Project Number 001907

DELIS

Dynamically Evolving, Large-scale Information Systems

Integrated Project

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

Deliverable D4.3.3

Evolutionary and socially inspired algorithms, tools and applications in dynamic nets

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:	December 2006
Actual submission date:	December 2006
Dissemination level:	PU – public
Work Package 4.3:	Models and Methods for Dynamics, Evolution and Self-Organization
Participants:	Universita di Bologna (UniBO), Italy Research Academic Computer Technologie Institute (CTI), Patras, Greece University of Cyprus (UCY), Nicosia, Cyprus Rheinisch Westfälische Technische Hochschule Aachen (RWTH), Germany
Authors of deliverable:	David Hales (hales@cs.unibo.it) Ozalp Babaoglu (babaoglu@cs.unibo.it) Stefano Arteconi (arteconi@cs.unibo.it) Edoardo Mollona (mollona@cs.unibo.it) Giovanni Rossi (giorossi@cs.unibo.it) Andrea Marcozzi (marcozzi@cs.unibo.it) Paul Spirakis (spirakis@cti.gr) Simon Fischer (fischer@cs.rwth-aachen.de) Berthold Vöcking (voecking@cs.rwth-aachen.de)

Abstract

This report comprises the report for the D4.3.3 deliverable (in association with the relevant software[1]) for workpackage WP4.3 in Subproject SP4 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. In this workpackage we explore the fundamental problem of coordination in networks, particularly when agents or nodes within the network behave in a selfish way, that is, considering their own benefits rather than system level benefits.

Our work follows two fundamental approaches a) classical rational action and b) evolutionary and bounded rationality / adaptive action. Some of the open problems using a) in distributed computer systems are addressed by a proposed authority mechanism which can, essentially, enforce rational behavior (section 1). If such a system could be implemented then systems could be provably designed to give desirable behavior. Work in b) assumes agents or nodes will not behave rationality but rather follow a boundedly rational evolutionary approach (or some adaptive heuristic). In this case what controls selfish behavior is the dynamic formation of social structures which mediate interaction. Interestingly, using adaptive routing mechanisms we show how efficient equilibrium can be found (section 2). In section 3 we overview work where we draw on emerging biological and social theory to inform our models. We show how an evolutionary approach can find socially optimal equilibria in a group co-orindation game on a network. We also present results from a model that incorporates emerging ideas from organizational theory to task allocation.[2]

[1]Available from http://peersim.sourceforge.net

[2]Most papers produced within DELIS are available from the DELIS website as DELIS Technical Reports. Where this is the case references are appended with the DELIS Tech Report number in square brackets. This indicates the paper was produced within the DELIS project, not some other project.

Contents

1 Game Authority for Robust Distributed Selfish-Computer Systems

1.1 Introduction

Game theory can model structural aspects of distributed selfish-computer systems as elegant social games. However, existing algorithmic game designs are immature because of several assumptions that they make. In particular, the entity that enforces the rules of the game should be explicitly constructed, and (software) agent rationality should be observed. Designs that do not consider the later imply that software cannot be faulty or mischievous. We design the first suiting middleware for a game authority that enforces the rules of the game. Moreover, we can detect agents that do not select their action according to the Nash criteria and, then disqualify them as game participants. We base our design on a self-stabilizing Byzantine agreement according to which the ma jority of participants audit the play while dealing with unexpected faults. We use several other cryptographic techniques for auditing mixed strategies while providing agents with privacy.

A key benefit of our design is that mischievous agents are depicted as Byzantine. Hence, the middleware transforms classical game models to distributed selfish-computer systems that tolerate mischievous behavior.

The algorithmic game designer is now provided with guarantees for stability while permitting free choice. Game theory characterizes the notation of Nash equilibrium as a state with guaranteed self-enforcing stability that comes with a price, e.g., the price of stability [2] and the price of anarchy [20, 34]. We suggest a new cost criterion for evaluating performances the multi-round anarchy cost. We show that the cost can be asymptotically optimal when repeated Nash selections are guaranteed. One may seek strategic coordination (not just by rough consensus). Our design also includes services that allow agents to share a collaborative effort for coalition optimization. We allow agents to simulate plays in the framework of their efforts to improve performances, say, by allowing them to deviate from a path that refines particular potential function of optimality.

Lastly, we offer a service for regulating group-preplay negotiation. Since there are no guarantees for achieving a self-enforcing agreement within a bounded time (e.g., it may not exist), we consider other approaches (that do not necessarily satisfy the Nash criteria, but provide elements of free choice). Unsuccessful negotiations are resolved deterministically with a touch of democratic flavor (e.g., parliamentarian voting).

