Project Number 001907

DELIS

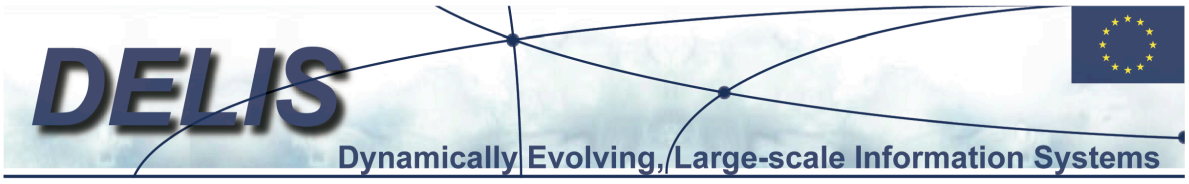
Dynamically Evolving, Large-scale Information Systems

Integrated Project

Member of the FET Proactive Initiative **Complex Systems**

Deliverable D5.2.1

Algorithms to Identify Locally Efficient Sub-graphs in Information Transfer Networks



Start date of the project:	January 2004
Duration:	48 months
Project Coordinator:	Prof. Dr. math. Friedhelm Meyer auf der Heide Heinz Nixdorf Institute, University of Paderborn, Germany
Due date of deliverable:	October 2004
Actual submission date:	November 2004
Re-submission date:	April 2005
Version:	2
Dissemination level:	PU – public
Work Package 5.2:	Evolved Tinkering and Degeneracy as Engineering Concepts
Participants:	Universitat Pompeu Fabra, Spain University of Bologna, Italy Telenor, Norway
Authors of deliverable:	Ricard V. Solé (ricard.sole@upf.edu) Sergi Valverde (svalverde@imim.es) Ozalp Babaoglu (babaoglu@cs.unibo.it) David Hales (dave@davidhales.com)

Preface

This report comprises the complete D5.2.1 deliverable as specified for workpackage WP5.2 in Subproject SP5 of the DELIS (Dynamically Evolving Large-scale Information Systems) Integrated Project.

The essential goal of the DELIS project is to understand, predict, engineer and control large evolving information systems. When dealing with information processing in complex systems, a number of problems arise. One is how to properly map information processing, storage and coding into architecture. In other words, what is the relation between structure and function. If we try to make explicit the intuitive links between biological and technological structures, a network description of the system structure and dynamics is necessary. Recent models of network growth have shown the presence of common evolution principles in a large variety of systems. In this report we will consider the evolution of three different computation systems: cellular networks, *in-silico* networks and class diagrams (software).

The first section of this report authored by Ricard V. Solé and Sergi Valverde comprises the major part of this deliverable, detailing on-going work exploring what appear to be quite general structural properties in engineered network structures which develop through dynamic evolutionary-like processes. The second section details on-going work from Bologna which demonstrates the power of a simple "duplicate and re-wire" evolutionary algorithm in a simulated peer-to-peer network.

The inspiration for this latter work has come from the social and biological sciences (the novel "tag" mechanism). It appears that such simple rules of topology construction may be selected for in many evolutionary domains. In future work we aim to apply the on-going, detailed, empirical and scientific work (section 1) to help in the practical construction (from an engineering perspective) of robust and scalable node level algorithms (section 2).

1 Algorithms to Identify Locally Efficient Sub-graphs in Information Transfer Networks

1.1 Introduction

Few people will disagree that efficient computation pervades complexity. Computation provides the basis for an unified view of the emergence of complexity in both natural and artificial complex adaptive systems (CAS). The degree of efficiency achieved by a given system gives us a measure of how well it has evolved to perform its tasks. Although there is little consensus concerning the definition of CAS, they all share some degree of anticipation to external changes which is mapped into some underlying computational task. To efficiently predict the future is a key ingredient of adaptation and its computational basis can be perceived at different scales.

When dealing with information processing in complex systems, a number of problems arise. One is how to properly map information processing, storage and coding into architecture. In other words, what is the relation between structure and function. If we try to make explicit the intuitive links between biological and technological structures, a network description of the system structure and dynamics is necessary. Networks have been shown to properly retain a large fraction of the internal complexity of a given system, and they are the roadmaps to our understanding of emergence of functionality. The importance of architecture within the context of efficient behavior is a key issue: both selective forces or optimal design are able to shape local and global structures. Efficient behavior is detectable in terms of sparse graphs (i. e. low cost), modular patterns (i. e. division of labour and limited damage spreading) and also network subgraphs with special features allowing them to be used as basic building blocks.

Low cost is a first pattern that seems to be shared by most complex networks performing information processing. Here hardware becomes an obvious example and, as shown by previous studies, it already displays remarkable features, such as small world behavior or heterogeneous patterns of

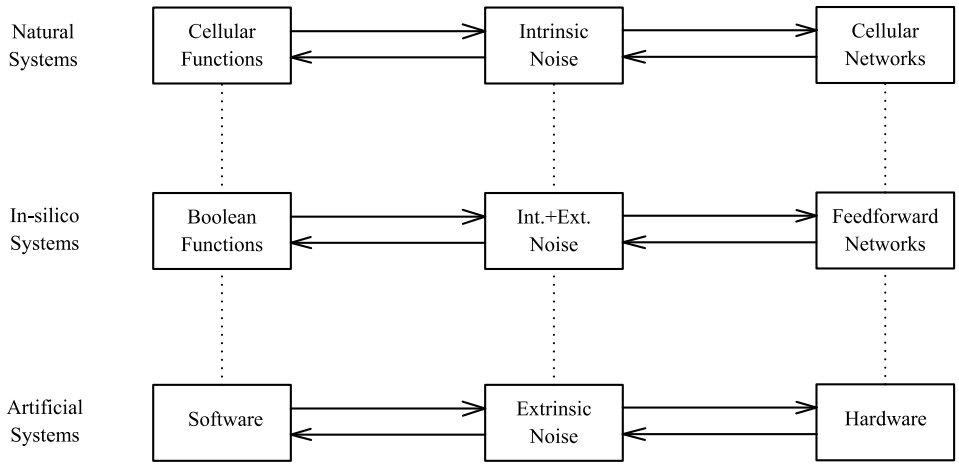


Figure 1: Classification of different natural and artificial computing systems. System operation can be affected by intrinsic, extrinsic or both types of noise.

connections (Ferrer and Sole, 2000). For the first class, function is a predefined concept and the wiring pattern is the result of optimal design, at least for small circuits. When dealing with large-scale designs, engineers make re-use of previously defined gates: a more complex funtion is obtained by combining parts that play a given desired sub-function.

Looking at natural structures, we appreciate a number of similarities and differences in relation with artificial designs. On the one hand, human-made artifacts are tipically fragile against even tiny failures of single components. This is a general trend in artificial systems, since they are not designed to be robust: tipically, they fail under subsystem failure unless they explicitly incorporate some degree of redundancy. In cellular networks, such as those involved in signal gathering and processing in cells, robustness appears to be a common feature. But robust behavior is constrained by the cost of links and their reliable behavior: efficiency appears as a complex tradeoff between functional needs and topological limitations. Untangling the mapping between dynamic responses and network architecture at different scales is one of the most fascinating problems in complexity. Our study has been focused in determining the relevance of small subgraphs within systems dealing with some computational task. Our results indicate that there is indeed a non-random distribution of subgraphs in the systems analysed, but the suggestion that they can be considered as the meaningful building blocks of network funcion seems to be flawed.

One of the most important limitations to efficient information processing is noise. Although noise inside cells is very high, signals are received and properly processed so that correct repsonses are obtained. When individual components fail transiently, the system is able to perform the functionality by using alternative pathways. This is achieved not through redundancy but instead through a different class of distributed robustness. Such type of robustness, also known as degeneracy, becomes evident when we remove whole genes and created a so called *knock out* organism. In many cases, the phenotypic impact of the genetic modification is small or even zero. This is the case in spite that the removed gene was known to be involved in some well-defined function.

While looking for specific quantitavive measures of graph structure, we have also faced a methodological problem. Since we want to obtain universal properties from exploring both artificial and natural systems and their in silico models, some rationale seems required. In figure 1, a summary of the three levels of analysis and their possible links is summarized. Here the top and bottom layers correspond to natural (cellular) and artificial (designed) systems. The left and right columns correspond to function and structure, respectively. The medium column involves the type of noise to deal with in each case. The intermediate line, involving models of in-silico functions and structures,

will allow us to establish the bridges between both large classes of system. The reason of making such analogies is that recent work has shown that, in terms of their structural features, networks spanning a broad class of systems, from protein-protein interactions to software graphs display some qualitatively (and sometimes quantitatively) similar features.

The spectrum wiring patterns of cellular networks, particularly those well known in performing some time of computation, such as transduction networks (STN), is rather illustrating. STN are cascades of changes propagated through interactions among molecules that perform both information transfer and processing. The process takes place under noisy conditions: many of these molecules are in low-copy numbers and yet the whole computation takes place successfully and efficiently. The way each system achieves efficient performance should be expected to be different from each other. Identifying the nature of such differences, both at the large and small scales, is a first step towards understanding information dynamics in our real world.

1.2 Evolutionary Models of Computational Networks

The classical theory of computation deals with static machine descriptions having a computational power dictated by an equivalent Turing machine. However, it is clear that computation systems evolve by a process that frequently resembles learning activities. Recent models of network growth have shown the presence of common evolution principles in a large variety of systems. In this report we will consider the evolution of three different computation systems: cellular networks, *in-silico* networks and class diagrams (software). The aforementioned systems can be described in terms of a directed network where nodes represent different components or single entities and links display the interactions between these items.

The network representation is a convenient means for capturing the structural features of each system and discarding many other details. Series of network snapshots are useful in understanding system evolution. For example, by counting the number of nodes in the network representation we obtain a good estimate of system size. But detailed analysis of consecutive network snaphots reveals a more interesting picture. In figure 2 there is a sequence of networks describing a one-year evolution history of a computer game subsystem [1]. It is possible to notice some order and coherent growing pattern in this picture. This raises several interesting questions: what is the origin of this pattern? what rules and principles drive its evolution? and is it possible to predict the outcome of evolution? During this research, we will try to answer these questions for each of the previous systems. The final outcome will be to uncover the connection between evolution and computation.

We must be cautious when drawing conclusions from the network analysis of system evolution. The mapping between system and network enables a comparison between different systems solely on the basis of their structural features. However, we should take in mind that comparison between different systems is not straightforward. Mapping may involve different scales of detail. In particular, the level of detail chosen for software analysis is much higher than scales considered in cellular and in-silico systems. Still, those limitations will not preclude us to stem some general evolution principles.

1.3 Deterministic Computation: Software and Electronic Circuits

This is an important family of computation networks that are mainly characterized by being artificial and deterministic at their smallest scales of detail. Interestingly enough, their global behavior is utterly complex and even unpredictable sometimes. While these systems are built from reliable blocks, they are subjected to unpredictable and stressing environments.

