Fault tolerance of cooperative and competitive coordination strategies in distributed information processing systems

Gajus Richard Dirkzwager
gdirkzwager@gmail.com
Radboud University Nijmegen

Supervisors: dr. Ida Sprinkhuizen-Kuyper and dr. Paul Kamsteeg

June 20, 2012

Abstract

In this study we employ cellular programming to design both cooperative and competitive mechanisms that coordinate social interactions between computational units in distributed information processing systems. Through a series of experiments based upon computer simulations, it is argued that units supplied with a healthy amount of competitive attitude towards one another are beneficial for the robustness of the system as a whole. Moreover, we argue cooperative coordination mechanisms are only stable when we can rely on the structural integrity of all computational units in the system.

Keywords: Evolution of Cooperation, Iterated Prisoners Dilemma, Multi-Agent Systems, Cellular Programming, Cellular Automata, Genetic Algorithms, Parallel Computation.

Contents

List of Figures

1 Introduction

Many modern day computing systems and information environments are distributed. For example, connected to the Internet, a computer has access to an almost unimaginable playground of information and resources scattered throughout the world. In fact, "distributed computation has become the dominant computational paradigm today" (Weiß, 1999).

Compared to centralised computing systems, distributed computing systems have a number of advantages. For example, as the number of computational elements which combine forces increase, so does the capacity to process information. In addition, distributed systems are also renown for their ability to perform in an acceptable manner while operating under faulty conditions (Sipper, 1997). Also, the development and maintenance of the increasingly complex computer systems and their applications can be constrained by breaking up systems into manageable parts.

It is also obvious that the lack of a central authority considerably complicates matters on a number of important issues. For example, it is not all that clear how to express computational tasks to a distributed collection of information processors. In many cases, decomposing problems into convenient parts requires at least some broad knowledge of the problem. This disagrees with the very nature of these systems as no computational unit has complete knowledge of the problem. A similar complication arises during assembly of an overall solution from sub-problems solved by individual computing elements. Between decomposing a problem and finding its solution lies yet a third challenge: how to coordinate the problem solving activities of individual processors in an efficient manner. According to Weiß(1999), "Two basic contrasting patterns of coordination are cooperation and competition. Cooperation involves working together by sharing knowledge and resources in order to achieve a common goal. In contrast, competition involves rivals in pursuit of a common goal. Cooperating computational elements try to accomplish as a team what individuals cannot, and so fail or succeed together. Competitive computational elements try to maximise their own benefit at the expense of others, and so the success of one implies the failure of others".

So what does this failure of others imply when in the future, computing systems may contain many computational elements. For systems containing such a large number of components, the issue of resilience can no longer be ignored since failure will be likely to occur with high probability (Sipper, 1997). Because computing elements which apply cooperative mechanisms to coordinate their activity depend on others, failure is likely to cause less harm in systems where computing elements apply competitive mechanisms to coordinate their activity. In short, our study is motivated by the following question:

Is fault tolerance of distributed processing systems which coordinate interactions amongst computational elements through competitive mechanisms better, compared to fault tolerance in distributed processing systems in which interactions between computational elements are coordinated by cooperative mechanisms?

To be able to investigate this question, we need two different types of distributed information processing systems which we can subject to various degrees of damage: one system in which coordination of interactions between individual computational units is accomplished through competition, the other in which coordination of interactions between individual computational units is accomplished through cooperation. By causing damage to the systems and then measuring which mechanism is more successful in coordinating interactions between processor units we can determine which type of system maintains its identity better under faulty conditions.

In the following sections, we will outline the methodology being used in this study. Next, the discussion will focus on the experimental setup. Finally the results and conclusion are discussed.

2 Methods

To be able to test our hypothesis, we need to decide upon a number of important issues. First we need to choose a computational model to represent the class of distributed information processing systems. Second we need to decide upon how to model cooperative and competitive behaviour. Third, we need to define the type of damage we afflict on the systems. Finally, we need a measure of how well (or poor) units in a system are able to coordinate their interactions with other units.

2.1 The Cellular Automata (CA) Model

A number of distributed information processing models qualify as suitable representatives. The most obvious candidates are the connectionist networks (CN), for example the interactive activation and competition model. To model cooperation for example, we could use excitatory connections between (groups of) neurons. Competition could then be modelled using inhibitory connections between (groups of) neurons. CN also display graceful degradation of performance when confronted with noisy input or erroneous operation of neurons. A major drawback of using this method is that it is difficult to determine precisely when a neuron is cooperative and when it is competitive as these values are measured on a continuous scale. We wish to use discrete representations of cooperative and competitive behaviour. Another drawback is that it seems reasonable from the human psychological perspective to understand cooperation as excitation and competition as inhibition, just like humans associate loud sounds with height, in nature this need not necessarily be true. In addition neurons that release neurotransmitters can have excitatory effects on some neurons and inhibitory effects on others. For this reason CN are less suitable.

An alternative candidate model for distributed information processing is grid computing. The best known example of grid computing is the Search for Extra

Terrestrial Intelligence or SETI project which uses computational resources from computers all over the world to search for intelligent radio signals from outer space. The problem with grid computing is however, that generally speaking grids still need a central authority to divide a problem into pieces and allocate different computational resources. Each resource is provided with a piece of the problem so that interaction between resources is minimised. So, although each computational resource is autonomous when it comes down to solving its own piece of the problem, the central authority is responsible for coordinating interactions amongst computational units. This makes these systems highly vulnerable to damage.

Cellular Automata (CA) represent a third general class of computational models in which distributed information processing occurs in parallel amongst autonomous processors. CA represent a viable model for a number of different reasons. First, CA provide a highly abstract discrete and simplified representation of reality, this allows us to simulate and visualise various systems and their components with very limited means. Second, CA support universal computation, this makes our model extremely powerful. For these reasons we intend to use CA as representative for the class of distributed information processors.