The work presented in this section is an overview summary of the work presented in [1]. Technical aspects, definitions and proofs can be found there.

1.2 The self(ish)-stabilizing system

We consider fault tolerance systems in the presence of selfish agents . Self-enforcing agreements should be established in order to prevent superiority of some agents that is achieved by their mischievous behavior. We consider runs with Byzantine processors (assuming that some standard requirements hold). Therefore, our settings can cope with systems that would have otherwise diverted from the preferable solution and/or never regain consis- tency. Moreover, our system design is integrated with the self-recovery mechanisms that can tolerate: (1) periods in which the environment introduces transient failures, and (2) periods in which the agents act upon short-lived myopic logic, say, due to transient computational resource shortage (see [21]). However, we assume that such periods do not occur during a normal run of the execution. The correctness of our system is demonstrated by considering every nice execution in which the behavior of the non-mischievous agents is according to the game model (e.g., rational strategic game with complete information), and in which the system assumption holds (e.g., regarding the Byzantine agreement). During a legal execution, the system is to facilities a play of the game according to its rules, and for an infinite number of game rounds.

1.3 Auditing the Nash criteria

The game authority guarantees that the play arguments, with which agents make their strategic choices, are well known. Therefore, pure strategic choices of non- mischievous agents can be predicted by any pseudo rational program. The case of mixed strategies is more complicated. At the beginning of a game round, we require the agents to declare publicly their profiles of probabilities support (PPS) for every pure action. At the end of every game round, the system can compare the performed actions with the declared profiles. The difficulty that arises is that it may be hard to ensure that an action (or a finite sequence of actions) is indeed random.

1.4 Punishment

In case a mischievous behavior is detected, the agents can strategically punish such a behavior in the course of the next game rounds. We require that all non-mischievous agents agree on the set of mischievous agents. Moreover, we assume that it is technically possible to exercise pun- ishment actions. Since the standard requirements for Byzantine agreement hold, such an agreement is possible. Moreover, standard requirements for Byzantine agreement allow us to assume that it is technically possible to exclude (for several game rounds) any processor that exhibits Byzantine behavior, and prevents all mischievous agents from participating in the game.

1.5 Examples of Applications

Traditionally, the applications of the computational game theory consider rough consensus among uncooperative agents. In this section, we consider applications that make use of the game authority, which ensures the legitimacy of agents actions and allows preplay negotiation. The market game. There is a marketplace with several stands selling different merchandise. The locations of the stands influence the selling potential. Obviously, each seller would like to have the prime location, in order to sell more of his/her merchandise. A certain percentage of the profit is deposited to a common account and is distributed evenly among the sellers.

In the social democratic policy every sales-person tries to organize the stall in a way that maximizes the overall sales. This is because the sales people change their positions and conduct the sales in a round robin fashion and, therefore, earn the same portion of the total market sales. In the capitalistic policy, the sales person that finds a better stall location arrangement will benefit from choosing the prime location. The ma jority will still benefit from the improvement due to the common (tax) account.

Another aspect of privacy can be demonstrated by proving the existence of necessity of sufficient resources (e.g., stock) for making a proposal that is believed to increase sales. The sales person that proposed the better way to organize the stalls might not be willing to reveal his/her exact solution unless it is chosen to be used (say, just in case it will be chosen later). The sales person may use zero-knowledge interactive proof [18] to prove to his/hers colleagues that his/her solution is indeed a better one and do so without revealing the exact manner in which he/she proposes to organize the stalls.

In terms of distributed systems, we consider computer networks in which users can offer processing time and memory for use for a certain payment. Networking configuration, such as configuration of the virtual private network (VPN) or virtual LAN, may influence the performance of each server.

1.6 Summary

We would like to draw the readers attention to the experimental sociological point of view for studying games that involve the human-factor (according to [39]). It is most common in empirical studies to have an external observer that records payoffs (with their measurements) and the appli- cable actions (with their statistics). One may view our design as a computational (and distributed) implementation of the above approach.

The game authority middleware proposed is useful for the construction of autonomous distributed systems. It provides all agents with online reports regarding their strategic play and allows them to negotiate in a regulated manner. We believe that our study of game theory and distributed computing illuminates their different interconnections.

2 Dynamic Traffic Engineering Based on Wardrop Routing Policies

The concept of Wardrop equilibria is similar to the Nash equilibria but applied to traffic routing through transportation networks. Essentially, the main idea is that if individuals can not unilaterally change routes and improve their individual performance (reducing travel cost) then a system is in equilibrium. Such equilibria are also efficient.

