Investigating Kauffman's NK Model for Agent-Based Modelling

Richard Edward Mellor

Bachelor of Science in Computer Science with Honours
The University of Bath
May 2007
This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed:
Investigating Kauffman’s NK Model for Agent-Based Modelling

Submitted by: Richard Edward Mellor

COPYRIGHT

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see http://www.bath.ac.uk/ordinances/#intelprop).

This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:
Abstract

Models are constantly being developed to help us to simulate and predict what the consequences of our actions may be. Whether it is how to make the perfect cup of tea or the behaviour of epistatic genes, if used correctly they can give us valuable insight. We begin by investigating Kauffman’s NK Model, studying the mechanics behind it, its popularity with business analysts and what it’s so far been applied to.

Finally we develop our own agent based version of the NK Model using the MASON Toolkit, comparing our results with Kauffman’s original results, concluding by considering extra functionality.
# Contents

CONTENTS..................................................................................................................................................1

LIST OF FIGURES ...................................................................................................................................... IV

LIST OF TABLES ......................................................................................................................................... V

ACKNOWLEDGEMENTS............................................................................................................................ VI

INTRODUCTION ......................................................................................................................................... 1

1.1 AIM ..................................................................................................................................................... 1

1.2 OBJECTIVES ..................................................................................................................................... 2

LITERATURE SURVEY ................................................................................................................................ 3

2.1 INTRODUCTION ................................................................................................................................... 3

2.2 FITNESS LANDSCAPES .......................................................................................................................... 3
  2.2.1 Fitness Landscape Structure ........................................................................................................... 4
  2.2.2 Fitness Landscape Models ............................................................................................................... 6
  2.2.3 Summary ....................................................................................................................................... 7

2.3 NK MODEL ........................................................................................................................................... 7
  2.3.1 Main Parameters .............................................................................................................................. 8
  2.3.2 Fitness Calculation .......................................................................................................................... 10
  2.3.3 Effects on the Landscape of Varying K ........................................................................................... 12
  2.3.4 Travelling the Landscape ............................................................................................................... 15
  2.3.5 Application of NK Model ............................................................................................................. 17
  2.3.6 Possible Extensions to The NK Model .......................................................................................... 19
  2.3.7 Critique of NK Model ................................................................................................................... 20
  2.3.8 Conclusion ................................................................................................................................... 21

2.4 AGENT BASED MODELLING ............................................................................................................... 22
  2.4.1 What Are Agents? .......................................................................................................................... 22
  2.4.2 Agent Toolkits ............................................................................................................................... 22
  2.4.3 JADE ............................................................................................................................................. 23
  2.4.4 Mason .......................................................................................................................................... 23
  2.4.5 Netlogo ....................................................................................................................................... 23
  2.4.6 Repast ........................................................................................................................................ 24
  2.4.7 Conclusion ................................................................................................................................... 24

REQUIREMENTS ........................................................................................................................................ 25

3.1 FUNCTIONAL REQUIREMENTS .............................................................................................................. 25
  3.1.1 Additional Requirement ................................................................................................................ 27

3.2 NON-FUNCTIONAL REQUIREMENTS .................................................................................................... 27

3.3 CONCLUSION ...................................................................................................................................... 28

DESIGN ....................................................................................................................................................... 29
Investigating Kauffman's NK Model for Agent-Based Modelling

4.1 HIGH LEVEL ARCHITECTURE .................................................................................. 29
4.2 MODEL CLASS .......................................................................................................... 30
  4.2.1 Outputting Results Generated ........................................................................... 30
4.3 NK_SPACE_GENCLASS ......................................................................................... 32
  4.3.1 Creating the Landscape ..................................................................................... 32
  4.3.2 Storing each Combination ................................................................................. 36
  4.3.3 Assigning fitness values ..................................................................................... 37
4.4 AGENT CLASS .......................................................................................................... 37
4.5 CONCLUSION ............................................................................................................ 38

IMPLEMENTATION AND TESTING .............................................................................. 39

5.1 BASIC INSTRUCTIONS TO RUN THE NK MODEL .................................................. 39
  5.1.1 Basic instructions to compile the NK Model from the command line ................... 40
5.2 AGENT TOOLKIT SELECTED ................................................................................... 40
  5.2.1 MersenneTwister Random Number Generator .................................................. 41
  5.2.2 Bag .................................................................................................................... 41
5.3 MAIN FEATURES OF PROGRAM ............................................................................ 41
  5.3.1 NKSpace_Gen Class ......................................................................................... 41
  5.3.2 Model Class ...................................................................................................... 43
  5.3.3 Agent Class ....................................................................................................... 43
  5.3.4 Model WithUI Class ......................................................................................... 43
5.4 CONCLUSION ............................................................................................................ 44
5.5 TESTING ................................................................................................................... 44

EXPERIMENTS .............................................................................................................. 45

6.1 MEAN FITNESS OF LOCAL OPTIMA (NEAREST-NEIGHBOUR INTERACTIONS) ........................................ 45
  6.1.1 Method ............................................................................................................... 45
  6.1.2 Hypothesis ........................................................................................................ 46
6.2 MEAN FITNESS OF LOCAL OPTIMA (RANDOM INTERACTIONS) .......................................................... 46
  6.2.1 Method ............................................................................................................... 47
  6.2.2 Hypothesis ........................................................................................................ 47
6.3 IN DEPTH MEAN FITNESS OF LOCAL OPTIMA (RANDOM INTERACTIONS) .................................................. 48
  6.3.1 Method ............................................................................................................... 48
  6.3.2 Hypothesis ........................................................................................................ 49
6.4 ENTREPRENEUR VS. SILENT INVESTOR ................................................................ 49
  6.4.1 Method ............................................................................................................... 50
  6.4.2 Hypothesis ........................................................................................................ 51

RESULTS ........................................................................................................................ 52

7.1 RESULTS – MEAN FITNESS OF LOCAL OPTIMA (NEAREST-NEIGHBOUR INTERACTIONS) ................... 52
  7.1.1 Slow Search Travel Results .............................................................................. 52
  7.1.2 Fast Search Travel Results ............................................................................... 53
  7.1.3 Conclusion ........................................................................................................ 53
7.2 RESULT – MEAN FITNESS OF LOCAL OPTIMA (RANDOM INTERACTIONS) ............................................... 54
  7.2.1 Slow Search Travel Results .............................................................................. 54
  7.2.2 Fast Search Travel Results ............................................................................... 55
  7.2.3 Slow Search Vs. Fast Search Comparison ......................................................... 56
  7.2.4 Conclusion ........................................................................................................ 56
7.3 IN DEPTH MEAN FITNESS OF LOCAL OPTIMA (RANDOM INTERACTIONS) .................................................. 57
7.4 ENTREPRENEUR VS. SILENT INVESTOR ................................................................ 58
7.5 CONCLUSION ............................................................................................................ 59
PROJECT CRITIQUE ................................................................................................................................. 61
  8.1 MAIN PROBLEMS WITH THE PROJECT .......................................................................................... 61
     8.1.1 Investigation Problems ........................................................................................................ 61
     8.1.2 Implementation Problems .................................................................................................... 62
  8.2 STRENGTHS .................................................................................................................................... 63
  8.3 SUMMARY ......................................................................................................................................... 63

CONCLUSIONS ........................................................................................................................................... 64
  9.1 WEAKNESSES & IMPROVEMENTS ............................................................................................... 64
  9.2 FURTHER WORK ............................................................................................................................ 65
  9.3 FINAL CONCLUSION ...................................................................................................................... 65

BIBLIOGRAPHY ......................................................................................................................................... 67

RESULTS ..................................................................................................................................................... 70
  IN DEPTH MEAN FITNESS OF LOCAL OPTIMA (RANDOM INTERACTIONS) .................................... 73

CODE ......................................................................................................................................................... 76
List of Figures

**Figure 1.** Genotype space of 2, 3 or 4 genes in a genotype with two-alleles. Redrawn from Wright (1932)........................................................................................................ 5

**Figure 2.** Example of K assignment on N4 when N = 7, K =4. (a) Nearest neighbour influences. (b) Randomly assigned influences .......................................................... 9

**Figure 3.** Calculating fitness, when N=3, K=2, A=2. (a), (b) Assignment of K =2 gene dependences to each gene (N). (c) Assignment of random fitness between 0.0 and 1.0 to each allele combination ($A^{K-1}$). (d) Assignment of each genes (N) fitness value ($W$), and each genotypes average overall fitness ($W$)........................................................................ 12

**Figure 4.** Impression of NK landscape when varying K. (a) A simple landscape with high correlation similar to ($K=0$). (b) A rugged landscape with several local peaks, similar to ($0<K<N-1$). (c) A completely random rugged landscape, similar to ($K=N-1$). ........................................................................................................................................ 13

**Figure 5.** Fitness landscape on the three dimensional Boolean cube, corresponding to the fitness values of the eight genotypes in (Figure 3d). Note that more than one local optimum exists................................................. 15

**Figure 6.** High level class architecture design................................................................. 29

**Figure 7.** Design of results output to file........................................................................ 31

**Figure 8.** Output from GUI Model, showing agents fitness value as it travels the landscape........................................................................................................................................ 44

**Figure 9.** Results - mean fitness of local optima (random interactions) – long search. Compared with Kauffman’s original results......................................................... 54

**Figure 10.** Results - mean fitness of local optima (random interactions) – fast search. Compared with Kauffman’s original results.......................................................... 55

**Figure 11.** Results - mean fitness of local optima (random interactions) – slow and fast search comparison ................................................................................................. 56

**Figure 12.** In depth mean fitness of local optima (random interactions) – slow search. For a range of ($N=2 – 16$ and $K=0 – 15$)......................................................................................... 57
List of Tables

**Table 1** Kauffman’s Results of Mean Fitness of Local Optima (Nearest-Neighbour Interactions). Taken from Kauffman (1993) Table 2.1. ...................................................... 46
**Table 2** Kauffman’s Results of Mean Fitness of Local Optima (Random Interactions). Taken from Kauffman (1993) Table 2.2. .............................................................. 47
Acknowledgements

I would just like to thank my supervisors Julian Paget and Richard Vidgen for all their help and assistance with this project. Also giving a special thanks to my parents who have supported and encouraged me in whatever I have wanted to achieve. Finally I would like to thank Stacey for her support, and dinner service she provided as the deadline approached.
Chapter 1

Introduction

As the interest in Kauffman’s NK Model grows, more business analysts appear to want to adapt Kauffman’s NK Model for their own use (e.g. McCarthy, 2002; Vidgen et al., 2006). Few implementations of the NK Model, appear to have been actually implemented and those that have, have been collectively used, maybe without providing the user with the necessary complete underlying knowledge of the model.

Kauffman’s (1993) NK Model was originally designed as a method of varying the correlation of fitness landscapes in biological evolution and speciation. Kauffman (1993) only defines the NK Model in the biological context which he designed it for and not as a generic version, for which he stated the model can be used. The NK Model has since become a key model in biological evolution and more recently become popular with business analysts applying it to anything from situated learning theory (Yuan et al., 2004) to manufacturing strategies (McCarthy, 2002).

1.1 Aim

The aim of this project is to investigate Kauffman’s NK Model by breaking it down to its core components, to gain a true understanding of the mechanics behind the design. With this knowledge we can then redefine and build Kauffman’s NK Model as a generic version.

Through doing this we will provide a template for a model which can be customised and enhanced for use in numerous areas of research.
1.2 Objectives

1. Research Kauffman’s NK Model and produce a formal generic definition of the NK Model.

2. Study the current uses of the NK Model in different research areas to gain an understanding of what is commonly required from the NK Model.

3. Using our knowledge gained on the current uses of the NK model along with our formal definition, investigate and appraise possible methods for implementing each part of the model. Given this information, evaluate the relative merits of each method in order to produce a comprehensive design of the NK Model.

4. Apply our comprehensive design and requirements of the NK Model to implement a generic agent based simulation of Kauffman’s NK Model.

5. Using Kauffman’s original experiments with the NK Model as a guide produce a comprehensive schedule of experiments to obtain results which can be compared with the Kauffman’s results from the original NK Model.

6. Consider additional features which have been applied to the NK Model along with our own additional suggestions and implement these additional functions to try and improve the extendibility of the NK Model.
Chapter 2

Literature Survey

2.1 Introduction

Before embarking on the development of our generic NK Model it is vital that we explore Kauffman’s NK model, examining the structure and mechanics behind it so we are able to successfully reproduce it. We will also investigate the current applications of the NK Model and explore alterations that have been made and the effect in which they had on the structure and results produced.

After a complete understanding of Kauffman’s NK Model and the current applications established, we move on to looking at agent based simulations; what they are, how they operate, the possible toolkits available and how they are currently used, with an aim to find the most appropriate toolkit for our proposed purpose.

The results of this research should provide a clear and well defined investigation into Kauffman’s NK model, giving a better understanding of its structure and operation. This knowledge combined with an understanding of the current agent toolkits available will enable us to compare and select the most appropriate toolkit for our proposed purpose.

2.2 Fitness Landscapes

Kauffman’s (1993) NK Model was based on the concept of rugged fitness landscapes, an idea introduced by the biologist Sewall Wright (1932) for visualizing biological evolution and speciation. Before proceeding with the investigation of Kauffman’s (1993) NK Model it is vital to first gain an adequate understanding of “fitness landscapes” in evolutionary biology.
Investigating Kauffman’s NK Model for Agent-Based Modelling

Biological evolution is an enormously complex process, influenced by a large number of genetic, environmental, developmental and ecological factors (Gavrilets, 2004). To understand such a complex phenomenon, it is extremely useful to have a simple metaphor. Wright’s (1932) “fitness landscape” metaphor also known as “adaptive landscapes,” “adaptive topographies,” and “surfaces of selective value,” and has become a standard tool for visualising evolution and speciation (Gavrilets, 2004). Wright’s metaphor is still today considered by many as one of his most important contributions to evolutionary biology (Coyne et al., 1997; Arnold et al., 2001). At the same time the metaphor has become subject to some controversy. Skipper (2004) contrasts Provine’s (1986) critiques of the metaphor with Ruse’s (1996) argument in defence of Wright’s fitness landscape.

A fundamental issue of Wright’s fitness landscapes highlighted by Provine (1986) was that there are actually two somewhat different versions of fitness landscapes which Wright himself used interchangeably (Gavrilets, 2004). Thus potentially causing confusion about the fitness landscapes exact meaning, dimensionality and justification. Perhaps Wright’s biggest mistake was in not formally defining the fitness landscape concept clearly enough before using it for the intent and purpose of which he designed it for.

Gavrilets (2004) tried to distinguish Wright’s fitness landscape concept, describing the first version, as a relationship between possible combinations of genes and fitness (or a fitness component), with a genotype space being the set of all possible genotypes. He therefore formally defines a fitness landscape as a map, which assigns a fitness value to each specific combination of genes in the genotype space. In Wright’s second version, a fitness landscape is a relationship between possible genetic structures of a population and its average fitness. A fitness landscape in this case can be formally defined as a map that assigns an average fitness value to each population state in the population genetic structure space (Gavrilets, 2004). Kauffman’s (1993) NK Model adopts the former definition of a Wright’s (1932) fitness landscape and it is this definition the investigation will focus on.

### 2.2.1 Fitness Landscape Structure.

In Wright’s 1932 original definition, fitness landscapes represent the fitness of gene combinations, with each gene having a specified amount of alleles, known as variants of a gene, see Campbell et al. (2002: 249). A fitness landscape is constructed by first specifying a genotype space (see Figure 1), the set of all possible genotypes, where a genotype is described as a sequence of genes at specific positions (Gavrilets, 2004). Fitness values are then assigned to each genotype within the genotype space, this represents the "height" of the landscape. Genotypes which are similar are said to be "close" to each other, while those that are different are "far" from each other. Through combining these two concepts of height and distance, we form the idea of a "landscape". Thus the set of all possible genotypes, the similarity between them, and their fitness values, creates a fitness landscape. (Gavrilets, 2004; Wright, 1932)
Investigating Kauffman's NK Model for Agent-Based Modelling

Illustrated in figure 1, the genotype space for a genotype made up of two, three and four genes each with two alleles, linked to their closest variant genotypes, with one gene difference. A genotype space can therefore be calculated by $A^g$ where $A$ is the number of alleles at each gene and $g$ is the number of genes within a genotype.

Wright’s Rugged Fitness Landscape

Wright’s image of a typical fitness landscape was that of a “rugged” surface containing many isolated local “fitness peaks” of different heights separated by “fitness valleys” of different depths. Fitness peaks signify high-fitness combinations of genes, offering alternative solutions to the problem of survival, which all biological organisms face, while fitness valleys between peaks representing low fitness contributions among the genes (Gavrilets, 2004; Wright, 1932).

Within the structure of fitness landscapes, adaptive evolution (moving up to a local fitness peak) is considered as “hill climbing”. Once at the top of a peak however there is no way to improve fitness any further, without any additional forces being involved. A vital concept of the fitness landscape is the fact that the peak which is reached does not necessarily have the highest fitness. On the contrary, it is much more plausible that the peak found has an intermediate height and higher peaks exist nearby. (Gavrilets, 2004; Wright, 1932).

Figure 1. Genotype space of 2, 3 or 4 genes in a genotype with two-alleles. Redrawn from Wright (1932).
Wright’s (1932) definition of a rugged fitness landscape leads us to an important question; how can fitness can be increased further when a local fitness peak has been reached? This is something that Kauffman’s (1993) discusses and what this investigation will look into further later on in this chapter.

**Fisher’s Single-peak Fitness Landscape**

In contrast to Wright, Fisher (1930, cited by Provine 1986, pp. 274-275) suggested that the number of dimensions in a fitness landscape, determined by the number of genotypes that can be obtained from a given genotype by changing single genes (Gavrilets, 2004), increases and local peaks in lower dimensions become saddle points in higher dimensions. In this case according to Fisher (1930, cited by Provine 1986, pp. 274-275), natural selection will be able to move the global peak without any additional force being required, which implies with Fisher’s view that a typical fitness landscape only has a single peak.

Kauffman et al. (1987) claim that Fisher's criticism is not justified, arguing that a typical fitness landscape is filled with local peaks and finding the global peak by selection alone is, in general, impossible. Kauffman et al. (1987) and Gavrilets (2004) however do accept that single-peak landscapes can occur in some simple models but the majority are to complex.

2.2.2 Fitness Landscape Models

Kauffman’s (1993) NK Model is not the only model created to use Wright’s (1932) fitness landscape concept as its foundations. This investigation briefly reviews three different models which all utilise Wright’s (1932) fitness landscape concept, providing an insight into the alternative models available before focusing on Kauffman’s (1993) NK Model.