Evolvable computing systems require two different ingredients: information processing machinery (hardware) and an encoded and an evolvable representation of behavior (software). This organization is an optimal trade-off between efficient performance and an enhanced ability to adapt to environmental changes. In order to set up the computer to solve a given problem, the software or sequence

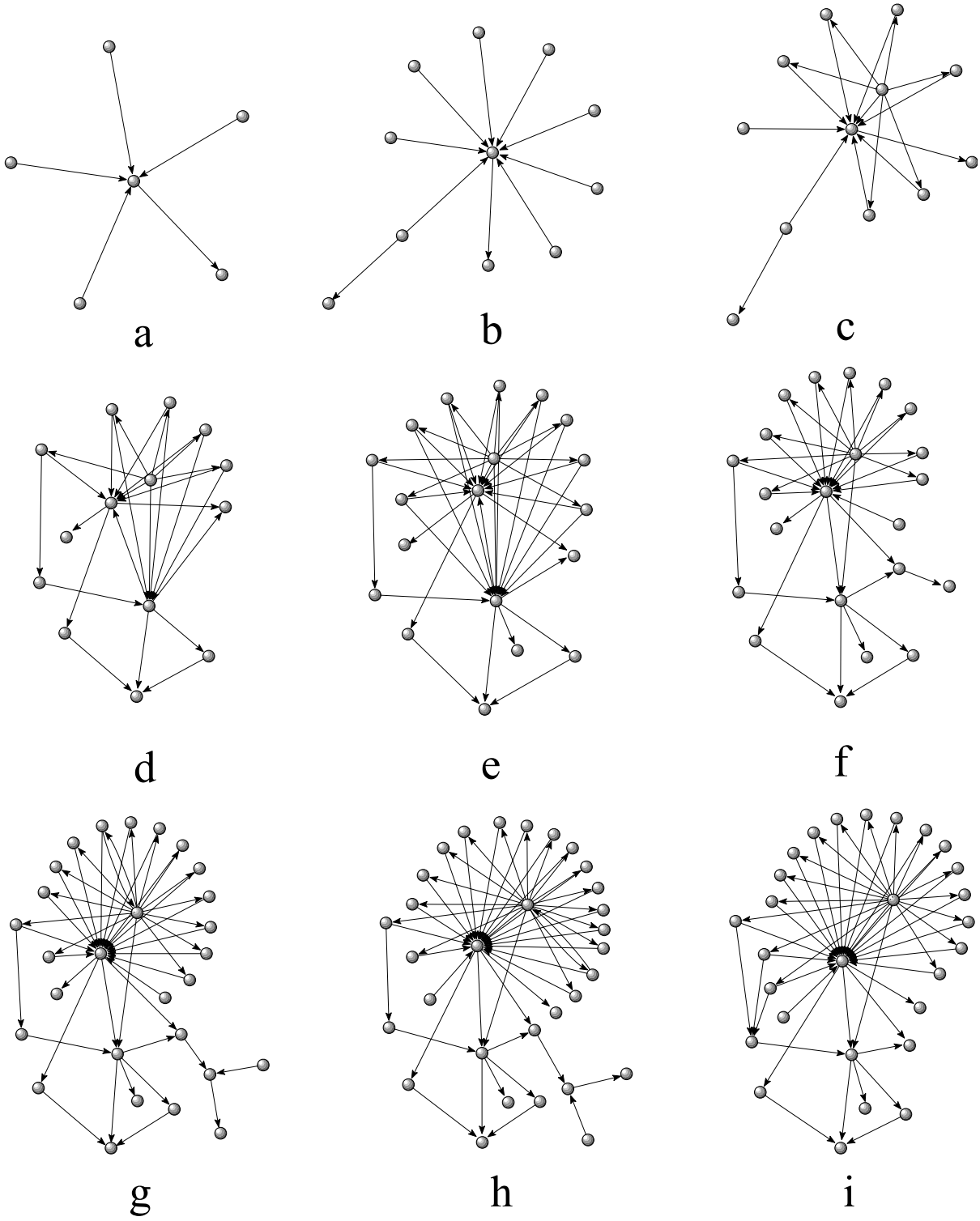


Figure 2: An example of the evolution of a software subsystem, displayed as several network snaphots corresponding to different consecutive evolution stages. Evolution goes from top to bottom and left to right. Only the largest connected component spanning subsystem classes is displayed here (from [1])

of instructions must be given to the hardware. The expected behavior is reproduced by the hardware as soon as the software is decoded and interpreted. For this purpose, software is encoded as a variable-sized bit string of length m, which is stored in the computer memory along with the data, very much like a strand of DNA within the body cell. The complexity of the task to be performed (i.e.: an aircraft control system or a complex developmental program) imposes a minimum number of instructions. Generally, the realization of the program involves a multi-step development process or an adaptive walk through the landscape of all possible instruction sequences.

In spite of the existing body of recipes and heuristic recommendations for building software systems, very little is know about the actual rules underlying real software development. In this context, empirical software engineering is a new and emerging sub-field that might provide insight into this important subject. However, these studies gather statistical trends and they still lack a solid theory explaining these observations. Here, we propose that the theory of complex networks is a solid grounds where a theory of software development might eventually develop. In order to understand the link between structure and evolution we have analysed a software model describing with a directed graph the intrinsic dependencies between different software entities arising from the design process. Dependencies exist at many different software levels. At low scales, software is described by a sequence of simple instructions performing basic computations. At higher scales, software is comprised by modules that are assembled with other modules in order to produce the whole system. These modules are software artifacts that group simpler low-level entities. Popular programming languages (like Java or C++) recognize the importance of modular software development and a first-class module entity (usually called "class") can be found in their toolsets.

It was noticed by us [14] that software architectures are nice examples of heterogeneous and uncorrelated networks that are properly described by their (undirected) degree distribution following an scaling law:

$$P(k) \approx k^{-\gamma}$$

where γ is about 2.5 (see figure 3). Interestingly, this pattern has been shown to be independent of the purpose and objectives of the particular software under study, suggesting that software structure (at the module/class level) is largely independent from function. These networks are also good examples of small-worlds. That is, the average shortest distance d between any pair of modules scales with the logarithm of system size N:

$$d \approx \frac{\log N}{\log < k >}$$

and the clustering coefficient C departs signficantly from random:

$$C >> C_{rand} \approx \frac{< k >}{N}$$

where $< k >$ is the average number of edges per node.

There are several network growing models that reproduce scale-free networks. Probably, the Barabasi-Albert preferential attachment model being the most popular of them. Unfortunately, this model fails to reproduce the high clustering observed in software systems thus suggesting that another mechanism will underlie software development. Also, there is the question if software systems are optimal designs in the sense they are shape in a way that minimizes the efforts and cost required to integrate new requirements [14]. It is well-know in software engineering that a good designer should avoid re-inventing the wheel again and again. In this context, it has been claimed that module reuse pays off in the long run. The simplest way of reusing is to duplicate some portion of the program and later adapt it to some specific needs. We have recently discovered that network models based on duplication and diversification mechanisms accounts for a significant part of the observed topological features [1]. This study was based on the analysis of common network motifs in software architectures.

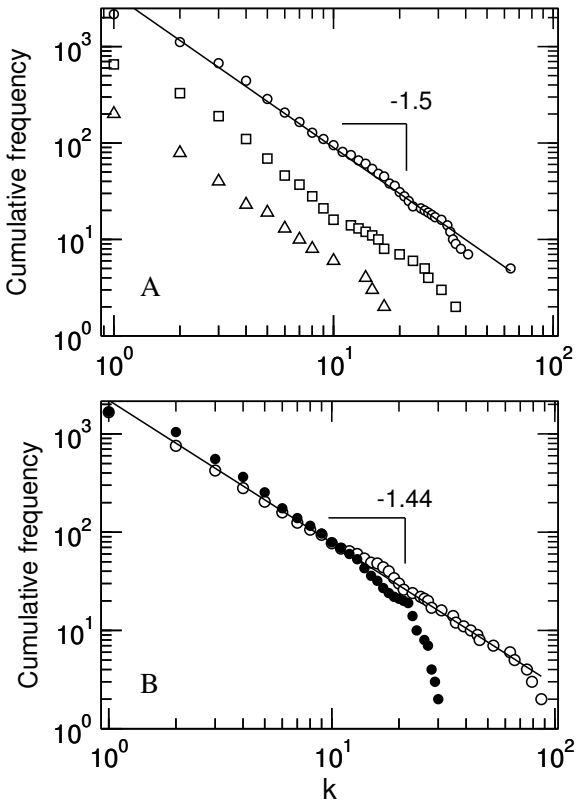


Figure 3: Cumulative degree distributions for software graphs. Log-log plots on top display the undirected degree distributions for eMule (N=129, triangles), Blender (N=495, squares) and CrystalSpace (N=1488, circles). Notice that distributions fit very well a power-law having exponent -2.5 in spite of different sizes and performing different tasks. The bottom plot displays the typical asymmetric pattern between in-degree (open circles) and out-degree distributions (filled circles) in directed versions of software graphs (here, we show the distributions for the computer game ProRally 2002). The in-degree distribution is the probability that a given module has been reused by other k_{in} modules. Conversely, the out-degree distribution is the probability that a module depends on k_{out} modules. Notice the sharp cut-off in the out-degree distribution (from [10]).

Network motifs are defined in terms of subgraphs which appear much more often than expected from pure chance. Specifically, they occur with a high frequency compared with the expected from an ensemble of randomized graphs with identical degree structure [24]. The random networks are generated by means of the switching rule. For every pair of links $i \to j$ and $u \to v$ in the original software network, we add the pair $i \to v$ and $j \to u$ in the randomized network. This rule keeps intact the in/out-degree sequences but destroys degree-degree correlations. The statistical significance of a given subgraph Ω_i is described by its Z score [24], defined as:

$$Z(\Omega_i) = \frac{N_{real}(\Omega_i) - N_{rand}(\Omega_i)}{\sigma(N_{rand}(\Omega_i))} \tag{1}$$

Here $N_{real}(\Omega_i)$ is the number of times the subgraph appears in the network, whereas $N_{rand}(\Omega_i)$ and $\sigma(N_{rand}(\Omega_i))$ refer to the mean and standard deviation of its appearances in the randomized ensemble, respectively. In order to be significant, it is required that $|Z(\Omega_i)| > 2$. When $Z(\Omega_i) > 2$ ($Z(\Omega_i) < -2$) the motif (antimotif) is considered to be more (less) common than expected from random (see fig. 4). Itzkovitz et al. [25] have shown that the average number of appearances G of a given subgraph is given by a product of moments of different orders of the indegree, outdegree and mutual degree distributions:

$$\langle G \rangle = \frac{aN^{n-g_a-g_m}}{\langle K \rangle^{g_a} \langle M \rangle^{g_m}} \prod_{j=1}^{n} \left\langle \begin{pmatrix} K_i \\ k_j \end{pmatrix} \begin{pmatrix} R_i \\ r_i \end{pmatrix} \begin{pmatrix} M_i \\ m_j \end{pmatrix} \right\rangle_i$$

where the approximation assumes uncorrelated, sparse networks ($K \ll N$). Both conditions are met by software maps [8]. These mean-field quantities can be used as a null model estimate of the number of motifs, and eventually to detect stray, significant deviations form randomness. Since different motifs are found in different systems [24] they can actually allow us to identify the basic functional blocks for a given class of networks.

However, it turns out that software motifs (at the module level) are not selected because they are required for doing some special function but because of the evolutionary dynamics. The frequency of network motifs in a software graph can be predicted by a very simple network growth model of duplication-diversification that does not take into account any functional requirement[1]. This model is based on previous models of proteome evolution but it has been extended in order to produce directed networks. Network growth is described by the algorithm in figure 5. First, an initial random (so-called backbone) network of $m_0 < N$ nodes is created. The backbone follows a tree-like structure. This simple assumption seems to be validated in initial stages of development concerning real software architectures. Duplication starts from the previous backbone by adding a node w at a time until N nodes have been added to the network. The new node w inherits its links by cloning them from another randomly selected node v. For each pair of original/redundant links remove one of them with probability β. Notice this copy process is separated in two loops for the incoming and outgoing links, respectively. In addition, a link $w \to v$ is introduced with probability γ.

In order to compare the network resulting from the previously described duplication-diversification process with a similar-sized real software network it is required to put the right parameters. We have employed the evolutionary algorithm originally introduced by [4]. The pseudocode for this algorithm is shown in figure 6.

A population of $N = 10^2$ networks is used to target a real software graph. The networks start with a given size m_0 and carry also the two parameters (δ, β). Each network is grown until it reaches the same number of links as the real one. The average degree k is then compared to k_{real}. Those networks closer to the real value are selected and the rest are removed and replaced by new networks new different pairs of parameter combinations. The winner through this selection process is used in searching for motif patterns.

