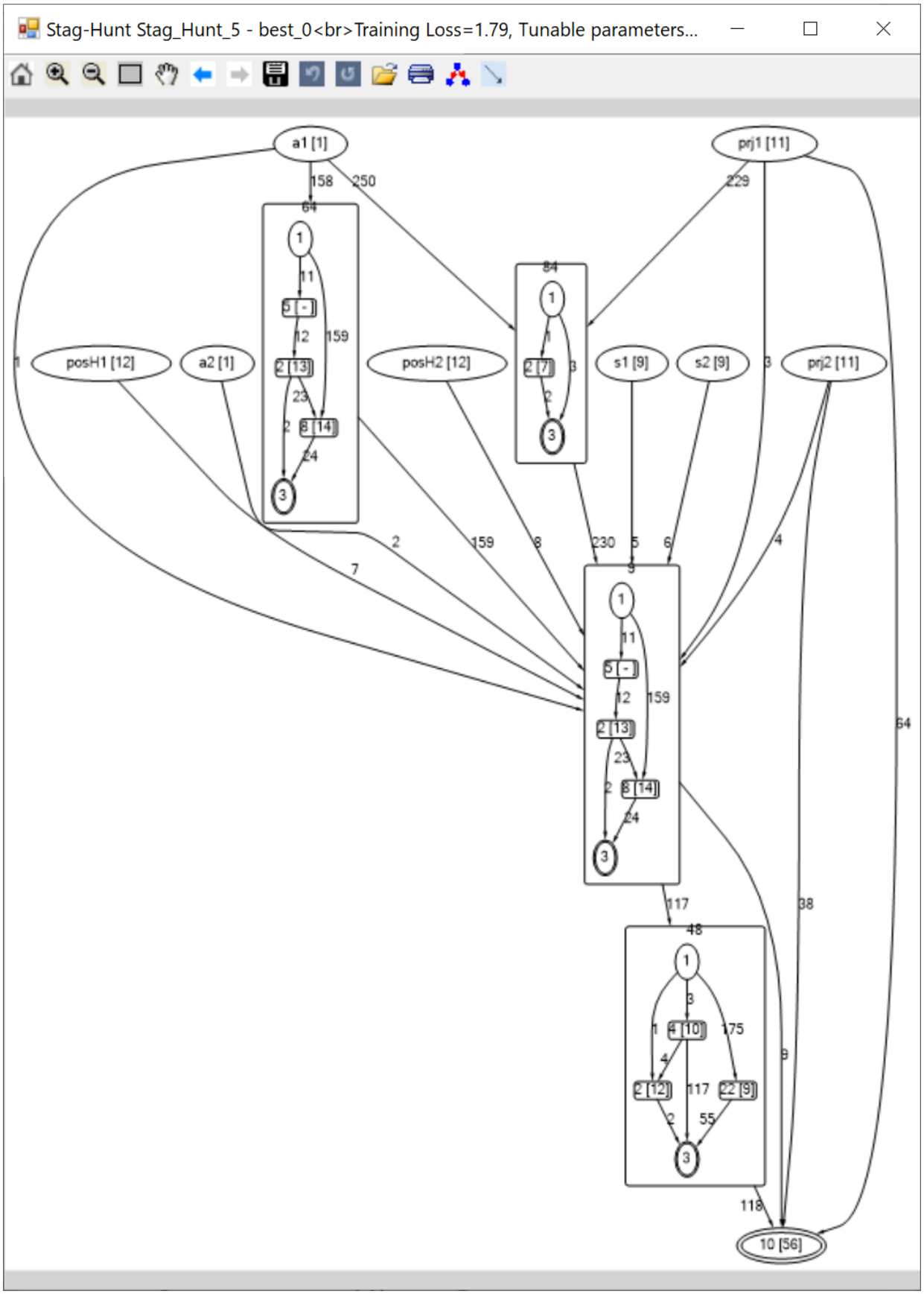

Stag-Hunt Stag_Hunt_2 - best_1
Training Loss=1.88, Tunable paramete...
prj2 [11]
a1 [1]
114
234
48
1
1
112
2 [5]
62
70
25 [5]
2
63
3
s2 [9]
a2 [1]
1
s1 [9]
posH2 [12]
prj1 [11]
115
2
5
8
3
6
9
1
1
2 [5]
151
52
25 [-]
53
3
59
9
10 [56]