**Russian Roulette Model**

Gavrilets (2004) describes the Russian Roulette Model to have two separate versions, two dimensional and hypercubes. Essentially the two model versions share the same fundamental principles, the latter merely being a more complex multidimensional version. In the Russian Roulette Model, fitness can only take two values, 0 and 1, potentially seen as a serious limitation, although Gavrilets (2004, pp.89-90) demonstrates otherwise. Gavrilets (2004) describes the Russian Roulette model as having an uncorrelated landscape, in which any genetic change (including the alteration of a single gene) results in an independent fitness value.

**Multiplicative Fitness Model**

In contrast to the Russian Roulette model, the Multiplicative Fitness Model represents alternative alleles as “advantageous” and “deleterious” and the fitness on an individual
genotype is dependent on k deleterious alleles within it Gavrilets (2004). The fitness landscape in this model is therefore considered be highly correlated, with similar genotypes having similar fitness values Gavrilets (2004).

**NK Model**

The Russian Roulette Model examined above provides an example of an extremely rugged fitness landscape, with a colossal amount of local peaks and total lack of correlation between the fitness of similar genotypes. While the Multiplicative Model, in contrast, lies at the opposite end of the spectrum, producing highly correlated fitness landscapes with fitness values of similar genotypes generally being similar. Kauffman (1993) however came up with a model for random fitness landscape in which the degree of correlation could be changed by altering a single parameter, (see Section 2.3).

### 2.2.3 Summary

Wright’s (1932) fitness landscape metaphor uses analogies with geographical landscapes to help with the visualisation of complex and generally unknown problem spaces. The importance of Wright’s (1993) metaphor is that it allows something very complex in biological evolution to be visualised as a concept everyone is familiar with. Provine (1986) criticises Wright’s (1932) fitness landscape concept, illustrating how he actually had two different versions which he used interchangeably, potentially causing confusion about the fitness landscapes exact meaning. This investigation shows that the two versions, are not necessarily a flaw within Wright’s (1932) concept but merely, inadequate in defining the fitness landscape concept formally before applying it in the context he designed it for. A question also raised by this investigation, is the value of Wright’s metaphor for visualising more complex landscapes which are multidimensional. As the human brain normally struggles with anything more than three dimensions, can we really visualise the correct landscape or is it actually misleading us?

Various models have been designed to utilise Wright’s (1932) fitness landscape concept, differing design to simulate extremely rugged landscapes with no correlation to highly correlated landscapes with potentially a single peak. The most intriguing model however was Kauffman’s (1993) NK Model which allows the correlation be changed by altering a single parameter, and it is Kauffman’s (1993) model the investigation will now focus on.

### 2.3 NK Model

Stuart A. Kauffman, a theoretical biologist and complex systems researcher is best known for arguing that the complexity of biological systems and organisms, may result as much from self-organisation and far-from-equilibrium dynamics as from Darwinian’s natural selection (Kauffman, 1993).
Investigating Kauffman’s NK Model for Agent-Based Modelling

Kauffman developed the NK Model, named after its two key parameters $N$ and $K$, as a formal model of rugged fitness landscapes. The original purpose of which was to model genetic interactions. It was also designed as a problem-independent model for constructing landscapes that could easily be changed from smooth to rugged (Kauffman, 1993).

### 2.3.1 Main Parameters

Quite surprisingly, due to the designed intent and purpose of the NK Model, Kauffman (1993) defines the model as only having four main parameters with a potential fifth parameter:

1. The number of parts in a system ($N$).
2. The number of other parts ($K$) which influence each $N$.
3. How the number of parts ($K$) influencing each $N$ are distributed within a system.
4. Number of alleles at each site ($A$).
5. The underlying distribution from which fitness values are assigned.

#### Number of Parts In A System ($N$)

The parameter $N$ represents the number of parts in a system, in Kauffman’s (1993) case referring to genes in a genotype and amino acids in a protein. Each part makes a fitness contribution to the overall system, which is dependent on the part itself and upon $K$ other parts within the system which $N$ is dependent on.

#### Number of Parts ($K$) Which Influence Each $N$

The second crucial parameter, which together with $N$ creates the fundamental structure to the NK Model is $K$, referring to how richly cross-coupled the system is. In genetics terms, with which Kauffman (1993) describes the model, $K$ measures the richness of epistatic interactions among the components of the system. Referring back to (Section 2.2) $K$ therefore essentially controls the amount of correlation within the landscape.

#### How The Number of Parts ($K$) Influencing Each $N$ Are Distributed Within A System

Kauffman (1993) describes two methods of assigning each $N$ its ($K$) dependent parts. The first method simply assigns the $K$ dependent parts for each $N$ to $N$’s flanking $K/2$ neighbours either side, (Figure 2a). Alternatively the second method randomly assigns the $K$ dependent
Investigating Kauffman's NK Model for Agent-Based Modelling

parts for each \( N \) to any of the other parts within the system (Figure 2b). Therefore following Kauffman’s (1993) definition each part \((N)\) of a system is dependent on \( K \) other parts plus itself i.e. \((K+1)\).

![Figure 2: Example of \( K \) assignment on \( n_4 \) when \( N = 7, K = 4 \). (a) Nearest neighbour influences. (b) Randomly assigned influences](image)

Kauffman (1993) later proves however that the distribution of \( K \) does not appear to have any dramatic effect of the results the model produces.

**Number of alleles at each site \((A)\)**

Kauffman (1993) maintains the same concept about alleles as previously discussed (see Section 2.2.1). Kauffman (1993) therefore states that each part \((N)\) can have any number of allele \((A)\). Kauffman (1993) gives the example, if \( A = 2 \) then each \( N \) could be either 0 or 1. Kauffman (1993) restricts \( A \) to two, to define and demonstrate the model. With \( A \) and \( N \) both defined the total amount of systems or as Kauffman (1993) preferably refers to, genotypes can be calculated. In the Kauffman’s (1993) example where \( A = 2 \) the total amount of systems/genotypes is \( 2^N \), when \( A \) is unrestricted however the total amount of systems/genotypes is \( A^N \).

**The Underlying Distribution From Which Fitness Values Are Assigned.**

Although Kauffman (1993) does not class this as one of the main parameters in the NK model, he does specify that it is worth consideration, and change to this parameter would alter the results of the NK Model. Kauffman (1993) explains how the range of fitness values assigned to the systems/genotypes is a very sensitive feature of the model. Normally drawn at random from the uniform interval between 0.0 and 1.0, changing the random assignment to a different underlying distribution would alter the models results. Kauffman (1993) discusses two alternatives, a peak Gaussian distribution between 0.0 and 1.0 in which the random decimals are more likely to be near 0.5 than near 0.0 and 1.0, therefore tending to compress fitness values assigned to all possible systems/genotypes closer to the mean of the distribution, 0.5. Alternatively as Kauffman (1993) suggests a U-shaped distribution between 0.0 and 1.0 in which fitness values are more likely to be nearer 1.0 or 0.0 than 0.5. Use of the U-shaped distribution would potentially expand the deviation of fitness values assigned to all possible genotypes farther away from the mean fitness of the ensemble, 0.5.
2.3.2 Fitness Calculation

The fitness calculations to Kauffman’s (1993) NK Model are possibly the key aspect to his model. These allow the landscape to easily be changed from a rugged uncorrelated landscape to a correlated single peak landscape, by altering a single parameter, $K$.

Kauffman (1993) describes the fitness calculation in just two short paragraphs, the comprehensibility of which is poor. Partially due to the description in a genetics context as well as having no clear structure to the process, this can easily lead to confusion with a lack of clear illustrations to help define the calculations better. After extensive research this investigation manages to extract Kauffman’s (1993) key stages to the calculation process, defining each in a comprehensible manner:

- Assign each gene/locus the $K$ genes which impinge upon it.
- The fitness contribution of each allele at each gene in the context of the $K$ other genes which impinge upon that gene must be specified.
  - Fitness contribution of the allele at the $i$th locus depends upon itself (whether it is 1 or 0) and on the alleles, 1 or 0, at $K$ other loci, hence upon $K + 1$ alleles.
  - Number of combinations of these alleles is just $2^{K+1}$
  - Assign to each of the $2^{K+1}$ combinations at random, a different fitness contribution drawn from the uniform distribution between 0.0 and 1.0
- For each gene, its fitness contribution is generated by random assignment of the $2^{K+1}$ allele combinations of the $K + 1$ genes which impinge upon it.
- The fitness of an entire genotype is the average of the contribution of all the loci/genes within it.

(Kauffman 1993, pp 42)

Assign each Gene/Locus the $K$ Genes which Impinge Upon it.

As previously discussed and illustrated (Figure 2), each gene $(N)$ in a genotype must be assigned the number of other genes $(K)$ which influence it. A complete assignment of each gene $(N)$ and the specified number of genes $(K)$ which influence each $N$ is illustrated in (Figure 3a and 3b).

The Fitness Contribution of each Allele at each Gene in the Context of the $K$ other Genes which Impinge Upon that Gene must be Specified.

Kauffman (1995) defines the fitness contribution of each gene $(N)$ to depend on the gene’s own allele state, plus the allele states of the $K$ other genes that affect that gene $(N)$. Therefore, for each gene $(N)$ connecting to $K$ other genes with different allele states, a different fitness value must be assigned. Therefore the total combinations of alleles
determine the number of generated fitness values required, calculated by \( A^{K+1} \). Kauffman (1993; 1995) assigns a random fitness value drawn from the uniform distribution between 0.0 and 1.0 to each allele (Figure 3c).

**For each Gene, its Fitness Contribution is Generated by Random Assignment of the \( 2^{K+1} \) Allele Combinations of the \( K + 1 \) Genes which Impinge Upon it.**

Kauffman (1993) states that the fitness value of \( (w_i) \), is dependent on its own allele state and the number of other gene \( (K) \) allele states which it is dependent on. In which all of the possible fitness combinations and values having been assigned (Figure 3c). A fitness value following these rules is then assigned to every gene \( (N) \) within a genotype (Figure 3d), for all the possible genotypes combinations, extracting their relevant fitness values form (Figure 3c).

**The Fitness of An Entire Genotype is the Average of the Contribution of all the Loci/Genes within it.**

Having assigned a fitness value to each gene \( (N) \) within each genotype, each genotypes final fitness value can be calculated \( (W) \). The final fitness values for each genotype are calculated as a mean average of the fitness contribution of all the genes within the specified genotype see Figure 3d, (Kauffman, 1993).
Investigating Kauffman’s NK Model for Agent-Based Modelling

**Figure 3.** Calculating Fitness, when $N=3$, $K=2$, $A=2$. (a), (b) Assignment of $K=2$ gene dependences to each gene ($N$). (c) Assignment of random fitness between 0.0 and 1.0 to each allele combination ($A^{K+1}$). (d) Assignment of each genes ($n_i$) fitness value ($w_i$), and each genotypes average overall fitness ($W$).

### 2.3.3 Effects on the Landscape of Varying $K$

Gavrilets (2004) suggests a key feature to Kauffman’s NK Model is the ability to alter the degree of correlation by changing a single parameter $K$, thus changing the landscape from smooth to rugged. Kauffman (1993) defines the terms rugged and smooth landscape as the correlation structure of the fitness landscape, dependent on how similar the fitness values of one-mutant neighbours in the space are. A smooth landscape is described as one in which neighbouring points have nearly the same fitness value, knowing the fitness value of one point carries a lot of information about the fitness value of neighbouring points (Kauffman, 1993). At the opposite extreme Kauffman (1993) describes a maximally rugged landscape as one in which the fitness values are entirely uncorrelated, knowing the fitness value at one point would therefore carry no information about the fitness of neighbouring points.
It is not possible to draw an NK landscape, similar to Wright’s (1932) fitness landscape concept, Kauffman (1993) describes the NK landscape to be $N$ dimensional. Applying Kauffman’s (1993) theory however of how the landscapes can evolve from a single peaked correlated landscape, when $K = 0$, to a multi peaked rugged landscape with no correlation, $(K = N - 1)$ we can draw a theoretical representation of varying $K$ (Figure 4) (Lazer et al., forthcoming; McCarthy, 2002).

![Figure 4. Impression of NK landscape when varying $K$.](image)

**Figure 4.** Impression of NK landscape when varying $K$. (a) A simple landscape with high correlation similar to $(K = 0)$. (b) A rugged landscape with several local peaks, similar to $(0 < K < N -1)$. (c) A completely random rugged landscape, similar to $(K = N -1)$.

**Single-peaked Correlated Fitness Landscape ($K = 0$)**

Kauffman (1993) examines the effect on the landscape of $K = 0$, with the further condition that each gene only has two alleles. Kauffman (1993) demonstrates how the structure of this fitness landscape has a single global optimal genotype (Figure 4a). Kauffman (1993) argues that all other genotypes are suboptimal and can climb to the global optimal via fitter neighbours, and all one-mutant neighbours have nearly the same fitness.

At each gene by chance either allele 0 or allele 1 will make the higher fitness contribution. Therefore, within the set of all possible genotype there will be one which contains the fitter allele for each gene, thus the global optimum genotype (Kauffman, 1993). In addition to this any other genotype, which will obviously have a lower fitness value, can sequentially be changed to the global optimum. As each suboptimal genotype lies on a connected pathway, by altering each gene which contains the allele with the weaker fitness value to the fitter allele with the stronger fitness value, the global optimum can be reached (Kauffman, 1993). Kauffman (1993) therefore states that there are no optima other than the single global optimum, which all other genotypes can climb to, as illustrated (Figure 4a).
Multi-peaked Fully Random Fitness Landscapes ($K < N-1$)

The largest possible value for $K$ is $N-1$, and this is when the fitness landscape at its highest possible ruggedness (Figure 4c). At this limit, each gene is affected by all the remaining genes in its genotype. Referring back to Kauffman’s (1993) fitness calculation (Figure 3), this means that every gene in each genotype will have a different random fitness value. In other words, as Kauffman (1993) states, the fitness value of one genotype gives no information about the fitness value of its one-mutant neighbours and therefore the fitness landscape is entirely uncorrelated.

In such extremely rugged fitness landscapes Kauffman (1993) shows a number of quite surprising features which are true on completely uncorrelated landscape, in particular he suggests:

1. The number of local fitness optima is extremely large.
2. The expected fraction of fitter one-mutant variants dwindles by $\frac{1}{2}$ on each improvement step.
3. The lengths of adaptive walks to optima are very short and increase only as a logarithmic function of $N$.
4. The number of searches tried to reach an optimum is proportional to the dimensionality of the space.
5. The ratio of accepted to tried searches scale is $\ln N/N$ for the two-allele case.
6. Any genotype can only climb to a small fraction of the local optima.
7. Only a small fraction of genotypes can climb to any given optimum.
8. As the number of genes within a genotype ($N$) increase, the local optima fall towards the mean fitness of the space of genotypes.

Kauffman (1993) suggests that point eight is perhaps the most important implication of such a landscape, which can also be seen in a large class of rugged but correlated landscapes. Kauffman (1993) describes this feature as a further kind of complexity catastrophe, pointing to a fundamental restraint on adaptive selection. Kauffman (1993) describes it as a consequence of attempting to optimise systems by increasing many conflicting constraints among the components, with causes accessible optima to become ever poorer and fitness peaks to dwindle in height.

Having studied $K$ values at both extremes and how it affects the landscape, with $K = 0$ corresponding to fully correlated smooth landscapes and $K = N-1$ corresponding to fully uncorrelated rugged landscapes. It is evident, as $K$ increases the landscape changes from
smooth, through a family of increasingly rugged landscapes, until it reaches a fully uncorrelated landscape \( K = N - 1 \) (Kauffman, 1993).

### 2.3.4 Travelling the Landscape

Kauffman (1993) describes travelling the landscape as an uphill climb, selecting a starting point at random and climbing to the highest fitness attainable. To reach this a method to travel the landscape needs to be devised. Kauffman (1993) discusses two alternative methods of travelling a NK landscape by “adaptive walks” and “long jumps”.

**Adaptive Walks**

An adaptive walk is the process of travelling the landscape via mutant neighbours. A mutant neighbour simply being a genotype with a specific amount of different gene allele states compared to the selected genotype (Kauffman, 1993; 1995). For example a one mutant neighbour of a genotype, would be any genotype which has the same gene structure as the selected genotype apart from one gene which has changed to an alternative allele state. Kauffman (1995) describes a simple adaptive walk as selecting a genotype and then randomly selecting one of its mutant neighbours, if the variant is fitter, moving to that genotype. If the mutant neighbour is not fitter, it will not move there, instead choosing another random mutant neighbour and moving to this genotype if it is fitter. This process repeats until a genotype is found which is fitter than all its mutant neighbours, any such genotype is a local optimum in the genotype space (Figure 5) (Kauffman, 1993). In the NK Model Kauffman (1993) only considers one-mutant neighbour walks across the landscape, hence for \( K = 0 \), the one-mutant neighbours are highly correlated as they only differ by one gene alleles fitness value, as \( K \) increases however the correlation decreases.

![Fitness Landscape Diagram](image)

**Figure 5.** Fitness landscape on the three dimensional Boolean cube, corresponding to the fitness values of the eight genotypes in (Figure 3d). Note that more than one local optimum exists.
Long Jumps

Kauffman (1993; 1995) describes a “long jump” as simultaneously making a large number of mutations that alter many genes allele state in a genotype at once. Thus rather than slowly stepping across the fitness landscape, similar to a one-mutant adaptive walk, a long jump across the fitness landscape occurs. Kauffman (1995) describes the purpose of a long jump on a rugged but still highly correlated landscape ($N = 1000, K = 5$), by changing 500 of the 1000 allele states, effectively leaping halfway across the landscape and potentially beyond the correlation length of the landscape. Therefore the fitness value found from performing the long jump would be totally random with respect to the fitness value of the genotype where the jump was initiated.

Kauffman (1995) describes a very simple law which governs a long-jump adaptation, with the results exactly mimicking adaptive walks via fitter single-mutant variants on random landscapes. Every time a jump occurs and a fitter variant is found, the expected number of tries to find an even better variant doubles.

The result of such a long jump is an initial rapid increase in fitness, followed by a very slow increase which strongly suggests an exponential slowing (Kauffman, 1993; 1995). Another important feature Kauffman (1995) highlights is as $N$ increases, enlarging the space of possibilities, long jump adaptation attains ever poorer results after the same number of tries. Kauffman (1995) describes this as another form of complexity catastrophe, as the number of genes increases, long jump adaptations become less and less successful.

Apart from the two key methods of travelling an NK landscape Kauffman (1993) also briefly discusses three different move algorithms; “greedy”, “fitter” and “random”. The greedy move specifies adaptive steps occurred via the fittest one-mutant variant, this corresponding to a greedy gradient ascent where only the fittest one-mutant neighbours are ever selected (Kauffman, 1993). In a fitter move procedure, one of the fitter one-mutant variants, if any exist, is chosen at random on each iteration. The random move procedure is performed by sampling a random one-mutant variant and if it is fitter, the adaptive process steps to that variant (Kauffman, 1993).