The earliest study of CA can be traced back to John von Neumann and his work on self replicating machines during the 1940s. The details were published after his death in the book *Theory of Self-Reproducing Automata* edited in 1966 by Arthur Burks (von Neumann and Burks, 1966). CA gained widespread popularity amongst the general public through a column by Martin Gardner (1980) in Scientific American who reported on the work of the Mathematician John Conway and his Game of Life.

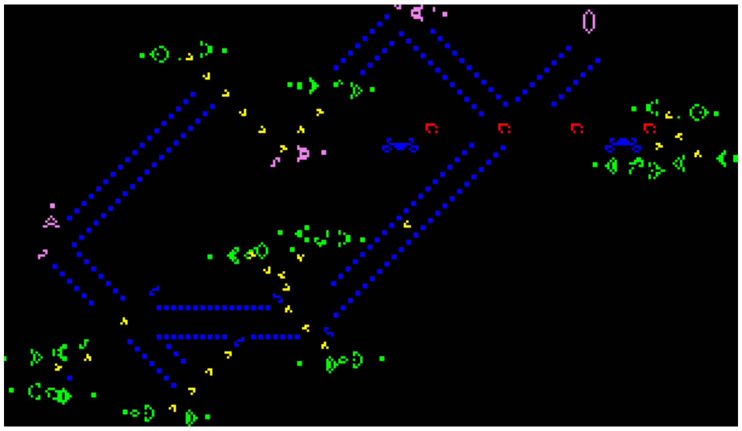


Figure 1: Conway's Game of Life. Green: Gun. Yellow: Glider. Red: Spaceship. Violet: Reflector. Blue: Eaters. [1]

It is difficult to captivate the true beauty of this automaton in one single

[1]http://en.wikipedia.org/wiki/File:Color_coded_racetrack_large_channel.gif

image. Figure 1 does however give a good impression of how we can imagine a CA. Traditionally processors are referred to as cells. Cells are usually aligned on a one or two dimensional array, but any finite number of dimensions are possible. At any point in time, each cell can be in one of a finite number of different states. In the Game of Life, there are two states, “alive”, white or “dead”, black (the different colours in the image simply identify the different “alive” forms). The state of a cell at $t + 1$ is determined by the current states of the other surrounding cells, and possibly its own.

Often, the function that maps states from one point in time to the next is referred to as the neighbourhood transition function or neighbourhood function.

$$\begin{array}{ccccc} & \text{N} & & & \\ \text{W} & & \text{E} & \mapsto & \text{C'} \\ & \text{S} & & & \end{array}$$

Figure 2: CA 5-Cell neighbourhood

Figure 2 shows a neighbourhood of five cells. The central cell and four other cells to the north, east, south and west. In the Game of Life for example the neighbourhood function can be characterised as follows:

- Each live cell with two or three live neighbours stays alive, otherwise it dies.
- Each dead cell comes to life with exactly three live neighbours.

Figure 3 shows an example of a CA which we will refer to as NESW555. The first part of its name comes from its four cell neighbourhood. The second part refers to the decimal representation of its binary rule: 0000001000101011, see Table 1 below. Depending on the states of cells to the north, east, south and west, each central cell can be in one of two states. Like the Game of Life, in NESW555 in Figure 3 each cell contains the same rule, or put differently, applies the same neighbourhood function. This situation is called a uniform CA. When cells contain different rules we refer to the CA as non-uniform.

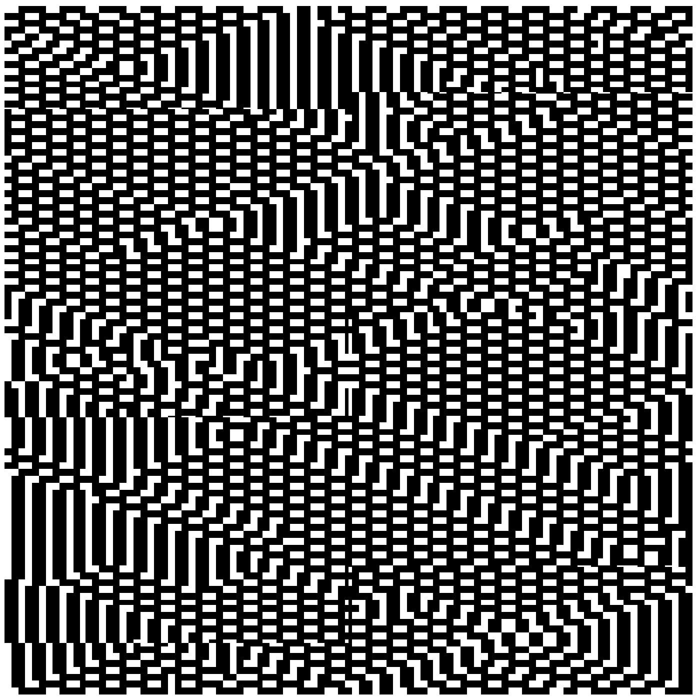


Figure 3: NESW555 after a simulation of 200 epochs

We can also represent the neighbourhood function of CA NESW555 more formally using a rule table.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
N_t	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
E_t	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
S_t	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
W_t	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
C_{t+1}	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	1

Table 1: Bottom row shows the rule NESW555. The top 5 rows showing the binary and decimal representations of neighbourhood configurations that map towards a particular bit in the rule.

Table 1 shows how the state of a cell at the next point in time, C_{t+1}, corresponds to the configuration of the neighbouring states N_t, E_t, S_t and W_t at this point in time. So for example, C_{t+1} will be 1, or "alive" only when N_t, E_t, S_t and W_t are either (1,0,0,1), (1,0,1,0), (1,1,0,0), (1,0,0,0) or (0,0,0,0). Periodic boundaries are used such that cells at opposite ends of the grid are neighbours of each other. This creates a toroidal grid, somewhat like a donut.