Network	Nodes	Edges	N_{real}	N_{rand}	Z_{score}	N_{real}	N_{rand}	Z_{score}	N_{real}	N_{rand}	Z_{score}	N_{real}	N_{rand}	Z_{score}	N_{real}	N_{rand}	Z_{score}
Software Networks (medium)					FFL S38			Bi-parallel S904			FFL extended S344						
Fairuse	106	180	41	11.6±3.3	8.94	18	7.3±3.5	3.01	33	9.5±5.5	4.25						
Aime	143	319		N/A		230	30.8±19	10.46	55	31.8±9.3	2.49						
Filezilla221a	186	331	77	29.4±6.1	7.86	30	10.2±4.5	4.38		n/a							
Aztec	255	391	68	26±5.4	7.82	25	10.6±4.6	3.15	68	14±5.9	9.08						
Exult	261	504	107	56.3±8.4	6.01	86	30.2±8	6.75	182	80.2±19.6	5.18						
Software Networks (large)					Bi-fan S204			Bi-parallel S904			Multi-X S2252			Multi-Z S206			Multi-Y S2190
blender226	495	834	486	138±30.3	11.4	33	16±5.2	3.2	123	7.8±5.8	20	22	3.7±3.6	5.04	18	4.2±3.2	4.37
gtk221	748	1147	644	189±33.8	13.4	119	25±7.5	12.5	173	26±16.7	8.8	21	2.0±1.7	10.9	19	4.2±2.7	5.41
vtk	771	1362	512	262±39.9	6.27	295	69±14.3	15.8	41	6.3±3.2	10.7	193	27.7±14.7	11.2	122	12.8±5.8	18.7
java2	1364	1947	816	189±35.5	17.7	173	48±10.8	11.5	345	18.4±14	23.3	22	2.2±2.1	9.5	17	3.8±2.2	5.99
prorally	1993	4987	22750	1840±171	122	3848	322±34.2	103.1	1080	144±50.6	18.4	210	28.7±9.2	19.8	1318	55.5±14.7	85.9
Software Networks (large)					FFL S38			S472			Multi-X incomp. S2186			Multi-X incomp S408			FFL extended S344
blender226			126	33.8±6.1	15		N/A		1976	766±162	7.43	436	196±65	3.7	94	26.2±8.6	7.88
gtk221			118	47.7±9.6	7.31	15	3.1±2.3	5.06	4177	1941±398	5.6	1462	748±261	2.73	188	68.1±13.7	8.73
vtk			229	81.6±10.6	13.9	30	13±6.7	2.53	707	388±61.6	5.17	333	217±44	2.62	718	212.1±49.1	10.3
java2			176	46.2±9	14.4	8	1.8±1.4	4.57	10212	6346±1180	3.2	2494	1397±522	2.1	257	52.5±17.6	11.6
prorally			1169	272±21	42.6	282	30.8±10.4	24.2	25099	15482±1750	5.5	5742	4163±752	2.1	2909	736±101.4	21.4
Gene Regulation (transcription)					Bi-fan S204			Multi-X incomp S2186			Multi-X incomp S408			Multi-X S2252			
E. coli	495	834	209	65±14.7	9.8	1270	349±185	4.9	33	16±5.2	3.2	33	16±5.2	3.2			
Neurons					FFL S38			Bi-fan S204			Bi-parallel S904						
C. elegans	252	509	209	65±14.7	9.8	127	55±13	5.3	227	35±10	20						

Figure 4: Network motifs with $n = 3, 4$ elements found in software graphs. The numbers of node and edges for each network are shown. The most frequent motifs where classified in distinct rows for each type of system: medium software systems, large software systems, cellular and neural networks. For each motif, we display the number of ocurrences in the network (N_{real}), the number of occurrences ($N_{rand} \pm SD$) in a set of 100 randomized network versions and a qualitative measure of its statistical significance as given by the Z score (see text). Medium and large software networks share a large amount of motifs but we found larger variability in the medium data set. A remarkable difference is motif 2190 (the last motif in the second row), which appeared only in the context of large software systems (from [1]).

```
FUNCTION Network_Growth ( N: INTEGER, m0: INTEGER,
gamma: REAL, beta: REAL)
   G: NETWORK;
   // Create Backbone
   FOR i := 1 TO m0 DO
      FOR k := 1 TO 2 DO
        j = RANDOM(0, i-1);
        ADD_LINK(G, i, j);
     NEXT k
   NEXT i
   // Duplication-based Evolution
   FOR i := m0 TO (N-1) DO
      v := RANDOM(0, i-1);
      FOR every link {v->j} DO
         IF (RANDOM(0,1.0) < beta THEN
             IF (RANDOM (0, 1.0)< 0.5 THEN
                  REMOVE_LINK(G, v, j);
                  ADD_LINK (G, i+1, j);
             ENDIF
         ELSE
             ADD_LINK (G, i+1, j);
         ENDIF
      NEXT
      FOR every link {j->v} DO
         IF (RANDOM(0,1.0) < beta THEN
             IF (RANDOM (0, 1.0)< 0.5 THEN
                  REMOVE_LINK(G, j, v);
                  ADD_LINK (G, j, i+1);
             ENDIF
         ELSE
             ADD_LINK (G, j, i+1);
         ENDIF
      NEXT
      IF (RANDOM(0, 1.0) <= gamma) THEN
         ADD_LINK (G, i+1, v);
      ENDIF
    NEXT i;
    RETURN G;
END
```

Figure 5: Network growing algorithm based on duplication and drift. The model has two parameters β and γ that control the rate of link removal and the probability of adding a link from the new node to its ancestor.

```
FUNCTION Macroevolution ( N: INTEGER, L: INTEGER)
  m0 : ARRAY [1..NGENES] OF INTEGER;
  gamma : ARRAY [1..NGENES] OF REAL;
  beta : ARRAY [1..NGENES] OF REAL;
  fit : ARRAY [1..NGENES] OF INTEGER;
  G, Gbest : NETWORK;
  fitbest: INTEGER;
  meanfit: REAL;

  FOR i:= 1 TO NGENES DO
    m0[i] := RANDOM (0, N-1);
    gamma[i] := RANDOM (0, 1.0);
    beta[i] := RANDOM (0, 1.0);
  NEXT i
  fitbest := +1E12;
  WHILE ( fitbest is not good ) DO
    meanfit := 0;
    FOR i:= 1 TO NGENES DO
      G := Network_Growth(N, m0[i], gamma[i], beta[i]);
      fit[i]:= ABS(NLINKS(G) - L);
      meanfit := meanfit + fit[i];
      IF fit[i] < fitbest THEN
        fitbest = fit[i];
        Gbest := G;
      ENDIF
    NEXT i
    meanfit := meanfit / NGENES;

    FOR i:= 1 TO NGENES DO
      IF (fit[i] < meanfit) AND
           (RANDOM (0, 1.0) > MUTRATE ) THEN
        Perturbes m0[i], gamma[i] and beta[i]
        by an small amount;
      ELSE
        m0[i] := RANDOM (0, N-1);
        gamma[i] := RANDOM (0, 1.0);
        beta[i] := RANDOM (0, 1.0);
      ENDIF
    NEXT i;
  END
  RETURN Gbest;
END
```

Figure 6: This evolutionary algorithm employs a population of networks in order to find the (m_0, β, γ) configuration that better fits the given number of nodes and links.

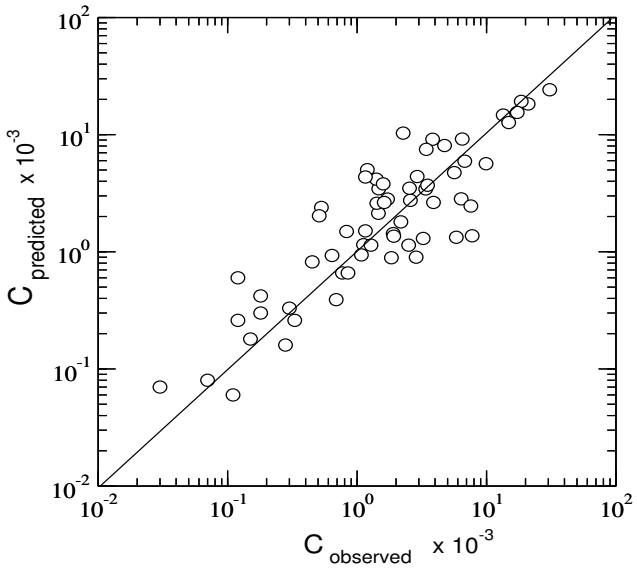


Figure 7: Comparison of given 4-motif concentrations for Prorally, Filezilla, Exult, Blender, GTK and VTK and their best-fit network obtained with the duplication-deletion model (see text). Here $C_{observed}, C_{predicted}$ indicate the concentrations ($\times 10^{-3}$) of motifs (from [1]).

Detailed comparison of the motifs found in real software networks and their synthetic counterparts show similar levels or motif concentrations(see figure 7). Here the concentration C is simply the number of appearances of the 4-subgraph over the total number of 4-subgraphs found in the network. This result indicates that software motifs (at the module level) emerge due to evolutionary constraints posed to the very same design process. It seems likely that later those motifs will be exploited for implementing the required functions and unlikely that they are selected previously by design as suggested in [24]. However, the link between structure and function might be reflected at lower software scales(such as flow diagrams). Some future work should be devoted in order to clarify this question. Software modules are a sort of black box that deals more with semantic and evolution than with functional features. Any motif study trying to uncover this link should be addressed below the module level analyzed here.

Electronic circuits offer a different, complementary view of technology. As shown by previous studies, we know that real electronic circuits display small-world structure together with a surprising degree of heterogeneity. In spite that those circuits share some topological features with cellular networks, they completely fail to perform their function when any single component fails. This means an extreme fragility that is not shared by biological systems, where failures of isolated components is compensated by a number of mechanisms. Although it was though for a long time that the reason of such reliability under component failure was redundancy, mounting evidence has shown that a different mechanism, so called degeneracy or distributed robustness, is responsible for this type of homeostatic behavior. The origins of degeneracy remain largely unknown.

In order to gain some insight into the origins of degeneracy, we have also analysed the outcome of evolving in silico electronic circuits. Although this type of analysis is not new (evolvable hardware is a whole field of research) no attention has been devoted to explore in detail the structure of the landscapes of evolving circuits and in particular the degree of redundancy and degeneracy of the resulting designs. Our work in progress has already shown that a large amount of distributed robustness is achieved for free as a byproduct of the evolutionary rules. The observed topologies are also consistent with patterns displayed by biological networks. This result is encouraging, since it suggests that we can answer some of the key questions formulated here both in terms of understanding

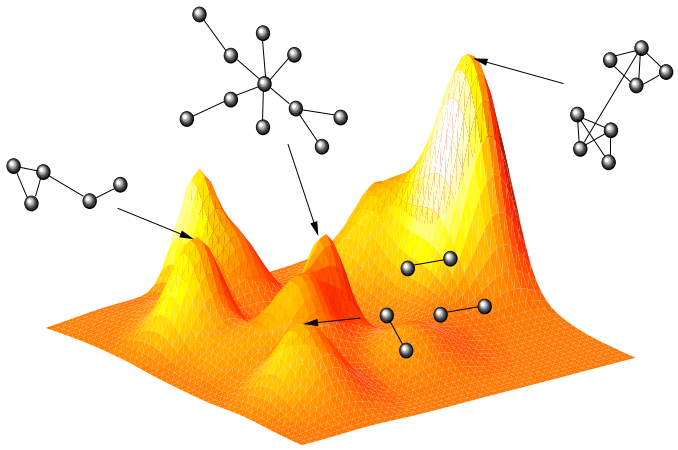


Figure 8: What is the structure of technological landscape networks? Is it related in some way to the one displayed by natural systems or their in silico counterparts? One key ingredient in order to determine the links between both types of systems involves dealing with the structure of their landscapes. Although theoretical studies have pointed out towards rugged landscapes (such as the one shown here for illustration) as the standard picture, increasing evidence suggests that real landscapes might actually display a high degree of neutrality.

degeneracy in bionetworks but also in applying it into new forms of design.