It is important to note that although Kauffman (1993) considers these three different move algorithms, he does only for an adapted version of the NK Model, customised to study the concept of “The Cambrian Explosion and Premian Quiescence” in evolutionary biology. Kauffman (1993) also demonstrates how these three move algorithms also all produce similar results, thus although it is worth considering these different move algorithms in this investigation they are not a key part to the original NK Model.

Rivkin (2000) highlights a fundamental issue with Kauffman’s NK Model, with the time it takes to explore the landscape. Performing an exhaustive search of an NK landscape for low
values of $N$ is essentially no problem. However as $N$ increases, the time taken for example, to perform a complete search of the landscape dramatically increases (Rivkin, 2000). If an assumption $A = 2$ and it takes one second to explore each genotype for low values of $N$ this will be extremely quick, (if $N = 5$, $2^N = 32$ genotypes, therefore the total search would take 32 seconds). As $N$ increases however ($N = 30$, $2^N$ possible genotypes, therefore the total search would take 34 years) the time taken dramatically increases. Therefore a key point to consider with Kauffman’s NK model is a full search of the landscape is only practical at low levels of $N$.

### 2.3.5 Application of NK Model

Apart from the NK Model’s original purpose of modelling genetic interaction, it has more recently been used in the analysis of problems outside of the biological framework. The NK Model has particularly become popular with business analysts as an approach for visually mapping the strategic options an organisation could pursue (McCarthy, 2002). This investigation proceeds to review several of these alternative applications for the NK Model, studying the NK Model’s generic capabilities and any extensions applied.

#### Manufacturing Fitness

McCarthy (2002) reviews Kauffman’s original definition of the NK model and explores how this theory relates to competitiveness and strategy, proposing a definition and model of manufacturing fitness. McCarthy (2002) takes Kauffman’s (1993) evolutionary biological notations for the NK Model and applies them to a manufacturing strategy context. McCarthy (2002) defines $N$ to be equal to the number of organisational parts, such as the facility location or the approach to quality. $K$ represents the amount of interconnectedness among the organisational parts, creating trade-offs or accumulative dependencies between capabilities such as quality and productivity. Finally $A$ represents the number of possible states an organisational part may have, for instance the quality control capability could have four states; inspection, quality control, quality assurance and total quality management.

Although McCarthy (2002) does not display any numerical results from his manufacturing adaptation of the NK Model, he provides a good illustration of how the NK Model can be applied in a different context other than evolutionary biology. McCarthy (2002) also suggests that evolutionary theory does not just apply to biological organisms. If an entity (technological, social or economical) evolves then systems research provides a framework to understand the evolutionary process. Manufacturing organisations like many others are complex adaptive systems which evolve over time. They do not just exist in equilibrium or chaos, but rather as a periodic order of evolutionary progress (McCarthy, 2002). McCarthy states that this periodic order can therefore be represented as a landscape of different configurations, with each point containing their own defining strategies and capabilities.
Investigating Kauffman’s NK Model for Agent-Based Modelling

**Situated Learning Theory**

Yuan *et al.*, (2004) applies Kauffman’s NK Model to situated learning theory, studying how interaction within groups alters performance. For this model Yuan *et al.*, (2004) defines $N$ as the number of people forming a group, while $K$ measures the number of communication linkages among people.

Yuan *et al.*, (2004) implements an agent based version of the NK Model in the context described above, with various values of $N$ and $K$. Although the results are similar to Kauffman’s (1993) original results, this is to be expected, Yuan *et al.*, (2004) discusses the implication of the results in the applied context and how they can be used. Yuan’s *et al.*, (2004) results of group learning illustrate an initial increase in group performance as the number of communication linkages among people increases, however as the number continues to increase the performance level changes direction and decreases. Kauffman (1993) labelled this effect in the biological context as the complexity catastrophe. Yuan *et al.*, (2004) demonstrate how this effect is also applicable in group performance, with too much communication linkages within a group causing a negative effect, thus being the theoretical result expected.

Yuan *et al.*, (2004) warns that the NK Model may not be able to achieve a high level of accuracy if it is made to simple and general, also suggesting several possible extensions to improve the NK Model. After the NK Model configuration has been setup at the initial stages of the simulation, it remains unchanged throughout the remaining duration of the simulation. Yuan *et al.*, (2004) suggest this can be a constraint in some research, and that through allowing the manipulation of $N$ and $K$ during the simulation a better exploration on the interaction of $N$ and $K$ could yet be discovered. Kauffman’s NK Model also only allows one network to dominate, Yuan *et al.*, (2004) argue that other networks can also have an influence and it would be interesting to be able to simulate two different networks at once and study how they interact and the affect they have on each other. Kauffman (1993) suggests an extension to the NK Model, which he labels the “C” factor which controls the ties between genotypes in different genotype spaces. Yuan *et al.*, (2004) discuss the advantages of this addition, distinguishing the NK Model from other similar models which focus on a specific group, enabling additional groups or environmental entities which may affect a group to be modelled. Allowing more complex and realistic simulations to be performed.

**The Network Structure of Exploration and Exploitation**

Lazer *et al.*, (forthcoming) use the NK Model to explore how the structure of communication networks among actors can affect system-level performance. Lazer *et al.*, (forthcoming) appears to make basic use of the NK Model by travelling the landscape and recording the fitness values found on different network configurations. Lazer *et al.*, (forthcoming) however does highlight one important, on the variety of different network structures available. A
possible extension to the NK Model, could therefore be to add additional network structures to how \( K \) is assigned, an example of this would be to assign \( K \) in a linear network structure rather than randomly, or to its neighbours on both sides. This theory and network structures could also be used for searching the landscape rather than just randomly selecting a one-mutant neighbour for comparison.

### 2.3.6 Possible Extensions to The NK Model

After reviewing three alternative adaptations of the NK Model, a number of possible extensions have already been suggested. Lazer et al. (forthcoming) used the NK Model to explore the effectiveness of various networks which lead to the suggestion of incorporating these networks into the structure of the NK Model. While Yuan et al. (2004) suggest allowing the main parameters \( N \) and \( K \) to evolve while the simulation is running. Yuan et al. (2004) also discuss the possibility to enable the NK Model to simulate two networks at once and study how they interact and the affect they have on each other.

Originally Kauffman’s NK Model did not consider how systems and their strategies may interact with developments of other systems within the same environment, in both biological and business systems this is not realistic (McCarthy, 2002). Therefore Kauffman (1993) introduced an additional parameter to the NK Model, described as the “C” factor, thus creating a NKC Model. This “C” factor component considers an ecosystem with species, thus providing the concept that systems rarely exist in isolation. For example, if a company gains a new high profile investor, then its share prices are likely to raise, this consequently affecting its main competitors. McCarthy (2002) argues that a landscape is not static, continually changing with new innovative ideas, emerging all the time, with competitors entering and leaving the market. Kauffman’s (1993) additional \( C \) parameter would therefore be another interesting extension to the NK model. Although this addition has been discussed in several journals (McCarthy, 2002; Yuan et al., 2004) no implementation appear to have yet been tried.

When Kauffman (1993) first defines the number of alternative alleles \( A \) a gene may have, he restricts the value to two. Therefore potential extension to the model would be to allow the number of alternative alleles at each gene to be greater than two. Through applying this extension the landscape size could be dramatically increased even with only small values of \( N \). One possible research area with this extension would be to analyse the effect of the landscapes correlation with higher values of \( A \) as \( K \) also increases in size. An important consideration must however be made, as Rivkin (2000) illustrates with \( A = 2 \) and moderate values of \( N \), exploring the whole landscape can potentially take years. Therefore by increasing \( A \) the time taken to explore the whole landscape will take even longer.

Kauffman (1993) defines \( K \) to stand for the average number of genes which affect the fitness contribution of each gene. Kauffman however is not consistent in the usage of the average in his definition and shows now evidence of the potential of each gene to have different \( K \)
values. Apart from bringing to light the comprehensibility of Kauffman’s NK Model it also provides an alternative approach to assigning $K$. For example when $K = 2$, rather then every gene influencing two other genes, some genes may influence 3 or 4 genes while other genes only influencing 1 other gene, while still on average over the genotype each gene would be influenced by two other genes. This extension could potentially have an extremely interesting effect on the correlation of the landscape, causing a landscape even with a low value of $K$, to contain areas with highly correlated genes as well as genes with low correlation.

Kauffman (1993; 1995) discusses two alternative methods to travel the NK landscape; via adaptive walks and long jumps. In addition to this a further extension could be to combine the two concepts. An adaptive walk would be performed until a local optimum was reached. A long jump would then be attempted for a specified amount of tries, or until a higher fitness value was found. This process could then be repeated until the long jump is unable to find a fitter variant in the specified number of attempts. This alternative method to travel the landscape offers another technique to simulate real life situations. For example, consider two investors, one investor is a silent partner (long jump strategy), jumping from one investment to another, when they see potential to make a profit/increase fitness. While the other investor is more hands on, and will explore their current market position looking for improvement before moving elsewhere to invest (adaptive walk with long jump).

On the NK landscape Kauffman (1993) restricts an adaptive walk to a genotypes one-mutant neighbours. Altering the number of mutant neighbours a genotype can step to in one step is therefore a further possible extension. This extension would almost be a controlled long jump, with the maximum number of mutations in one step being specified. Thus as the number of mutant neighbours increases the expected correlation between the two fitness values is likely to dwindle.

### 2.3.7 Critique of NK Model

Possibly the main critique of Kauffman’s NK model is the comprehensibility of his definition. Once an understanding of Kauffman’s NK Model has been gained, the model itself appears to be a reasonably straight forward concept, which is not overly complex. However gaining an adequate understanding of the NK Model was extremely difficult. When Kauffman (1993) originally defines the NK Model he does so only in the biological context he designed the model for, assuming the reader has some background biological knowledge. Kauffman’s (1993) inadequate formal definition of the NK Model is poorly structured, introducing new components and additional features of the NK Model, hidden amongst discussions of biological evolution. Combined with the poor comprehensibility of Kauffman’s definition, it is extremely complex to gain the understanding required to take full advantage of the capabilities the NK Model can offer. Kauffman (1995) does however make a second attempt to define the NK Model, which still suffers from the majority of issues already highlighted albeit at a slightly more comprehensible level.
Another key criticism of Kauffman’s NK Model is what Yuan et al., (2004) describe as the lack of “humanisation”, suggesting that the NK Model does not always representing real life situations. Kauffman (1993) does discuss this himself, suggesting an additional component (C) which allows the model to also study the effects different groups in an environment have on each other, thus enabling the NK Model to cover the concept that systems rarely exist in isolation and allowing it to produce simulations which are a closer match to reality.

2.3.8 Conclusion

After a thorough exploration of the NK Model studying the underlying theory, this investigation has managed to take Kauffman’s (1993) original definition and redefine it a more comprehensible manner. Providing an understanding of the capabilities and mechanics required to reproduce a successful replica of Kauffman’s NK Model.

On reviewing Kauffman’s NK Model and the applications to which it has been applied, it was clear a thorough understanding of the NK Model does not always appear to be present. Some publications displaying a somewhat reiterated version of what Kauffman originally said, simply express their own interpretation and mathematical formula to explain the model further. This has lead to some papers becoming more confusing (e.g. McCarthy, 2002) fitness calculation) than helpful in gaining the correct understanding of Kauffman’s NK Model. The investigation therefore focused primary on Kauffman’s original work and definition of the NK Model to ensure that the correct interpretation of the model was achieved.

Once a comprehensive understanding of the NK Model has been achieved it is clear that the applications in which the model can be applied to are vast. This has been demonstrated with the investigation looking into just a few of the current applications, see (Section 2.3.5). With the additions of the possible extensions this investigation also suggests the possible applications are endless. Wickenberg et al. (2002) applies an agent based simulation to the software development process, being just one of the new areas in which the NK Model could be applied to.

After reviewing Kauffman’s NK Model and the potential capabilities available to it, this investigation tries to consider why it is not more popular than it is, and why it is only just starting to become popular within the business analyst community. The only suggestion the present analysis could make, was due to the poor comprehensibility of Kauffman’s original definition, thus making it a lot harder to fully understand and replicate the NK Model.

Finally the present study has found the actual number of NK Models available for use is limited, with some having been written in the potentially confusing biological context rather
than a generic version. Many analysts appear to be borrowing currently available versions and running their simulations, potentially without the knowledge of the underlying mechanics. Alternatively analysts are altering the available model to get it to meet their requirements (e.g. Yuan et al., 2004). Therefore it is clear that a well defined generic template version of the NK Model would be extremely useful to the research community and could potentially increase the popularity of the NK Model dramatically.

2.4 Agent Based Modelling

Agents are being used in an increasing number of applications, from fairly small systems such as e-mail filters to large and complex mission critical systems such as air traffic control (Jennings et al., 1998)

Software agent technology is a developing area of research. Application domains where agents are currently being applied or research is being carried out include; workflow management, telecommunications network management, air traffic control, business processing re-engineering, data mining and much more. (Nwana, 1996 cited by Jennings et al., 1998)

Agent based modelling is most commonly used in the simulation of social processes, it has however also been used in other areas, such as biology, psychology and traffic vehicle simulations. (Gulyas, 2005)

2.4.1 What Are Agents?

‘An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives’ (Wooldridge et al., 1995).

Gulyas (2005) states that an agent based model is made up of a set of interacting agents embedded in a shared normally active environment.

2.4.2 Agent Toolkits

A multitude of agent toolkits exist for building and running agents, using various different programming languages. As well as differing in programming languages, toolkits tend to be designed for different purposes, therefore it is important the investigation studies a variety of toolkits in order to select the correct toolkit.

- JADE
Investigating Kauffman’s NK Model for Agent-Based Modelling

- Mason
- Netlogo
- Repast

2.4.3 JADE

JADE (Java Agent Development Framework) is a software environment used in building agent systems for management of network information resources in compliance with FIPA specification. (Rafael, 2005)

JADE is one of the most powerful agent platforms available, due to its multitude of features. Therefore it is commonly used for developing heavy weight agent systems. Therefore JADE is not really appropriate to use for this project as Kauffman’s NK model should not need such a complex toolkit.

2.4.4 Mason

Luke et al. (2004) describe Mason as a quick, easily to extend discrete-event multi-agent simulation toolkit in Java. Mason like Repast was designed to serve as a basis for a wide range of agent simulations.

Mason also distinguishes between model and visualisation, allowing models can be detached or attached to visualises dynamically. Mason also has a basic architectural design. (Luke et al. 2004)

MASON similarly to Repast also offers a wide range of extensions and tutorials to help get you started. Essentially however, MASON is reported to be 1.5x faster than Repast making one of the quicker agent based toolkits. (Luke et al. 2004)

2.4.5 Netlogo

Netlogo is particularly well suited for modelling complex systems developing over time. The language is Logo dialect extended to support agents and concurrency. It has a simple language structure, fully programmable and enables you to view your model in either 2D or 3D.
Investigating Kauffman's NK Model for Agent-Based Modelling

2.4.6 Repast

Repast is a commonly used open-source agent-based modelling and simulation toolkit. Repast is currently available on three platforms; Java, Microsoft .NET framework and repast for Python scripting, each containing the same core features. North et al. (2005)

Repast is more of a lightweight development framework compared to JADE. Agents are just normal java objects that the modeller is made to think of as agents. Repast is a well established multipurpose toolkit which has been used on several top end projects.

2.4.7 Conclusion

JADE appears to be the biggest toolkit in terms of complexity, however this is not essentially an advantage, as the NK model should not need anything that complex. We especially don’t want to over complicate the simulation as this is what we are trying to get away from in previous versions of the NK model.

Due to Netlogo being part of the logo dialect and not appearing to have as much support as some of the other more well developed languages we will not be using NetLogo for the project. (Wilensky, 1998) As well as having to learn a new language or part of one there does not seem to be such a well established community for software support.

Essentially the choice is between Repast and MASON, both appear adequate for the job and either would be suitable to use. However MASON is reportedly faster than Repast which could be a key advantage when creating an NK model with a large landscape. MASON also separates the model and visualisation allowing models to be detached or attached to visualise dynamically (Luck et al. 2004). This is essentially another key feature which could be useful in the development of an NK Model, ensuring the environment is not over complicated with visualisation code, thus allowing the code to be easier to read and therefore more adaptable as people require extensions to be added to the model.
Chapter 3

Requirements

The investigation has led to the point where an understanding of Kauffman’s NK Model has been established and a set of formal requirement for the model can be produced. These requirements will be used to define exactly what is required of the NK Model and provide guidance in the design and implement of the model.

It is important to note, a key objective of this project is to create a generic version of the NK Model, to help improve the understanding and enable it to be easily applied to various situations. Following on from Kauffman’s (1993) vague and brief generic definition of the key parameters in the NK Model, for the duration of this project they will be defined as follows:

- \( N \) – The number of parts in a system.
- \( K \) – The number of other parts which influence a specified part.
- \( A \) – The number of alternative types for each part

3.1 Functional Requirements

With the investigation performed into Kauffman’s NK Model (see Chapter 2) and Kauffman’s (1993; 1995) formal definition, a set of functional requirement for the NK Model can be produced.

1. **Build The Landscape**
   
   1.1. Set number of parts in a system \( (N) \).
Investigating Kauffman's NK Model for Agent-Based Modelling

Allows the user to specify the number of parts each system is made up of.

1.2. Set the number of other parts which \((K)\) which influence each part \((N)\).

Allows the user to specify the number of dependences each part \((N)\) has. (Must be between 0 and \(K - 1\)).

1.3. Set the number of alternative parts \((A)\) each part \((N)\) can have.

Allows the user to specify the number of alternative parts each part can have (default is 2).

1.4. Calculate all combinations of parts possible within a system.

Calculates the number of unique possible systems, where each system is different by at least one alternative part type.

1.5. Create landscape.

Creates the landscape by assigning each system a unique combination of parts, until every combination has been assigned to a system.

2. **Calculate Fitness**

2.1. Generate the fitness values required.

Create a random fitness value between the uniform distribution of 0.0 and 1.0 for all possible alternative part combinations in relation to \(K\) (i.e. \(A^{K+1}\)).

2.2. Assign each part its fitness value.

Assigns a fitness value to each part, with the fitness value assigned being dependent on each parts \(K\) other dependent parts state.

2.3. Calculate fitness value for each system.

Adds the fitness value together, for each part within the system to get the average fitness value for the overall system.

3. **Travel The Landscape**

3.1. Perform an adaptive walk

Travels the landscape via randomly selecting one part variant systems and comparing fitness values.