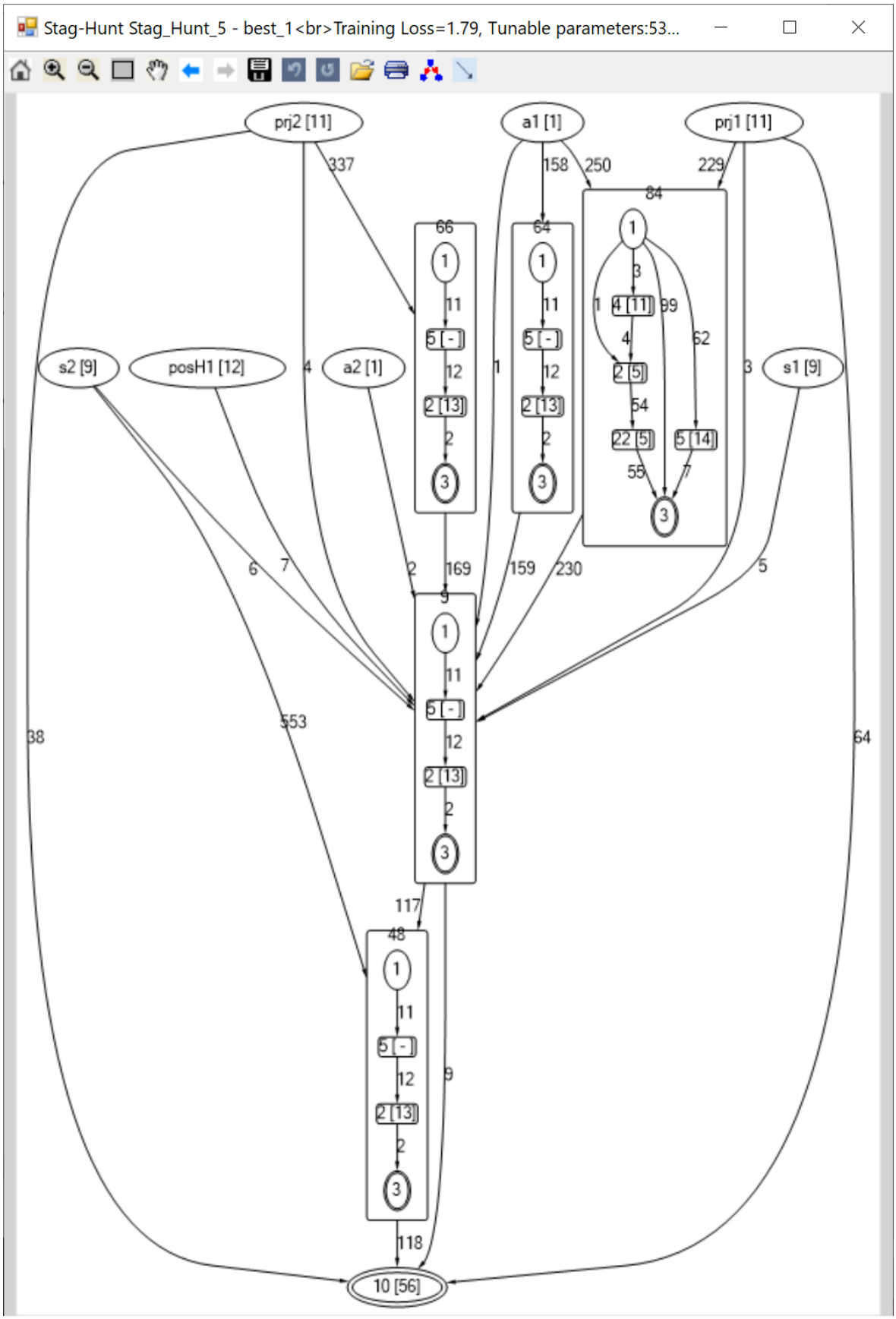

Stag-Hunt Stag_Hunt_3 - best_0
Training Loss=1.88, Tunable par...
s1 [9]
s2 [9]
a2 [1]
a1 [1]
prj1 [11]
posH2 [12]
5
6
2
1
3
8
9
1
1
2 [5]
87
15
8 [-]
16
3
249
84
1
1
2 [15]
2
3
prj2 [11]
228
250
10 [56]

