Real Life: A Cellular Automaton for Investigating Competition between Pleiotropy and Redundancy

Teck L. Hoo[a], Andrew Ting[a], Erin O'Neill[b], Andrew Allison[a] and Derek Abbott[a]

[a]Centre for Biomedical Engineering (CBME) and Dept. of Electrical & Electronic Engineering, SA 5005, Australia
[b]School of Biochemistry and Molecular Biology (BaMBi), Faculty of Science, Australian National University, Canberra, ACT, Australia

ABSTRACT

Redundancy is where multiple agents perform one task. On the other hand, pleiotropy is the inverse of redundancy – that is, where one agent multitasks. In real systems it is usual to find a mixture of both pleiotropic and redundant agents. In engineered systems we may see this in communication networks, computer systems, smart structures, nano-self-assembled systems etc. In biological systems, we can also observe the interplay of pleiotropy and redundancy from neural networks through to DNA coding. The open question is how to design a given complex system with the correct trade-off between redundancy and pleiotropy, in order to confer maximum robustness for lowest cost. Here we propose an evolutionary computational approach for exploring this trade-off in a toy model cellular automaton, dubbed Real Life.

Keywords: Pleiotropy, Redundancy, Cellular Automata, Evolutionary Computation, Genetic Algorithms, Complexity and Complex Systems

1. INTRODUCTION

Here we investigate the competition between *pleiotropy*[1,2] and *redundancy*. The investigation involves the design and coding of a genetic algorithm written in Java, using the principles of cellular automata[3,4,5] and evolutionary computation to observe the behaviour of a system. The algorithm follows a set of rules and specifications, and tries to better fulfil those specifications with each generation of evolution.

Most engineers are familiar with the term *redundancy*, and we use it to describe the concept of having a surplus of capable agents for doing a specified task. *Pleiotropy*, on the other hand, is the opposite of redundancy, i.e. the ability for one agent to do more than one task. These are illustrated for a simple case in Figure 1.

The engineering question is raised, given a problem and a *state space*, as to what is the optimal design solution or set of solutions, and how do we maximise efficiency? A real-world example is the management of human resources within a company, to best fulfil the requirements of certain sections. This must be done by keeping the *cost* down and allowing an adequate amount of *redundancy,* to give the system a degree of *robustness*. The problem involves a set of rules or restrictions that must either be strictly followed or must be followed as closely as possible, within a certain degree of error.

The *state space* in which this question will be discussed is in a 2D lattice of cells. The other concepts and topics that will be used in this discussion are *cellular automata*, *genetic algorithms*, *evolutionary computation*, *John Conway's Game of Life* [6], *cost function* and *efficiency*.

The world is made up of many people, each with their own capabilities, limitations and rate of learning. A company has a pool of contractors and a set of projects that need to be done by certain due dates. How can a company best utilise its human resources, such that it has the right people doing the tasks they are most suited for? How does the company reduce the cost (in the number of workers), leaving a suitable amount of redundancy to compensate for the risk of absence due to illness, retirements, death or resignations of the workers? All of these scenarios can be translated into subsystems (the projects), cells (the contractors), states (capabilities of the workers), and a system (the collection of the company's projects). Each of these projects has certain requirements that must be met before they are started, and the requirements may remain the same for the whole depending on

Electronics and Structures for MEMS II, Neil W. Bergmann, Derek Abbott, Ahsan J. Hariz, Vijay K. Varadan, Editors, Proceedings of SPIE Vol. 4591 (2001) © 2001 SPIE · 0277-786X/01/$15.00