3.1.1. Select a system by random.

3.1.2. Select another system by random which only has one part different to the originally selected system.

3.1.3. Compare fitness values.

3.1.4. Set position at system with highest fitness value.

3.2. Perform a long jump

Jumps to a random system on the landscape comparing the system it is located at with another random system, moving to the new system if a higher fitness value is found. Doubling the amount of tries until stopping on every fitter system found.

3.2.1. Select a system by random.
3.2.2. Select another system by random
3.2.3. Compare fitness values.
3.2.4. Set position at system with highest fitness value.

3.3. Perform adaptive walk and long jump combined
Combine the two original methods of travel the landscape (adaptive walk and long jump) into one hybrid method.

3.3.1. Perform adaptive walk.
3.3.2. Perform long jump.

3.1.1 Additional Requirement
It is important to note that several additional functional requirements have been added which were not originally defined by Kauffman. After the investigation into Kauffman’s NK Model (see Chapter 2) the decision was made to add these additional components, listed below, due to the additional functionality they enabled the NK Model to offer.

Requirement 1.3, has been altered slightly from Kauffman’s original specification, where he restricts the number of alternative parts (A) each part can have to two, where in this specification “A” is set by default to two, but can be altered. Additionally requirement 3.3, has been added as an additional method to travel the landscape, this extension was originally discussed in chapter 2, and has been included into the main specification of the NK Model due to its similarity to real life scenarios.

3.2 Non-Functional Requirements
From the investigation into Kauffman’s NK Model (see Chapter 2) and Kauffman’s (1993; 1995) formal definition, a set of non-functional requirement were compiled.

- The model should be produced for a generic context, to allow easy manipulation and understanding.
- This model should be created as a template, to be used by others and adapted to their specific requirements.
- The model should be able to handle large values of N and K.
- The model should be capable of creating and storing large a number of part combinations (i.e. 2^24).
- Efficient in searching the landscape to maximise productivity and results attainable.
Simulations with a moderate value of $N$ (up to $N = 24$) should run in within a maximum of two days maximum).

- Results should be output in their raw format so they can be easily presented as required by the user.
- The model’s core fundamental mechanics must be an accurate replica of Kauffman’s original NK Model.
- The results produced must show a similarity to Kauffman’s NK Model’s original results.

### 3.3 Conclusion

A formal set of requirements have now been produced from the earlier investigation into Kauffman’s NK Model. These requirements will be a vital aid in the design and implementation of the NK Model to ensure an accurate representation is created and that it conforms to the key requirement suggested in the investigation.

When formulating the requirement it became apparent there was a conflict between several of the requirements. A main requirement of the model is to run quickly to maximise the productivity and results attainable, however another crucial requirement is for the model to be able to create and store a large number of part combinations. Therefore by meeting the second requirement and designing a model which can store a large number of combinations may potentially cause the model to slow down. This conflict will have to be addressed further in the design section and a compromise may have to be made.
Chapter 4

Design

Essentially the architectural structure of the NK Model should be fairly straight forward, as the model itself does not appear to be too complex. It is the way in which the model was originally defined which complicated the understanding and therefore making it appear more difficult than actually was.

This investigation now studies the essential design structure of the NK Model, highlighting the key design decisions made along the way, referring back to the requirements for guidance on any critical design decisions.

4.1 High Level Architecture

![Diagram of high level architecture]

**Figure 6.** High level class architecture design.
The Model class should fundamentally be the core class where everything is initiated and ended. This class should be used to store all the main parameters required to generate the NK Model, as well as the results it produces, which the other classes can inherit.

The NK_Space_Gen class should be responsible for creating all the possible system combinations and assigning the correct fitness values. This class could potentially be included within the Model class. However as each unique system combination and fitness calculation is a substantial part to the overall model and none of which is required by the Agent class. To enable a better comprehensibility and structure to the code, it will be placed in its own class. Through placing this core component of the model into its own class it also meets the requirements of creating the model as a template for additional components, as the model is potentially made more adaptable.

Finally the Agent class should be responsible for travelling the system. An instance of an agent should be made, which can collect its instructions from the Model class on how to travel the landscape. The agent should then travel the landscape returning its results to the Model class.

### 4.2 Model Class

As discussed above this class should be the core to the NK Models structure and operation. As this class is used primarily to set the fundamental parameter of the model it does not have many exciting features apart from the arbitrary method calls to initiate and end the model. The Model class however should have one other vital role within the structure of the NK Model. This role is to output the results the NK Model generates, and is something which needs further investigation.

#### 4.2.1 Outputting Results Generated

Essentially this is the most important feature to the NK Model and is therefore worth further discussion and consideration. If the results are displayed inadequately even the best NK Model in the world with hundreds of additional features could, fundamentally be useless. Therefore a study of two alternative methods of outputting the results are considered and a conclusion is made on the most appropriate design to implement.

**Graphical Representation of Results**

Potentially with the use of an agent based simulation results could be displayed to the screen in a visual representation as the model is running, thus potentially almost showing the results in real time. Therefore charts could automatically be produced plotting the results as the simulation runs, providing the user with a quick method to view and analyse the results.
produced from the NK Model. With reference to the formal requirements (see Chapter 3) however this could dramatically restrict the models usability and adaptability. With so many potential possibilities for the NK Model, as highlighted in Chapter 2, and the possible methods in which the results are displayed, it would be almost impossible and extremely complex to automatically generate results for every alternative method. Therefore if the results were not displayed in the correct form for the user the NK Model could potentially be deemed useless. This would also not comply with our non-functional requirements of the model to produce the results in the raw format so the user has the ability to use them as they see fit as well as potentially stopping the model from being used as an effective template for users to customise for their own use.

**Write Results to File**

An alternative approach which would make the results produced from the model’s simulation much easier to manipulate and apply is a user specified manner would be to write the results to file. This approach would allow the user to display and apply the results as they desired and therefore conforming to the requirements previously specified in Chapter 3.

A crucial consideration however is how the results should be displayed in the file. One method would be to display the results in an aesthetically pleasing structured format which from initial view is easy to read. However a consideration needs to be made for how the results can easily be manipulated and potentially imported into other applications for further analysis once the simulation has complete. Therefore an alternative method would be to display the results in a format which would be compatible with third party applications to import the results for further manipulation (Figure 7). As illustrated in Figure 7, the results could be labelled and then listed along the sample line with a comma (",")) dividing them, to represent the start of a new column, thus allowing the data to be easily imported into third party applications for further analysis, such as Microsoft Excel.

![Figure 7. Design of results output to file](image-url)
Conclusion

It is clear that writing the results of the models to file in a format in which third party applications are easily able to import the data is the best method to meet the requirements of the model. However this is possibly the worst method for the user to be able to easily view the results the simulation without having to format them in any way. Therefore, as well as the results getting written to a file, for easily manipulation the results should also be displayed graphically when the simulation is running, through the assistance of the modelling tool kit. Possible displays could include:

- A chart displaying the current fitness currently found over the time it has taken to find it.
- A chart displaying the current fitness currently found over the number of steps it has taken to find it.
- A chart displaying the end fitness value reached after of each run and the mean fitness calculation
- A chart displaying fitness values of a randomly selected systems neighbours
- A 3D representation of the landscape.

4.3 NK_Space_Gen Class

Essentially the NK_Space_Gen class should be responsible for building the landscape and assigning an appropriate fitness value to each system once the model structure has been specified in the Model class. Within this class there are three main functions which are extremely important to the successful operation of the model:

- Creating the landscape (all the unique system combinations).
- Storing each combination.
- Assigning fitness values.

4.3.1 Creating the Landscape

From the earlier investigation into Kauffman’s NK Model (see Chapter 2, section 2.3) each system contains a specified number of parts ($N$) with each part containing a number of alternative types ($A$), usually represented by 0 or 1, when there are only two alternative types available. Therefore the number of unique systems which can be created from a specified number of parts is calculated by $A^N$. Once the number of possible combinations has been calculated, each combination needs to be actually created, thus building the landscape. Here three alternative methods are compared to ensure the correct design decision is made:
Investigating Kauffman’s NK Model for Agent-Based Modelling

- Binary Assignment
- Random Assignment
- Systematic Assignment

**Binary Assignment**

The first possible option is to use binary assignment, allowing 0 to represent one alternative part and 1 to represent another, thus each combination up to the value of \( A^N \), would be represented by its binary equivalent (Listing 1).

**Listing 1** Pseudocode for binary assignment design.

```plaintext
Calculate possible combinations
While combinations created is less than the amount of possible combinations
    Convert combinations created value to binary
    Store the binary value
    Increase combinations created value by one
End while
```

This method has several main advantages, primarily it is a reasonably easy and quick process to perform, thus meeting the requirements of ensuring the model is efficient. This method also keeps the same order of combinations for each system which Kauffman uses, potentially avoiding any confusion to a user who is familiar with Kauffman’s representation and providing familiarity.

The main drawback to this method of generating the combinations is “\( A \)” could only ever equal two. This would be ok for Kauffman’s version of the NK Model where he restricts the value of \( A \) to be two. However a key objective of this project is to try and make a more customisable version of the NK Model, which can be used as a template for users to customise as required. Using this method would essentially stop the model from meeting this requirement (set in Chapter 3), making it potentially a lot less useful for users. Another disadvantage of this method is \( N \) can potentially only be as big as the highest binary representation, thus causing this method to potentially fail another requirement, it being specified as capable of handling large values of \( N \).

**Random Assignment**

Random assignment of each part type within a system, checking the finished system combination has not already been generated is essentially a brute force method (Listing 2).
Investigating Kauffman's NK Model for Agent-Based Modelling

Listing 2 Pseudocode for random assignment design.

<table>
<thead>
<tr>
<th>Pseudocode for random assignment design.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify the number of alternative parts a part can have</td>
</tr>
<tr>
<td>Calculate possible combinations</td>
</tr>
<tr>
<td><strong>While</strong> combinations created is less than the amount of possible combinations</td>
</tr>
<tr>
<td><strong>While</strong> part is less than number of parts in a system</td>
</tr>
<tr>
<td>Randomly assign part a value between 0 and the number of alternative parts a part can have</td>
</tr>
<tr>
<td>Increase part value by one</td>
</tr>
<tr>
<td><strong>End while</strong></td>
</tr>
<tr>
<td><strong>If</strong> system part combination has not already been stored</td>
</tr>
<tr>
<td>Store combination</td>
</tr>
<tr>
<td>Increase combinations created by one</td>
</tr>
<tr>
<td><strong>End if</strong></td>
</tr>
<tr>
<td><strong>End while</strong></td>
</tr>
</tbody>
</table>

The only advantage of this method is it should eventually create all the possible combinations available independent of what values N or A are assigned. This method however is a very inefficient method to create all the possible combinations. As more combinations are stored it gets harder to find another unique combination and therefore it is likely the amount of duplicate combinations will increase thus slowing the model down, and making it unpredictable how long it will take to create the set of all possible combinations. Although this method meets the requirement to have larger numbers of A it fails to meet the efficiency requirement as this is an extremely slow method especially as N increases in size.

**Systematic Assignment**

This method reproduces the combinations produced from the binary method by finding a pattern to generate the combinations which could then potentially be expanded to create combinations for systems with more than two alternative parts. One pattern found is as the part number \((n_1, n_2, \ldots, n_N)\) increases the amount of combinations produced before it needs to change to an alternative part doubles from the amount of the previous part (Listing 3).
Listing 3 Pseudocode for systematic assignment design

```
Calculate possible combinations
While combinations created is less than the amount of possible combinations

Calculate number of combinations before each part changes
Part1 changes every 1 combination
Part2 changes every (part1 x 2)
Part3 changes every (part2 x 2)
Part4 changed every (part3 x 2)
Etc...

Store number of combinations made for each part.
Part1 part Count
Part2 part Count
Part3 part Count
Part4 part Count
Etc...

While part is less than number of parts in a system
  If number of combinations before part changes = 0
    Set part value to 1
    add one to part count
    when part count equals original number of combinations
    before each part changes, number of combinations before
    each part changes is reset to original value
  Else
    Set part to value to 0
    Minus 1 from number of combinations before part changes
  End if
End while
End while
```

This method has several advantages, one of which, as with the binary method the combinations can replicate the same order Kauffman produced his sequences making it more familiar with users who have previously studied Kauffman’s work. This method also has the extendibility create combinations with higher values of (4) this making it more adaptable and a good template for which the model can be enhanced later.

The main disadvantage of this method is the time taken to create a combination, although the process should not take to long as N increases in size the performance will decrease also. This being a potential problem in meeting the efficiency requirement specified in Chapter 3.
Conclusion

After having considered three alternative methods to build the landscape and create every combination of systems from the number parts specified, the method which is going to be implemented is the systematic assignment. Although this method does potentially have a lack of efficiency over the binary assignment method, it offers the model better extendibility for the future. Therefore a design decision was made to have the slight loss in efficiency but gain in extendibility.

4.3.2 Storing each Combination

As each system combination is created an appropriate method of storing it must be devised in which it can be accessed at any point when the simulation is running. Two alternative methods are considered for storing the combinations, in an array or a file.

Array Storage

Essentially an array is the obvious choice, allowing each combination to be easily stored and access quickly when required. However an essential problem occurs for this method of storage as the total number of combinations increases beyond an arrays capacity. Therefore if this method is used essentially only low values of \( N \) could be handled by the model, thus failing one of the main requirements set out in Chapter 3. This would be a major limitation also stopping the model from being very adaptable and possibly even useful in some cases where higher values of \( N \) are required to get a better view of the simulation results.

File Storage

An alternative method is to store the combinations in a file. The main disadvantage of this is it would lead to a slight loss of performance compared to the array method, in the time it takes for storage and retrieval. This performance loss would also be likely to increase as the file increases in size (i.e. more combinations are stored within it). To minimise the loss in performance however the data could be stored in the machines format, and not human in a readable format. This should make it slightly quicker to store and access the file, however it does mean the file could not be opened and read by the user, although there should be no need to for the user to do this.

To facilitate an increase in the access time in the file storage method, would be to store every combination in a different file, naming the file after the combination number so it could easily be accessed when required. This could possibly slow down the initial storage of the combinations slightly, however the access time, would not deteriorate as the number of possible combinations increased, thus ensuring the actual simulation, after the setup of the all the combinations would run more efficiently.
Investigating Kauffman's NK Model for Agent-Based Modelling

The main advantage with the file storage method is the amount of combinations which could potentially be stored is endless. This ensures the requirements are met in regard to ensuring large values of $N$ and $K$ are accepted and the model is adaptable.

**Conclusion**

Comparing the two alternative methods highlights a conflict between conforming to the requirement of efficiency and speed (i.e. using the array method), or conforming to the requirements of adaptability and enabling the model to support larger values of $N$ and $K$ (file store method). Essentially a compromise has to be made, and on review the file store method as individual files has been chosen. This decision was based on the fact that the array method would simply limit the model to much and could make it useless to many users who want higher $N$ values. The decision was made to store each combination as an individual file as the actually key simulation will start once the landscape has been created (i.e. all combinations stored) therefore this provided the most efficient way to retrieve a combination.

**4.3.3 Assigning fitness values.**

This component of the model contains two key parts. First the random generation of fitness values, which from the earlier investigation into Kauffman’s NK Model (see Chapter 2, section 2.3) defines a fitness value to be randomly generated from the uniform distribution of 0.0 and 1.0. Therefore this should be a reasonably basic concept which could be carried out by the default programming language random number generator. However a possible further investigation into the efficiency and randomness of the algorithm should possibly be considered once the development language and toolkit have been decided upon.

Secondly the storage of the fitness values, after the values have initially been generated they will need to be stored in a file as there is potentially the chance that of having the same number of fitness values as the number of combinations (i.e. $K = N - 1$). Thus the same storage method can be adopted to store the fitness values as was adopted to store the system combinations, however these should be stored in a single file. As these values are only required temporarily while each system combination is assigned its appropriate fitness value. Therefore when a combination is assigned a fitness value the fitness value could be permanently stored with the combination, thus increasing the efficiency of the system, as only one file would be read to retrieve both the combination and its fitness value.

**4.4 Agent Class**

The agent class will be used to travel the landscape. This means it will contain all the methods required to perform an adaptive walk, long jump and a hybrid of the two. The functionality and structure of these methods to travel the landscape have already been
formally defined by Kauffman (1993; 1995) and discussed again in the investigation into the NK Model (see chapter 2.) As these features are quite straightforward and restricted by Kauffman’s definition there is not much variation available. Therefore no further design discussion really needs to be considered on these methods, apart from implementing them as Kauffman defines them.

4.5 Conclusion

In this chapter several crucial design decisions have been made which affect both the performance and operational abilities of the NK Model. The design has highlighted several conflicts within the requirements where compromises had to be made. These conflicts usually appeared to be between the speed and functionality of the model and often a balance between the two had to be found in the design method. Essentially the balance of speed and functionality in the model is extremely important, if the model runs too slowly it could be deemed useless, simply taking too long to gather the results required. While at the same time if it hardly contains any functionality or the capability to extend the model then it could also be deemed useless. The next stage will be to implement the design decisions made to ensure they were the correct decisions and an appropriate NK Model is produced.
Chapter 5

Implementation and Testing

The implementation process examines the essential decisions made when creating the NK Model as well as providing the essential instructions on how to run the model. All the code for this implementation can be seen in Appendix B.

5.1 Basic Instructions to run the NK Model

There are various methods to compile a MASON program, the basic command line operations have been listed below. The recommended method for this project is to compile through eclipse as the has been used throughout the development. For further details please review document entitled:

*How to Set Up MASON in Eclipse* Prepared by: Steve Lytinen  
([http://condor.depaul.edu/~slytinen/](http://condor.depaul.edu/~slytinen/)) and Steve Railsback (Lang, Railsback & Associates)

It is also important to note as well as having MASON version 11 installed, the extra libraries are also required for the GUI Version. All these downloads can be found on the MASON Home Page:

5.1.1 Basic instructions to compile the NK Model from the command line

Compiling from the OS X / Linux Makefile

Ensure the CLASSPATH is set up right. Then:

```
cd mason; make 3d
```

...will compile all the files. For additional help type:

```
cd mason; make help
```

Running an App from the Command Line

Every GUI app class ends with the extension "WithUI.java". Assuming your CLASSPATH is set up properly, you can run it directly from the command line. For example:

// GUI Version
```
java sim.app.NK_Model.ModelWithUI
```  

or

// Command line version (parameter require being changed manually from the code)
```
java sim.app.NK_Model.Model
```

5.2 Agent Toolkit Selected

From the previous investigation into the various agent toolkits available (see Chapter 2, section 2.4) a decision has been made to use MASON as the agent toolkit to be used to implement the NK Model. The main reasons are listed below, (also discussed in Section 2.4):

- Based on java – a familiar programming langue with plenty of support and is cross platform compatible.
- Speed – MASON is know as one of the faster agent based simulation toolkits and this is essentially extremely import, to meet the requirements specified in Chapter 3.
- Tutorials – MASON offers several introductory tutorials which will be useful to help learn how to use the application
- Separate GUI – This allows the GUI to be separated from the rest of the program ensuring the code is easier to read and providing an independent command line version if required. Thus also meeting the adaptability requirements set out in Chapter 3.
Investigating Kauffman's NK Model for Agent-Based Modelling

- Mailing List – MASON does have a distribution list which could be used if help is required with implementation issues.