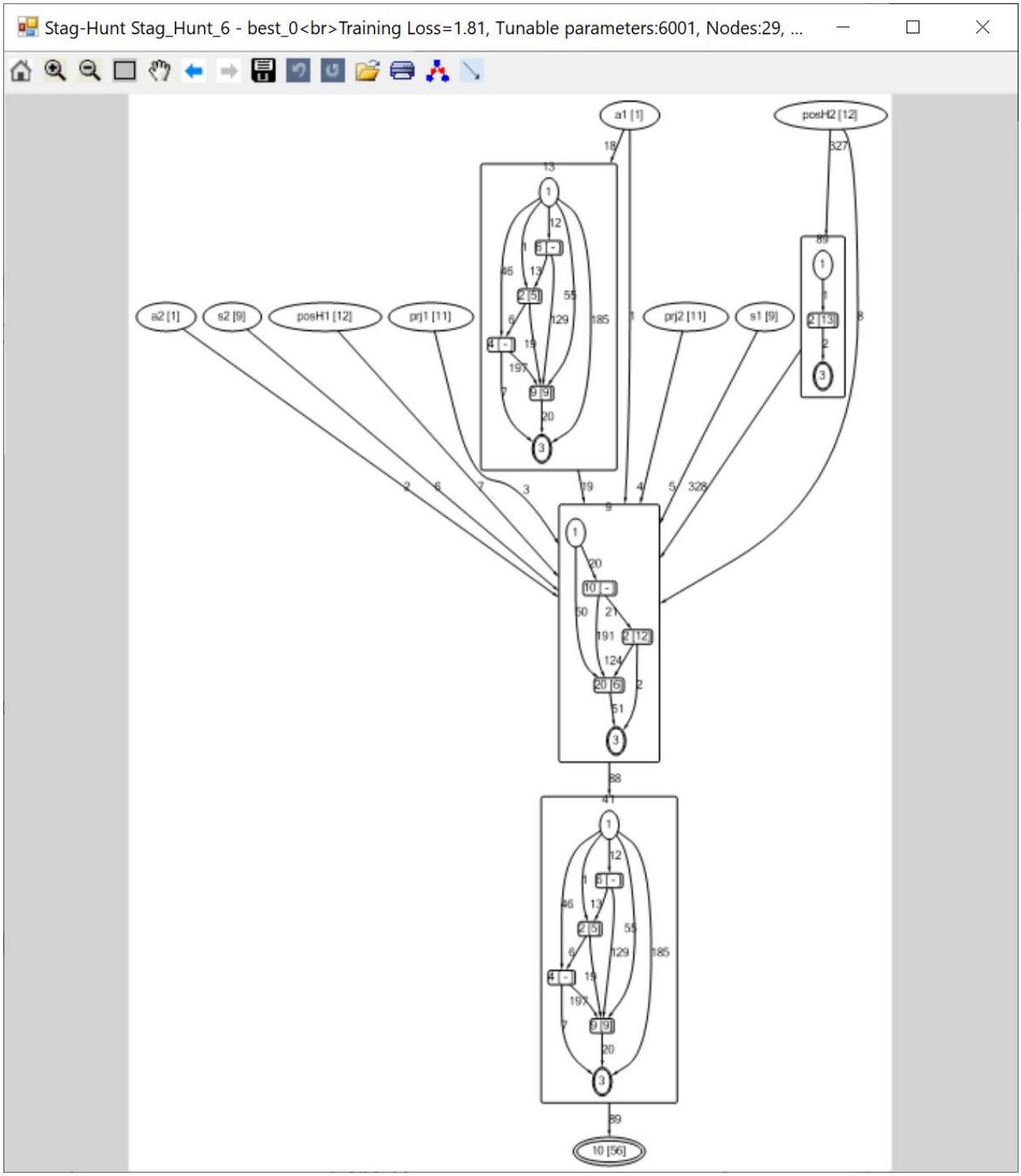

Stag-Hunt Stag_Hunt_3 - best_1
Training Loss=1.88, Tunable parameters:2293, ...
s1 [9]
a2 [1]
a1 [1]
prj1 [11]
posH2 [12]
5
2
1
3
8
9
1
1
2 [6]
49
20
9 [-]
21
3
249
84
1
1
2 [13]
2
3
prj2 [11]
228
250
10 [56]