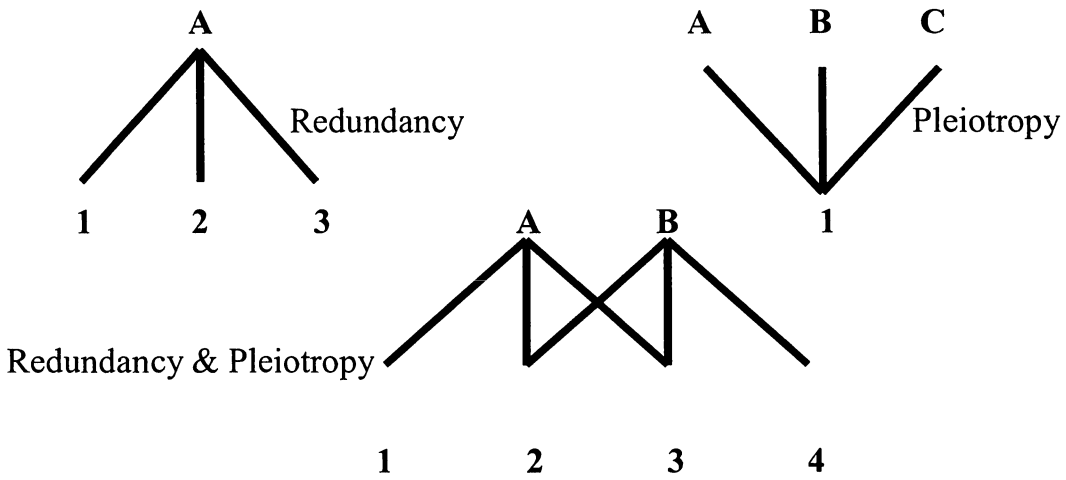

Figure 1. The top left diagram illustrates redundancy where multiple agents are able to perform the same task. We see in the top right that pleiotropy is the inverse of this, where one agent performs multiple tasks. The bottom diagram shows and example where agents can be redundant and pleiotropic simultaneously, to perform multiple tasks. It can be intuitively seen that greater robustness per cost per output task is achieved in this case.

the rules, restrictions and specifications. The program "Real Life" demonstrates the behaviour of a complex system, where each cell has its own characteristics and interacts with neighbouring cells according to a set of rules, which involve the use of random decision making determined by predefined probabilities

2. CELLULAR AUTOMATA AND JOHN CONWAY'S "THE GAME OF LIFE"

Cellular automata consist of "cells" that have 2 or more possible states and this state space (usually 2D) evolves through multiple "generations." The state of a cell at the end of a generation is dependent on the state it is at present, and the state, which its neighbours are in. John Conway developed a simple program that demonstrates the principles of cellular automata, and the many patterns that evolve naturally through the application of a simple set of rules. Conway's rules, using "dead" and "alive" states were:

1. If the cell is alive and has exactly 2 or 3 neighbours, then remain alive for the next generation, else it dies from loneliness (0 or 1 live neighbours) or overcrowding (4,5,6,7,8 alive neighbours).
2. If the cell is dead and has exactly 3 neighbours, then it becomes alive in the next generation.

The simple patterns that form from these rules include the toggling traffic light; the stable 2x2 block; the stable beehive; shooting guns, travelling gliders, and many others. It demonstrates the concept of cellular automata at a very low level, and only demonstrates very simple patterns in nature.

3. GENETIC ALGORITHMS AND EVOLUTIONARY COMPUTATION

Genetic Algorithms (GAs) and Evolutionary Computation (EC) are associated with cellular automata, by the fact there is a set of rules or restrictions that determine the next step within that evolution. In GAs, there is a state space consisting of "chromosomes" that, in turn, consist of "genes" or "cells." The state then undergoes an evolution, whereby a certain percentage of the chromosomes are retained intact, a certain percentage crossed-over with each other, and the remaining percentage partly or wholly mutated. The retained selection may depend on the set of chromosomes that provide the best set of results or be a completely random selection. The crossed-over percentage may have genes crossed with genes that perform the same task and are under the same restrictions. The mutated percentage may have part of all of their genes randomised or "mutated" and then altered to meet the restrictions that are required by genes in their position. From this generation, the next generation evolves, and

after a certain number of generations, a set of results are obtained, that usually give a better solution than previous generations. This method is widely used in engineering to determine the set of optimal settings for variables to best satisfy a certain set of conditions or restrictions.

4. COST FUNCTION

Another factor that regulates the pleiotropic and redundant nature of a system is the use of a "cost" function, which may include the measures of efficiency. If there was not a cost function, involved with the creation of a system, a system could consist of fully pleiotropic cells with abundant redundancy, ie. every cell in the entire system capable of doing every task.

5. REDUNDANCY

When investigating the properties of redundancy, we see its "safety-backup" nature, whereby a system with some degree of redundancy can still function if there were some redundant cells removed from it. This property gives the system a degree of *robustness* with high integrity, and margin for error. The problem with a system with large redundancy is that the cost for maintaining the system is large, because, as the name suggests, there are many redundant and wasteful cells. Theses cells are either capable of doing tasks that can already be done by a number of other cells.