1.4 Computation in Networks: Neutrality, Reliability and Noise

One particularly well-defined class of network where a full analysis of the landscape structure and its meaning in terms of computation, phenotype-genotype mapping and evolvability is given by feedworward networks (FFN). Such networks involve directional cascades of changes with no loops. They are well known within the context of parallel distributed processing and have been used as metaphors for signaling cascades and cortical networks. However, little is known about their fitness landscapes. In this project we have started an exploration of such landscapes by using deterministic FFN computing a large number of Boolean functions.

Our first results are reveal a remarkable feature: the structure of these landscapes is surprisingly similar in many ways to the neutral landscapes found by Peter Stadler, Walter Fontana, Peter Schuster and co-workers in relation with RNA folding. For this particular problem, structure and function are described in terms of a sequence (the RNA sequence) and the way the chain folds in space to adopt a given shape. The shape is responsible for the function, since it allows the RNA molecule to bind and recognize specific targets. In our study we have seen that neutral networks percolate phenotype space: (a) there are always single-mutation neighbours (i. e. networks differing in a single change of a given link) of a given web exhibiting the same phenotype (i. e. performing the same computation) and (b) there is no need to search in large domains of genotype (wiring) space to find a given phenotype (i. e. a Boolean computation being performed) since all phenotypes are already present in a small neighborhood of any given sequence. These two observations, shared by previous studies on RNA folding, indicate that there is in principle a vast neutral space in which networks might not experience changes in their performance but allowing to explore a potentially large set of diverse states providing a strong source of evolvability.

A second project that has already provided very interesting results concerns the presence of noise in networks performing computations. The questions explored here involve the processes character-

istic of computational systems in nature and their correspondence with specific network structures. Although a rich literature (going back to the 1950s) exists on reliable computation under the presence of noise, such studies involved the assumption of well-defined hierarchical structures with predefined topology (either at the large scale or at the module scale). However, natural computations seldom occur in a fully hierarchical and sequential fashion, as it is common in standard computers. Standard models of computation typically assume a noise-free scenario with reliable units. Early studies of computation by Alan Turing and John von Neumann already explored the interplay between architecture (topology), computation and noise. The stochasticity was essentially associated to the nonreliable character of units. The fundamental question was then what sort of topological structures would be consistent with reliable computation from unreliable units. The most obvious consequence of such studies was that a high degree of redundancy was required in order to avoid system's failures.

Our ongoing research, in which Boolean functions have to be computed under the presence of faulty, noisy units, includes an important component: network evolution. In contrast with most previous works, we explicitly consider evolutionary rules that drive network evolution and are coupled to fitness. These models show that computations can be successfully done provided that some particular types of topologies are adopted. preliminary results indicate that the link between topology and reliable function is strongly associated to optimal coding.

1.5 Conclusions and prospects

Our ongoing research within DELIS is being developed towards understanding the origins of network complexity in information-processing systems and how it can be obtained under biologically-inspired approaches. In this context, we have been exploring the architecture of complex software graphs, cellular networks and in silico models of evolved circuits. All these systems appear to share some relevant features, including high degrees of heterogeneity, modular structure and some additional features that appear to come "for free", such as degeneracy. A first step in our approach is the exploration of the subgraph structure present in large software systems described at the class level. Since software systems are assumed to be the result of engineered design, a key assumption is that they should have been designed as efficient systems. When looking at the subgraph level, we find a set of common subgraphs, some of them shared by all systems and others more specific. Although most of these common subgraphs are shared by some biological networks (such as genetic regulatory webs and neural maps) we have found evidence for a mechanism of subgraph generation essentially based on graph growth through duplication and rewiring. Using a simple model (previously applied to genome growth dynamics) lacking any kind of functionality -and thus not optimized for efficient performance- we have shown that it is able to reproduce the concentrations of subgraphs found in real software architectures. Work in progress is extending this analysis towards other related systems, such as electronic circuits, both designed and evolved. By using different levels of description and considering functionality in explicit form, we hope to understand the origins of subgraph frequencies and its possible meaning, or either if they might be a by-product of network growth dynamics.

1.6 Acknowledgements

This work partially supported by the EU within the 6th Framework Program under contract 001907 (DELIS), grants BFM 2001-2154 and by the Santa Fe Institute.

2 Duplication and Re-wiring for Cooperation in Peer-to-Peer Networks

In this chapter we describe a simple algorithm that allows selfish adaptive peers in a network to maintain high levels of cooperation whilst performing the collective task of file sharing. The algorithm is adapted from novel tag models of cooperation that do not rely on explicit reciprocity, reputation or

trust mechanisms. Tags have not previously been applied to computer networks but have been used as a possible new kind of biological or cultural evolutionary mechanism that might explain certain kinds of seemingly irrational group-like behaviour in animals and humans.

We present a sequence of three simulation models - starting with an abstract model of tag-based cooperation (TagWorld) and finishing with a peer-to-peer file-sharing model (FileWorld) that puts the technique to work.

From analysis of extensive computer simulations, we demonstrate the technique to be scalable, robust and decentralized requiring no central servers or authorities. The simplicity of the algorithm means peers do not need to store additional trust information about other nodes or to perform significant additional processing.

Interestingly, the proposed algorithm (Evolutionary Rewiring Algorithm - ERA), makes use of simple node level rules in which there is *duplication and re-wiring*. In the current simulations network size is constant, but the same rules could grow a network.

An open question is how the evolving topologies of ERA-based networks would relate to the kinds of patterns identified in the analysis given in chapter 1. More interestingly, it would seem that results obtained from empirical analysis of existing functional networks (such as software graphs and existing P2P systems) may inform the design of node level algorithms for producing and maintaining functional structures in dynamic networks.

2.1 Introduction

Recently a number of novel cooperation models based on tags [27]-[30] have been presented. They were designed to explore theories of cooperation and coordination in the social sciences. We show how they can be applied to produce efficient peer-to-peer (P2P) networks.

We migrate the tag mechanism from a social science oriented model to a practically inspired P2P file-sharing model. In order to achieve this migration we present a sequence of three models moving from the initially abstract model to the final applied simulation model.

The initial model, TagWorld (section 2.3), demonstrates the emergence of cooperation in a population of evolving agents that interact randomly in pairs to play a cooperation game called the Prisoners Dilemma (PD) game. The second model, NetWorld (section 2.4), adapts the TagWorld by situating interactions between neighbors on a dynamic graph topologically similar to a P2P network. In the final model, FileWorld (section 2.5), we maintain the graph topology but change the task from the PD game to file sharing.

As we adapt the models, we check that we preserve the desirable scalability and robustness properties of the initial model while still emerging the required level of cooperation.

After presenting the models and simulation results, we compare related work (section 2.6) and conclude with a brief discussion (section 2.7).

Initially, in the next section, we introduce the motivating problem, how to control selfishness in P2P systems, and our general method drawing on existing social simulation models.

2.2 Social Mechanisms for P2P Systems

Social Scientists have always been interested in complex, self-organizing, emergent systems because their target is naturally occurring societies that display these properties.

Recently a wealth of, agent-based, computer models of social phenomena have been advanced [31]-[35]. Such models specify individual rules followed by multiple agents and the consequent emergent results when they are executed in a shared environment. Results often demonstrate properties that would appear desirable in engineered large-scale distributed systems.

Here then, it would seem, is a developing body of work that can be used by engineers of self-organizing systems. However, sociologically inspired models are realized at a high level of abstraction

- very different from the application task domains in which engineers are interested. How can the results from such models be used for the solution of practical engineering problems?

The approach we use here is to move, via a succession of models, from an initial abstract model to the actual application by introducing piecemeal refinements, while preserving the, emergent, phenomena of interest [36].

In this article, we take a sociologically inspired model that produces high cooperation between selfish entities and develop two further models that move closer to an application domain, peer-to-peer file sharing, in which high cooperation is desirable. Our initial results are extremely encouraging indicating that novel and useful techniques can be imported from sociologically inspired models.

2.2.1 The tragedy of the commons in P2P networks

Peer-to-Peer file sharing systems using unstructured overlay type networks have become very popular over recent years [37]-[39]. At a given time millions of users (peers) may be running, connected over the Internet. Each peer (or node) connects to some set of known peers forming a graph type network. These applications are self-organizing in that peer software is provided freely for downloading with users deciding when to download and execute these peers on their hardware. There is no centralized administration or control and such systems are open because there is nothing stopping peers from modifying their software (and hence behavior). In addition, users can choose to share nothing or only poor quality files. Given these latter considerations and empirical evidence [37] it can be argued that a major problem in such applications is to develop mechanisms that discourage selfish behavior, where peers download files without uploading them, and encourage altruistic behavior.

These kinds of situations, where all individuals benefit if all act altruistically but each has an incentive to act selfishly, called commons tragedies [40], are well studied in the social sciences since they occur in many situations in naturally occurring social systems (e.g. over-grazing on a common plot of land or polluting the environment).

In P2P systems, this problem is evident in many applications, not just file sharing (e.g. the sharing of processing power or storage, the passing of messages and performing remote operations). Hence, techniques that can address the commons tragedy would appear to have wide applications within P2P systems.

Existing models from the social and biological sciences suggest many possible candidate mechanisms for addressing this problem, including, reciprocal altruism [41], [42] group reputation [43] and interaction on fixed topological structures [44].

In this chapter, we focus on a recent novel tag model that solves an abstract form of a commons tragedy without requiring complex algorithms, reputational knowledge or fixed interaction topologies. We start with the abstract model, TagWorld (section 2.3), demonstrating how the tag mechanism produces cooperation and then progressively modify it and produce two further models: NetWorld (section 2.4) and FileWorld (section2.5). NetWorld situates interaction on a P2P type network topology. FileWorld changes the task domain from an abstract game to a simulated P2P file-sharing task.

2.3 The TagWorld Model - How Tags Work

Here we introduce the TagWorld model [27] that demonstrates a mechanism promoting cooperation between simple rational entities acting in a self-interested way. The simulations show that self-interested agents playing the game, normally known as, the Prisoner Dilemma (PD), that usually leads to non-cooperation, can instead be made to cooperate by the use of tags. The results from the TagWorld experiments serve as the foundation upon which further experiments are performed - firstly, to a networked, and then, to a P2P scenario, in which the tag technique is used as a mechanism to incentivise cooperation.

In the remainder of this section, we introduce the PD game, then the "tag" concept, then the model and results.

2.3.1 The Prisoner's Dilemma

The Prisoner's Dilemma (PD) game captures a situation in which there is a contradiction between collective and self-interest. It can be considered as a minimal and abstracted form of a commons tragedy. Two players interact by selecting one of two choices: to "cooperate" (C) or "defect" (D). For the four possible outcomes of the game, players receive specified payoffs. Both players receive a reward payoff (R) and a punishment payoff (P) for mutual cooperation and mutual defection respectively. However, when individuals select different moves, differential payoffs of temptation (T) and sucker (S) are awarded to the defector and the cooperator respectively. Assuming that neither player can know in advance which move the other will make and wishes to maximize her own payoff, the dilemma is evident in the ranking of payoffs: $T > R > P > S$ and the constraint that $2R > T+S$. Although both players would prefer T, only one can attain it. No player wants S. No matter what the other player does, by selecting a D move a player ensures she gets either a better or equal payoff to her partner. In this sense a D move can't be bettered since playing D ensures that the defector cannot be suckered. This is the so-called "Nash" equilibrium for the single round game hence an ideally rational selfish player would always choose D. It is also an evolutionary stable strategy (ESS) for a population of randomly paired players, where reproduction fitness is proportional to payoff hence evolution also selects D [45].

Therefore, the dilemma is that if both individuals selected a cooperative move they would both be better off but both evolutionary pressure and game theoretical rationality result in defection.