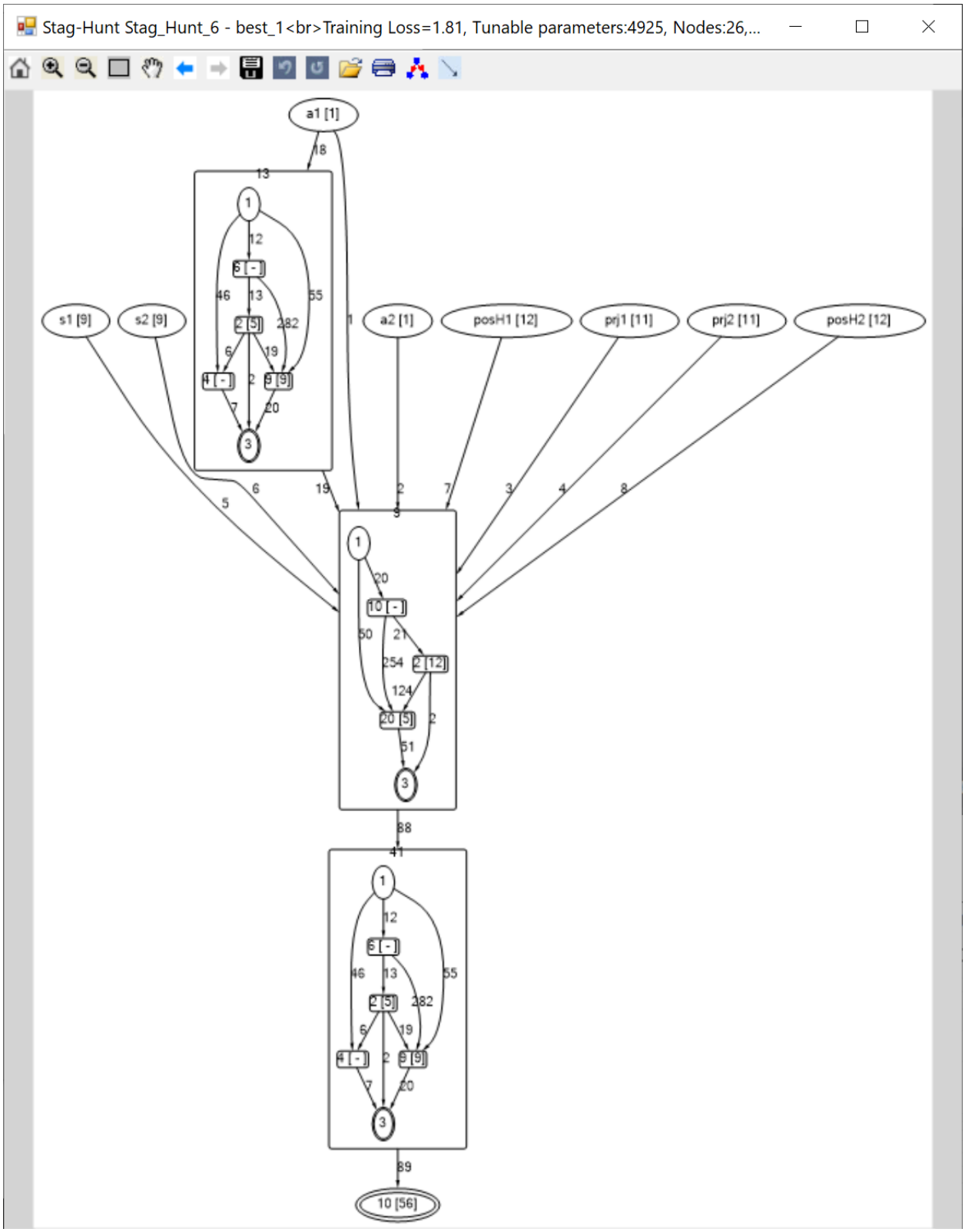

Stag-Hunt Stag_Hunt_4 - best_0
Training Loss=1.88, Tunable parameters:9103, Nodes:26, Con...
s2 [9]
a1 [1]
posH1 [12]
prj2 [11]
s1 [9]
a2 [1]
posH2 [12]
prj1 [11]
10 [56]
121
47
167
41 [-]
145
188
48
6
241
122
24
15
76
84
2 [13]
148
5
2
49
242
22
8
3
9
108
10
172
28
17
29


Stag-Hunt Stag_Hunt_4 - best_1
Training Loss=1.89, Tunable parameters...
s2 [9]
121
47
1
167
41 [-]
145
3
a2 [1]
posH1 [12]
prj2 [11]
306
48
6
122
286
241
76
24
15
39
14 [5]
40
2 [9]
153
2 [5]
prj1 [11]
s1 [9]
posH2 [12]
a1 [1]
3
5
49
8
22
242
1
148
172
108
28
17
10
29
10 [56]


Stag-Hunt Stag_Hunt_5 - best_0
Training Loss=1.79, Tunable parameters...
a1 [1]
prj1 [11]
posH1 [12]
a2 [1]
posH2 [12]
s1 [9]
s2 [9]
prj2 [11]
10 [56]


Stag-Hunt Stag_Hunt_5 - best_1
Training Loss=1.79, Tunable parameters:53...
prj2 [11]
a1 [1]
prj1 [11]
s2 [9]
posH1 [12]
a2 [1]
s1 [9]
10 [56]
337
158
250
229
84
66
64
11
5 [-]
12
2 [13]
2
3
4 [11]
99
62
4
2 [5]
54
22 [5]
5 [14]
55
7
1
6
7
2
169
159
230
5
9
553
38
64
117
48
118


Stag-Hunt Stag_Hunt_6 - best_0
Training Loss=1.81, Tunable parameters:6001, Nodes:29, ...
a1 [1]
posH2 [12]
a2 [1]
s2 [9]
posH1 [12]
prj1 [11]
prj2 [11]
s1 [9]
10 [56]


Stag-Hunt Stag_Hunt_6 - best_1
Training Loss=1.81, Tunable parameters:4925, Nodes:26,...
a1 [1]
s1 [9]
s2 [9]
a2 [1]
posH1 [12]
prj1 [11]
prj2 [11]
posH2 [12]
10 [56]

REFERENCES

Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, *22*, pp. 207-216. Retrieved from https://academic.microsoft.com/paper/2166559705

Ali, M. (2008). *Using Cultural Algorithms to Solve Optimization Problems with a Social Fabric Approach.* Wayne State University, Dissertation.

Ali, M. Z., Suganthan, P. N., Reynolds, R. G., & Al-Badarneh, A. F. (2016). Leveraged Neighborhood Restructuring in Cultural Algorithms for Solving Real-World Numerical Optimization Problems. *IEEE Transactions on Evolutionary Computation, 20*(2), 218-231. doi:10.1109/TEVC.2015.2450018

Al-Tirawi, A., & Reynolds, R. (2019). Using Common Value Auction In Cultural Algorithm to Enhance Robustness and Resilience of Social Knowledge Distribution Systems . *AAAI Spring Symposium: Interpretable AI for Well-being.* Retrieved from https://academic.microsoft.com/paper/2982706073

Al-Tirawi, A., & Reynolds, R. G. (2018). Using Common Value Auction In Cultural Algorithm to Enhance Robustness and Resilience of Social Knowledge Distribution Systems. *IEEE Symposium Series on Computational Intelligence (SSCI)*, (pp. 755-764). Bangalore.