6. PLEIOTROPY

A pleiotropic nature implies the capacity for multi-skilling or multi-tasking. A cell with the capabilities of being able to do many different tasks is more valuable than a cell with only limited skill capabilities. However it is expensive either to create an employee with many skills or to hire someone with many capabilities – likewise, the cost of creating or maintaining a cell is high. Every cell has its own capabilities, limitations, and capacity for growth, but the cost of a cell may only be dependent on what it is doing for the system now. This is analogous to a workplace recruiting a new graduate where, in most cases, a graduate will get paid the same as the next graduate at the same company, even though he or she may be a lot smarter, but have the same qualifications.

7. MIXED REDUNDANCY AND PLEIOTROPY

If a system consisted of purely pleiotropic cells, which satisfied the required specifications, we would have the highest degree of *compactness* and flexibility within the system, i.e. it would consist of the least number of cells possible with respect to the state space. The advantage with a high degree of compactness is management of the system is much simpler than a redundant system, as there are fewer cells to monitor and the cost will be minimal. This is due to the amount overhead involved with even the smallest and least useful cell, as it would usually cost less to have cells which can do tasks 1,2,3,4, rather than have two cells, one doing tasks 1,2,3, and the other doing task 4. The problem associated with a system that consists of highly pleiotropic cells, is that if a cell "dies" or is disabled, a vital function may be totally lost. Another question arises as to how much pleiotropy and redundancy is needed within a system? The optimal solution depends on the system requirements. The decision criteria are as follows:

1. Meet the minimum requirements for the system to function at optimal efficiency.
2. Meet minimal requirements for the system to operate, even though factors such as efficiency of operation may suffer, as there are fewer than the minimal numbers of cells capable of doing a required task.
3. Meet the minimal requirements of the system with a minimal operating efficiency.
4. Meet the minimal requirements of the system with a minimal redundancy for each (or some critical) task or have an overall redundancy greater than a certain threshold level.
5. Meet the minimal requirements at the lowest cost.
6. Meet the minimal requirements at the lowest cost, with a certain minimal operating efficiency.

7. Meet the minimal requirements at the lowest cost, with a certain minimal operating efficiency and certain minimal redundancy.

There are many other combinations that might be considered with cost, operating efficiency, redundancy all being a factor to varying degrees. These are only a few of the factors that are responsible for the dynamic changes within a system. The interactions between cells is also very important with each cell having different properties, and the needs within each subsystem being different.

8. REAL LIFE

An investigation of the above concepts was carried out via the design, creation and testing of a simulator, that simulated the behaviour of a living system. This simulator was written in Java and was named "Real Life" (RL). Its predecessor was written in Matlab and was a prototype named "Intermediate Life" that demonstrated the basic functionality of "Real Life" with fewer rules. The RL system was divided into *departments*, and the departments divided into quadrants and subsystems, each having their own set of requirements and set of rules.

In John Conway's "The Game of Life," the same rules are applied uniformly to every cell within the system and the rules are deterministic, because given the same starting pattern the resultant behaviour for each generation is the same, no matter how many times the program is run. In Real Life, there are some choices that are made and, as in life, no two behaviours are the same given the same starting pattern.

8.1 The Cell

The cell is the building block upon which subsystems, departments and systems are based upon. Each cell has its own present capabilities as far as the number of skills it can do, and what they are, and a Maximum Potential Index, MPI, (maximum of 8 and minimum of 1). MPI is the highest number of skills a cell can have at any one time and perform without any penalty. There are 16 jobs ranging from task 0 to task 15. Tasks 0,1,2, and 3 are classified as "management" tasks or capabilities, which have a higher probability of being learnt by cells on their own than for tasks 4 to 15. Finally, there is an attitude modulator that takes into account the attitude of a cell, luck and "being in the right place at the right time" scenario.

8.2 States

There are 9 states associated with "Real Life" ranging from 0 to 8. These states are broken up into 3 main categories, Managers (8), Trainables (4 to 7), and Sackables (0 to 3). Each category has its own properties and capabilities.

8.3 Managers

Managers are the major driving force behind the system. They have the property of being able to increment the state of a neighbouring cell, by teaching that cell one of the skills that the manager knows that the student cell does not know. Theoretically, if a manager happens to be next to a student cell all of its life, and the student's MPI is 8 and starting skill count was 0, the manager could teach all the skills it knows to the student cell and have an exact replica of the manager cell. Each manager cell is unique, and has its own set of random skills that it knows. Besides being able to teach a cell a skill, it also has the capability of moving, by swapping itself with the position of a random neighbouring sackable cell. If a manager was surrounded by trainables, it could not go anywhere. A manager has a 75% chance of training a trainable, 20% chance of training a sackable and 5% chance of dying.