One of the main problems computer programmers run into when dealing with distributed computation, is how to control the global behaviour of the system with instruction sets that provide only local operators. A simple calculation shows one can construct $2^{(2^4)^{2500}}$ different CA from a grid containing $50 * 50$ cells with a neighbourhood of 4 and each cell allowed to be in one of two possible states. To program each cell individually is an impossible task for any human programmer. To design global behaviour of a CA from the bottom up, we need a different approach. For this reason we will take a look at Cellular Programming which we will discuss in the next section.

2.1.1 Cellular Programming (CP)

According to Talia (2003), "it is better to design emergent systems by means of paradigms that allow for expressing the behaviour of the single simple elements and their interactions. The global behaviour of these systems then emerges from the evolution and interaction of a massive number of elements; hence it does not need to be explicitly coded". With this in mind we shall apply the framework of CP presented by Moche Sipper (Sipper, 1997) which involves designing parallel cellular machines through the application of artificial evolution to solve global computational problems.

When compared to the classic CA model, the extended model by Sipper has a number of important differences. First and foremost: not all cells have to be operational. Some cells my be vacant meaning they contain no rule table and can be regarded as empty or inactive. This means that instead of one, we need two bits to characterise a cell. One bit representing the state of a cell: 0 for defect, 1 for cooperate. The second bit representing the mode of a cell: 0 for vacant and 1 for operational. Although the term "alive" and "dead" might suggest cells in the Game of Life may be either vacant or operational, there is no concept of mode. In the Game of Life, "alive" and "dead" are simply labels invented to describe different states.

Another important difference is that cells are not only able to change their own state, but also the state of neighbouring cells. This means one cell can force one of its neighbours to defect or cooperate the next point in time. A third important difference is that different rules can spread throughout the CA because operational cells are able to "wake up" or "bring to life" vacant cells by copying their rule table into the vacant cell. This then changes the mode of a cell from vacant to operational. Cells cannot change the mode of other cells from operational to vacant.

To summarise, a cell maintains both a state and a mode. A state refers to a cell's disposition towards other cells. Defect as a negative disposition toward working together with other cells, and cooperate as a positive disposition toward working together with other cells. A mode refers to a cell's operational capability, vacant when a cell contains no rule table and as such has no direct influence on the behaviour of other cells, operational when a cell does contain a rule table which allows it to interact and directly influence the behaviour of other cells. Compared to the classic CA model, cells can also be regarded as more active because they can directly influence the state of other cells and indirectly influence the mode of other cells.

$$\begin{array}{ccccccc} & N & & & & N' & \\ W & C & E & \mapsto & W' & C' & E' \\ & S & & & & S' & \end{array}$$

Figure 4: CP 5-Cell neighbourhood

Figure 4 shows how the neighbourhood function of the CP model relates to a state of a cell and that of the other cells surrounding it. The neighbourhood function does not take into account the mode of other cells, of course it would be interesting to extend the model of Sipper to study the effect this has. The neighbourhood consists of a total of $2^5 = 32$ different configurations which are mapped upon a rule table containing between 1 and 32 different rules, each rule g_i containing a series of $5 * 2 = 10$ instructions: two bits for each cell in the neighbourhood. A "state change" bit S_x which specifies the state the respective cell becomes the next point in time, and a "copy rule" bit C_x which specifies if the cell wants to copy its entire rule table into another cell (or itself of course). In addition, cells may contain different rules which means the CP model is a non-uniform CA.

To better understand how the neighbourhood function operates, we adopt some notation from Sipper.

$$CNESW \Rightarrow Z_C Z_N Z_E Z_S Z_W \tag{1}$$

Where $CNESW$ represents the neighbourhood configuration and $Z_x = (S_x, C_x), x \in \{C, N, E, S, W\}$.

g_i - rule i where i corresponds to the decimal representation of the binary configuration of states in the neighbourhood in the order $CNESW$. Thus, neighbourhood configuration 01111 corresponds to g_{15}. In contrast to the CA in Figure 3, each of the $2^5 = 32$ different neighbourhood configurations can map to a different rule. In total there are $2^{(5*2)} = 1024$ different rules that can be expressed by being associated to one of the 32 different neighbourhood configurations. This means that besides each cell being able to contain a different rule table, each of the 32 different configuration of states in the neighbourhood can lead to the expression of a different rule.

S_x - state-change bit, where $x \in \{C, N, E, S, W\}$ denotes one of the five neighbours. This bit specifies the state change to be effected upon the appropriate neighbouring cell. For example, $S_E = 0$ means "change the east cell's state to defect". When $S_C = 1$ the bit means "change the central cell's state to cooperate". State changes can be effected by operational cells on both operational and vacant cells.

C_x - copy-rule bit, where $x \in \{C, N, E, S, W\}$ denotes one of the five neighbours. This bit specifies whether or not an operational cell has to copy its entire rule table into cell x. It is important to note the difference in semantics compared to the S_x bit. Where the S_x bit has a direct influence on the state of cell x, the C_x bit can only indirectly influence the mode of vacant cells which changes to operational when a rule table has been received. When a rule table is copied into an operational cell, this cell remains operational.

Because the CP framework operates on a five cell neighbourhood function, it becomes impractical to visualise the rule table in a manner similar to Table 1. In practice, there are only two important differences. First, the rule table consists of not 16 but 32 different configurations. Thus, there are 32 columns indexed by the binary configuration of states $CNESW$, each column g_i representing one of the 1024 rules. Second, instead of the neighbourhood function mapping on a one-bit action specified in the C_{t+1} row of Table 1, the neighbourhood function of the CP model maps towards a ten-bit action specifying $Z_C Z_N Z_E Z_S Z_W$. This gives us a rule table of 32 columns wide and 10 rows high where each column specifies rule $Z_C Z_N Z_E Z_S Z_W$.