Anderton, C. H. (2003). Conflict and Trade in a Predator/Prey Economy. *Review of Development Economics, 7*(1), 15 - 29.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., . . . Zaremba, W. (2017). Hindsight Experience Replay. *Advances in Neural Information Processing Systems*, (pp. 5048-5058). Retrieved from https://academic.microsoft.com/paper/2964001908

Annen, K. (2019). On the first mover advantage in Stackelberg quantity games. *Journal of Economics, 126*, 249–258.

Axelrod, R., & Hamilton, W. D. (1981). The Evolution of Cooperation. *Science, New Series, 211*(4489), 1390-1396.

Bhuyan, F., Lu, S., Reynolds, R., Zhang, J., & Ahmed, I. (2019). A Security Framework for Scientific Workflow Provenance Access Control Policies. *IEEE Transactions on Services Computing*, 1-1. Retrieved from https://academic.microsoft.com/paper/2951250714

Boudreau, J. W., Rentschler, L., & Sanders, S. (2019). Stag hunt contests and alliance formation. *Public Choice, 179*(3-4), 267.

Chappelow, J. (2019). *Tragedy of The Commons*. Retrieved from Investopedia: https://www.investopedia.com/terms/t/tragedy-of-the-commons.asp

Charnov, E. (1976). Optimal foraging, the marginal value theorem. *Theoretical Population Biology, 9*(2), 129.

Che, X. (2009). *Weaving the Social Fabric: Optimization Problem Solving in Cultural Algorithms using the Cultural Engine.* Wayne State University, Dissertation.

Chu, C. Y., & Thawonmas, R. (2017). Applying Fuzzy Control in Fighting Game AI. *Proceedings of the 77th Convention of Information Processing Society of Japan*, (pp. 101-102). Shiga, Japan. Retrieved from http://www.ice.ci.ritsumei.ac.jp/~ruck/PAP/ipsj4P-03.pdf

Chung, C. (1997). *Knowledge-based approaches to self-adaptation in cultural algorithms.* Phd Dissertation, Wayne State University.

Colon, D. L. (2012). *Ubiquitous Learning Laboratory For Pediatric Nursing: A Cultural Algorithm Approach.* Wayne State University Theses. 203. Retrieved from http://digitalcommons.wayne.edu/oa_theses/203

Cosma Shalizi CMU. (2009). *Estimating Distributions and Densities*. Retrieved from www.stat.cmu.edu: http://www.stat.cmu.edu/~cshalizi/350/lectures/28/lecture-28.pdf

Dawkins, R. (1976). *The Selfish Gene (Book).* Oxford University Press.

DeCanio, S. J., & Fremstad, A. (2011). Game theory and climate diplomacy. *Ecological Economics, 85*(0), 177-187. doi:10.1016/j.ecolecon.2011.04.016

Deep Mind. (2017, 10 18). *AlphaGo Zero: Starting from scratch*. Retrieved from https://deepmind.com/blog/article/alphago-zero-starting-scratch

Dong, Y., Xu, H., & Fan, S. (2019). Memory-based stag hunt game on regular lattices. *Physica A: Statistical Mechanics and its Applications, 519*, 247-255.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, 26(1)*(1083-4419), 29-41.

Evans, A. J. (2014). *Markets for Managers: A Managerial Economics Primer.* John Wiley & Sons.

Gong, C., Chunyang, L., Xu, Z., Lin, P., Junjie, W., Yang, H., . . . Xiaoting, H. (2014). Method for identifying microblog key users based on improved Page Rank. Retrieved from https://academic.microsoft.com/paper/2813726404

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* Cambride, MA: MIT Press. Retrieved from http://www.deeplearningbook.org/

Harrington, J. E., & Zhao, W. (2012). Signaling and tacit collusion in an infinitely repeated Prisoners' Dilemma. *Mathematical Social Sciences, 64*(3), 277 - 289.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*. Retrieved from https://academic.microsoft.com/paper/2949650786

Holland, J. H. (1992). Genetic algorithms. *267*(1), 66-72.

Hoole, J. (2018, 09 2017). *How animals vote to make group decision.* Retrieved from theconversation.com: https://theconversation.com/how-animals-vote-to-make-group-decisions-84134

Howard, D. J., & Berlocher, S. H. (1998). *Endless forms : species and speciation.* New York: Oxford University Press.

Intelligent Computer Entertainment lab., Ritsumeikan University. (2018). *Fighting Game AI Competition.* Retrieved from http://www.ice.ci.ritsumei.ac.jp/~ftgaic/

Intelligent Computer Entertainment lab., Ritsumeikan University. (n.d.). *FightingICE 2018 Championship Results*. Retrieved from http://www.ice.ci.ritsumei.ac.jp/~ftgaic/index-R18.html

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of The 32nd International Conference on Machine Learning*, (pp. 448-456). Retrieved from https://academic.microsoft.com/paper/1836465849

Jalali, A. (2016). Supporting Social Network Analysis Using Chord Diagram in Process Mining. *International Conference on Business Informatics Research*, (pp. 16-32). Retrieved from https://academic.microsoft.com/paper/2516851116

Jayyousi, T. W., & Reynolds, R. G. (2014). Extracting Urban Occupational Plans Using Cultural Algorithms [Application Notes]. *IEEE Computational Intelligence Magazine, 9*(3), 66-87.