8.4 Trainables

Trainables are the set of cells that are just below managers. They cannot teach other cells skills, but have a higher probability to learn a skill each generation than sackables. They also receive higher "group bonus" advancement probabilities if they make up the majority of a subsystem than sackables. Trainable by themselves, with no group

bonus probability, they have a 1/8 chance of advancing in a skill and a 1/5 chance of advancing if they make up more than 65% of the total alive cells in the subsystem.

8.5 Sackables

Sackables are the lowest and least capable of the three categories, and are divided again into two categories, "alive" and "dead." Alive sackables are those sackables with 1,2, or 3 skills present, while dead sackables are those that have 0 present skills. The difference between the two categories is the expense, as dead sackables do not cost anything but can still be swapped with managers. Sackables by themselves have a 1/16 chance of advancing in a skill, but if they make up over 65% of the alive cells within a subsystem, them they have a 1/10 chance of advancing.

8.6 Environment

The environment, in which a cell lives in, is one of 64 cells within the subsystem. A subsystem is an 8x8 matrix of cells, which has its own set of minimum specifications dependent on where it is located within a department. A department consists of 81 subsystems (9x9 matrix), with the centre subsystem called the homebase. This homebase is the origin of the department and from where all activity starts. The homebase is where all four corners of the quadrants of the department meet. A quadrant has its own "quadrant skill" or "level 1 skill," which is different to all the other quadrant skills from the other quadrants.

8.7 Requirements of subsystems

As mentioned before, the requirements of each subsystem differ depending on where they are located within the department – more specifically in which quadrant, and at what level in the quadrant. The closer a subsystem comes to the corner of a department, the more requirements and specifications are placed upon it to operate. In the following, there will be references to skill points, which is just another way of specifying the number of cells capable of doing the task. For example, 3 required skill points indicate that there are 3 cells required to do the task.

Each subsystem has one common requirement, a set of 4 skill points for each of the management tasks (0,1,2, and 3). This is equivalent to real life in needing basic management and communication skills, besides the technical knowledge to set up a project.

8.8 Requirements for each level in a quadrant

Level 1: At level 1 of the quadrant there is cross over with neighbouring quadrants. Due to this cross over, this level is special in that it requires skills from both quadrants. An arbitrary figure of 5 skill points, from each quadrant, is the skill required in order for the subsystem to start operating.

Final requirements: $4x\{0,1,2,3\} + 5q1 + 5q2$ (assume you are on the border between quadrant 1 and quadrant 2, and q1=quadrant skill for quadrant 1 and q2=quadrant skill for level 2).

The requirements chosen, for each of the subsystems at increasing levels, were linear – though in reality an exponential relationship could be more realistic.

8.9 Choosing of skills for levels

When a manager swaps its way into a subsystem outside the homebase, and comes across an uninitiated subsystem, it will choose a skill other than the 4 management skill for that quadrant(s) skill level, from its list of skills that it knows. If it is a skill quadrant, it must not be the same as any of the other quadrant skills. If it is a level skill, it must not be the same as any of the skills already in that quadrant, but can be the same as a skill in another quadrant.

Level 2: At level 2 there is an additional skill that is chosen for the quadrant, and there are no effects of any skill requirements from other quadrants. The effects of the skill requirement from the previous level are reduced, and a new skill becomes the dominant skill.

Final requirements: 4x{0,1,2,3} + 5lv2 + 4q2

(assuming you are in quadrant 2, you need 5 skill points of the level 2 skill of quadrant 1 and 4 skill points of quadrant 1 level 1 skill).

Level 3, Level 4, and Level 5: At level 3,4, and 5, another skill is introduced to the quadrant and the effects of the skill requirements from previous levels are reduced.