In [3] we have seen that a large population of agents is capable of computing approximate Wardrop equilibria efficiently using a simple load-adaptive rerouting policy (cf. DELIS work package WP3.1). From these positive theoretical results the question arises whether this policy can be realised in practice. A main obstacle in implementing such protocols that operate on the timescale of seconds is the danger of oscillation due to stale traffic information. In [3] this issue is treated carefully.

In [2] we present an implementation of an online traffic engineering scheme based on the above-mentioned policy within the SSFNet simulation framework. In our protocol, called ReplEx, routers simulate the behaviour of agents sitting at the end hosts of a network. They do so by assigning weights to routes and distributing packets with equal destination for which multiple alternative routes exist according to these weights. Route weights are adapted over time depending on the quality of the paths. Since routers can directly observe only the quality of adjacent links, they need to exchange messages once in a while.

Our traffic engineering scheme was simulated in various artificial topologies as well as a real autonomous system consisting of 172 routers. In total, 5,400 Web clients and 417 Web servers were connected to the access and border routers, and the traffic between these end hosts was simulated using a realistic web workload generator. It turned out that, when parameterised correctly, the traffic engineering protocol converges quickly (within two minutes) and computes weights that bring the system to a Wardrop equilibrium. Furthermore, a significant increase of throughput is achieved.

3 Evolving Networks for Social Optimum

3.1 Introduction

In this section we present recent work in which evolving network structures guide a system toward cooperative or socially optimal behavior. Firstly we consider the application of a simple network rewiring scheme which we have produced previously, but we apply it to a scenario which requires coordination between a groups (not just pairs) of nodes: the Weakest Link Game. Secondly we explore a set of rewiring mechanisms that are inspired by subcontracting of jobs and tasks between and within firms: FirmNet. Here we give an overview of the main results from these works, further detail is available in the associated technical reports [13, 11].

3.2 Weakest Link

In recent work we introduced a simple "copy and rewire" algorithm (SLAC) that structures networks of nodes towards socially optimal behaviors when they interact with neighbors to generate utility [4]. The algorithm demonstrates properties such as self-organisation, scalability, robustness to node turn-over and free-riding behavior without the need for central control.

These properties are highly desirable for use in self-organising applications such as peer-to-peer (P2P) overlay networks where central control is problematical and free-riding and node failure are prevalent. Hence SLAC has been proposed as a P2P protocol for a number of scenarios [5, 6].

In these previous works, the social optimum, by which we mean the maximum total sum of utility that can be generated over all nodes, required a non-equilibirum strategy. We found that SLAC produced networks close to the social optimum even though nodes behaved in a boundedly rational and myopic selfish way. The interactions between nodes were modeled as pairwise interactions between network neighbors with a focus on the elimination of free-riding and hence the one-shot, two-player Prisoner's Dilemma (PD) game was used.

In [13] we apply the same algorithm to another game called the Weakest Link (WL) which, although having a well defined social optimum strategy, has several different properties from the PD. Firstly, the WL requires simultaneous interactions between several node neighbors to produce utility, secondly, the game has several possible equilibria each offering increasing social optimality and thirdly, there is a larger space of pure strategies. This means that a higher degree of coordination is required between nodes to reach a social optimum but when the optimum is achieved it is an equilibrium (i.e. there is no incentive for an individual to free-ride). We describe the WL game in detail in section 3.2.1.

We found that SLAC produced networks that evolved in stages toward the social optimum equilibrium passing through each of the less optimal equilibria. The network stayed in each equilibria for some time before jumping to the next. This process was monotonic with the network always jumping to a higher equilibrium but never to a lower one. This process was robust even when large numbers of nodes were introduced with strategies that exploit these higher equilibria. Hence we describe SLAC as functioning rather like a "social ratchet" always selecting strategies that move to higher social optimum. We also found that if we switched off "mutation" events - a feature of the previous SLAC algorithm - where nodes spontaneously change their strategy and links we could control the transition of the network to higher equilibrium by injecting a "seed" into the system: a pair of nodes following a higher equilibrium strategy. In this sense we see how the network "learns" from examples dynamically inserted into it and rapidly spreads the new strategy over the entire network.

The SLAC algorithm was adapted from a model developed within computational sociology based on the "tag" concept introduced by Holland [7] and developed by Riolo [12]. The interpretation here is of a cultural evolutionary process within a society of agents forming groups based on observable markings or social cues. From a sociological perspective SLAC can be seen as an adaptation of the algorithm in which agents interact over, and construct, dynamic social networks based on social cues.