There is the possibility for contention to occur when two or more cells try to copy their state or rule table into a third cell. This situation is resolved by the cell that is subject of the dispute by appointing randomly one of the contenders (possibly itself).

2.1.2 Evolutionary Design

In an investigation into the social structure of territoriality by Axelrod, a CA was constructed with help of experts in game theory from around the world who submitted a computer program which plays the Iterated Prisoners Dilemma (IPD) (Axelrod, 1984). Each program was assigned randomly to a number of different cells. Each iteration, a program attains a success score measured by the average payoff against its four neighbours. Then, if a program has one or more successful neighbours, the program converts to a more successful program occupying one of its neighbours. Because the programs are defined at the offset of the simulation, and are not allowed to adapt to changing environmental conditions over time, the approach of Axelrod is unrealistic in many real life situations (Sipper, 1997).

We have already seen that CP provides cells with a mechanism to interact with other cells in their environment. However, we still don't have a mechanism to form new rules and to design global behaviour. Put otherwise, we can manually program the rule table of cells in a way similar to Axelrod. Like with Axelrol, no new behaviour will ever emerge.

To deal with this problem we need to augment the computational process with an additional step. Similar to Axelrod, we start by evaluating the behaviour of individual cells and assign a score to that behaviour. We then construct new rules by selecting successful cells and combining their rules with the rules of other successful cells in their neighbourhood. Success or fitness can be measured in a number of different ways. Poli, Langdon, and McFee (2008) suggest one of the following:

- The amount of error between its output and the desired output.
- The amount of time (fuel, money, etc.) required to bring a system to a desired target state.
- The accuracy of the program in recognising patterns or classifying objects.
- The payoff that a game-playing program produces.
- The compliance of a structure with user-specified design criteria.

Because we focus on distributed system with no global operators and as such have to evaluate performance of individual cells to obtain a desired behaviour of the system on a global level, not all of the above mentioned measures are equally suitable. If however, we interpret fitness in the context of a game playing situation we can build upon the prisoners dilemma that has been studied extensively as a mechanism to design global cooperative behaviour. This means we can also use the payoff mechanism of the prisoners dilemma as a fitness measure to quantify how well a cell has done. Table 2 shows the payoff matrix we apply to the interactions between cells:

	Cooperate	Defect
Cooperate	3	0
Defect	5	1

Table 2: Payoff matrix

In this matrix defect refers to a negative inclination towards working together with other cells. Consider the following situation: (D, C), cell A defects and cell B cooperates. We position cell A as the row player and cell B as the column player. The payoff for cell A becomes 5. To determine the payoff for cell B, we position cell B as the row player and cell A as the column player. The payoff for cell B becomes 0, the sucker's payoff. The total payoff for cells A and B is also 5 which is a suboptimal score. If both cells would be willing to risk the sucker's payoff and choose to cooperate, the total payoff would be 6. In a single round prisoners dilemma, defection is the rational strategy because a cell is always assured a minimal payoff of 1 when it defects. With a cooperative strategy a cell can only assure itself of a minimal payoff of 0. For any cooperation to emerge amongst rational players (which we assume our cells to be), the game must be played more than once, to be more precisely infinitely. Then cells can build up trust relations and take previous actions of their counterparts into account.

A cell's total fitness during one generation is computed as the sum of the payoffs a cell receives after playing one game of the IPD with each its four neighbours. Only operational cells play the game and build up payoff, the opponent however does not need to be operational. When the total payoff of all cells has been computed we can start with the process of natural selection. First all operational cells with a fitter neighbour die. This leaves (clusters of) cells having the same fitness. The offspring for the next generation is produced by having the remaining operational cells mate with another operational cell in their neighbourhood. By applying the genetic operators crossover and mutation we obtain the new rule table for the cells' offspring. To keep in line with the Genetic Algorithms literature we shall refer to a rule table as the cells genome which consists of 32 different genes instead of rules. In turn each gene can be associated with one of 1024 different expressions. Crossover is then performed by a cell by randomly selecting an operational neighbour and then copying (part of) its neighbour's genes into its own genome. If no other operational cells are present in the neighbourhood, a cell has to mate with itself. In this case crossover is useless. Otherwise the crossover point is determined by the following equation:

$$\frac{f(i,j) + f(i_n, j_n)}{2 * 4 * (D,C)} \tag{2}$$

Where $f(i,j)$ denotes the total payoff of the first cell and $f(i_n, j_n)$ denotes the total payoff of the second cell. Furthermore (D,C) denotes the maximum payoff possible, that of defecting while the opponent cooperates. In this way the more payoff cells receive, the more genetic material is copied during crossover. So, if we think of the genome containing 32 columns and 10 rows, a crossover point of 0.60 means the bits in the last 6 rows are copied from one genome to the other. By slicing genes in half and copying rows, the pieces of genetic material that are selected for crossover cover all neighbourhood configurations with equal probability. If on the other hand columns (entire genes), are copied, the probability that g_{31} is copied, would be considerably higher than the probability g_0 is copied. A second reason is that slicing genes also facilitates the assembly of new genes by which new types of behaviour can emerge. Experiments show however, that both row based and column based crossover behave quite similar, ultimately leading to the same end result. Note that the crossover operator is asymmetrical which is somewhat different compared to the classic Genetic Algorithms approach. It is argued by Sipper that this slightly decreases coupling between cells, thus enhancing locality and generality (Sipper, 1997).