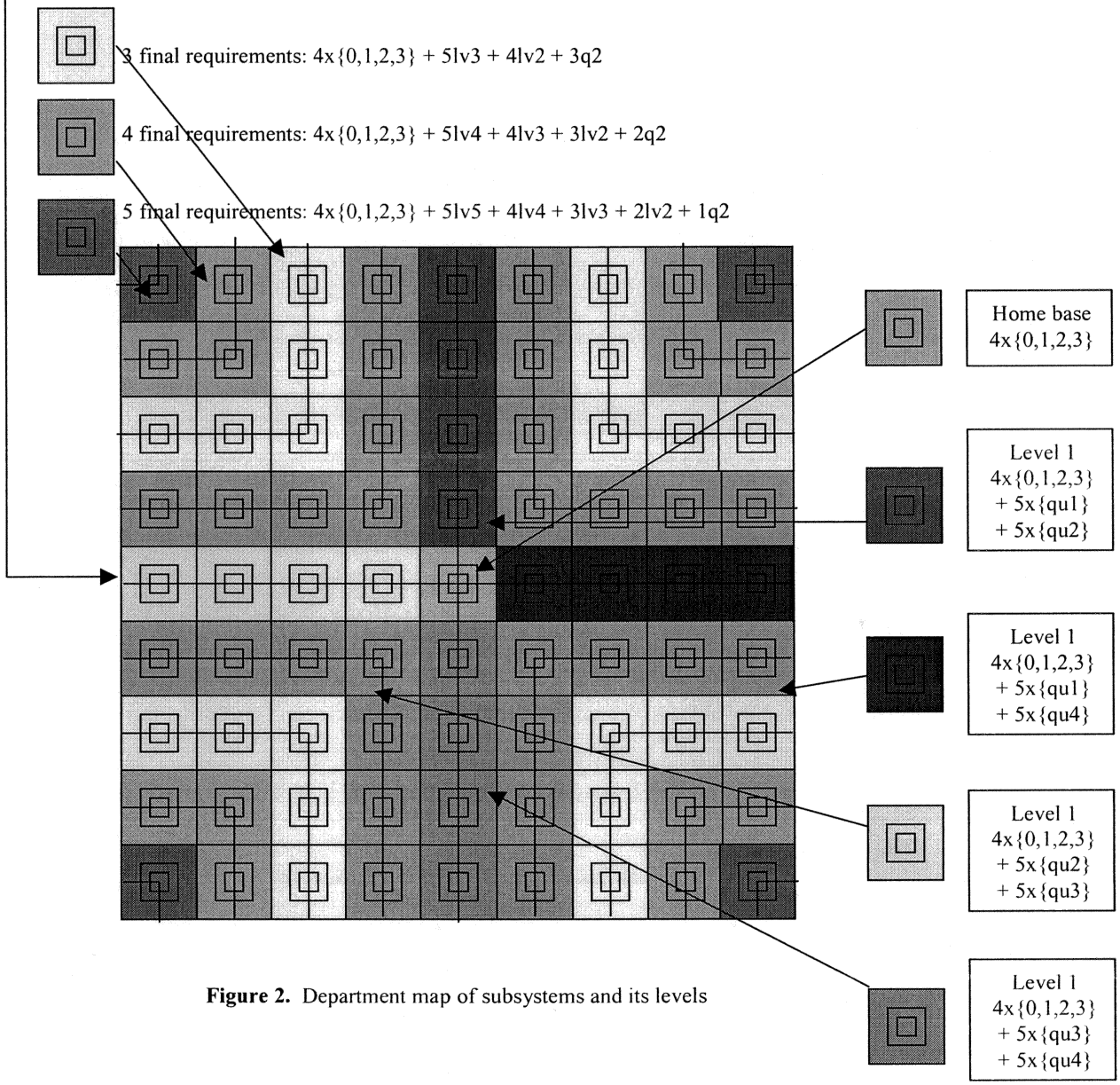

Figure 2. Department map of subsystems and its levels

In general, the higher the level the higher the need is for more diverse skilled (pleiotropic) cells. In Level 1, the total skill points required other than the basics is 10; level 2 requires 9 skill points; level 3 requires 12; level 4 requires 14, and level 5 requires 15.

The actual number of skills required for each level is an arbitrary amount, with higher numbers creating tighter restrictions for the subsystem to start operating. In any case, the numbers should be set in increasing order with reduced effects from skills in previous levels, which leads to an increase in the pleiotropic nature of the subsystem in higher levels.