2.3.2 Tags

Tags are markings or social cues that are attached to individuals (agents) and are observable by others [46]. Often represented in models by a single number or a bit string; they evolve like any other trait in a given evolutionary model. The key point is that the tags have no direct behavioral implication for the agents that carry them. But through indirect effects, such as the restriction of interaction to those with the same tag value, they can evolve from initially random values into complex ever changing patterns that serve to structure interactions. A number of tag models have been applied to social dilemma type scenarios [27], [29], [30] but only the TagWorld model appears to produce cooperation in the single-round PD game (i.e. interactions with strangers) and we envisage this would be something that would be beneficial in our target P2P application.

2.3.3 Description of TagWorld

In TagWorld agents play the PD in pairs in a population with no topological structure. The model is composed of very simple agents. Each agent is represented by a small string of bits. On-going interaction involves pairs of randomly selected agents, with matching tags, playing a single round of PD. Agent bits are initialized informally at random. One bit is designated as the PD strategy bit: agents possessing a 1 bit play C but those possessing a 0 bit play D. The other (L) bits represent the agents tag a binary string. Tag bits do not affect the PD strategy played by the agent but they are observable by all other agents. Fig. 9 gives an outline of the simulation algorithm used.

Each agent is selected in turn to play a single-round of PD. Agents do not selected an opponent randomly but selectively based on the tag string. The opponent is selected randomly from the subset of the population sharing the same tag string as the agent. If this subset is empty, because no other agents have an identical tag, the agent plays against some randomly chosen partner from the entire population whatever their tag values.

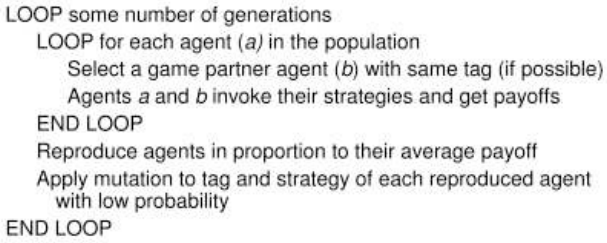


Figure 9: Pseudo-code algorithm for main loop of the model.

After each pair of agents plays a game of PD the payoffs are accumulated against each agent. When all agents have been selected in turn, and played a game, agents are reproduced probabilistically in proportion to the average payoff they received (using a roulette wheel selection algorithm). With a small probability, each bit of each reproduced agent is mutated (i.e. flipped).

2.3.4 Results from TagWorld

Extensive experimentation varying a number of parameters showed that for a large enough number of tag bits; high levels of cooperation quickly predominate in the population. High cooperation is obtain when there is at least $L \geq 32$ tag bits (per agent) for a population of 100 agents with a mutation rate of $m = 0.001$ and PD payoffs of $T = 1.9, R = 1, P = S = 0.0001$. P and S are set to the same small value for simplicity. If a small value is added to P (enforcing $T > R > P > S$) results are not significantly changed. If tags are removed from the model making game pairing completely random then the population quickly goes to complete defection - the Nash equilibrium for the single-round PD.

More interesting still, if all the agents are initially set to select action D (as opposed to randomly set) then the time required to achieve a system where C actions predominate is found to monotonically decrease as population size increases. This is an inverse scaling phenomena: the more agents, the better. Additionally, the fact that the system can recover from a state of total D actions to almost total C actions, under conditions of constant mutation, demonstrates robustness to noise.

The TagWorld produces an efficient, scalable and robust solution based on very simple individual learning methods here modeled as reproduction with mutation. How do tags produce this result? We discuss this in the next section.

2.3.5 How Tags Work

The key to understanding the tag process is to realize that agents with identical tags can be seen as forming an interaction group or tribe. The population can be considered as partitioned into a set of such groups. If a group happens to be entirely composed of agents selecting action C (a cooperative group) then the agents within the group will outperform agents in a group composed entirely of agents selecting action D (a selfish group). This means that individuals in cooperative groups will tend to reproduce more than agents in selfish groups because they will obtain higher average payoffs. If an agent happens to select action D within a cooperative group then it will individually outperform any C acting agent in that group and, initially at least, any other C acting agent in the population remember the T payoff is 1.9 but the best a C acting agent can do is R = 1.

However, due to its high payoff such a D acting agent will tend to reproduce many copies of itself and then the group to which it belongs becomes very quickly dominated by the newly reproduced D acting agents. The group then becomes a selfish group and the relative advantage of the lone D

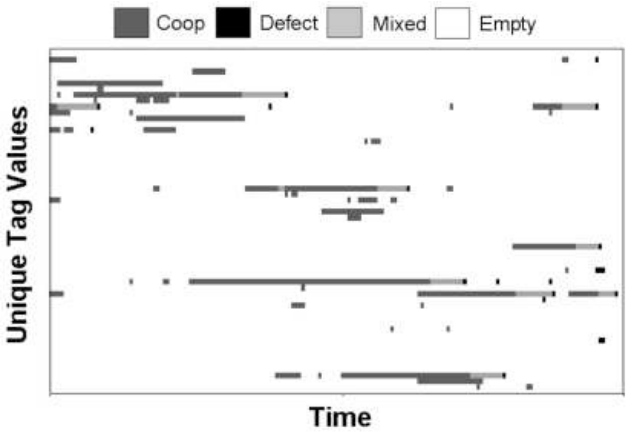


Figure 10: Visualization of 200 cycles (generations) from a single simulation run showing cooperative groups coming into and going out of existence. See the text for an explanation

acting agent is lost the group snuffs itself out due to the interaction being kept within the group. So by selecting the D action an agent destroys its group very quickly (remember groups are agents all sharing an identical tag). Fig. 10 visualizes this group process in a typical single run. Each line on the vertical axis represents a unique tag string (i.e. a possible group). Groups composed of all C action agents are shown in light gray, mixed groups of C and D agents are dark gray and groups composed of all D are black.

2.3.6 TagWorld Discussion

The results indicate that by partitioning the population into distinct interaction groups (or tribes) tags facilitate a kind of group selective process in which defection is kept in check. However, from these results, it is unclear how important the binary string form of the tags is does the large string (remember $L \geq 32$ bits for high cooperation) impose some kind of necessary topological structure? Previous experiments with the model in which the tag was replaced by a single value (a real or integer number) produced low cooperation so it appeared that the bit string structure was important. However, after further experimentation, we discovered that the significant factor was differential mutation between tag and strategy. An artifact of a long tag string is that, given mutation is applied with equal probability to each bit; the mutation rate applied to the tag as a whole relative to the PD strategy is significantly higher. Mutation applied to the tag as a whole is $1-(1-m)^L \approx 0.0315$ (where $m = 0.001$ and $L = 32$). This means that the effective mutation rate on the tag is well over one order of magnitude higher than on the strategy.

In order to test if differential mutation alone produced the high cooperation we re-implemented the TagWorld model but replaced the binary tags with a single real number, allowing tags to take values between [0..1], and increased mutation on the tag by a factor (f) relative to the strategy bit. We experimented with different values for f and found that indeed a high f produced high cooperation whereas a low f gives low cooperation. Typical results are shown in Fig. 11. Notice that cooperation comes in three phases a low phase with $f < 4$, a high phase with $f > 6$, and an uncertain phase between the two, where the populations can go into either high or low cooperative regimes depending on initial conditions and on-going stochasticities (i.e. different numbers from the pseudo random number generator in each run).

Minimally these results indicate that there is nothing special about binary strings of tags rather it is differential mutation that is important. The tag needs to change faster than the strategy. Intuitively this makes some sense, the mechanism of cooperation is driven by cooperative groups

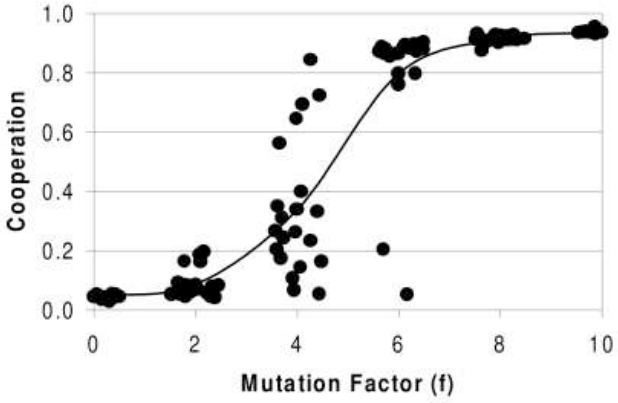


Figure 11: Results from each simulation run plotting mutation factor (f) against cooperation. There are 20 runs per f value (slight noise has been added to the x-axis for readability). The solid line represents a smoothed average curve through the runs.

forming more quickly than defectors can invade and destroy them. With high mutation on the tag, a cooperative group, as it grows, will tend to spawn more new cooperative groups. A group containing defectors does not grow and quickly deflates and dies (as discussed above).

We wish to move these desirable cooperative properties into a scenario that is closer to our target P2P file-sharing application. We begin this process by implementing a new model in which we transplant the PD interactions onto a graph or network topology as a first step towards our P2P application. Will cooperation still be produced if interactions occur on a network?

2.4 The NetWorld Model - From Tags to Networks

We now consider how to translate the cooperation producing tag mechanism into a network. We represent the network as an undirected graph in which each vertex represents a node and each edge represents a link between nodes. We assume each node has a fixed capacity (a maximum degree) of links defining its neighbors (a neighbor list).

The underlying mechanism driving cooperation within the TagWorld is the formation and dissolution of sharply delineated groups of agents (identified by sharing the same tag). Each agent could locate group members from the entire population. Each member of the group had an equiprobable chance of interacting with any agent in the population sharing the same tag. In this sense each agent could determine which agents were in their group and always selected an in-group agent to play PD with if this was possible.

If we consider a sparse P2P network in which each node (or peer) knows of some small number of other nodes (neighbors) and those neighborhoods are highly interconnected (clustered) such that most neighbors share a large proportion of other neighbors then we have something similar to the tag-like groupings (or tribes) in TagWorld. Instead of a tag (an observable marker) we have an explicit list of neighbors. In a highly clustered network the same list will be shared by most of the neighborhood. In this sense, one can visualize the table of known peers (the neighbor list) stored in each node as something similar to a tag. It is shared by the group and is the key by which the group can directly interact with each other. To this extent it defines a group boundary. A nice feature of this way of defining group boundaries is that it offers a watertight method of isolating nodes into their neighborhoods since nodes cannot directly interact with other nodes that it does not know of (i.e. that are not in their neighbor list).

Initially we will consider only direct interactions between neighbors in our network model. In

a sense this is all that can ever happen in P2P systems. Indirect interactions between nodes that do not share neighborhoods have to be mediated by direct interactions between intermediate nodes. Essentially, one can view all interaction as with neighbors - even if those neighbors are actually proxies for other more remote nodes. If cooperation can be established between the majority of neighborhoods in a network then it follows that any pair of nodes in the network that are indirectly connected will have a good chance of being able to find a path of cooperation through the network. In order to capture this kind of neighborhood interaction in the simplest possible way we have each node in the network play a single round of PD (see above) with a randomly chosen neighbor. No information is stored or communicated about past interactions and the topology is not fixed (see below).

2.4.1 The Evolutionary Rewiring Algorithm (ERA)

In the TagWorld model change was produced over time by mutation and differential reproduction based on average payoff. How can these be translated into the network? The interpretations placed on previous tag models have been biological or cultural. But in NetWorld we do not view nodes as reproducing in a biological or cultural sense. We translate this process into the network by allowing nodes to move or rewire within the network. It is consistent with or initial assumptions that nodes may relocate to a new neighborhood in which a node is performing better. That is, we assume that periodically nodes make a comparison of their performance against another node randomly chosen from the network. Suppose node i compares itself to j. If j has a higher average payoff then i erases its neighbor list and copies the strategy and neighbor list of j also adding j into the list. This process of copying can be visualized as movement of the node into a new neighborhood that appears more desirable.