Jiang, C., Chen, Y., & Liu, K. J. (2014). Graphical Evolutionary Game for Information Diffusion Over Social Networks. *IEEE Journal of Selected Topics in Signal Processing, 8*(4), 524–536.

Jin, X., & Reynolds, R. (1999). Using Knowledge-Based System with Hierarchical Architecture to Guide the Search of Evolutionary Computation. *Eleventh IEEE International Conference on Tools with Artificial Intelligence*, (pp. 29 - 36).

Judeh, T., Jayyousi, T., Acharya, L., Reynolds, R. G., & Zhu, D. (2014). GSCA: Reconstructing biological pathway topologies using a cultural algorithms approach. *EEE Congress on Evolutionary Computation.* Beijing.

Julien, L. A. (2011). A note on Stackelberg competition. *Journal of Economics, 103*(2), 171-87.

Kennedy, J., & Eberhart, R. C. (1995). Particle Swarm Optimization. *Proceeding of the IEEE International Conference on Neural Networks.* Perth, Australia.

Kober, J., Bagnell, J., & Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research, 32*(11), 1238-1274. Retrieved from https://academic.microsoft.com/paper/1977655452

Konda, V., & Tsitsiklis, J. (2002). *Actor-Critic Algorithms.* Retrieved from https://academic.microsoft.com/paper/2156737235

Kumar, A., Fu, J., Tucker, G., & Levine, S. (2019). Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. *Conference on Neural Information Processing Systems (NeurIPS).* Vancouver, Canada.

Kumar, R., Raghavan, P., Rajagopalan, S., & Tomkins, A. (1999). Trawling the Web for emerging cyber-communities. *the web conference, 31*(11), 1481-1493. Retrieved from https://academic.microsoft.com/paper/2151626491

Lancichinetti, A., & Fortunato, S. (2009). Community detection algorithms: A comparative analysis. *Physical Review E, 80*(5), 56117. Retrieved from https://academic.microsoft.com/paper/1995996823

Langton, C. (1990). Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D: Nonlinear Phenomena, 42*(1-3), Pages 12-37.

Lu, T., Wang, X., Gao, Q., & Liu, C. (2015). Modified Page Rank Model in Investigation of Criminal Gang. *International Journal of Science and Engineering Applications, 4*, 100-104. Retrieved from https://academic.microsoft.com/paper/418250562

Matej Črepinšek, S. L. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR), 45*(3). doi:http://dx.doi.org.proxy.lib.wayne.edu/10.1145/2480741.2480752

Maynard Smith, J., & Price, G. (1973). The logic of animal conflict. *Nature, 246*(0), 15-18.

Microsoft. (2019). *Power BI*. Retrieved from Power BI: https://powerbi.microsoft.com

Microsoft Corporation. (n.d.). *F# Generics*. Retrieved from https://docs.microsoft.com/en-us/dotnet/fsharp/language-reference/generics/

Microsoft Research. (n.d.). *Microsoft Automatic Graph Layout* . Retrieved from https://www.microsoft.com/en-us/research/project/microsoft-automatic-graph-layout/

Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., . . . Hodjat, B. (2017). Evolving Deep Neural Networks. *CoRR*. Retrieved from https://arxiv.org/abs/1703.00548

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . Ku, D. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.

Moeckli, D. (2018). Referendums: Tyranny of the Majority? *Swiss Political Science Review, 24*(3), 335-341.

Mojab, S., Ebrahimi, M., Reynolds, R., & Lu, S. (2019). iCATS: Scheduling Big Data Workflows in the Cloud Using Cultural Algorithms. *2019 IEEE Fifth International Conference on Big Data Computing Service and Applications (BigDataService)*, (pp. 99-106). Retrieved from https://academic.microsoft.com/paper/2975462880

Morrison, R., & De Jong, K. (1999). A test problem generator for nonstationary environments. *Proceedings of the 1999 Congress on Evolutionary Computation.* Washington, DC.

Nash, J. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences, 36*(1), 48-49.

Peng, B. (2005). *Knowledge Swarms in Cultural Algorithms for Dynamic.* Wayne State University PhD Thesis.

Perreault, C. (2012). The Pace of Cultural Evolution. *PLoS ONE*, 7(9). Retrieved from http://groups.engin.umd.umich.edu/vi/w2_workshops/cultural_alg_reynolds_w2.pdf

Pershina, M., He, Y., & Grishman, R. (2015). Personalized Page Rank for Named Entity Disambiguation. *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, (pp. 238-243). Retrieved from https://academic.microsoft.com/paper/2295227292

Reynolds, R. G. (1978). On modeling the evolution of hunter-gatherer decision-making systems. *Geographical Analysis, 10*(1), 31-46.

Reynolds, R. G. (2019). *Culture on the Edge of Chaos: Cultural Algorithms and the Foundations of Social Intelligence.* SpringerBriefs in Computer Science.