MASON also contains several additional features specific to MASON, several of these features have been highlighted below which could be extremely useful in the development of the NK Model.

5.2.1 MersenneTwiser Random Number Generator

Although java does have an inbuilt random number generator, the MersenneTwister random generator is supposedly over 1.5 times faster and generally produces better random values. These are two extremely important issues, first ensuring a random, as possible number can be produced from a computer algorithm, which is essentially not that easy. Secondly although 1.5 times faster may not sound that much faster, however the amount of random numbers required within this program is vast. (For all the fitness calculations and random selection) This should make the implemented model more efficient. Thus covering the requirement set in Chapter 3. A special version of MersenneTwiser is also available called MersenneTwiserFast which is even faster and could therefore increase efficiency even more.

5.2.2 Bag

Bag is an extensible object array with public access. It is significantly 3-4 times faster than an ArrayList or Vector, and could therefore be an extremely useful object for storing values, again meet the efficiency requirement set in Chapter 3. Bag also has another excellent feature, if an item is required to be deleted from a Bag, the object has a fast method which simply reduces the bag size by one and inserts the top value from the Bag into the deleted items space, thus saving a lot of time not having to resort or copy to a new Bag.

5.3 Main Features of Program

Below the key features of the implementation have been divided into its appropriate class and highlighted.

5.3.1 NKSpace_Gen Class

This class as described in the design process is responsible for creating and storing the fitness values and part combinations. One of the key additions to this class not specified in the design was the fast search method (Listing 4). This allows the combinations to be generated as access is required to them, for example if a start position is randomly selected of 1043, the 1043 sequence of numbers can be generated without the other combination first having to be created. This means in landscape with large values of $N$ where there are
Investigating Kauffman’s NK Model for Agent-Based Modelling

potentially millions of combinations only a fraction will actually have to be created, thus saving an massive amount of time and allowing more simulations to be performed. An estimated amount can actually be calculate using Kauffman’s predictions for how many steps each $N$ and $K$ combination need to make to find a local peak.

Listing 4 demonstrates how this method is carried out, using the count values discussed in the design stage and getting a remainder over 2, if equal to 0, 0 is placed in the NK part and if equal to 1, 1 is placed in the NK part.

**Listing 4** Fast Search – creating a specific combination and getting fitness value

```java
// used for fast search - calculates a combination from the NK Space requester
public void singleSpace(int nk) {
    NKSpace = new int[N];
    for (int col = 0; col < N; col++) {
        int divVal = 0;
        int remVal = 0;
        divVal = nk / CountValues[col];
        remVal = divVal % 2;
        NKSpace[col] = remVal;
    }
    FitnessCalc(nk);
    writeToFile(nk);
    for (int i = 0; i < N; i++) {
        System.out.print("[") + NKSpace[i] + "]");
    }
    System.out.println();
}
```

Essentially one major compromise was made in the implementation of this model, due to the conflict between speed and the number on combination able to be produced. Within this model several of MASON’s aditionaial features have been used, such as the Bag object and MersenneTwister random number generator. However these features just as the Bag object have not been made to use the “long” data type. Thus meaning to gain the additional speed and efficiency of the model the N value had to be limited. This decision was made as with the time restaints of the project it seemed impractical to produce a model which would taken longer to run and therefore less results would be able to be collect and the higher values of N would not be run anyway. The code however (see Appendix B) has been
implemented in such a way then if required the user could change the data types to long or even BigInt to get even larger values and then use a slower method then the Bag object to store numbers when required.

5.3.2 Model Class

The model class contains one of the smallest but hardest part of this implementation. This was to stop the agents schedule repeating steppable once a specific point had been reached in the simulation, i.e. a local peak was reached. The MASON toolkit allows the user to specify a steppable to stop after a specific number of steps or time but does not when a specific condition is met. Therefore it took over a week of implementation time to try and get the simulation to stop and not continue forever. Eventually this was achieved by actually creating a second steppable which checked the condition and could then call the stop method (see Appendix B). This however does seem like a slightly inefficient method to do it, rather than just calling the stop from the step method itself, which does nothing.

5.3.3 Agent Class

The agent class was essentially the core class which travelled the landscape. Most of this implementation is straight forward performing what was specified from the design and investigation into Kauffman’s NK Model (see chapter 2). One key feature used in the class though was the IntBag object, this was used primarily to store all the possible one mutant neighbours a combination could have. One would then be selected from the bag randomly and the fitness value would be compared, if it was not fitter the value could quickly be deleted from the bag with minimal system resources, as described in Section 5.2.2.

5.3.4 Model WithUI Class

MASON initialises all GUI parts of the simulation from a different class, in this model it is the ModelWithUI class which is called if the user want to run the simulation with a GUI, effectively once this class is called the underlying Model class to run the simulation is called from within the ModelWithUI class. The GUI version is a lot more user friendly and allows the user to specify the models parameters from the console control panel. A chart display is also available (Figure 8), showing the user the current value of the agent as it climbs a peak.

There is however one fundamental problem with this class, after a long effort of trying to get the GUI to repeat the same simulation a specified number of times the discovery was made that this is actually a limitation in MASON. After talking to the MASON developer I was advised that this was not possible unless I really hacked the program, and unfortunately due to project time constraints I was unable to do so. This is a good example of the key problems I encountered when developing in MASON, due to the lack of documentation and support I was unaware of this limitation until I had spent a lot of time, effectively wasted trying to get it to work.
5.4 Conclusion

The implementation has been an extremely difficult process with many major issues and problems along the way, as well as having to learn the agent based programming concept from scratch and how to use MASON. Overall an implementation of an NK Model has been successfully completed, although not as well developed as originally planned. However after all the time taken to gain an appropriate level of understanding of Kauffman’s model the expected time to run the simulations the implementation time was very short and with the problems encountered along the way this caused even more problems and setbacks.

5.5 TESTING

Essentially it is extremely hard to test a simulation if not impossible, depending on the simulation. You could essentially be creating the simulation to simulate something as you don’t know the result and want to find out. This investigation wants to try and replicate an experiment and compare its output with the original results to confirm if it is a correct representation. Therefore normal testing is not required. Instead the implementation will be tested through the experiments carried out in the next section. The results can then be compared with Kauffman’s original results and a comparison can be made to determine if the implemented version of the NK model is an accurate replication or not.
Chapter 6

Experiments

To ensure the implementation of the NK Model is correct several experiments need to be run and the results compared with Kauffman’s original NK Model results. Although this cannot guarantee an exact replica of the NK Model, if on comparison a similarity of the results and behaviour to Kauffman’s Model is found, the assumption can be made that the implementation of Kauffman’s NK Model is an accurate replication.

Once the implementation of Kauffman’s NK model has been verified as an accurate replication, further in depth experiments will be carried out into the structure and statistical properties of the landscape on a specific range of $N$ and $K$. Finally the model will be used to mimic a real situation using Kauffman’s original methods of travelling the landscape as well as the hybrid method which was added as an extension to the original NK Model.

6.1 Mean fitness of local optima (nearest-neighbour interactions)

Kauffman’s first experiment on the NK Model was to examine the statistical properties of the landscapes for different values of the fundamental parameters ($N$ and $K$). Therefore this experiment will replicate this to confirm the implemented NK Model is an accurate replica.

6.1.1 Method

In this experiment, the number of other parts ($K$) for each part is assigned to each part’s nearest neighbours. The simulations should be carried out for different random examples of the NK landscape for fixed $N$ and $K$ values (see Table 1). An adaptive walk should start from a randomly selected position on the landscape and then proceed like normal until a local optima is reached. This process should be repeated 100 times on different randomly generated landscapes for each value of $N$ and $K$. The mean average should then be calculated.
Investigating Kauffman’s NK Model for Agent-Based Modelling

for each combination of $N$ and $K$ after the 100 runs and placed in the results table similar to Table 1.

This experiment should be carried out twice, first with the original concept of travelling the landscape, where the whole landscape is created first and then an adaptive walk occurs. Secondly the revised quick method of travelling the landscape should be tested, where the combination and fitness value are assigned as the landscape is travelled. This will ensure that both methods results are similar to Kauffman’s original model and ensuring both approaches are an accurate replica of how the NK Model travels the landscape.

### 6.1.2 Hypothesis

A simple prediction can be made, for both the traditional method of travelling the landscape and the fast method. The results should be similar to Kauffman’s (1993) original results (Table 1). A slight degree of variation should be expected however as the landscapes are created with random numbers and therefore the landscape used will not be the same as Kauffman’s. The same pattern of the results however should be apparent with the highest optima being found for any $N$ value having a $K$ value of two or four and then slowing decreasing in value.

**Table 1** Kauffman’s Results of mean fitness of local optima (nearest-neighbour interactions). Taken from Kauffman (1993) Table 2.1.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.65</td>
<td>0.65</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td>2</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
<td>0.71</td>
</tr>
<tr>
<td>4</td>
<td>0.70</td>
<td>0.71</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>8</td>
<td>0.66</td>
<td>0.68</td>
<td>0.68</td>
<td>0.69</td>
<td>0.68</td>
</tr>
<tr>
<td>16</td>
<td>0.65</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>0.63</td>
<td>0.64</td>
<td>0.64</td>
<td>0.61</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td>0.60</td>
<td>0.61</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.58</td>
</tr>
</tbody>
</table>

When $K = N$, actual $K$ value is $K - 1$.

### 6.2 Mean fitness of local optima (random interactions)

Kauffman (1993) also performed the same experiment to examine the statistical properties of the landscapes for different values of the fundamental parameters ($N$ and $K$) when $K$ was assigned randomly to other parts within a parts system. Therefore this experiment will replicate this alternative version to confirm the implemented NK Model is also an accurate replica when the $K$ values are assigned randomly within a system.
Investigating Kauffman’s NK Model for Agent-Based Modelling

6.2.1 Method

In this experiment, the number of other parts \( K \) for each part is assigned randomly to other parts within the same system. The experiment should then be carried out in exactly the same way as the previous experiment (Section 6.1). With different random examples of the NK landscape for fixed \( N \) and \( K \) values (see Table 2) being generated. An adaptive walk should start from a randomly selected position on the landscape and then proceeds like normal until a local optima is reached. This process should be repeated 100 times on different randomly generated landscapes for each value of \( N \) and \( K \). The mean average should then be calculated for each combination of \( N \) and \( K \) after the 100 runs and placed in the results table, similar to Table 2.

This experiment should also be carried out twice, first with the original concept of travelling the landscape, where the whole landscape is created first and then an adaptive walk occurs. Secondly the revised quick method of travelling the landscape should be tested, where the combination and fitness value are assigned as the landscape is travelled. This will ensure that both methods results are similar to Kauffman’s original model and ensuring both approaches are an accurate replica of how the NK Model travels the landscape.

6.2.2 Hypothesis

A simple prediction can be made, for both the traditional method of travelling the landscape and the fast method. The results should be similar to Kauffman’s (1993) original results (Table 2) and also be similar to Kauffman’s results where \( K \) is assigned to each parts nearest neighbours (Table 1), as from the investigations in Chapter 2, the assignment of \( K \) appears to not have an effect on the results produced. A slight degree of variation should be expected however as the landscapes are created with random numbers and therefore the landscape used will not be the same as Kauffman’s. The same pattern of the results however should be apparent with the highest optima being found for any \( N \) value having a \( K \) value of two or four and then slowing decreasing in value.

Table 2 Kauffman’s Results of mean fitness of local optima (random interactions). Taken from Kauffman (1993) Table 2.2.

<table>
<thead>
<tr>
<th>( K )</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>48</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.70</td>
<td>0.71</td>
<td>0.71</td>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td>4</td>
<td>0.68</td>
<td>0.71</td>
<td>0.71</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td>8</td>
<td>0.66</td>
<td>0.69</td>
<td>0.69</td>
<td>0.70</td>
<td>0.71</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>0.65</td>
<td>0.65</td>
<td>0.67</td>
<td>0.68</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>0.63</td>
<td>0.65</td>
<td>0.66</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td>0.60</td>
<td>0.62</td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.58</td>
</tr>
</tbody>
</table>

When \( K = N \), actual \( K \) value is \( K - 1 \).
6.3 In depth mean fitness of local optima (random interactions)

Kauffman (1993) performs the experiment to examine the statistical properties of the landscapes for different values of the fundamental parameters ($N$ and $K$). Kauffman does this by studying a small number of $N$ values over a reasonable distance (starting at eight and multiplying by two until ninety-six), with the $K$ values following the same pattern starting from two. This experiment focuses on the opposite, studying every $N$ and $K$ values result between the values of $N = 2 – 16$ and $K = 0 – 15$.

The aim of this experiment is to try and get a more detailed view of the values created by the model, in an aim to gain a better understanding of the NK Model. For example, as $K$ increases by one, is the fitness value always going to be slightly higher than the previous $K$ fitness value until it reaches the point where the fitness value starts to decline? Once the fitness value is declining, will a constant decline in fitness values throughout the rest of the $K$ values be seen? Alternatively are there some fitness values which are higher than the previous $K$ fitness values, when essentially Kauffman (1993) suggests the value should be lower?

This experiment will be carried out through assigning $K$ randomly only, as Kauffman (1993) states that there is little difference in the results between the two methods of assigning $K$ the nearest-neighbour assignment of $K$ will not be used in this experiment. This is due to two main reasons; first if both versions produce similar results the same thing would effectively be produced twice. Secondly with the time restraints of the project and the time it take to run the simulation, there would not be enough time to run the simulation for each version of $N = 2 – 16$ twice. Therefore if both versions of assigning $K$ were carried out the range of $N$ would have to be decreased and for this experiment to have a larger range of $N$ would be more beneficial for the results and analyse.

This experiment will however use both the original slow method of creating the landscape (where the whole landscape is created before anywhere is travelled) and the faster method (where the combination is generated and fitness value access when the simulation is running). This will prove two comparable versions of results and also enable a further comparison of the two slow and fast method methods in which to travel the landscape.

6.3.1 Method

In this experiment, the number of other parts ($K$) for each part is assigned randomly to other parts within the same system. The simulation will be run for every combination of $N = 2 – 16$ and $K = 0 - 16$. 
An adaptive walk should start from a randomly selected position on the landscape and then proceeds like normal until a local optima is reached. This process should be repeated 100 times on different randomly generated landscapes for each value of $N$ and $K$. The mean average should then be calculated for each combination of $N = 2 – 16$ and $K = 0 – 16$ after the 100 runs.

6.3.2 Hypothesis

From the understanding gained from the earlier investigation into Kauffman’s NK Model, the expected fitness value for each $N$ should start around 0.60 gradually increasing until $K = 2$ or slightly higher. The fitness values for each $N$ should then start to decline in value as the complexity ($K$) of the model increase, eventually becoming lower than the original fitness value of $K = 0$.

6.4 Entrepreneur Vs. Silent Investor

This final experiment tried to show a method in which the NK Model can mimic a real life situation and the potential information it can provide. Through the use of the long jump method and the hybrid of an adaptive walk and long jump, an extension added to the NK Model, the experiment models two different business investment strategies.

First for this experiment the generic definition of the main parameter for the NK Model need to be redefined for the intended purpose.

- $N$ – The number of characteristics within an organisation
- $K$ – The number of other characteristics which influence each characteristic.
- $A$ – The number of alternative characteristics within each characteristic.
- Landscape - The market in which all the organisations are in.

For example, if an organisation is made up of five characteristics ($N = 5$) and the number of alternative characteristics in an organisation is two ($A = 2$, identified by 0 or 1). An organisation represented by 00000 would be very similar to an organisation represented by 00001, however another organisation represented by 11111 would be a totally different. The parameter $K$ represents each characteristic of the market and how it is influenced by other characteristics, for example if a characteristic for an organisation is to employ a lot of people this may effect the organisations financial characteristic.

With this concept established we can look at the two alternative investment strategies. The entrepreneur, who would select an organisation and invest in that area, spending time
looking for better solutions in the nearby market, before moving onto a different organisation, (i.e. hybrid travel, of adaptive walk and then long jump). The silent investor however has a strategy of investing where they can increase their profit without any further work (or exploration), therefore adopting the long jump method, jumping from one organisation to another where an increase in profit is attainable.

6.4.1 Method

In this experiment, the number of other influencing characteristics \( K \) for each characteristic in an organisation is assigned randomly. The simulation will be run for the following values \( (N = 5, 10, 15, 20, 25 \text{ and } K = 0 – 24) \), \( A \) will be kept with the default of two.

This experiment will only be carried out using the fast method of creating the landscape (where combinations and fitness calculated when required). This is to allow a higher range of \( N \) to be tested in the time available and the slow method should produce similar results but just take a lot longer to do so.

Entrepreneur

In this part of the experiment all the combinations of the values of \( N \) and \( K \) specified above will be run 100 times and the mean fitness will be calculated. Each simulation will initiate from a randomly selected organisation and then perform the hybrid search method, performing an adaptive walk until a local peak is reached. The simulation should attempt to perform a long jump, which will be set to only have up to eight attempts. If a higher fitness value is found within these attempts that location will be moved to and the adaptive walk and long jump process repeated until no higher fitness is found in the eight long jump attempts.

The number of long jumps attempted has been specified to eight as if the entrepreneur cannot find a better organisation within this amount of attempt, it is assumed they give up and stay where they are.

Silent Investor

In the same method in which the entrepreneur simulation was run all the combinations of the values of \( N \) and \( K \) specified above will be run 100 times and the mean fitness will be calculated. Each simulation will initiate from a randomly selected organisation, the long jump method will then be used to travel the landscape until it stops from its natural end condition set out in the long jump specification (see Chapter 2, section 2.3.4).
6.4.2 Hypothesis

From this experiment there are several predictions we can make. First it is expected that the entrepreneur will achieve a higher fitness value and therefore be more successful overall with their investment strategy. However the silent investor is likely to find a better investment a lot quicker than the entrepreneur, potentially missing the opportunity to invest in another organisation which is nearby and therefore not performing as well overall. Finally as the number of characteristics within an organisation grows the expected fitness value attainable for the silent investor is likely to drop as potentially they are too overwhelmed with the possibilities of organisations to choose from making it harder to locate a better investment. The entrepreneur however should not experience this problem as much as they have a better understanding and awareness of the organisations around them.