Mutation in the TagWord model was applied after reproduction. Each bit of the tag and the strategy was mutated (flipped) with low probability. Since we are using the same one bit strategy we can apply mutation to the strategy in the same way. We therefore flip the strategy bit of a node with low probability (m) immediately after reproduction (the movement to a new neighborhood as described above). Since we treat the list of neighbors in each node as the tag, a mutation operation implies changing the list in some way. However, we cant simply randomly change the list; we need to change the list in such a way as to produce an effect that closely follows the functional effect when mutation is applied in the TagWord model. In that model, tag mutation tended to give agents unique tags i.e. tags not shared by other agents at that time. However, agents could interact with a randomly chosen agent with non-matching tags if none existed with identical tags. In this way tag mutation lead to the founding of new tag groups. In the network model we dont want to isolate the node completely from the network otherwise it will not be able to interact at all. Neither do we want to move into an existing neighborhood (as with reproduction) but rather to do something that may initiate the founding of a new neighborhood. So we pragmatically express tag mutation as the replacement of the existing neighbor list with a single neighbor drawn at random from the network.

We now have analogues of reproduction and mutation for the network model. Reproduction involves the nodes copying the neighbor lists and strategies of others obtaining higher average scores. Mutation involves flipping the strategy with low probability and replacing the neighbor list with a single randomly chosen node with a low probability, see Fig. 12 4.

Perhaps the biggest shift we have made here is in the group-level topology. In the TagWorld model groups had distinct boundaries (those sharing the same tag) and interactions was strictly within groups. Here, groups may overlap since neighboring peers will not necessarily share the same neighbors. In this sense the group boundaries are no longer absolute. This change could be significant and may destroy the cooperation process. For example, a previous tag model of altruism [29], [47] with overlapping and changeable group boundaries is not able to solve a social dilemma such as the PD [48].

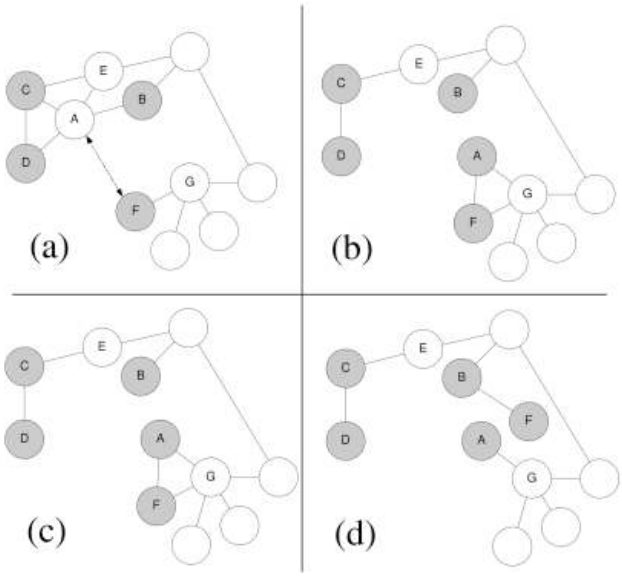


Figure 12: An illustration of replication and mutation as applied in the Evolutionary Rewiring Algorithm (ERA). Shading of nodes represents strategy. In (a) the arrowed link represents a comparison of utility between A and F. Assuming F has higher utility then (b) shows the state of the network after A copies Fs links and strategy and links to F. A possible result of applying mutation to As links is shown in (c) and the strategy is mutated in (d).

In the next section, we describe the NetWorld model and then give results of simulation experiments. We find that although significant properties of the cooperation process are changed; scalable, robust and high cooperation is still produced.

2.4.2 Description of NetWord

The NetWorld model is composed of a set N of nodes (or peers). Each node stores a list of other nodes it knows about (we term this the neighbor list). In addition to the neighbor list each node stores a single strategy bit indicating if it is to cooperate or defect in a single round game of the PD. Neither the strategy bit nor the list is normally visible to other nodes. Initially nodes are allocated a small number of neighbors randomly from the population. Periodically each node selects a neighbor at random from its list and plays a game of PD with it. Each node plays the strategy indicated by its strategy bit. After a game the relevant payoffs are distributed to each agent. Periodically pairs of randomly chosen nodes (i, j) compare average payoffs. If one node has a lower payoff then the strategy and neighbor list from the other node is copied (effectively moving the lower scoring node to a new neighborhood). Mutation is applied to the strategy with probability m and to the neighbor list with probability mf. In all cases given here $f = 10$. Therefore, the mutation on the neighbor table is one order of magnitude higher than on the strategy. We take this from our analysis of relative mutation rates in the previous TagWorld model see later discussion concerning the effect of f in this model. Mutation of the strategy involves flipping the bit. Mutation of the neighbor list involves clearing the list and replacing it with a single randomly chosen node from the population. Fig.13 gives an outline pseudo-code algorithm of the NetWorld simulation main loop.

The neighbor lists are limited in size to a small number of entries. The entries are symmetrical between neighbors (i.e. if node i has an entry for node j in its list then node j will have node i in its list). If a link is made to a node that has a full neighbor list then it discards a randomly chosen

```
LOOP some number of generations
    LOOP for each node (i) in the population of size N
        Select a game partner node (j) randomly from neighbor list
        Node i and j invoke their strategies and get payoffs
    END LOOP
    Select N/2 random pairs of agents (i, j)
    Copy neighbor list and strategy of higher scoring node to lower
      scoring node
    Apply mutation to neighbor list and strategy of each copied
      node with probability m
END LOOP
```

Figure 13: Outline pseudo-code algorithm of the NetWorld simulation main loop.

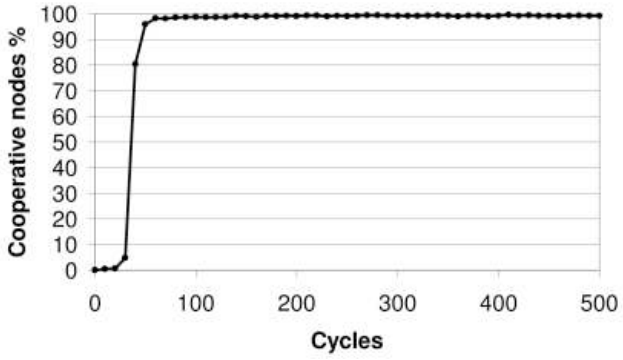


Figure 14: Shows a typical NetWorld time series ($N = 10,000$ nodes) giving the percentage of cooperative nodes at each cycle.

neighbor link in order to make space for the new link. If a node is found to have no neighbors when attempting to play a game of PD (this can happen if all neighbors have moved away) then a randomly chosen node is made a neighbor.

2.4.3 Results from NetWord

Fig. 14 and Fig. 15 give some typical results up to $N = 2 \times 10^4$. Similar results were obtained up to $N = 10^6$. In these experiments mutation rate was $m = 0.001$ (increased by a factor of $f = 10$ when applied to mutation of the neighbor list) and the PD payoffs were as previously described in the TagWorld model. Notice that although there appears to be no scaling cost (with convergence to high cooperation taking approximately the same number of cycles for different N) we have lost the reverse scaling cost property observed in the TagWorld model. Interestingly, when the mutation factor (f) was decreased to 1, making mutation rates equal for links and strategy, high cooperation still occurred but the time to reach it did not scale. We found, what appeared to be, an exponentially increasing non-linear upper bound scaling cost (in time). This meant that some runs were not converging after thousands of cycles for large $N(> 10,000)$. We still, therefore, need the high mutation rate on the links to produce scalable results but the relationship of the mutation factor (f) to the results has changed. If we could understand fully why these scaling costs have changed over the TagWorld model then a deeper understanding of the mechanisms related to scaling costs could be gained. Currently however, this requires further analysis and we leave this for future work.

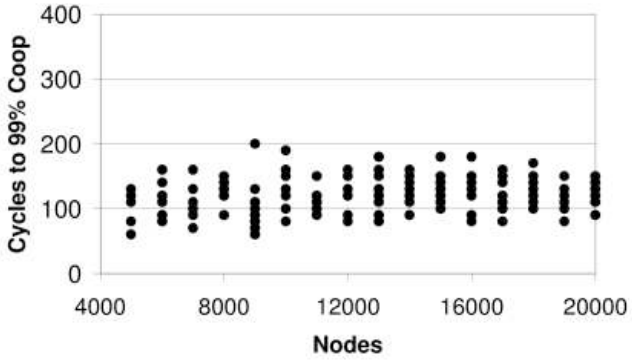


Figure 15: Shows the number of cycles before 99 precent of nodes are cooperative for various values of N in NetWorld. Each dot is an individual run.

2.4.4 NetWorld discussion

The PD task, used in NetWorld, although capturing a minimal form of a commons tragedy, is rather abstract. The behaviors and coordination required is trivial - although the dilemma itself is not trivial. In order to test if the results obtained so far can be carried over into a less abstract task domain we apply the same basic NetWorld algorithm in a new model, FileWorld, which implements a P2P file-sharing scenario.

In the next section we present the model and results. We find that we can indeed use the basic rewiring technique to produce non-selfish file sharing at the network level even when nodes act selfishly at the individual level. We also note a number of important issues that moving from the PD game to something more realistic raises.

2.5 The FileWorld Model - From PD To File Sharing

In this section we outline the results of applying the evolutionary node-rewiring algorithm used in NetWorld in a simple simulated P2P file-sharing scenario [50], [55]. It models a flood-fill query process where nodes periodically generate and forward queries to their neighbors. Neighbors either process the queries, by producing a hit or forwarding to all their neighbors, or ignore them - depending on their capacities and internal state variables.

Each node i is defined by three state variables: an answering power Ai, a questioning power Pi and a capacity Ci. Both Ai and Pi are real values in the range [0..1]. Ci takes some cardinal value greater than zero. The values of each variable quantifies the behavior of a node over some unit of time t. Ci indicates the capacity of the node, given as total number of queries. When a node generates or answers a query this takes one unit of capacity. Pi gives the proportion of the capacity Ci that will be allocated to generating new queries. Conversely, $1 - Pi$ of the capacity will be allocated to answering queries from other nodes. The answering capacity, Ai, gives a probability that a node can directly match a query - producing a hit. It represents, indirectly, the amount and quality of files served by the node. For the experiments given here all nodes have fixed values of $Ai = 0.4$ and $Ci = 100$ but we allow Pi to be adapted by the node (see below).

Over a single time period each node i may process a total of Ci queries. This capacity is divided between generating $Pi \times Ci$ new queries (passed to neighbor nodes) and reserving enough capacity to process $(1 - Pi)C$i queries from neighbors. Pi therefore represents a measure of selfishness. If $Pi = 1$ then node i uses all its capacity to generate new queries - ignoring queries from neighbors. If $Pi = 0$ then i uses all its capacity processing queries from neighbors - generating none itself.

2.5.1 Outline of a simulated time period

In a simulated time period, $C \times N$ nodes (N = node population, $C = 100$, the same as each Ci) are selected randomly from the population, with replacement, and fired. If a fired node still has capacity to generate queries it generates one query and passes this to its neighbors otherwise the node takes no action. When a node i receives a query, if it has spare capacity, it processes the query. With probability Ai a hit is produced for the query. If no hit is produced the query is passed to the neighbor nodes of i. If a node has no capacity left to process a query it is ignored no action is taken and the query is not passed on. Queries are not passed on indefinitely but have a preset time-to-live (TTL) after which they are ignored by all nodes. In all experiments presented here TTL = 3 - meaning that queries never get more than a maximum of 3 nodes depth from the originating node. The process of firing nodes in random order with replacement introduces noise in the form of some nodes firing more often than others and some nodes not being able to generate their full quota of queries. We view this as reasonable since it introduces realistic kinds of noise such as non-synchronized nodes with differential processing speeds etc.