8.10 Cost of a subsystem

The cost of a subsystem is dependent upon the total cost of the cells that exist within it. Those cells with zero number of total skills are treated as empty space and do not cost the subsystem anything. The other cells with 1 to 8 total skills have their cost determined by the following table:

Skill number	Cost
0	0
1	30
2	35
3	40
4	50
5	60
6	70
7	80
8	100

Figure 3. Table of costs for the skill number of a particular cell

The costs were chosen with respect to each cell's capabilities to learn and advance (determined by what category they are classified under) besides the current skill level of the cell. There is an increase of 5 for each level in the alive sackable category (1,2,3), and an increase of 10 for each level in the trainable category (4,5,6,7) and finally a jump of 20 from 7 to 8 (manager category). In each case, the cell's cost could have been under or over estimated, but these numbers have been chosen as an example set of costs.

8.11 Efficiency and overloading of a subsystem

The efficiency of a subsystem is determined the efficiency of its constituent cells, and whether it has met the minimum number of skill points for each of its required skills. When a subsystem is initially non-operational and then finally meets its requirements, it becomes operational. As generations go past, the subsystem grows, and the manager which may have started the subsystem could leave, and due to its pleiotropic nature, could have been responsible for many of the tasks required for the subsystem. In that case, the other cells of the subsystem must "overload" to make up for the loss in required skill points. The cells that are capable of overloading are those cells whose skill number equals their MPI. Those who have their number of skills less than their MPI cannot overload. A cell can overload for up to 3 generations without a penalty to its efficiency, after which its suffers from an efficiency penalty that exponentially increases with the number of overloaded tasks, which the cell is doing. A cell cannot have its total number of skills and total number of overloaded skills greater than 8, so a cell with an MPI of 6, can only overload a maximum of 2 tasks, while a cell with an MPI of 1 can overload a maximum of 7 tasks.

The following table is how the efficiency penalty is determined:

Overloaded tasks	Penalty calculation	Penalty total
1	5*2^0	=5%
2	5*2^1	=10%
3	5*2^2	=20%
4	5*2^3	=40%
5	5*2^4	=80%
6	5*2^5	=160%
7	5*2^6	=320%

Figure 4. Efficiency penalties for overloading

An example of how the efficiency for a subsystem is determined is as follows:

If skills 0,1,2, and 3 have 3 cells in the system capable of performing the task instead of the required 4, and qu3 and qu4 have 4 cells instead of the required 5, the efficiency for the subsystem is (3*(1/4) + 3*(1/4) + 3*(1/4) + 3*(1/4) + 4*(1/5) + 4*(1/5))/6 =(0.75+0.75+0.75+0.75+0.8+0.8)/6= 77%. The efficiency penalty for the subsystem is then determined afterwards and modifies the original efficiency total.

If 2 cells are overloaded by 2 tasks => 10% efficiency penalty each and the total number of alive cells in subsystem=20.

Modification = (2/20)*10%=0.1% efficiency penalty for the entire system=>77-0.1=76.9%

If for the above situation, 8 of the 20 alive cells were overloaded by 6 tasks instead then =>160% efficiency penalty each.

Modification =(8/20)*160%=64% => 77-64=13%

If at any stage the efficiency goes below 0%, then the project is scrapped and the subsystem is abandoned.

8.12 Redundancy calculations

The number of additional skill points required for 100% redundancy was arbitrarily chosen with only some reference to the fact that an emergent skill is relatively moderate – 2 becoming unavailable is low to moderate and 3 or more quite low. The penalty applied to the efficiency is inversely proportional to the minimum number of skill points required for that task in the subsystem.

That is, a task requiring 3 skill points will have an efficiency penalty of 33% for every skill point below the 3 to total of 100%, if there are no skill points for that task. Therefore being undermanned by 1 worker in a task which requires a minimum of 10 workers has not got as great an efficiency penalty (10%) as a task with only 3 people required (33%).

Minimum Number required for 100% efficiency	Level of redundancy, in terms of number of cells extra that must be able to perform task to have 100% redundancy	Penalty applied in terms of efficiency for every cell below minimum required.
1	3	100%
2	3	50%
3	3	33%
4	4	25%
5	4	20%
6	4	16.6%
7	4	14.2%
8	4	12.5%
9	4	11.1%
10	5	10%
11	5	9.09%
12	5	8.3%
13	5	7.7%
14	5	7.1%
15	5	6.67%
16	5	6.25%

Figure 5. Redundancy requirements for each level of minimum skill points, and their efficiency penalties

The redundancy is calculated like the efficiency and is determined by how much of the required number for full redundancy is present. For example, if a task requiring 5 cells, which has a redundancy requirement of 4 cells for 100% redundancy and it only has 3 cells, the task has 75% redundancy. The redundancy for each task is then determined and averaged over the number of tasks under consideration, eg. 4x{0,1,2,3} + 5x{qu3} + 5x{qu4} This requires 4 redundant cells for skills 0,1,2,3,qu3 and qu4. If skill 0,1,2,3 have 2 redundant cells, and qu3 and qu4 has 4 redundant cells, the redundancy for the subsystem is (½ + ½ + ½ + ½ + 1 +1)/6 =4/6=66%.