Finally, the mutation operator is applied according to some fixed probability, similar to Sipper we set the probability of the mutation operator being applied to 1%. This means each bit in the entire genome has about 1% probability of being flipped. In contrast to crossover, operational cells that have to mate with themselves because no operational neighbours exist are subject to mutation. To recapitulate the procedure for one generation of cells is:

1. In parallel: Let all operational cells execute a rule (gene expression) from their rule table (genome) according to the configuration of states in their neighbourhood. State changes are set according to the S_x bits in the rule. Rule tables are copied where C_x bits are 1 and contentions are resolved.

2. In parallel: Make vacant cells that have received a new genome from one of their neighbours operational.

3. In parallel: Let all operational cells calculate the value of their gene expression by playing one game of the prisoners dilemma with all of their neighbours. Cells don't play the game with them selves. When an operational cell plays the game its opponent does not have to be operational, the operational cell is however the only one that receives payoff.

4. In parallel: Evaluate fitness of all cells which have previously played one round of the IPD with each of their neighbours. Cells with fitter operational neighbours become vacant.

5. In parallel: First apply the crossover and then the mutation operator to the genome of all remaining operational cells.

2.2 Conditions

Of course there are many ways in which a distributed system can be damaged to the extent that normal operation becomes difficult. When only a few isolated components are damaged, error occurs at local regions in space. Another possibility is that damage occurs uniformly amongst certain types of components, producing errors throughout the system. Wires for example, are often much more exposed and vulnerable when compared to resistors or a LED. Also, the degree to which a damaged component produces errors can vary. In practice, damage is most likely to occur in varying degrees at different physical places in space. To avoid complicating matters, our conceptual understanding of damage applies only to systems where damage occurs with the same probability amongst all components that allow cells to communicate with other cells. For example, imagine wires are soldered in some poor way. As a result, noise is added to each bit transferred through the wire causing misinterpretations a certain amount of time. To maintain the reciprocal dependency at each end of the wire, a cell takes account not only for its own misinterpretations, but those of its counterpart too. So, if for example cell A transmits a 0 and cell B transmits a 1, furthermore cell B misinterprets cell A, the game becomes $(1, 1)$ for both cells and they receive payoff accordingly.

The different conditions are specified by various degrees to which bits transferred between cells are incorrectly perceived. The base condition specifies a zero probability of component error in order to establish a measure of performance while the system operates under normal conditions. The second condition specifies one percent probability that each bit transferred is perceived incorrectly. The motivation for this value is based upon the work of Axelrod on the evolution of cooperation (Axelrod, 1984). In his famous IPD computer tournaments each player is assumed to have perfect knowledge about other players' prior actions. To explore the implications of misperception, he ran the first round of the tournament again with the modification that the choice of every player had a one percent chance of being misperceived by the other player. The results indicate a great deal more defection but show the cooperative strategy TIT FOR TAT was still the best (Axelrod, 1984).

To summarise, there are two conditions. One in which all communications between cells function properly and a second in which communications between cells fail to work properly 1% of the time.

2.3 Experiment

The experiment was set up using two groups of 50 randomly generated systems each containing a total of $50 * 50$ cells, this amounts to a population size of 2500 individual cells in each system. The first group of 50 systems consists of competitive systems, the second group of cooperative systems.

In the following sections, all stages that involve the experiment on one system will be discussed. First, we will consider the initial setup configuration that we use to kickstart the evolutionary process. Next we will discuss the training of the system and how we measure the degree and type of coordination in a system. In the final section we will discuss the second stage of the experiment in which a system is subjected to both conditions.

2.3.1 Initial Conditions

At offset, the state of each cell was randomised to cooperate or defect with about 50% probability. Also, the initial genetic makeup of each cell was selected randomly from a pool of genes found to be common during pilot simulations of both types of systems. In cooperative systems for example, gene expressions occur in a number of different ways. If we take a closer look at the content of g_{31} which is a nice example of a gene that is expressed often in cooperative systems (neighbourhood configuration 11111), we see that g_{31} is associated with a number of different gene expressions. Most commonly the expression 1023 that written in binary (1111111111) contains only S_x and C_x bits that are 1. On occasion one of the C_x bits may be 0. For example expression 831 (1110111111) which sets all states in the neighbourhood to cooperate and copies its genome to all cells in the neighbourhood with exception to its neighbour to the north. Competitive systems however, tend to be composed of expressions associated with g_{15} which corresponds to neighbourhood configuration 01111. An example of one such expression is 597 or 1001010101. This expression causes cells to be locked into a pattern of alternate defect because it changes the current cells state to become cooperate, and the genome table to be copied into all neighboring cells, with their state changed to defect. This is an interesting strategy in that an operational cell ensures cooperation of the cell it occupies and then defects to a neighboring cell (Sipper, 1997).

The number of vacant cells was set to 15%, this leaves a larger amount of operational cells having to compete with each other. As such, the initial gene population should contain a larger variety of genes.

2.3.2 Training

Each experiment consists of two separate stages. During the first stage we try to evolve one of either systems. Sit back and let evolution take its course until a sufficiently cooperative or competitive system has evolved. On average it takes about 200 generations to evolve cooperative systems, competitive systems take on average another 800 extra. The reason for this is that both cooperative as competitive cells benefit from having cooperative neighbours. The resulting systems often offer nice surprises. For example, the right side of Figure 5 shows a cooperative system with a few "pockets of resistance". This is most representative for cooperative systems. It does however occur that all 2500 cells cooperate, where after some time you see that due to genetic mutations competitive cells slowly start to gain the upper hand. The right hand side of Figure 5 shows a typical competitive system with only a few operational cooperating cells. This is most likely caused by genetic mutations because when the mutation operator is set to 0 a system of absolute alternate defection evolves in which only one gene is likely to survive. Convergence towards cooperative systems does however occur about five times more often compared to competitive systems. If after 3000 generations no sufficient coordination mechanism has evolved, the model is discarded.