2.5.2 Application of the evolutionary rewiring algorithm

After each time period (that is, after $N \times C$ node firings) the evolutionary rewiring algorithm (ERA), as used in NetWorld above, is applied. $N/2$ pairs of nodes (i, j) are selected from the population at random with replacement. If $Ui > Uj$ then node j drops all existing links and copies node is links and additionally links to node i itself. Also Pj is set to Pi (copying the query handling behavior of i). If $Ui < Uj$ then the mirror process is performed (i copying j). In the case $Ui = Uj$ then a randomly selected node (i or j) is designated winner and the process proceeds as if that node had a higher U value.

For the experiments presented here we used a utility (U) value equal to the total number of hits obtained by a node in the time period. Obviously this would tend to be higher if P was higher (generating more queries). We used the utility value of total hits (per node) since this gives an apparent incentive for freeloading acting selfishly by generating queries but not answering them. If the average hits per query is used there is no commons tragedy because nodes wont generally increase their utility by performing more queries.

After any node i copies another node j it applies mutation, with low probability, to the links and the Pi value. With probability m, Pi is changed to a random value selected uniformly from the range [0..1]. With probability $10m$ the links from i are removed and replaced with a single link to a randomly chosen node from the population.

2.5.3 Description of FileWorld

For the purposes of simulation we represent the network as an undirected graph in which the degree of any node is fixed at a maximum value (20 in all cases). When any operation requires a further link from a node, the node simply deletes a randomly chosen existing link and continues with the new link operation. We experimented with various initial topologies for the graph, including randomly connected, lattice, small world and completely disconnected. All produced similar results to those presented here. We also experimented with different initial P values. Again we found we obtained similar results (even when all P values are initially set to zero). Essentially then, we found the results obtained were insensitive to the initial conditions of the network. The results given, unless otherwise stated, start with initially random graphs and randomly selected P values

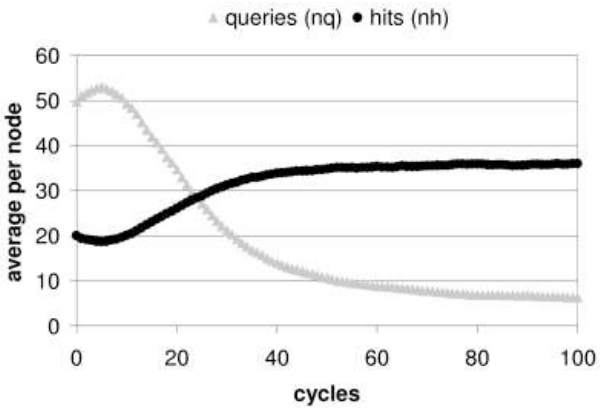


Figure 16: A typical run for a 10^4 node network. Notice that nq and nh values initially get worse than the baseline values ($nq \approx 50, nh \approx 20$) but then quickly improve.

2.5.4 Results from FileWorld

In order to gain a baseline benchmark, that measures how the network behaves without the application of the evolutionary re-wiring algorithm, we ran 10 trials for 10 cycles on static networks with randomly initialized topologies and P values. We did this for a number of network sizes N = [200..51200]. All other values were kept as previously described. Since in the static case nothing changes, the averaging over 10 cycles is done simply to smooth out the stocasticities of the model. Averaging over 10 different trials (with different pseudo-random number seeds) averages over the different initial random network topologies and P values.

We considered the following two measures for benchmarking: the average number of queries generated per node in a cycle (nq) and the average number of hits per node generated per cycle (nh). We found that the baseline level for these measures was, with low variance, nq = 49.45 and nh = 20.13 in all cases. Calculating nh / nq gives an average hit rate per query generated = 0.41. We might expect nq = 0.5 since the P values are selected uniformly randomly but this slightly lower value is a result of the (random selection with replacement) method of firing nodes as described earlier.

Given these baseline values for nq and nh we can investigate the effect of applying the evolutionary rewiring algorithm (ERA). If results give a consistently higher number of hits (nh) by keeping the number of queries generated (nq) low then ERA is suppressing the self-interest of the nodes and thus benefiting the network as a whole. Fig. 16 shows a time series for a typical run for a network of size N = 104 with ERA enabled. As can be seen, over time, nq decreases and nh increases. Notice also that initially these values move in the opposite direction, indicating an initial favoring of selfish behavior, but this is soon corrected. This shows that the evolutionary process (forming cooperative groups within the network) takes a few cycles to get started from the initially randomly initialized network.

Fig. 17 shows nq and nh measures averaged over cycles 40-50 for different network sizes with 10 independent runs for each network size. Notice that as the size of the network increases the variance of the individual runs decreases indicating that larger networks are less sensitive to ongoing stocasticities and initial conditions, since we are using average values this appears intuitive. Fig. 18 shows the numbers of cycles before high hit values are attained (when $nh > 30$). Again 10 independent runs are shown for various network sizes. As before the variance of results decreases as network size increases and there is no significant increase in the number of cycles required for larger networks - suggesting that ERA scales well in this scenario also (i.e. there is no cost for larger networks).

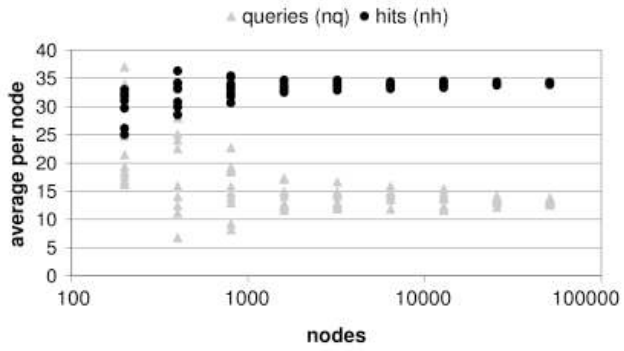


Figure 17: Shows the values of nq and nh (averaged over cycles 40..50) for different network sizes. Ten independent runs for each for each network size.

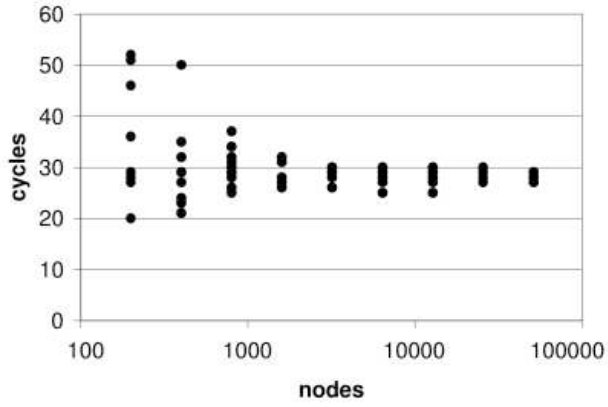


Figure 18: Shows cycles required before high hit values ($nh > 30$) are attained. Ten independent runs for each for each network size are shown.

2.5.5 FileWorld discussion

Our initial experiments with the FileWorld model suggest that ERA does indeed control the self-interest of the nodes by keeping down the number of queries generated and hence increasing total hits. We appear to have preserved the zero scaling cost properties from the NetWorld model. This implies that the ERA has, at least some, general applicability it performed well without changing the basic algorithm for the FileWorld task domain.

Although the ERA performs well in the FileWorld domain, this is currently a rather generous task domain. By fixing the answering power of each node to a relatively high level (40 percent of queries can be answered) we simulate file rich environment [49]. It would be of interest to perform experiments with an initially reduced answering power or add a cost to nodes having high answering powers (diminishing utility in some way) and allow them to adapt in the same way as their links and questioning power. Under those conditions would the system still produce high hit rates? In addition, we have not explored, in depth, the kinds of topologies produced in the NetWorld or FileWorld domains.

In both these scenarios, a network disconnected into a number of components will still produce respectable functionality (since long range routing is not a requirement of these task domains). In future work we may refine the task domain to include long range routing between distant nodes.

Our initial investigations of NetWorld and FileWorld topologies shows that networks tend to form into disconnected components that constantly grow and reform (rather like the tag groups in TagWorld). Interestingly, since the network is in constant flux it would appear that forms of temporal routing might be applicable for long-range task domains [47].

2.6 Related Work

The mechanism we apply is distributed; each node only has to concern itself with its own interactions. In this way, it contrasts to centralized systems of trust that could provide similar kinds of functionality [52].

ERA bears similarities to the more complex SLIC algorithm [49]. However, in that work (from which we adapted our FileWorld task scenario) incentive structures are explicitly programmed, with each node monitoring the service it receives from its neighbors and updating weights which moderate the future service it offers to others. There is therefore explicit retaliation programmed into the model similar to that applied in actual deployed peer applications [53].

In FileWorld model presented here the incentives effectively emerge from the dynamic behavior of the nodes (moving in the network) rather than being explicitly programmed in. Nodes therefore do not need to monitor or store the performance of others reducing overheads. In addition, in SLIC [49] simulations are only applied to scenarios in which single probe nodes behave selfishly nodes do not adapt their behavior to increase their utilities network wide. Consequently, it is not clear how that model would react when all nodes are acting selfishly rather than just a small number (however, this is mentioned as future work).

The APT protocol [54] compares closely with SLIC and ERA but in APT nodes explicitly store trust values based on past interaction and move within the network to maximize these rather than a selfish utility (such as the number of high quality files personally downloaded by the node). In this latter sense, the nodes are programmed to act in a non-selfish way. However, APT is shown to be effective against a number of attach scenarios in which selfish or malicious nodes enter the network and to produce efficient topologies based around content level semantic clustering.

Previous models inspired by game theoretical approaches offer some similar insights [55]. In this latter work, however, network topology is not explicitly modeled and the strategies rely on repeated game interactions between the same entities (i.e. the tit-for-tat strategy in the iterated PD [41], [42]). Such iterated strategies require on-going interactions with recognizable individuals our model does

not rely on this since it is based on mechanisms that work well in the single-round game allowing interactions with strangers. Our model bears some comparison to the social simulation of leadership dynamics presented in [56]. However, this model is deterministic and relies on agents knowing the strategies of others when moving. Never-the-less, agents in the model move around a social network, making and breaking links based on self-interest and play the single round PD insights from this model may be applicable to our approach.

2.7 Concluding Discussion

The desirable properties from novel tag models [27], [28], [30] have been carried over into a P2P file-sharing scenario. The ERA algorithm potentially offers a generally applicable mechanism for controlling selfish behavior in many possible P2P task domains without the need to program and test explicit incentive mechanisms for each domain. In the work presented here, the ERA algorithm effectively emerges an incentive mechanism from the selfish moving behavior of the nodes. This happens because, although it may do well for a while, a very selfish node will tend to lose neighbors as they find other nodes that are members of more cooperative groupings and hence have higher utilities. Additionally, selfish nodes doing well (exploiting neighbors) are a signal for copycats to latch onto them and their neighborhood and exploit it - speeding up the dissolution of the cluster.

We have moved closer to our target, P2P, application by a progressive succession of models. The results appear encouraging and we are confident that further iterations of the process would lead to a sequence of models finishing in a deployable implementation. There are general issues when adapting a model that has started life in the social or biological sphere, towards engineering applicability: firstly, how to capture conceptions of utility and utility comparison; secondly how to interpret and implement reproduction; thirdly, how to deal with non-adaptive agents; fourthly, implementing random selection in the population.

Utility is a fundamental concept in much evolutionary modeling. It is assumed that some value or measure can be determined for each agent that defines its utility or fitness. In the models described in this article the relative simplicity of the scenarios means that selecting a utility measure is easy this may not be the case for more complex domains. This is especially true when rewards are delayed. It would be desirable therefore to eliminate a simplistic conception of utility and move towards a more general performance targets arrangement. We have assumed in our model that agents can freely compare utility but this would be problematical in many application domains. Two possible workarounds are possible; we could abandon maximization of utility and apply a satisficing metric. When agents satisfice they do not change behavior after they reach some aspiration level of performance. This has some intuitive appeal and would appear easy to implement but it raises new questions: how does one determine the aspiration level? How and when, if at all, should it change? Interestingly, because humans are considered to often satisfice - rather than optimize, there are bodies of work, including simulations, exploring this technique going back to the ideas of Simon [57]. The other potential solution is for nodes to monitor the performance they receive from at least two subsets of their neighbors and then drop and replace the links to the poorer performing neighbor subset.