To summarise the predicted results are:

1. The entrepreneur will achieve a higher fitness value
2. Silent investor is likely to find a better investment a lot quicker than entrepreneur
3. The silent investors fitness value attainable is likely to drop as the number of characteristics within an organisation increases.
4. The entrepreneur should have a better understanding and awareness of the market and the organisations around them
Chapter 7

Results

Performing the experiments have produced some interesting and frustrating results, highlight some more problems with the chosen development software. Overall however most of the experiments were a success, with the results matching the hypothesis. A set of all the results in labelled tables can be found in Appendix A.

A major problem encountered while running these experiments was the amount of time it took to run larger $N$ and $K$ values 100 times. The decision however was made not to compromise on this issue by reducing the number of simulations per $N$ and $K$ value. This decision was primarily made due to it seeming extremely import to the whole project and requirements (see Chapter 3), that the implemented model is proven to be an accurate replica of Kauffman’s original NK Model.

7.1 Results – Mean fitness of local optima (nearest-neighbour interactions)

Effectively two experiments were run in this section, once with the original slow method to travel the landscape and secondly with the faster method.

7.1.1 Slow Search Travel Results

Due to the time restraints of the project and the time it took to run some of the simulations with larger values of $N$ and $K$, results were only attained up to $N = 24$ and $K = 23$. Therefore the last two values of $N$ (48, and 96) which Kauffman had in his original results we not attained. This decision was made due to the fact it was taking about a week to run the value of $N = 24$ on a reasonably high specification PC. Although this potentially isn’t a massive
amount of time compared to some simulations with a tight project deadline approaching it was decided this experiment should be stopped at this point so other experiments could be performed.

From the results produced in this experiment (see appendix A) a comparable match to Kauffman’s originally results can be seen. When $K = 0$ the fitness of optima are independent of $N$ and equal to about 0.65. This is expected as discussed in the earlier investigation (see Chapter 2) fitness values were drawn at random between 0.0 and 1.0 for both alternative parts. Kauffman (1993) argues that ordered statistics shows that the average value of the less fit pare will be $\frac{1}{3}$ and that of the fitter part will be $\frac{2}{3}$. Since each site contributes additively to the overall fitness, which is the mean, the global optimum should be independent of $N$.

As the value of $K$ increases, the effect Kauffman describes as a form of complexity catastrophe occurs (see Chapter 2), thus causing the mean fitness value to steadily decline as $K$ increases. These results therefore to demonstrate Kauffman’s theory and replicate the behaviour predicted. On these grounds the assumption can be made that this method of travelling the landscape is an accurate replica of Kauffman’s original method.

### 7.1.2 Fast Search Travel Results

Unfortunately due to a slight limitation of the present implementation method to enable the fast search to work the results for $N = 96$ were unobtainable. This is due to an issue discussed in the implementation (Chapter 5). However as this experiment took advantage of the fast search method the time taken to complete the simulations was a lot quicker, taking less than half an hour to run all the simulations up to $N = 24$. It is important to note however, the model had to be altered slightly to accommodate $N = 48$, and this simulation did take slightly longer, to implement this value, several data types had to be changed to the type “long” and an alternative to the IntBag was used. This is discussed more in the implementation section (Chapter 5).

Again as seen in Appendix A, these results conform to the same predicted behaviour discussed above and are comparable to Kauffman’s original results. Hence the assumption can also be made that this method of travelling the landscape is also an accurate replica of Kauffman’s original method.

### 7.1.3 Conclusion

Two alternative methods of performing the same operation have been directly compared to Kauffman’s results for travelling the landscape with $K$ being assigned to its nearest neighbours. The results reinforce Kauffman’s original theory and are very similar to his own results. This allowing the assumption to be made that both the fast and the slow method of
travelling the landscape with nearest neighbour assignment are a true replica to Kauffman’s original NK Model.

### 7.2 Result – Mean fitness of local optima (random interactions)

Again effectively two experiments were run in this section, one with the original slow method to travel the landscape and secondly with the faster method.

#### 7.2.1 Slow Search Travel Results

As for the same reasons as in the previous experiment, results were only attained up to $N = 24$ and $K = 23$.

Comparing the results retrieved from this experiment a clear similarity with Kauffman’s original results can be seen (Figure 9). The table of results (appendix A) and Figure 9, clearly illustrate a similarity and therefore potentially proving the implemented NK Model is a good replica of Kauffman’s original model for random $K$ assignment using the long search approach.

![Mean fitness of local optima (random interactions) - Long Search](image)

**Figure 9.** Results - mean fitness of local optima (random interactions) – long search. Compared with Kauffman’s original results
Obviously some difference is expected when comparing the result as random number are used to generate the landscape and therefore the same values would not have been used that Kauffman originally used. The key trend however is clear, what Kauffman describes as a form of complexity catastrophe is seen in both Kauffman’s original results and our version of the NK Model. The only slight down side here is the failure to get the whole range of result through to N = 96, unfortunately however the time restraints just wouldn’t allow this.

### 7.2.2 Fast Search Travel Results

Unfortunately due to a slight limitation of the present implementation method to enable the fast search to work the results for $N = 96$ were unobtainable. This is due to the same reasons previously stated in Section 2.1.2, and the same solution and methods were applied.

Comparing the results against Kauffman’s original results, the same trend can be seen from what was originally predicted from the investigation into Kauffman’s NK Model (see Chapter 2). The results show, when $K$ is increased the fitness value originally rises but as the complexity increases further the fitness value obtainable begins to decline at an increasing rate (Figure 10). One observable point of this fast search technique is that the fitness values for different $N$ values appear to be a lot closer together than Kauffman’s results. This could potentially be by chance and a total coincidence from the random numbers generated, to see if this is the case however or if in fact the fast search some how makes the fitness values closer. This is potentially an opportunity for a further investigation and could be quite interesting as essentially the search and fitness value assignment is done in exactly the same way, it is just the timing of when it is done which has been changed.

![Mean fitness of local optima (random interactions) - Fast Search](image)

**Figure 10.** Results - mean fitness of local optima (random interactions) – fast search. Compared with Kauffman’s original results
7.2.3 Slow Search Vs. Fast Search Comparison

Comparing the two different search methods (Figure 11), it is clear to see that fundamentally they have the same pattern, as the complexity increases the overall fitness value attainable increases. However comparing the two it is apparent that the fast search method seems to have a closer relationship in fitness values for each \( N \) value, as also observed on comparing the results with Kauffman’s original work. The fundamental concept and results however are still correct and therefore this in not a major issue at present.

![Mean fitness of local optima (random interactions) - Slow Vs. Fast Search Comparison](image)

**Figure 11.** Results - mean fitness of local optima (random interactions) – slow and fast search comparison.

7.2.4 Conclusion

On comparison of the two search methods against Kauffman’s original results and against each other a conclusion can be made that this implemented version of Kauffman’s NK Model for the random assignment of \( K \) is a successful replica. From these limited results however it is noticeable that the fast search method does not have as such wide spread fitness values for each \( N \) value and therefore a further investigation could be made to try and find out why this is the case. It is however important to remember that although there appears to be a big difference it is essentially only about 0.05 or less of a difference in the actual spread of the results. However it is potentially a question why it does this when the method to search is exactly the same apart from when the fitness value is actually assigned.
7.3 In depth mean fitness of local optima (random interactions)

This experiment was designed to try and provide a better insight into the results produced by an NK Model, focusing on a specific range of values ($N = 2 – 16$ and $K = 0 – 15$).

On review of the previous experiments carried out it was decided to just carry out this simulation using the slow search method. Although this is the slower method of the two from the previous experiment results, this method was found to be the slightly more accurate method. Even though overall both search methods are suitable to replicate an NK Models simulation for this in-depth investigation it was decided that accuracy over time should prevail.

On examination of the results (see Appendix A, for more details) it is clear that this experiment still supports Kauffman’s theory. It does however also highlight some other issues which previously have not been discussed by Kauffman. From the results shown in Figure 12, once the highest peak is reach, usually around the $K = 2$ value, the fitness value should start to decline it appears, however that there are actually moments when a sudden unexpected increase in fitness can occur again and a mini climb to a higher fitness occurs.

![In depth mean fitness of local optima (random interactions)](image)

**Figure 12.** In depth mean fitness of local optima (random interactions) – slow search. For a range of $(N = 2 – 16$ and $K = 0 – 15$)

If we study $N = 15$ in more detail (see Appendix A), once the initial maximum fitness value is reach $K = 4$, the fitness value does for each higher value of $K$ generally decreases in value.
However there are moments for a single change of $K$ where the fitness actually increases again. This arguably could be put down to the fact that the numbers are assigned randomly. Even still effectively over a very small range of $N$ this occurs several times with different values of $N$ and $K$.

Potentially a further investigation should be carried out into these results, studying a larger range of $N$ and $K$ values to study the frequency of the occurrence and if any pattern occurs, if any pattern were to be found then this could potentially contradict Kauffman’s concept of the complexity catastrophe occurrence as $K$ increases.

A suggestion why Kauffman has not discussed this previously is due to the fact, he never seemed to study a range of values close together, which effectively gives a better illustration the detailed workings of the model. Due to the infrequency of the mini climbs it could be quite easy to miss when on looking at different values of $N$ and $K$ which are a distance apart from each other.

### 7.4 Entrepreneur Vs. Silent Investor

Unfortunately due to the extent of time spent on running the other experiment to gain a full insight into the mechanics of the NK Model, studying the results produced and running an array of simulations to gain a step by step incrimination of results this final experiment was not completed. Therefore the results have not been compiled due to the time restrictions of the project and the numerous problems which had occurred on the way.

Apart from this an implementation error within the method required to produce the results appears to have occurred. This had previously been tested and working perfectly, however the random number generator required to select a location on the landscape stopped functioning correctly, always producing the same number, thus the jump was always finished early (see Listing 5). Again due to the numerous other implementation problems experience, the problem could not be rectified and the results produced in time. This particular random number generator was a component of MASON, thus adding to the list of implementation problems experienced with this agent toolkit.
Investigating Kauffman’s NK Model for Agent-Based Modelling

Listing 5 Extract of code which stopped producing random numbers, and always produced the same number.

```java
public void longJump(int nk)
{
    NK = nk;
    while (jumpFinished == false)
    {
        int tempNK;
        double NKSpaceFitness;
        double tempNKSpaceFitness;
        NKSpaceFitness = getFitness(NK);
        if (jumpAttempt < maxLongJumps)
        {
            tempNK = random.nextInt(totalNKSpaces); // Returns an integer drawn uniformly from 0 to n-1
            System.out.println("Random NK Space Selected: "+tempNK);
            tempNKSpaceFitness = getFitness(tempNK);
            System.out.println("***** "+ tempNKSpaceFitness);
            if (tempNKSpaceFitness > NKSpaceFitness)
            {
                NK = tempNK;
                NKSpaceFitness = tempNKSpaceFitness;
                maxLongJumps = maxLongJumps * 2;
            }
            else
            {
                jumpAttempt ++;
            }
        }
        else
        {
            jumpFinished = true;
            System.out.println("Jump finished highest fitness found: "+ NKSpaceFitness);
        }
    }
}
```

7.5 Conclusion

From the first two experiments and analysis of the results, it is clear the implementation of the NK model can and does replicate similar results to Kauffman’s original version. The faster search method however didn’t seem quite as accurate although still producing a comparable set of results the fitness values of different $N$ values at each $K$ appeared to be slightly closer together than in the slow search method and Kauffman’s original.
Investigating Kauffman's NK Model for Agent-Based Modelling

The third experiment and perhaps the most interesting was investigating Kauffman’s model in more detail. This showed that where previously it was expected for the fitness value to continually decline as $K$ increased this was not always the case, with mini climbs in several simulations where the actual value increased slightly.

The forth experiment unfortunately was unsuccessful; this was primarily due to the time restraints and having potentially spent to much time on the other simulations and development of the model.

These experiments have however brought some considerations to light and suggested future work. First the fast search method could possible be investigated further to study if it is just coincidence that the values appear to be slightly more compacted or if there is another reason behind it. Secondly a further in-depth study should be carried out, increasing the range of $N$ and $K$ values to determine if the unexpected mini climbs in the landscape are purely due to the random number assignment or if there is a potential fault in Kauffman’s concept.
Chapter 8

Project Critique

In this section I look back over the past year and reflect on my attitude, methods, effort and achievements in the creation of this project.

8.1 Main Problems with the Project

I begin with the problems, as this is what I have had most of and on reflection a lot of my strengths have been overcoming these problems. From the outset this projected seemed to attract issues which seemed to complicate otherwise straightforward issues.

8.1.1 Investigation Problems

A key part of this project has been the investigation into Kauffman’s NK Model, having completely rewritten my literature review from scratch twice, as well as constantly amending sections as the project unfolded.

Essentially a straightforward concept was devised to investigate Kauffman’s NK Model and turn it into an agent based simulation. With more business analysts wanting to use the model Richard Vidgen (my second supervisor) from the universities business department was keen to have a model to use himself. However once the initial investigation into the model started it was clear it wasn’t going to be as straightforward as first anticipated. With regular group meetings discussing our findings and trying to understand Kauffman’s extremely bad explanation and definition of the NK Model, I eventually thought I understood the concept. Richard confirmed this method, with another business analyst who has written several papers about Kauffman’s model. I was then happy with my understanding of Kauffman’s concept and though I was on the right path to developing a successful model.
Investigating Kauffman’s NK Model for Agent-Based Modelling

However as I moved further into the project it became clear that this definition was incorrect and like many of the other papers available on the NK Model I had been mislead slightly, from being told we had the correct concept. This in effect meant I had to go back to the drawing board, going over and reviewing all the literature again, primarily focusing on Kauffman’s original definition. As other literature written by authors who had used Kauffman’s definition as a base appeared in places to be more misleading than helpful. Eventually after several weeks of studying the literature and trying to calculate what Kauffman has done, I eventually got the correct definition. This however meant that the time left to complete the rest of the project was very limited and my literature review would again have to be altered.

8.1.2 Implementation Problems

Moving onto the implementation of the project, MASON was chosen as the development toolkit (for the reasons specified in Chapter 5), this was possibly my biggest mistake of the entire project. Having never done any agent programming before, I was slightly unaware of what to expect but willing to learn. After choosing MASON for its speed and 3D modelling GUI possibilities I was excited at the possible outcome of the project.

Having to initially take time out to learn MASON going through the available tutorials, I thought I had become confident in MASON and could produce and extremely impressive visually pleasing project. When it came to implementing the actual NK model I didn’t have any major implementation issues other than what you would expect of a normal implementation. However when it came to trying to get the simulations to run correctly for the purpose I required it became extremely difficult.

One major issue I experience was the lack of support, no book, forum or anything, apart from a set of “stupid model”, (basic sample implantations of simulations in MASON) but you had to be doing the same as what they where showing for them to be much help. A mailing list however was also available and this became my only source of help once I was really stuck. To highlight two of the key issues I experienced:

The MASON toolkit has not been designed well to stop a repeating steppable simulation once a specific action has been performed and this is essential for the implementation of the NK Model, (this also then lead to charting issues to). This took several weeks to figure out after I did not get a successful reply from the mailing list I was left on my own to find a way to do it. Secondly another major issue which really effected the project is the GUI version of MASON is not designed to run the same simulation more than once, recording the results, i.e. running the simulation 100 times and getting the mean fitness. Again this took me over a week trying to find out how to perform this operation, playing around with my implementation until eventually I was informed it was not possible from the developer of mason, unless you really hacked around with the system, and by this point I did not have the time to do so.
Essentially these are just the main highlights of the difficulties using MASON. However saying this I would not say it is a bad agent development toolkit it just has a poor support community to make it more usable.

8.2 Strengths

My key strength thought out this project has been my ability to take a problem and resolve it on my own. Having faced an enormous amount of issues during this development I feel I have really learnt to deal with really difficult issues and structured and efficient format

Another main strength I would say has been in my project planning and control, although the project has run until the last minute, this has been due to the ever changing situations and problems experienced along the way. Several time having had to sit down and reorganise the project and prioritise issues of importance in order to achieve the most in the time left available.

I also feel throughout the project I have been very strong in my determination to successfully complete this project even though so many problem have been placed in my way. One key issue is the literature review and investigation into Kauffman’s NK Model, probably the hardest part of my project, being dyslexic I really struggled with this section and it potentially took me a lot longer than it should have. Combining this with the fact I had to totally rewrite it from scratch, I feel this is a good demonstration of the effort I have put into the project.

8.3 Summary

Overall I feel I have done extremely well in tackling all the problems and issues which I have come against throughout this project. However due to all these issues I am aware the project implementation has suffered and the final product is only really half of what I ideally would have liked to created. Essentially having added more of the extensions highlighted in my investigation, performing more experiments and having a completely working GUI with 3D models.
Chapter 9

Conclusions

On review of the project, referring back to my original objectives specified in Chapter 1, I am pleased to say I have completed all but the final one completely, and this final objective has partly been done except for the implementation of additional features to the problems experienced throughout and the lack of time available.

9.1 Weaknesses & Improvements

One of my key weaknesses of for this project was my programming ability, having only ever done the essential programming required to learn a new concept of agent and implement it successfully has been extremely difficult.

Another key weakness was may intrigue to investigate Kauffman’s model further and gain a better understanding, once the concept had eventually been understood. Spending too much time on simulating results to study the actual model operation and not therefore not having enough time to apply the model to alternative concepts. I do however consider it to be extremely important to have gained a complete understanding of the model and do not regret this decision. After also being asked to possible present the results and explanation of the model to other researchers in the business department there does seem to be a very successful side to the work I have produced.

My final main weakness has also been my dyslexia, finding it extremely hard, when it came to the amount of literature and formal writing required to produce a project to a high standard.
Improvements to the model would have to include completing the GUI section of the model so the 3D visualisations of the NK landscape are included and better usability of the model is available. Also another key improvement would be complete the implementation of the additional components suggested in my investigation to make the NK Model’s application possibilities potentially endless. For example developing the NK Model into an NKC Model.

Another main improvement to this project would be to get something officially published on the operation of the NK model, providing a comprehensible definition of a generic version could potentially assist so many people in there own use of the NK Model. This would also how to demonstrate the possibilities available from using this essentially simple model, however it is only simple if it is defined in a comprehensible manner.

9.2 Further Work

In addition to what I have done and the improvements I have suggested, I would like to carry out the fourth experiment which this project did not allow time for. This experiment was essentially a simple easy to comprehend example of how useful the NK Model can be when applied with an understanding of the underlying mechanics.