Reynolds, R. G. (2020). *Cultural Algorithms: Tools to Model Complex Dynamic Social Systems.* IEEE Press Series on Computational Intelligence.

Reynolds, R. G., & Saleem, S. M. (2005). The impact of environmental dynamics on cultural emergence. *Perspectives on Adaptions in Natural and Artificial Systems* , 253-280.

Reynolds, R. G., Gawasmeh, Y. A., & Salaymeh, A. (2015). The impact of subcultures in cultural algorithm problem solving. *IEEE Symposium Series on Computational Intelligence* (pp. 1876-1884). Cape Town: Institute of Electrical and Electronics Engineers Inc.

Reynolds, R., & Kinnaird-Heether, L. (2013). Optimization problem solving with auctions in Cultural Algorithms. *Memetic Computing, 5*(2), 83–94. doi:doi:10.1007/s12293-013-0112-8

Reynolds, R., & Kinnaird-Heether, L. (2019). Diversity as a Necessity for Sustainability in Cultural Systems: Collective Problem-Solving in Cultural Algorithms. *2019 IEEE International Conference on Humanized Computing and Communication (HCC).* Retrieved from https://academic.microsoft.com/paper/2999579549

Riehmann, P., Hanfler, M., & Froehlich, B. (2005). Interactive Sankey diagrams. *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, (pp. 31-31). Retrieved from https://academic.microsoft.com/paper/1932144848

Roweis, S., & Saul, L. (2000). Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science, 290*(5500), 2323-2326. Retrieved from https://academic.microsoft.com/paper/2053186076

Saleem, S. (2001). *Knowledge-based Solution to Dynamic Optimization Problems Using Cultural Algorithms.* Wayne State University, Ph. D. Thesis.

Schelling, T. C. (1971). Dynamic Models of Segregation. *Journal of Mathematical Sociology, 1*(2), 143-186.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., . . . Silver, D. (2019). Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *ArXiv Preprint*. Retrieved from https://arxiv.org/abs/1911.08265

Schulman, J., Levine, S., Moritz, P., Jordan, M., & Abbeel, P. (2015). Trust Region Policy Optimization. *arXiv preprint arXiv:1502.05477*. Retrieved from https://academic.microsoft.com/paper/2949608212

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*. Retrieved from https://academic.microsoft.com/paper/2736601468

Seide, F., & Agarwal, A. (2016). CNTK: Microsoft's Open-Source Deep-Learning Toolkit. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (pp. 2135-2135). Retrieved from https://academic.microsoft.com/paper/2513383847

Shi, Y. (2011). Brain Storm Optimization Algorithm. *International conference in swarm intelligence.* Berlin, Heidelberg: Springer.

Shoham, Y., & Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations (Book).* Cambridge University Press.

Silver, D., Schrittwieser, Julian, Simonyan, Karen, Antonoglou, Ioannis, Huang, Aja, Guez, Arthur, . . . Hassabis, Demis. (2017). Mastering the game of Go without human knowledge. *Nature, 550*(7676), 354-359.

Skyrms, B. (2004). *The Stag Hunt and Evolution of Social Structure.* Cambridge, UK: Cambridge University Press.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation, 10*(2), 99-127.

Stanley, K. O., Bryant, B. D., & Miikkulainen, R. (2005). Real-Time Neuroevolution in the NERO Video Game. *IEEE Transactions on Evolutionary Computation 2005, 9*(6), 653-668.

Stanley, S. D. (2020). *CAPSO: A Multi-Objective Cultural Algorithm System to Predict Locations of Ancient Sites.* Wayne State University.

Storn, R., & Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11*(4), 341–359.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning : An Introduction (second edition).* Cambridge, Mass: MIT Press.

Sutton, R., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems 12*, *12*, pp. 1057-1063. Retrieved from https://academic.microsoft.com/paper/2155027007

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (pp. 1-9). Retrieved from https://academic.microsoft.com/paper/2097117768

Takano, M., Wada, K., & Fukuda, I. (2016). Reciprocal Altruism-based Cooperation in a Social Network Game. *New Generation Computing, 34*(3), 257–272.

Veelen, M. v., & Spreij, P. (2009). Evolution in games with a continuous action space. *Econ Theory, 39*, 355–376.

von Neumann, J., & Morgenstern, O. (1947). *Theory of games and economic behavior, 2nd edition.* Princeton, NJ: Princeton University Press.

Wang, X., Luo, C., Ding, S., & Wang, J. (2018). Imitating Contributed Players Promotes Cooperation in the Prisoner's Dilemma Game. *IEEE Access, 6*, 53265-53271.

Waris, F., & Reynolds, R. G. (2015). Using Cultural Algorithms to Improve Wearable Device Gesture Recognition Performance,. *IEEE Symposium Series on Computational Intelligence.* Cape Town.

Waris, F., & Reynolds, R. G. (2018). Optimizing AI Pipelines: A Game-Theoretic Cultural Algorithms Approach. *IEEE Congress on Evolutionary Computation (CEC).* Rio de Janeiro.

Weibull, J. W. (1995). *Evolutionary game theory.* MIT Press.