For simulation models of biological or cultural phenomena the idea of reproduction makes sense successful behaviors are copied and increase. Within an application domain such as a P2P system, nodes dont reproduce directly. It can be claimed [55] that we can model nodes as reproducing because this captures the notion of human users of systems installing and using clients that appear to offer desirable results - fast file downloading say. It is important to realize the dramatic implications of this interpretation however; it is that we are no longer modeling potential deployable mechanisms alone but rather the behavior of a system with humans within the loop. Currently there appears insufficient data (although there is some [37]) to know how humans behave in these contexts. This latter problem leads onto our next major issue how do you ensure agents (or nodes in our case) do

in fact follow an adaptive process. In our models we have assumed all the agents follow the same adaptive process but in a real system why would they? A robust system cannot assume this. Our conception of robustness has been that the system is resistant to mutation on selected components under the assumption of utility maximization but what if a subset of the population simply stopped adapting and acted in a non-greedy way: ignoring the fact that others were doing better? How robust would our model be then? This kind of whitewashing behavior can be explored, in each domain, by running various attack scenarios. We intend to test the robustness of the ERA against such attacks in a number of task domains in future work.

The ERA algorithm presented allows nodes to move around a network based on self-interest. Nodes are not required to store any additional information about neighbors or to calculate trust scores. The ERA makes no distinction between stranger nodes and those for which there has been on-going interactions and therefore is suited to a highly dynamic environment. We find it promising that such a simple algorithm has performed well in several simulated scenarios with little or no tweaking of parameters. We aim to developed the ERA and apply it to a number of real P2P application task domains based on newly emerging tools and infrastructures [58].

References

[1] Valverde, S., & Solé, R. V., *Network Motifs in Computational Graphs: A Case Study in Software Architecture*, submitted to *Phys. Rev. E* (2004), also Santa Fe Institute Working Paper (DELIS) [1].

[2] E. W. Dijkstra, "The Structure of the 'T.H.E.' multiprogramming system", Comm. of the ACM, vol. 11, no. 5, pp. 453-457, (1968).

[3] G. Booch, *Object-Oriented Analysis and Design with Applications, Second Edition*, Addison-Wesley, Menlo Park CA, (1993).

[4] J. Marin and R. V. Solé, IEEE Trans. Evol. Comp. **3** (1999) 272; Phys. Rev. E65 (2002) 26207.

[5] J. Martin and J. Odell, *Object oriented methods: a foundation, (UML Edition)*, Prentice Hall, Englewood Cliffs, NJ, (1995).

[6] Gamma, E., Helm R., Johnson R., Vlissides J. (1994) *Design Patterns Elements of Reusable Object-Oriented Software* (Addison-Wesley, New York)

[7] Stroustrup, B., (1986) *The C++ Programming Language* (Addison-Wesley, Reading, MA)

[8] Solé, R. V. and Valverde, S., in: *Complex Networks*, E. Ben-Naim, H. Frauenfelder, and Z. Toroczkai (eds.), Lecture Notes in Physics, pp. 169-190. Springer, Berlin (2004), (DELIS).

[9] R. V. Solé, R. Ferrer-Cancho, J. M. Montoya and S. Valverde, "Selection, Tinkering and Emergence in Complex Networks", Complexity, vol. 8(1), 20-33, (2002). (NON-DELIS)

[10] S. Valverde and R. V. Solé, "Hierarchical Small-Worlds in Software Architecture", Santa Fe Institute Working Paper SFI/03-07-044,(2003). (NON-DELIS).

[11] B . Bollobás. *Random Graphs*, Academic Press, 1985.

[12] S.N. Dorogovtsev and J. F.F. Mendes, *Evolution of Networks: From Biological Nets to the Internet and the WWW*, Oxford, New York, (2003).

[1]Note: Items followed by "(DELIS)' indicate the paper is a DELIS output and has been suitably credited. Items listed as (NON-DELIS) were not DELIS funded

[13] D.J. Watts and S.H. Strogatz, "Collective Dynamics of Small-World Networks", Nature, vol. 393, no. 440, (1998).

[14] S. Valverde, R. Ferrer-Cancho and R. V. Solé, "Scale-Free Networks from Optimal Design", Europhysics Letters, 60, pp. 512-517, (2002).

[15] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks", Science, vol. 286, pp. 509-512, (1999)

[16] Jeong, H., Mason, S., Barabási, A. L. and Oltvai, Z. N. *Nature* **411** (2001) 41.

[17] Jeong, H., Tombor, B., Albert, R., Oltvai, Z. N. and Barabasi, A.-L. *Nature* **407** (2000) 651.

[18] Newman, M. E. J. *Proc. Natl. Acad. Sci. USA* **84** (2001) 404.

[19] Ferrer i Cancho, R. and Solé, R. V. *Procs. Roy. Soc. London B*, **268** (2001) 2261.

[20] H-S. Chae, Y-S. Kwon and D-H. Bae, "A Cohesion measure for object-oriented classes", Soft. Pract. and Exp., vol. 30, 12, pp. 1405-1431, (2000).

[21] M.E.J Newman, S.H. Strogatz and D. J. Watts, "Random Graphs with arbitrary degree distributions and their applications", Phys. Rev. E, vol. 64, 026118, (2001).

[22] J.-L. Guillaume and M. Latapy, "A Realistic Model for Complex Networks", cond-mat/0307095, (2003).

[23] Shen-Orr, S., Milo, R., Mangan, S. & Alon, U., *Nature Genetics*, 31, 64-68 (2002).

[24] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D. & Alon, U., *Science* **298** (2002) 824-827.

[25] Itzkovitz, S., Milo, R., Kashtan, N., Ziv G., & Alon, U. *Phys. Rev. E 68* 026127 (2003)

[26] Alon, U., *Science*, **301**, 1866-1867 (2003)

[27] D. Hales, Cooperation without Space or Memory: Tags, Groups and the Prisoner's Dilemma, in Proc. 2nd Int. Workshop on Multi-Agent-Based Simulation - LNAI 1979, Berlin: Springer, 2000, pp.157-166.

[28] D. Hales, Tag Based Cooperation in Artificial Societies, Ph.D. dissertation, Dept. Comp. Sci., Essex Univ., Colchester, UK, 2000.

[29] R. L. Riolo, M. D. Cohen and R. Axelrod, Evolution of cooperation without reciprocity,. Nature 414, 2001, pp. 441-443.

[30] D. Hales and B. Edmonds, Evolving Social Rationality for MAS using Tags, in Proc. 2nd Int. Conf. on Autonomous Agents and Multi-agent Systems, Melbourne, 2003, ACM Press, pp. 497-503.

[31] J. Epstein and R. Axtell, Growing Artificial Societies: Social Science from the Bottom Up. Cambridge, MA: The MIT Press, 1996.

[32] N. Gilbert and R. Conte, Artificial Societies: the Computer Simulation of Social Life. London: UCL Press, 1995.

[33] N. Gilbert and J. Doran, Simulating Societies: the Computer Simulation of Social Phenomena. London: UCL Press, 1994.

[34] D. Hales, B. Edmonds, E. Norling and J. Rouchier (2003) Multi-Agent-Based Simulation III, LNAI 2927. Berlin: Springer, 2003.

[35] JASSS, The Journal of Artificial Societies and Social Simulation (Online). Available: http://jasss.soc.surrey.ac.uk/JASSS.html

[36] B. Edmonds and J. Bryson, The Insufficiency of Formal Design Methods - the necessity of an experimental approach for the understanding and control of complex MAS, in Proc. 3rd Int. Conf. Autonomous Agents and Multi-agent Systems, New York, 2004, ACM Press.

[37] E. Adar and B. A. Huberman. (2000, October). Free Riding on Gnutella. First Monday (Online). 5(10).

[38] Gnutella. http://www.gnutella.com

[39] Kazaa. http://www.kazaa.com

[40] G. Hardin, The Tragedy of the Commons, Science, 162, 1968, pp. 1243-1248.

[41] R. Axelrod, The Evolution of Cooperation. New York: Basic Books, 1984.

[42] R. Trivers, The evolution of reciprocal altruism, Q. Rev. Biol. 46, 1971, pp. 35-57.

[43] D. Hales, (2002, October). Group Reputation Supports Beneficent Norms. J. of Artificial Societies and Social Simulation (Online). 5(4). Available: http://www.soc.surrey.ac.uk/JASSS/5/4/4.html

[44] M. Nowak and R. May, Evolutionary Games and Spatial Chaos, Nature, 359, 1992, pp. 826-929.

[45] J. M. Smith, Evolution and the Theory of Games. Cambridge: Cambridge University Press, 1982.

[46] J. Holland, The Effect of Labels (Tags) on Social Interactions, Santa Fe Institute Working Paper 93-10-064. Santa Fe, NM, 1993.

[47] K. Sigmund. and M. A. Nowak, Tides of Tolerance, Nature 414, 2001, pp. 403-405.

[48] B. Edmonds and D. Hales. (2003, October). Replication, Replication and Replication - Some Hard Lessons from Model Alignment. J. of Artificial Societies and Social Simulation (Online). 6(4). Available: http://jasss.soc.surrey.ac.uk/6/4/11

[49] Q. Sun and H. Garcia-Molina, SLIC: A Selfish Link-based Incentive Mechanism for Unstructured Peer-to-Peer Networks, in Proc. of 24th IEEE Int. Conf. on Distributed Systems, IEEE Computer Society, 2004.

[50] D. Hales, From Selfish Nodes to Cooperative Networks Emergent Link-based Incentives in Peer-to-Peer Networks, in Proc. of the 4th Int. Conf. on Peer-to-Peer Computing, IEEE Computer Society, 2004. (DELIS)

[51] D. Kempe, J. Kleinberg and A. Kumar, Connectivity and inference problems for temporal networks, in Proc. 32nd ACM Symposium on Theory of Computing, 2000.

[52] S. Kamvar, M. Scholosser and H. Garcia-Molina, The EigenTrust Algorithm for Reputation Management in P2P networks, in Proc.12th Int. World Wide Web Conf., 2003. Available: http://www2003.org/cdrom/

[53] B.Cohen, Incentives build robustness in BitTorrent, presented at the Workshop on Economics in Peer-to-Peer Systems, Berkeley, CA, June 5-6, 2003. Available: http://www.sims.berkeley.edu/research/conferences/p2pecon/

[54] T. Condie, S. Kamvar, H. Garcia-Molina, (2004) Adaptive Peer-To-Peer Topologies, in Proc. of the 4th Int. Conf. on Peer-to-Peer Computing, IEEE Computer Society, 2004.

[55] K. Lai, M. Feldman, I. Stoica, and J. Chuang, Incentives for cooperation in peer-to-peer networks, presented at the Workshop on Economics in Peer-to-Peer Systems, Berkeley, CA, June 5-6, 2003. Available: http://www.sims.berkeley.edu/research/conferences/p2pecon/

[56] M.G. Zimmermann, V. M. Egufluz and M. San Miguel, Cooperation, adaptation and the emergence of leadership, in Economics with Heterogeneous Interacting Agents. A. Kirman and J. B. Zimmermann Eds. Berlin: Springer, 2001, pp. 73-86.

[57] H. A. Simon, Administrative Behavior. 3rd edition. New York: The Free Press, 1976.

[58] M. Jelasity, A. Montresor and O. Babaoglu, (2004), A Modular Paradigm for Building Self-Organizing Peer-to-Peer Applications, in Engineering Self-Organising Systems - Nature-Inspired Approaches to Software Engineering, LNAI 297, G. Di Marzo Serugendo, A. Karageorgos, O. F. Rana, F. Zambonelli Eds. Berlin: Springer, 2004, pp. 265-282. (NON-DELIS)