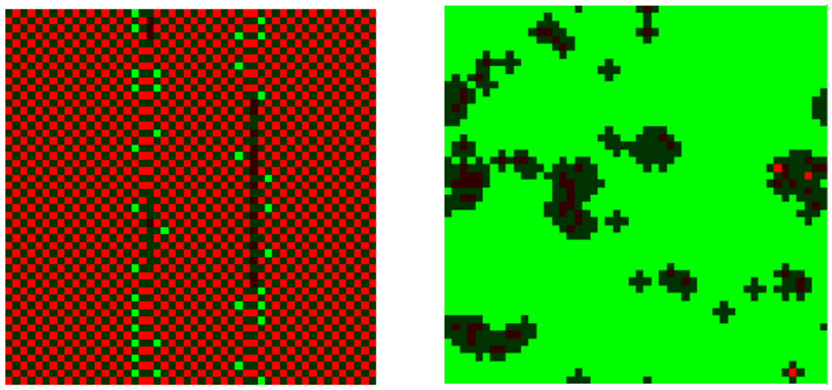
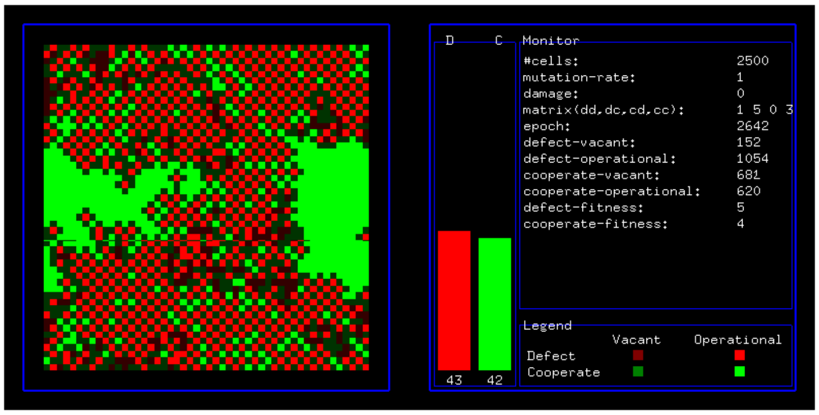


Figure 5: Typical Subjects. On the left hand side a competitive system where defecting cells occupy most of the space, on the right hand side cooperating cells dominate. Light Green: Operational-Cooperate. Dark Green: Vacant-Cooperate. Light Red: Operational-Defect. Dark Red: Vacant-Defect.

Figure 5 above shows an impression of two system representatives. In order to determine if a system is suitable to participate further in the experiment, we first need to classify it in either cooperative or competitive. Second, we need to be able to evaluate this global behaviour over time. To make the situation more concrete, consider Figure 6.



```
Monitor
#cells:                      2500
mutation-rate:               1
damage:                      0
matrix(dd,dc,cd,cc):         1 5 0 3
epoch:                       2642
defect-vacant:               152
defect-operational:          1054
cooperate-vacant:            681
cooperate-operational:       620
defect-fitness:              5
cooperate-fitness:           4
```

Legend

	Vacant	Operational
Defect	🟥	🟥
Cooperate	🟩	🟩

Figure 6: Simulation console. On the left hand side a graphic visualisation of a system. On the right hand side the system monitor which allows to observe different variables.

Figure 6 shows a system where both cooperative and competitive strategies have a considerable momentum. At this stage, the case is quite obviously (still) undecided, it does however illustrate why we cannot simply count red and green operational cells to classify a system. Competitive cells benefit only at the expense of others and therefore competitive systems harbour less operational cells, at most half of all cells at one point in time. In addition this method would provide an indirect measure of how well cells coordinate their behaviour with other cells. If for example all cells cooperate, we could infer cooperative cells must have been quite successful. What we need is a measure that reflects the degree to which cells succeed in their interactions with other cells.

If we assume that the more the states of other cells in the neighbourhood of one cell corresponds to the desires of that cell the previous point in time, the better the cell was able to coordinate his interactions with other cells. To get a global picture of how well cells following a particular strategy are able to coordinate their interactions with other cells, we simply have to calculate the average degree of coordination amongst cells following that particular strategy. This can be viewed on the left hand side of the system monitor in Figure 6 which shows coordination amongst competitive cells is about 0.43 and coordination amongst cooperative cells is about 0.42. To understand how we compute the degree of coordination, consider the following situation in Figure 7:

g_i	C	N	E	S	W	S_C	C_C	S_N	C_N	S_E	C_E	S_Z	C_Z	S_W	C_W
12	0	1	1	0	0	0	1	1	0	1	0	1	0	1	0

Table 3: Entry in the genome table of the central cell in Figure 7 showing expression 426 (0110101010) that was associated with gene 12 (01100) at t=x.

Table 3 shows part of the genome table of the central cell of Figure 7, in

$$\begin{array}{|c|c|c|}\hline \text{x} & 1 & \text{x} \\ \hline 0 & \mathbf{0} & 1 \\ \hline \text{x} & 0 & \text{x} \\ \hline \end{array} \mapsto \begin{array}{|c|c|c|}\hline \text{x} & 1 & \text{x} \\ \hline 0 & \mathbf{0} & 1 \\ \hline \text{x} & 1 & \text{x} \\ \hline \end{array}$$

Figure 7: Example of two consecutive points in time. Left *t=x* where the central cell is surrounded by two cooperating cells and two defecting cells. Right, *t=x+1* where the central cell is surrounded by one defecting cell and three cooperative cells.