Additionally I would like to study Kauffman’s NK Model further looking deeper into the structure, operation and other potential features Kauffman possibly suggested which are hidden away within his book. Further more I would like to continue the work on the NK Model in the generic context I have produced, enabling it to be easier to understand and use. Thus allowing future research to be carried out using the help my NK Model definition, and template implementation.

9.3 Final Conclusion

It has become apparent throughout this project that the NK Model is massive concept and research area in which people are currently doing a PHD on. It has therefore been a constant struggle to try and fully review and produce an NK Model in a relatively short period of time. This project appeared to grow in size from the outset becoming bigger than anyone originally anticipated and is something which could be continued next year by another student.

NK Model is an extremely versatile model with an abundance of features and possibilities. However its main restraint is its poor incomprehensible definition its creator gave it and is possibly one of its only weaknesses, however without the correct understand the model is almost useless and therefore the poor definition is a fundamental fault in the NK Model.
Reflecting on this information, and reviewing the original objectives and requirements set, this investigation has achieved all of them to some degree. Without the various problems along the way, more could have been archived, but this is part of the learning process and has essentially developed my understanding of the NK Model a lot further than otherwise. Overall I have managed to achieve the fundamental aim of this project, to develop a correct version of a NK Model, proven to replicate Kauffman’s original results. Therefore I have conformed to all the requirements specified in Chapter 3 apart from when conflicts in the requirements were discovered and compromises had to be made.
Bibliography


Investigating Kauffman’s NK Model for Agent-Based Modelling


MABS (2002), Third International workshop on Multi Agent Based Simulation, proceedings of an international conference of Multi Agent Based Simulation, held Italy, 6 September 2002, Blekinge Institute of Technology, Bologna, Italy.


TIC (2002), Tackling Industrial Complexity: the ideas that make a difference, proceedings of an international conference of the Manufacturing Complexity Network organised by the University of Cambridge, held Cambridge, 9-10 April 2002, Institute for Manufacturing, Cambridge, UK.


Investigating Kauffman’s NK Model for Agent-Based Modelling


Appendix A

Results
Investigating Kauffman’s NK Model for Agent-Based Modelling

Mean fitness of local optima (nearest neighbour interactions)

Implemented NK Model Results on Slow Search Method

<table>
<thead>
<tr>
<th>$N$</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>48</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.6499866</td>
<td>0.6578673</td>
<td>0.6665498</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>0.7111679</td>
<td>0.7113217</td>
<td>0.7194984</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>0.6939773</td>
<td>0.706542</td>
<td>0.7098466</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>0.6715703</td>
<td>0.6749847</td>
<td>0.6789466</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>0.6504984</td>
<td>0.6549847</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>0.6216549</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

When $K = N$, actual $K$ value is $K - 1$.

Implemented NK Model Results on Fast Search Method

<table>
<thead>
<tr>
<th>$N$</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>48</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.64196195</td>
<td>0.64605821</td>
<td>0.64024309</td>
<td>0.6416517</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>0.70051825</td>
<td>0.69663625</td>
<td>0.69798955</td>
<td>0.7094981</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>0.69371912</td>
<td>0.69596112</td>
<td>0.69504531</td>
<td>0.6946546</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>0.6718361</td>
<td>0.67782872</td>
<td>0.67758696</td>
<td>0.6760132</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>0.6451577</td>
<td>0.65546292</td>
<td>0.6506465</td>
<td>-</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>0.64316942</td>
<td>0.6432165</td>
<td>-</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td>0.6398498</td>
<td>-</td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

When $K = N$, actual $K$ value is $K - 1$.

Kauffman’s Results of mean fitness of local optima (random interactions). Taken from Kauffman (1993) Table 2.1.

<table>
<thead>
<tr>
<th>$N$</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>48</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0.65</td>
<td>0.65</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td>2</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
<td>0.71</td>
</tr>
<tr>
<td>4</td>
<td>0.70</td>
<td>0.71</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>8</td>
<td>0.66</td>
<td>0.68</td>
<td>0.68</td>
<td>0.69</td>
<td>0.68</td>
</tr>
<tr>
<td>16</td>
<td>0.65</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
<td>0.66</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>0.63</td>
<td>0.64</td>
<td>0.64</td>
<td>0.64</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td>0.60</td>
<td>0.61</td>
<td>0.58</td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When $K = N$, actual $K$ value is $K - 1$.
Mean fitness of local optima (random interactions)

### Implemented NK Model Results on Slow Search Method

<table>
<thead>
<tr>
<th>$K$</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>48</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.703998342</td>
<td>0.710680068</td>
<td>0.716544865</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>0.687601839</td>
<td>0.705637211</td>
<td>0.708465466</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>0.657607424</td>
<td>0.679301912</td>
<td>0.686506415</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>0.642799780</td>
<td>0.6427997680</td>
<td>0.642799708</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>0.634049494</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

When $K = N$, actual $K$ value is $K - 1$.

### Implemented NK Model Results on Fast Search Method

<table>
<thead>
<tr>
<th>$K$</th>
<th>8</th>
<th>16</th>
<th>24</th>
<th>48</th>
<th>96</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.687415371</td>
<td>0.682012329</td>
<td>0.6850737</td>
<td>0.6886546</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>0.689862944</td>
<td>0.67527382</td>
<td>0.6850592</td>
<td>0.6812332</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>0.670859809</td>
<td>0.666152191</td>
<td>0.6721727</td>
<td>0.6740554</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>0.63786227</td>
<td>0.6408922</td>
<td>0.6548496</td>
<td>-</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>0.6384655</td>
<td>0.6440857</td>
<td>-</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td>0.6146546</td>
<td>-</td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

When $K = N$, actual $K$ value is $K - 1$.

Kauffman’s Results of mean fitness of local optima (random interactions). Taken from Kauffman (1993) Table 2.2.
Investigating Kauffman's NK Model for Agent-Based Modeling

<table>
<thead>
<tr>
<th>Implemented NK Model Results on Slow Search Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
</tr>
<tr>
<td>K</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

When $K = N$, select $K$ value is $K - 1$. In depth mean fitness of local optima (random interactions)
Investigating Kauffman's NK Model for Agent-Based Modelling

![Graph showing the relationship between Fitness and K Value for different NK models.](image)

- Fitness values range from 0.74 to 0.82.
- K Value values range from 0 to 16.

The graph illustrates the in-depth mean fitness of local optima (random interactions) for various NK models with different K values.
Investigating Kauffman's NK Model for Agent-Based Modelling

In depth mean fitness of local optima (random interactions) $N = 15$

![Graph showing the relationship between fitness and K value for $N = 15$. The fitness values range from 0.62 to 0.72, with a peak around K = 5.]
Appendix B

Code

- Model Class
- ModelWithUI Class
- NKSpace_Gen Class
- Agent Class
Investigating Kauffman’s NK Model for Agent-Based Modelling

**Model Class**

```java
package sim.app.NK_Model;

import java.io.*;
import ec.util.MersenneTwisterFast;
import sim.app.tutorial1and2.Tutorial1;
import sim.engine.*;
import sim.field.grid.*;
import ec.util.*;
import java.math.BigInteger;

/**<p>@author Rick Mellor</p>* Key Operations of Model Class:
* 1) Obtains Parameter for which to Create an NK Model
* 2) Calls methods and objects required to create the NK Model on
the parameters gained.
* 3) Obtains Results from the landscape
*/

public class Model extends SimState {
    public static int N = 5; // key parameter (parts within a
    system)
    public static int K = 3; // key parameter (number of
    parts which effect each part)
    public static int A = 2; // key parameter (alternative
    parts)

    public static boolean peakFound;

    public static double meanFitness; // stores mean fitnes over the
    number of runs
    public static int numRuns; // number of times to run
    the simulation
    public static int runCount;
    public boolean stop;
    public static int totalNKSpaces = (int) Math.pow(A, N); // total
    number of NK spaces posible i.e. A^N (i.e. 2^N)
    int[] CountValues; // Used For 0,1 Combinations
    int[] tempCountValues; // Used For 0,1 Combinations
    int[] temp; // Used For 0,1 Combinations
    int Count; // Used For 0,1 Combinations

    static boolean quickRun;
    static boolean walk;
    static boolean randomLinks;
    static boolean jump;
```
public Model(long seed)
{
    super(seed);
    Count = 1;
    CountValues = new int[N];
    tempCountValues = new int[N];
    temp = new int[N];
    CountValues[0] = 1;
    tempCountValues[0] = 1;
    //meanFitness = 0.0;
    numRuns = 100;

    for (int i = 1; i < N; i++)
    {
        Count = Count * 2;
        CountValues[i] = Count;
        tempCountValues[i] = Count;
    }
}

public int getN() { return N; }
public void setN(int val) { if (val > 0 ) N = val;
gettotalNKSpace();}
public int getK() { return K; }
public void setK(int val) { if (val >= 0 && val < N) K = val;
; }
public int getA() { return A; }
public void setA(int val) { if (val > 0) A = val;
gettotalNKSpace();} // Need to add A's limit!
public int gettotalNKSpace() { return totalNKSpaces = (int)Math.pow(A, N); }
public int getNumRuns() { return numRuns; }
public void setNumRuns(int val) { if (val > 0) numRuns = val;
}

public void start()
{
    super.start(); // very important! This resets and cleans out the Schedule.

    NKSpace_Gen nkGen = new NKSpace_Gen(System.currentTimeMillis());
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
nkGen.initialSetup();
nkGen.Build_Space();
nkGen.printAllFiles();

Agent agent1 = new Agent(System.currentTimeMillis());
agent1.selectPoint();
agent1.getOneMutantNeighbours();
schedule.scheduleRepeating(agent1);

schedule.scheduleRepeating(0,3,new Steppable() {  
    public void step(SimState state) {
        checkIfDone(state);
    }
});

// *** NEED TO MAKE CLASS WRITE RESULTS TO FILE ***

public void checkIfDone(SimState state) {
    Model theModel = (Model) state;
    if (peakFound == true) {
        peakFound = false;
        theModel.schedule.reset();
        theModel.finish();
        // System.exit(0);
    }
}

// step main
public static void main(String[] args) {
    // Model model = new Model(System.currentTimeMillis());
    // model.start();
    // doLoop(Model.class, args);
    // System.exit(0);

    quickRun = false;
    walk = true;
    randomLinks = true;
    jump = false;
```
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
NKSpace_Gen nkGen = new NKSpace_Gen(System.currentTimeMillis());

for (int runs = 0; runs < numRuns; runs++)
{
    if (walk == true)
    {
        nkGen.initialSetup();
        Agent agent1 = new Agent(System.currentTimeMillis());
        System.out.println("Run: " + runs);
        if (quickRun == false)
        {
            nkGen.Build_Space();
            nkGen.printAllFiles();
            agent1.selectPoint();
            agent1.getOneMutantNeighbours();
            agent1.travel();
            nkGen.FitnessGen(); // Sets A fitness values for each A at each loci
            nkGen.setRandomLinks();
            agent1.peakFound = false;
        }
        else
        {
            MersenneTwisterFast random = new MersenneTwisterFast();
            int tempNK = random.nextInt(totalNKSpaces);
            nkGen.singleSpace(tempNK);
            System.out.println("Run: " + runs);
            agent1.selectedPoint(tempNK);
            agent1.getOneMutantNeighbours();
            agent1.travel();
            agent1.peakFound = false;
        }
    }

    if (jump == true)
    {
        nkGen.initialSetup();
        Agent agent1 = new Agent(System.currentTimeMillis());
        MersenneTwisterFast random = new MersenneTwisterFast();
        int tempNK = random.nextInt(totalNKSpaces);
        nkGen.singleSpace(tempNK);
        System.out.println("Run: " + runs);
        agent1.selectedPoint(tempNK);
        agent1.getOneMutantNeighbours();
        agent1.travel();
        agent1.peakFound = false;
    }
}
```

80
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
    meanFitness = meanFitness / numRuns;
    System.out.println("After "+numRuns+" runs, mean fitness =" +meanFitness +" For N="+N+" K="+K);
    System.out.println("Highest Score was:"+nkGen.highestScore);
```
package sim.app.NK_Model;

import sim.engine.*;
import sim.display.*;
import sim.portrayal.grid.*;
import java.awt.*;
import javax.swing.*;
import sim.util.media.*;

public class ModelWithUI extends GUIState
{
    public Display2D display; // defining display
    public JFrame displayFrame; // defining frame
    org.jfree.data.xy.XYSeries series; // the chart data series
    add to
    sim.util.media.ChartGenerator chart; // the charting facility

    public ModelWithUI()
    {
        super(new Model(System.currentTimeMillis()));
    }

    // calls underlying Model class's step method
    public ModelWithUI(SimState state) { super(state); }

    // Creates Display Info in MASON Program
    public static String getName() { return "NK Model: Version 1"; }

    // Creates Display Info in MASON Program
    public static Object getInfo()
    {
        return
        "<H2>Kauffmans NK Model</H2>" +
        "<p>... created as a generic template version of the NK
        Model";
    }

    // to get model parameter input in the GUI i.e. N =? K =? etc...
    (only allows to be input at a start of a sim! which is good
    public Object getSimulationInspectedObject()
    {
        return state;
    }
}
// start method (run on console button press)
public void start()
{
  super.start();

  display.reset(); // reschedule the displayer
  display.repaint(); // redraw the display

  chart.removeAllSeries(); // cleans chart
  series = new org.jfree.data.xy.XYSeries("Fitness", false);
  // sets axis value
  chart.addSeries(series, null);
  scheduleImmediateRepeat(true, new Steppable()
  {
    public void step(SimState state)
    {
      // need an X value and a Y value. The Y value
      // is whatever data I'm extracting from the
      // simulation. For purposes of illustration,

      double x = state.schedule.time(); // set value of chart
      double y = Agent.localPeak; // sets value of chart

      // now add the data
      series.add(x, y, true);
    }
  });
}

public void init(Controller c)
{
  super.init(c);

  // Makes the Display2D.
  Model mod = (Model)state;
  display = new Display2D(mod.N * 4, mod.N * 4, this, 1);
  displayFrame = display.createFrame();
  c.registerFrame(displayFrame); // register the frame so it appears
  // in the "Display" list
  displayFrame.setVisible(true);

  display.setBackdrop(Color.black);

  // Sets up chart (Creat/set title/ set axis/ put in frame)
  chart = new sim.util.media.ChartGenerator();
  chart.setTitle("Time/Peak Value - Mapping Each Result From A");
Investigating Kauffman’s NK Model for Agent-Based Modelling

Walk of The NK Model

chart.setRangeAxisLabel("Fitness");
chart.setDomainAxisLabel("Time");
JFrame frame = chart.createFrame(this);

frame.show();
frame.pack();
c.registerFrame(frame);

}

public static void main(String[] args)
{
    ModelWithUI modelWithUI = new ModelWithUI(); // creates instance which calls underlying model method
    Console c = new Console(modelWithUI); // Sets console
    c.setVisible(true); // makes it visible
}
Investigating Kauffman’s NK Model for Agent-Based Modelling

NKSpace_Gen Class

package sim.app.NK_Model;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import ec.util.*; // random method contained in there
import java.io.*;
import sim.util.*; // IntBag

/**<n *
 * @author Rick
 * NKSpace_Gen.class Key Operations
 * 1) Creates all possible A^N combinations of NK Spaces
 * 2) Calculates fitness values For all NK Spaces (depending on K)
 * 3) Saves Each NK Space to an indervidual file for quick reference later
 */

public class NKSpace_Gen extends Model
{
    int[] NKSpace; // Stores a combination of 0,1

    static double[][] AFitnessValues; // stores Fitness Values
    double NKSpaceScore; // stores a combination

    int numScores; // calculates number of fitness values required

    static int[][] links; // stores links between the genes including the gene itself
    double highestScore; //int[][] OneMutantNeighbours; // stores an NKSpaces one mutant neighbours i.e. if N = 5 it has 5 neighbours

    public NKSpace_Gen(long seed)
    {
        super(seed);

        numScores = ((int) Math.pow(A, K+1)); // calculates number of fitness values required
    }

    public void initialSetup()
    {

    }
Investigating Kauffman's NK Model for Agent-Based Modelling