8.13 Measure of compactness

The measure of compactness is simply determined by the ratio of total number of possible alive cells/number of actual alive cells, eg. for 10 alive cells within a subsystem => 64/10=6.4

Another measure of compactness is the number of skills per cell within a system, which is determined by the total number of skill points of subsystem/total number of alive cells, eg. for 10 alive cells within a subsystem containing a total of 60 skill points => 60/10=6 skill points per cell

8.14 Recruits

The last concept that will be introduced is the idea of recruits. In an initially random state space, where there are random cells with random properties placed everywhere in the system, they are not counted towards the cost of the system. As a manager moves out of the original homebase of a department, it interacts with other cells, swapping with sackables (dead or alive) and have a possibility of training dead sackables (0) to skill level 1, or may come across other alive sackables or trainables and train them. When the manager trains them, they become part of the cost of the system, and they make their way towards the homebase, which is at the centre of the department. If they come across a subsystem that needs their particular skills to meet the specified requirements, they enter the subsystem and become part of it, if not they move along the boundary of the subsystem until it meets either another subsystem that will accept them or the boundary of the quadrant. The main purpose of recruits is to improve the redundancy of subsystems as subsystems will only begin operating once they reach the threshold requirements of the subsystem, after which they grow to try and achieve the other specifications such as redundancy.

8.15 Program Options

The last section on recruits may be included or left out in the coding, as the coding complexity associated with monitoring recruits and their movements is quite enormous. With the exclusion of the recruits, the system is much less dynamic, and initialisation of new subsystems is completely determined by managers' activities, and overloading, redundancy and efficiency issues become more relevant as recruits do not move back into old subsystems. The old subsystems are left to grow by themselves, training themselves up or having managers do the training, and from these processes, the efficiency, redundancy and overloading is determined.

9. CONCLUSIONS

This project can be extended in many ways, by altering the efficiency, cost, redundancy, and compactness requirements. A loosening of the specifications will allow for faster growth of the system, while a more strict set of rules would involve a slower growth process. The initialisation of the system could either occur with a system completely filled with dead sackables that have their MPI's between 1 and 8, but have no initial skills, and then place a group of cells with different or the same properties within in and observe how fast the subsystem grows, and what characteristics it takes upon itself. There is a very large set of combinations that could be investigated, with each relating to a different set of circumstances. With the correct amount of redundancy and pleiotropy within a system, a suitable balance between cost, redundancy, and compactness can be achieved, and a higher total integrity and flexibility can be achieved compared to a system that was fully pleiotropic or redundant. Constant monitoring of all operational subsystems can give regular feedback on the status of all key factors of the subsystems. A log can also be created giving status reports on all major advancements within the system. There is a myriad of possibilities, but only one program, and that is what "Real Life" can simulate.

ACKNOWLEGEMENTS

Support from Adelaide University and GTECH Corporation Australia is gratefully acknowledged.

REFERENCES

1. S.N. Coppersmith, R.D. Black and L.P. Kadanoff, "Analysis of a population genetics model with mutation, selection, and pleiotropy," *J. Statistical Physics*, **Vol. 97**, No.3-4, 1999, pp.429-457

2. M. Morange, "Gene function," *C.R. Acad. Sci. III,* **Vol. 323**, No. 12, 2000, pp.1147-1153.

3. J. von Neumann, *The Theory of Self-Reproducing Automata.* (Edited by A. W. Burks), University of Illinios Press, 1966.

4. A.W. Burks, *Essays on Cellular Automata,* University of Illinios Press, 1970.

5. S. Wolfram (Ed.), *Theory and Applications of Cellular Automata*, World Scientific Press, Singapore, 1986

6. M. Gardner, "Life," *Sci. Am.*, **Vol. 223**, 1970, p. 120.