particular the expression 426 associated with gene 12. The central cell has attempted to set the state of all cells in its neighbourhood except for itself into "cooperate" and has succeeded in four out of five of its objectives, this means its rate of success is $\frac{4}{5}$. To get a better picture of how we calculate the degree of coordination in the entire system, lets assume that there are 2000 operational cells of which 1990 cells are in state cooperate and 10 cells are in state defect. Furthermore, of the 5 cells in the neighbourhood of a cooperative cell, an average of 4.98 cells have actually become what was specified by the S_x bitsof that cell. The degree of cooperative coordination amongst operational cells then becomes $0.991 = \frac{1990*\frac{4.98}{5}}{2000}$. Furthermore, if we assume that 4.8 out of 5 neighbours of a competitive cell actually have become what was specified by the S_x bit of that cell, we can calculate the degree of competitive coordination amongst operational cells as follows: $0.005 = \frac{10*\frac{4.8}{5}}{2000}$. We can summarise this using the following equations:

$$C = \frac{\sum\limits_{x=0}^{2499}(M(x) = 1 \wedge S(x) = 1) * \sum\limits_{x=0}^{2499} \frac{\sum\limits_{r=0}^{4} G(x)[i][2*r] \wedge S(N(x)[r])}{5}}{\sum_{x=0}^{2499} M(x) = 1} \tag{3}$$

$$D = \frac{\sum\limits_{x=0}^{2499}(M(x) = 1 \wedge S(x) = 0) * \sum\limits_{x=0}^{2499} \frac{\sum\limits_{r=0}^{4} G(x)[i][2*r] \wedge S(N(x)[r])}{5}}{\sum_{x=0}^{2499} M(x) = 1} \tag{4}$$

To decide if a system is cooperative or competitive we use the threshold $\theta >= 0.99$. When $C >= \theta$ then the system can be classified as cooperative, when $D >= \theta$ then the system can be classified as competitive.

To understand these equations a number of important issues should be discussed. First we use the logic conjunction operator to return 0 when the result is false and 1 when the result is true. x is one of the 2500 cells, furthermore $M(x)$ is the mode operator returning 0 when a cell is vacant and 1 when a cell is operational. Similar $S(x)$ is the operator returning 0 if the state of a cell is defect and 1 if the state of a cell is cooperate. $N(x)$ is the operator that returns the neighbourhood of cell x in the form of an array of cells indexed by r. The first cell in this neighbourhood is the cell itself, then the cell to the north continuing clockwise until west. Finally, G is the operator returning the genome of cell x indexed by i and r where i is the index to the most recent expressed gene g_i and r is the index used to refer to cells in the neighbourhood. Note we index the S_x bit for a particular neighbour in the gene through $2 * r$ because each Z_x bit corresponds to one neighbour and has both an S_x and C_x part.

Compared to counting the number of operational cells that follow a particular strategy, the measure of coordination outlined in this section is better because it not only takes into account the number of operational cells and the strategy they follow, but also depends heavily on the degree to which cells can maintain the operational capacity of their preferred coordination strategy.

2.3.3 Testing

When either $C >= \theta$ or $D >= \theta$ the second stage of the experiment can begin. It is during this stage that a system is subjected to the various tests. To avoid interaction between the two different conditions, a replica of the system is made prior to offset of the second stage. This allows us to test both conditions in parallel starting from the same system state. In total, 5000 generations are simulated. During the first 1000 generations, operation of the models are simulated under normal conditions. This gives us a clear point of distruction at $t = 1000$. The remaining 4000 generations are simulated under faulty conditions. One more important issue to consider is that during the second stage of the simulations the crossover and mutation operators remain active. By allowing the systems to adapt to the damage that has been caused, we can measure if a strategy is stable while confronted with changing environmental conditions.

3 Results

In order to obtain a typical picture of how each group behaves over time, the results for each group are averaged over all 50 subjects. Note Figures 8 and 9 only depict the second stage of the experiment. It would be difficult to incorporate the results of the first stage as it takes a varying number of generations to evolve a system.

Figure 8 shows that at the start, the level of coordination amongst competitive cells is about 0.99, this is visible from the red line: “0% Damage - Defect” and blue line: “1% Damage - Defect”. On the other hand, coordination amongst cooperative cells is only about 0.1%, this is visible from the green and purple line: “0% Damage - Cooperate” and “1% Damage - Cooperate”. Slowly but steadily cooperation evolves until a system is either damaged or not at $t = 1000$. The coordination amongst competitive cells at this point in time has decreased to around 0.65 whereas coordination amongst cooperative cells has increased to about 0.20. This trend continues steadily when a system is left undamaged. At $t = 5000$, coordination amongst competitive cells has slowly decreased to about 0.20 whereas coordination amongst cooperative cells has slowly increased towards about 0.70. When a system is damaged however, coordination amongst competitive cells increases again towards 0.70. This is the blue line: “1% Dam-

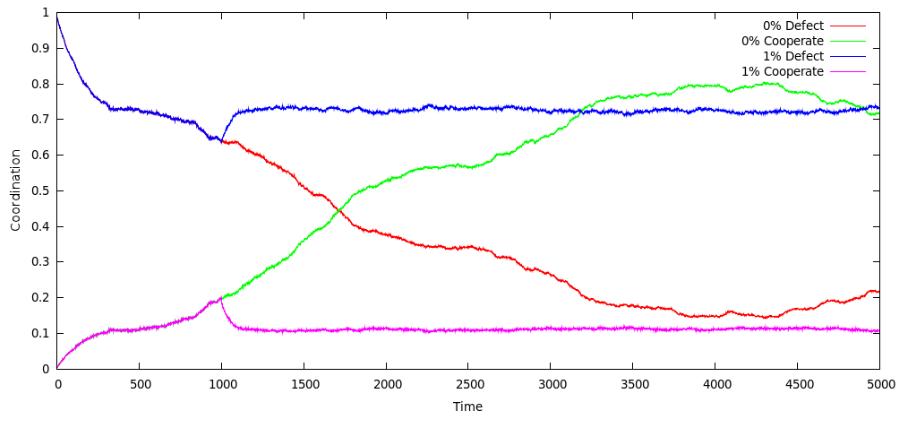


Figure 8: Subject group A: Competitive system. On the horizontal axis the time or number of generations that have elapsed. On the vertical axis the degree of coordination which is measured in the group at that particular point in time.

age - Defect". Coordination amongst cooperative cells decreases towards about 0.10. This is the purple line: "1% Damage - Cooperate". This all takes place in only a few generations, to be more precise between $t = 1000$ and $t = 1100$.