```
links = new int[N][K+1]; // stores links between the genes including the gene itself
AFitnessValues = new double[numScores][N]; // array to store all initial fitness values [A row] = 0 or 1 or 2 etc.. A value [N column] = for each loci the A fitness value relates to
highestScore = 0.0;
FitnessGen(); // Sets A fitness values for each A at each loci
if (randomLinks == true)
{
    setRandomLinks();
} else
{
    setNearestNeighbourLinks();
}
public void Build_Space()
{
    int NK = 0; // Effectively the NKSpace count i.e. each NK row is one NKSpace Combination
    for (NK = 0; NK < totalNKSpaces; NK++) // while total number of combinations have not been made
    {
        NKSpace = new int[N]; // temporary stores an NKSpace i.e. combination
        for (int col = 0; col < N; col++)
        {
            if (tempCountValues[col] == 0)
            {
                NKSpace[col] = 1;
                temp[col] = temp[col] + 1;
                if (temp[col] == CountValues[col]) // count values used to determine when a 1 or 0 should be inserted
                {
                    tempCountValues[col] = CountValues[col];
                    temp[col] = 0;
                }
            } else
            {
                NKSpace[col] = 0;
                tempCountValues[col] = tempCountValues[col] -1;
            }
        }
        FitnessCalc(NK); // retrieves the fitness value for the newly created NK Space
        writeToFile(NK); // saves the NK Space combination and fitness value to a file
        System.out.println(NK+", ");
    }
}
```
Investigating Kauffman's NK Model for Agent-Based Modelling

    // used for fast search - calculates a combination from the NK Space requester
public void singleSpace(int nk)
{
    NKSpace = new int[N];
    for (int col = 0; col < N; col++)
    {
        int divVal = 0;
        int remVal = 0;
        divVal = nk / CountValues[col];
        // System.out.println(divVal);
        remVal = divVal % 2;
        NKSpace[col] = remVal;
    }
    FitnessCalc(nk);
    writeToFile(nk);

    for (int i = 0; i < N; i++)
    {
        System.out.print("[" + NKSpace[i]+ "]");
    }
    System.out.println();
}

public void setRandomLinks()
{

    // *** FILL THE LINKS *** // prints N in first column and then the value of the other n's N is linked to through k
    System.out.println("Links Random");
    for (int i = 0; i < N; i++)
    {
        links[i][0] = i; // sets first column of every row to n
        // Fills a bag with all possible loci N can link to appart from its self, as one is selected as a link the location is removed from the bad so N cannot link to the same loci twice
        IntBag nLinks = new IntBag(N);
        for (int x = 0; x < N; x++)
        {
            if (x != i)
            {
                nLinks.add(x);
            }
        }

        System.out.print("[" + links[i][0] + "]");
        int x =1;
        for (int j = 1; j < K + 1; j++)
        {
            System.out.print("[" + links[i][j] + "]");
        }
    }
}
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
{
    int randomSelect = random.nextInt(N-x); // so can pick a
    random number from a smaller list as the bag gets smaller each time
    links[i][j] = nLinks.objs[randomSelect];
    nLinks.remove(randomSelect); // removes the loci location
    from the bag so N cannot link to the same loci again!
    System.out.print("[" + links[i][j] + "]");
    x++;
}
System.out.println();
}

// Goes through assigning links to each N's neighbour until K
value is reached
// process is seen as circular so if get to end moves back to the
beginning
public void setNearestNeighbourLinks()
{
    System.out.println("Links Nearest-Neighbour");
    for (int i = 0; i < N; i++)
    {
        links[i][0] = i; // sets first column of every row to n
        System.out.print("[" + links[i][0] + "]");
        boolean right = true; // checks if it is the turn to
        assign a value from the right or left
        int rightCount = i + 1; // keep count of current
        location to assign link to for right hand side
        int leftCount = i - 1; // keep count of current
        location to assign link to for left hand side
        for (int j = 1; j < K + 1; j++)
        {
            if (right == true)
            {
                if (rightCount < N && rightCount > 0)
                {
                    links[i][j] = rightCount;
                    right = false;
                    rightCount ++;
                }
                else
                {
                    rightCount = 0;
                    links[i][j] = rightCount;
                    right = false;
                    rightCount ++;
                }
            }
            else
            {
            }
        }
    }
}
```
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
{   if (leftCount < N && leftCount >= 0)
    {
        links[i][j] = leftCount;
        right = true;
        leftCount --;
    }
    else
    {
        leftCount = N -1;
        links[i][j] = leftCount;
        right = true;
        leftCount ++;
    }
}
System.out.print("[" + links[i][j] + "]");
}
System.out.println();
}
```

// randomly assigns score for each possible combination
public void FitnessGen()
{
    // *** Fill The Scores ***
    int numScores = (int) Math.pow(A, K+1); // for how many each n looks at

    for (int aVal = 0; aVal < numScores; aVal++)
    {
        for (int loci = 0; loci < N; loci++)
        {
            AFitnessValues[aVal][loci] = random.nextDouble(); // ***
            MersenneTwisterFast 1.5x faster than java.random and better randomness!!! ***
        }
    }
}

// calculates fitness calculation for a specific combination
public void FitnessCalc(int NK)
{
    NKSpaceScore = 0;
    for (int loci =0; loci < N; loci++)
    {
        ...
    }
}
```
```java
int tempCount = 0;
// System.out.print(" ["+NKSpace[loci]+ "] - ");
if (K == 0)
{
    int aVal =0;
    while (NKSpace[loci] != aVal)
    {
        aVal++;
    }
    NKSpaceScore += AFitnessValues[aVal][loci];
}
else
{
    if (K < N)
    {
        if (NKSpace[loci] == 1)
        {
            tempCount += (numScores /2);
        }
        for (int link=1; link < (K + 1); link++)
        {
            if (NKSpace[links[loci][link]] == 1)
            {
                tempCount += (link); // took out link +1 changed to link
            }
        }
        // System.out.print(AFitnessValues[tempCount][loci]);
        NKSpaceScore += AFitnessValues[tempCount][loci];
    }
}
}
NKSpaceScore = NKSpaceScore / N;

System.out.print(" NK: "+NK+ " Fitness: "+NKSpaceScore+ "");
if (NKSpaceScore > highestScore)
{
    highestScore = NKSpaceScore;
}

public void writeToFile(int NK)
{
    String title = ("NKSpace_N"+N+"K"+K+"_"+NK+NK+.nk"); // puts count to identify unique genome combination and .nk to show what file used for.
    //int[] NKSpaceStore = new int[N];
```
try {
    DataOutputStream out = new DataOutputStream(new
          BufferedOutputStream(new FileOutputStream(title)));
    out.writeDouble(NKSpaceScore);
    for(int i=0; i < N; i++)
        { out.writeLong(NKSpace[i]); }
    out.close();
}
catch (IOException ioe)
    { System.out.println("IO Exception"); }
}

public void writeText()
    {
        String title = ("Fitness Scores.txt"); // puts count to
        identify unique genome combination and .nk to show what file used
        for.
        //int[] NKSpaceStore = new int[N];
        try
        { // FileWriter writer = new FileWriter(title);

            BufferedWriter out = new BufferedWriter(new
                  FileWriter(title));
            for (int NK =0; NK < totalNKSpaces; NK++)
                { NKSpaceScore = getFitness(NK);

                    out.write("," + NKSpaceScore);

                }
            out.close();

        }
        catch (IOException e) { }
    }

public void readFile(int NK)
    {
        String file = ("NKSpace_N"+N+"K"+K+"_"+NK+".nk");
Investigating Kauffman's NK Model for Agent-Based Modelling

// *** Load NK Spaces *** //
try {
    DataInputStream in = new DataInputStream(new
    BufferedInputStream(new FileInputStream(file)));
    NKSpaceScore = in.readDouble();
    NKSpace = new int[N];
    //System.out.println(file + " loading...");
    System.out.println("NKSpace: "+ NK + " (read from file):");

    for (int i = 0; i < N; i++)
    {
        NKSpace[i] = in.readInt();
        System.out.print("["+ NKSpace[i]+"]");
    }

    System.out.print("NK Space Fitness Value: "+NKSpaceScore);
    System.out.println();
}
catch (IOException e)
{
    System.out.println("IO Exception" + e);
}

public double getFitness(int nk)
{   // *** Loads NK Space Fitness *** //
    String file = ("NKSpace_N"+N+"K"+K+"_nk.nk");
    double fitness = 0.0;
    try {
        DataInputStream in = new DataInputStream(new
        BufferedInputStream(new FileInputStream(file)));
        //System.out.println(file + " loading...");
        //System.out.println("NKSpace (TO GET FITNESS VALUE): "+
        NK + " (read from file):");
        fitness = in.readDouble();
        //System.out.print("NK Space Fitness Value: "+fitness);
    }
    catch (IOException e)
    {
        System.out.println("IO Exception" + e);
    }
    return fitness;
}

public void printLinks()
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
{
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < K + 1; j++)
        {
            System.out.print("[" + links[i][j] + "]");
        }
        System.out.println();
    }
}

public void Print()
{
    for (int x =0; x < N; x++)
    {
        System.out.print("[" + CountValues[x] + "]");
    }
    System.out.println();
    System.out.println("temp Array");
    for (int x =0; x < N; x++)
    {
        System.out.print("[" + tempCountValues[x] + "]");
    }
    System.out.println();
}

public void Print2()
{
    for (int x =0; x < N; x++)
    {
        System.out.print("[" + NKSpace[x] + "]");
    }
    System.out.println();
}

public void PrintAfitnessValues()
{
    System.out.println("Fill the Scores");
    for (int aVal = 0; aVal < numScores; aVal++)
    {
        for (int locus =0; locus < N; locus++)
        {
            System.out.print("[" + AFitnessValues[aVal][locus] + "]");
        }
        System.out.println();
    }
}

public void printAllFiles()
{
}

93
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
{ 
for (int NK = 0; NK < totalNKSpaces; NK++) 
{ 
    readFile(NK);
}
}

public static void main(String[] args) 
{ 
    // These will come from fields within the model.
    NKSpace_Gen easy = new NKSpace_Gen(System.currentTimeMillis());
    Model model = new Model(System.currentTimeMillis());
    Agent agent1 = new Agent(System.currentTimeMillis());
    for (int runs = 0; runs < numRuns; runs++)
    { 
        agent1.NK = 0;
        agent1.higherFitness = false;
        agent1.peakFound = false;
        System.out.println("Run: " +runs);
        easy.Build_Space();
        // easy.printAllFiles();
        // easy.printAllFiles();
        agent1.selectPoint();
        agent1.getOneMutantNeighbours();
        agent1.travel();
    }
    meanFitness = meanFitness / numRuns;
    System.out.println("After " +numRuns+ " runs, mean fitness =" +meanFitness +" For N=" +N+" K=" +K);
}
```
Investigating Kauffman's NK Model for Agent-Based Modelling

Agent Class

// randomly selects a point on the landscape
// travels the landscape to find highest fitness value

package sim.app.NK_Model;

import java.io.*;
import ec.util.MersenneTwisterFast;
import sim.util.IntBag;
import sim.engine.*;
import sim.display.*;

public class Agent extends Model implements Steppable {

    int[][] OneMutantNeighbours; // stores an NKSpaces one mutant neighbours i.e. if N = 5 it has 5 neighbours
    static double localPeak;
    static double currentFitness;
    int NK;
    int[] currentNKSpace;
    boolean higherFitness;

    int maxLongJumps;
    int jumpAttempt;
    boolean jumpFinished;

    //boolean peakFound;
    //MersenneTwisterFast random = new MersenneTwisterFast();
    //MersenneTwisterFast random = new MersenneTwisterFast();
    //IntBag mutantNeighbours;

    public Agent(long seed) {
        super(seed);
        NK = 0;
        higherFitness = false;
        maxLongJumps = 1;
        jumpAttempt = 0;
        jumpFinished = false;
    }

    public void step(SimState state) {
        if (mutantNeighbours.numObjs > 0 && higherFitness == false)
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
{
    walkLandscape();

    if (higherFitness == true)
    {
        higherFitness = false;
        getOneMutantNeighbours();
    }
    else
    {
        // Prints Each Higher Fitness Found Top is where ended bottom
        // where started.
        System.out.println("*** Higher Fitness Values Found ***");
        System.out.println("Fitness Value: "+ localPeak);
        meanFitness += localPeak;
        peakFound = true;
        stop = true;
    }

    if (peakFound == false)
    {
        System.out.println("*** Current Fitness Searched ***");
        System.out.println("Fitness Value: "+ currentFitness);
    }
}

public void travel()
{
    while (peakFound == false)
    {
        if (mutantNeighbours.numObjs > 0 && higherFitness == false)
        {
            if (quickRun == true)
            {
                walkLand(); // fast
            }
            else
            {
                walkLandscape(); // when whole landscape made.
            }

            if (higherFitness == true)
            {
                higherFitness = false;
                getOneMutantNeighbours();
            }
        }
    }
}
```
Investigating Kauffman's NK Model for Agent-Based Modelling

else {
    // Prints Each Higher Fitness Found Top is where ended bottom 
    where started.
    System.out.println("*** Higher Fitness Values Found ***");
    System.out.println("Fitness Value: "+ localPeak);
    meanFitness += localPeak;
    System.out.println(meanFitness);
    peakFound = true;
}
if (peakFound == false) {
    System.out.println("*** Current Fitness Searched ***");
    System.out.println("Fitness Value: "+ currentFitness);
}

// ** Used in full landscape making ***
public void selectPoint() {
    NK = random.nextInt(totalNKSpaces); //nextInt(int n) Returns 
an integer drawn uniformly from 0 to n-1
    System.out.println("Random NK Space Selected: "+NK);
    currentNKSpace = getNKSpace(NK);
    currentFitness = getFitness(NK);
    localPeak = currentFitness;
    //getOneMutantNeighbours(getNKSpace(locNK), locNK);
}

// used from fast search - randomly selects a point on the 
landscape
public void selectedPoint(int nk) {
    NK = nk;
    System.out.println("Random NK Space Selected: "+NK);
    currentNKSpace = getNKSpace(NK);
    currentFitness = getFitness(NK);
    localPeak = currentFitness;
    //getOneMutantNeighbours(getNKSpace(locNK), locNK);
}

public void getOneMutantNeighbours() {
    mutantNeighbours = new IntBag(N);
Investigating Kauffman's NK Model for Agent-Based Modelling

System.out.print("Selected NK Space: ");
for (int x = 0; x < N; x++)
{
    System.out.print("[" + currentNKSpace[x] + "]");
}
System.out.println();
System.out.println();
for (int loci = 0; loci < N; loci++)
{
    int tempNK = NK;
    if (currentNKSpace[loci] == 1)
    {
        tempNK = tempNK - CountValues[loci];
        if (tempNK < 0)
        {
            tempNK = tempNK + totalNKSpaces; // wraps around to begin of array again
        }
    }
    else
    {
        if (currentNKSpace[loci] == 0)
        {
            tempNK = tempNK + CountValues[loci];
            if (tempNK > totalNKSpaces)
            {
                tempNK = tempNK - totalNKSpaces; // wraps around to begin of array again
            }
        }
    }
    mutantNeighbours.add(tempNK);
    // for (int k = 0; k < mutantNeighbours.numObjs; k++)
    // {
    //     System.out.println("Mutant Neighbour NK Number From IntBag: " + mutantNeighbours.objs[loci]);
    // }
    // printOneMutNeighbours(mutantNeighbours);
    // System.out.println("Travel Landscape!");
    // walkLandscape(currentNKSpace, NK, mutantNeighbours);
}

public void walkLandscape()
{
    int tempNK;
    double NKSpaceFitness;
double tempNKSpaceFitness;
//boolean higherFitness = false;

NKSpaceFitness = getFitness(NK);

//while (mutantNeighbours.numObs > 0 && higherFitness == false)
//{
    int randomSelect = random.nextInt(mutantNeighbours.numObjs);
    tempNK = mutantNeighbours.objs[randomSelect];
    tempNKSpaceFitness = getFitness(tempNK);
    currentFitness = tempNKSpaceFitness;

    if (tempNKSpaceFitness > NKSpaceFitness)
        {   higherFitness = true;
            localPeak = tempNKSpaceFitness;
            // need to call methods to search again with new higher
            // fitness as the main one
            NK = tempNK;
            currentNKSpace = getNKSpace(NK);
            System.out.print("*** Higher Fitness Neighbour Found: ");
            for (int i = 0; i < N; i++)
                {  System.out.print("["+currentNKSpace[i]+"]");
                }
            System.out.println(" Fitness is: "+tempNKSpaceFitness);
        }
    else
    {  mutantNeighbours.remove(randomSelect); // check it resizes
    }
// Prints Each Higher Fitness Found Top is where ended bottom
//System.out.println("*** Higher Fitness Values Found ***");
//System.out.println("Fitness Value: "+ NKSpaceFitness);

// *** For fast build ***
public void walkLand()
{
    int tempNK;
    double NKSpaceFitness;
    double tempNKSpaceFitness;
    //boolean higherFitness = false;

    NKSpaceFitness = getFitness(NK);
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
// while (mutantNeighbours.numObjs > 0 && higherFitness == false)
// {
    int randomSelect = random.nextInt(mutantNeighbours.numObjs);
    tempNK = mutantNeighbours.objs[randomSelect];
    NKSpace_Gen nkgen = new NKSpace_Gen(System.currentTimeMillis());
    nkgen.singleSpace(tempNK);
    tempNKSpaceFitness = getFitness(tempNK);
    currentFitness = tempNKSpaceFitness;
    if (tempNKSpaceFitness > NKSpaceFitness)
    {
        higherFitness = true;
        localPeak = tempNKSpaceFitness;
        // need to call methods to search again with new higher
        // fitness as the main one
        NK = tempNK;
        currentNKSpace = getNKSpace(NK);
        System.out.print("*** Higher Fitness Neighbour Found: ");
        for (int i = 0; i < N; i++)
        {
            System.out.print("["+currentNKSpace[i]+"]");
        }
        System.out.println(" Fitness is: " +tempNKSpaceFitness);
    }
    else
    {
        mutantNeighbours.remove(randomSelect); // check it resizes
    }
}

public double getFitness(int nk)
{
    // *** Loads NK Space Fitness*** //
    String file = ("NKSpace_N"+N+"K"+K+"n"+nk+".nk");
    double fitness = 0.0;
    try
    {
        DataInputStream in = new DataInputStream(new
        BufferedInputStream(new FileInputStream(file)));
        System.out.println(file+" loading...");
        // System.out.println("NKSpace (TO GET FITNESS VALUE): "+ NK+
        (read from file):");
        fitness = in.readDouble();
        // System.out.println("NK Space Fitness Value: "+fitness);
    } catch (IOException e)
```
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
{
    System.out.println("IO Exception" + e);
}
return fitness;
}

public int[] getNKSpace(int nk)
{
    // *** Load NK Space Loci Combination *** //
    String file = ("NKSpace_N"+N+"K"+K+"_nk"+nk+".nk");
    int[] NKSpace = new int[N]; // Stores a combination of 0,1
    try
    {
        DataInputStream in = new DataInputStream(new
            BufferedInputStream(new FileInputStream(file)));
        double temp = in.readDouble();
        //System.out.println(file+" loading...");
        //System.out.println("NKSpace: "+ NK+" (read from file):");
        for (int i =0; i < N; i++)
        {
            NKSpace[i] = in.readInt();
            //System.out.print("["+ NKSpace[i]+""])"
        }
    }
    catch (IOException e)
    {
        System.out.println("IO Exception" + e);
    }
    return NKSpace;
}

public void printOneMutNeighbours(IntBag mutNeighbours)
{
    OneMutantNeighbours = new int[N][N];
    int[] NKSpace = new int[N];
    for (int k = 0; k < mutNeighbours.numObjs; k++)
    {
        NKSpace = getNKSpace(mutNeighbours.objs[k]);
        for (int i =0; i < N; i++)
        {
            OneMutantNeighbours[k][i] = NKSpace[i];
        }
    }
}
```
System.out.println();

System.out.println("1 Mutant Neighbours:");

for (int aVal = 0; aVal < N; aVal++)
{
    for (int locus = 0; locus < N; locus++)
    {
        System.out.print("[" + OneMutantNeighbours[aVal][locus] + "]");
    }
    System.out.println();//" Fitness: " + NKSpaceScore);
}

public void jump()
{
    ...
}

public void longJump(int nk)
{
    NK = nk;
    while (jumpFinished == false)
    {
        int tempNK;
        double NKSpaceFitness;
        double tempNKSpaceFitness;
        NKSpaceFitness = getFitness(NK);
        if (jumpAttempt < maxLongJumps)
        {
            MersenneTwisterFast random = new MersenneTwisterFast();
            tempNK = random.nextInt(totalNKSpaces); // nextInt(int n)
            System.out.println("Random NK Space Selected: " +tempNK);
            tempNKSpaceFitness = getFitness(tempNK);
            System.out.println("*****" + tempNKSpaceFitness);
            if (tempNKSpaceFitness > NKSpaceFitness)
            {
                NK = tempNK;
                NKSpaceFitness = tempNKSpaceFitness;
                maxLongJumps = maxLongJumps * 2;
            }
            else
            {
                jumpAttempt ++;
            }
        }
    }
}
else
{
Investigating Kauffman's NK Model for Agent-Based Modelling

```java
{  
jumpFinished = true;
    System.out.println("Jump finished highest fitness found: "+ NKSpaceFitness);

```
}