Wills, R. (2006). Google's page rank: the Math behind the search engine. *The Mathematical Intelligencer, 28*(4), 6-11. Retrieved from https://academic.microsoft.com/paper/2946417399

Wu, X., Kumar, V., Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., . . . Steinberg, D. (2007). Top 10 algorithms in data mining. *Knowledge and Information Systems, 14*(1), 1-37. Retrieved from https://academic.microsoft.com/paper/2142827986

Yao, X., & Darwen, P. (1994). An Experimental Study of N-Person Iterated Prisoner's Dilemma Games. *Informatica (slovenia), 18*(4), 435-450. Retrieved from https://academic.microsoft.com/paper/2108238158

Yu, M., & Hong, S. H. (2016). Supply–demand balancing for power management in smart grid: A Stackelberg game approach. *Applied Energy, 164*, 702-710.

ABSTRACT

CATGAME: A TOOL FOR PROBLEM SOLVING IN COMPLEX DYNAMIC SYSTEMS USING GAME THEORETIC KNOWLEDGE DISTRIBUTION IN CULTURAL ALGORITHMS, AND ITS APPLICATION (CATNEURO) TO THE DEEP LEARNING OF GAME CONTROLLER

by

FAISAL WARIS

May 2020

Advisor: Dr. Robert G. Reynolds

Major: Computer Science

Degree: Doctor of Philosophy

Cultural Algorithms (CA) are knowledge-intensive, population-based stochastic optimization methods that are modeled after human cultures and are suited to solving problems in complex environments. The CA Belief Space stores knowledge harvested from prior generations and re-distributes it to future generations via a knowledge distribution (KD) mechanism. Each of the population individuals is then guided through the search space via the associated knowledge. Previously, CA implementations have used only competitive KD mechanisms that have performed well for problems embedded in static environments. Relatively recently, CA research has evolved to encompass dynamic problem environments. Given increasing environmental complexity, a natural question arises about whether KD mechanisms that also incorporate cooperation can perform better in such environments than purely competitive ones? Borrowing from game theory,

game-based KD mechanisms are implemented and tested against the default competitive mechanism – Weighted Majority (WTD).

Two different concepts of complexity are addressed – numerical optimization under dynamic environments and hierarchal, multi-objective optimization for evolving deep learning models. The former is addressed with the CATGame software system and the later with CATNeuro.

CATGame implements three types of games that span both cooperation and competition for knowledge distribution, namely: Iterated Prisoner's Dilemma (IPD), Stag-Hunt and Stackelberg. The performance of the three game mechanisms is compared with the aid of a dynamic problem generator called Cones World. Weighted Majority, aka “wisdom of the crowd”, the default CA competitive KD mechanism is used as the benchmark. It is shown that games that support both cooperation and competition do indeed perform better but not in all cases. The results shed light on what kinds of games are suited to problem solving in complex, dynamic environments. Specifically, games that balance exploration and exploitation using the local signal of ‘social’ rank – Stag-Hunt and IPD – perform better. Stag-Hunt which is also the most cooperative of the games tested, performed the best overall. Dynamic analysis of the ‘social’ aspects of the CA test runs shows that Stag-Hunt allocates compute resources more consistently than the others in response to environmental complexity changes. Stackelberg where the allocation decisions are centralized, like in a centrally planned economic system, is found to be the least adaptive.

CATNeuro is for solving neural architecture search (NAS) problems. Contemporary ‘deep learning’ neural network models are proven effective. However, the network topologies may be complex and not immediately obvious for the problem at hand. This has given rise to the secondary field of neural architecture search. It is still nascent with many frameworks and

approaches now becoming available. This paper describes a NAS method based on graph evolution pioneered by NEAT (Neuroevolution of Augmenting Topologies) but driven by the evolutionary mechanisms under Cultural Algorithms. Here CATNeuro is applied to find optimal network topologies to play a 2D fighting game called FightingICE (derived from “The Rumble Fish” video game). A policy-based, reinforcement learning method is used to create the training data for network optimization. CATNeuro is still evolving. To inform the development of CATNeuro, in this primary foray into NAS, we contrast the performance of CATNeuro with two different knowledge distribution mechanisms – the stalwart Weighted Majority and a new one based on the Stag-Hunt game from evolutionary game theory that performed the best in CATGame. The research shows that Stag-Hunt has a distinct edge over WTD in terms of game performance, model accuracy, and model size. It is therefore deemed to be the preferred mechanism for complex, hierarchical optimization tasks such as NAS and is planned to be used as the default KD mechanism in CATNeuro going forward.

AUTOBIOGRAPHICAL STATEMENT

Mr. Faisal Waris is currently a PhD Candidate in the Depart of Computer Science, College of Engineering. His received his Master's in Business Administration from York University, Toronto in 1986. He started his PhD program formally in 2011 after working for several years in the various industries, including banking, insurance, telemarketing and automotive. Mr. Waris was with Ford Motor Company as a consultant for 15 years. He is currently employed by General Motors as a Senior Data Scientist. Mr. Waris has several publications related to research in Cultural Algorithms with several others in progress.