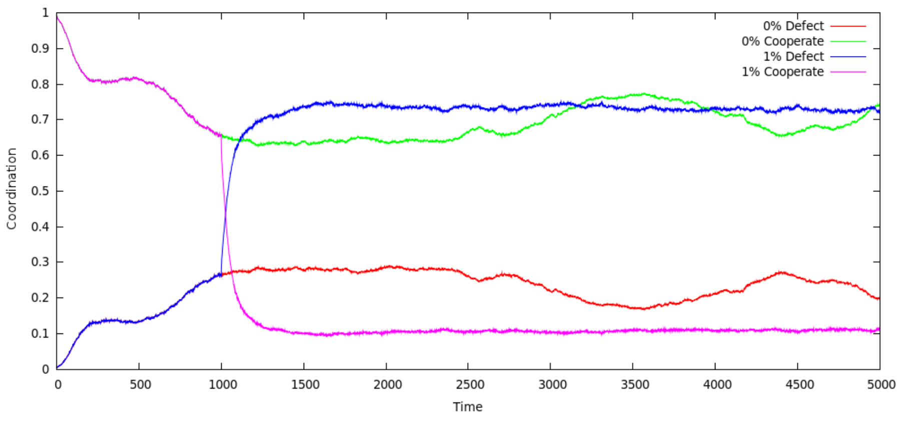


Figure 9: Subject group B: Cooperative system. On the horizontal axis the time or number of generations that have elapsed. On the vertical axis the degree of coordination which is measured in the group at that particular point in time.

Figure 9 tells a similar story. It is clearly visible that a cooperative system is steadfast when structural integrity is preserved. But, as soon as damage was caused at $t = 1000$ cooperative strategies get into more difficulty coordinating their interactions with other cells. This is represented by the purple line: "1% Damage - Cooperate". On the other hand, defection does not succeed in gaining any momentum under normal operating conditions, this can be observed when looking at the red line: “0% Damage - Defect” which stays around 0.25. It does however do a lot better when operating under difficult conditions which we can see by looking at the blue line: “1% Damage - Defect” which increases dramatically after the systems are damaged.

4 Discussion

In this study we have applied Cellular Programming combined with Genetic Algorithms to investigate how coordination of interactions between autonomous information processors influences the fault tolerance of distributed information processing systems. Two types of coordination strategies were considered: coordination through cooperation and coordination through competition. By degrading the means by which processors communicate with each other, we can measure how well a coordination strategy performs under degraded conditions.

4.1 Conclusion

The experiments show that when the communication between processors in a system is obstructed, cooperative systems fail to retain coordination compared to operation under normal conditions. Moreover, competitive strategies quickly gain advantage after only a few generations through genetic mutation. In competitive systems however, the situation is somewhat the opposite. Under normal conditions, defection is evolutionary unstable. Over time, mutant strategies slowly gain advantage allowing cooperation to emerge over defection. When structural integrity is disrupted, competitive systems are able to retain existing coordination mechanisms. In other words, distributed systems using competitive coordination mechanisms can actually benefit from a certain level of corrupt and unreliable behaviour amongst computational units.

Considering all the evidence, let us take a step back and see what the consequences are for our research question:

Is fault tolerance of distributed processing systems which coordinate interactions amongst computational elements through competitive mechanisms better, compared to fault tolerance in distributed processing systems in which interactions between computational elements are coordinated by cooperative mechanisms?

The answer is yes. We can even say that certain levels of noise are beneficial for competitive systems because competitive cells stand to loose less from errors compared to cooperative cells. Extra care must be taken to minimise error in cooperative systems. We can understand these results in terms of cooperating units relying on trust relationships with other cells. They are therefore more vulnerable to deceit.

4.2 Future Research

During our investigation, A number of questions have also arisen which would be interesting topics for future research. Our conditions were based upon a level of damage of 1% amongst components which are involved in communication between cells. This value is based upon the work of Axelrod (1984) and from any practical point of view is highly unfounded. If for example one considers a modern CPU which contains about 2.000.000.000 transistors upon an integrated circuit, the probability that 1% of these transistors fails to work properly is not very high. However, when units are loosely coupled and communicate through the Internet, the probability communication fails increases a great deal. A different assumption we made was for damage to occur with the same probability amongst all components that allow cells to communicate with other cells. When computational units are located at different geographical locations, this assumption is not very strong. A more realistic situation would be to repeat the experiment and causing damage to clusters of units. A final issue we would like to address is the fact that our study has not determined which levels of damage are acceptable. Clearly one percent is problematic so testing levels of damage between 0% and 1% should provide more insight.

5 References

[1] von Neumann, J. Burks, A. *Theory of Self-Reproducing Automata*, University of Illinois 1966.
[2] Gardner, M. *Scientific American 223*, 1970: 120-123.
[3] Axelrod, R. *The Evolution of Cooperation*, Basic Books 1984.
[4] Sipper, M. *Evolution of Parallel Cellular Machines*, Springer 1997.
[5] Weiß, G. *Multiagent Systems: A modern approach to Artificial Intelligence*, MIT Press 1999.
[6] Poli, L. Langdon, W. McFee, N. *A Field Guide to Genetic Programming*, Lulu Enterprises 2008.
[7] Talia, D. *Parallel Cellular Programming for Developing Massively Parallel Emergent Systems*, 2003.