3.2.1 Weakest Link (WL) Game Definition

In a strategic (or non-cooperative) game there are $n \geq 2$ players each of which takes some action, and everyone's utility depends on the n-tuple of taken actions. Formally, a game consists of a triple $\Gamma = (N, \mathcal{S}, u)$ where $N = \{1, \ldots, n\}$ is a finite player set, $\mathcal{S}$ is the n-fold product of action or strategy sets (one for each player), and $u : \mathcal{S} \rightarrow \mathcal{R}^n$ represents players' preferences: $u_i(s)$ is the utility attained by $i \in N$ when the strategy profile is $s \in \mathcal{S}$. A (pure-strategy) Nash equilibrium NE is any strategy profile $s \in \mathcal{S}$ with respect to which no player has an incentive to (unilaterally) deviate. That is, a situation where each player is playing a best response to the $n-1$-tuple of others' actions.

In a weakest link (WL) game each player $i \in N$ chooses an effort level s_i from some finite set $\{0, 1, \ldots, K\}$ ($K = 1$ above), and i's payoff $u_i(s) = u_i(s_1, \ldots s_n)$ depends only on the chosen effort level s_i and the minimum $\min_{j \in N} s_j$ across all players. In particular, it is increasing in s_i if $s_i = \min_{j \in N} s_j$, and decreasing in s_i if $s_i > \min_{j \in N} s_j$. In words, *each player wants to select exactly the minimum of the other players, and everyone wants the minimum to be as high as possible. But selecting high actions is risky because other players may select low actions.*

In our setting $K = 19$ and the payoff is

	$\min_{j\in N} s_j = 0$	$\min_{j\in N} s_j = 1$	$\min_{j\in N} s_j = 2$	$\cdots$	$\min_{j\in N} s_j = 19$
$s_i = 0$	$u_i(s) = 1$	$\backslash$	$\backslash$	$\cdots$	$\backslash$
$s_i = 1$	$u_i(s) = \frac{1}{2}$	$u_i(s) = 2$	$\backslash$	$\cdots$	$\backslash$
$s_i = 2$	$u_i(s) = \frac{1}{3}$	$u_i(s) = 1$	$u_i(s) = 3$	$\cdots$	$\backslash$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\backslash$
$s_i = 19$	$u_i(s) = \frac{1}{20}$	$u_i(s) = \frac{2}{19}$	$u_i(s) = \frac{3}{18}$	$\cdots$	$u_i(s) = 20$

Table 1: Weakest link payoff table with 20 possible strategy and payoff function in equation 1

$$u_i(s) = \frac{1 + \min_{j\in N} s_j}{1 + s_i - \min_{j\in N} s_j} \tag{1}$$

where $s_i - \min_{j\in N} s_j \geq 0$ (of course), and thus the payoff matrix is the one given in table 1.

It is easily seen that this game has precisely 20 equilibria, each of which attains when $s_i = k$ for every $i \in N$ as $k = 0, 1, \ldots, 19$. Furthermore, these equilibria are *Pareto-rankable*; that is to say, $s_i = 19$ for every $i \in N$ is the unique social optimum, and for $0 \leq h < k \leq 19$, each player prefers equilibrium $s_i = k$ for every $i \in N$ rather than $s_i = h$ for every $i \in N$.

3.2.2 SLAC algorithm applied to WL game

The basic SLAC algorithm specifies how nodes should update their strategies and network neighborhood under the assumption that they are involved in some on-going game interactions with neighbors. Each node generates a *utility* measure (U) according to some interaction with its neighbors. The higher the value of U the better the node is performing.

The algorithm is executed by each node and consists in it periodically comparing its own utility (say U_i) with another node (say U_j) randomly chosen from the network. If $U_i \leq U_j$ then node i drops all of his current links and copies all j's links (adding a link to j itself) and j's strategy - see figure 1.

```
i ← this node
do periodically:
   j ← GetRandomNode()
   if Ui ≤ Uj
     i.links ← j.links ∪ j
     i.Strategy ← j.Strategy
     with low probability mutate(i)
```

Figure 1: SLAC algorithm's pseudocode.

In SLAC all the rewiring operation are symmetric — if node i makes a link to j then node j makes a link to i and on the other hand if i drops a link to j the link from j to i has to be dropped as well. Each node can maintain a maximum amount of links (called *viewsize*), if a new node has to be added in an already full view, a randomly selected neighbor is dropped to make space for the new link. The SLAC algorithm provides a rewiring mechanism analogous to tournament selection within evolutionary computing. Nodes act in a highly boundedly rational way hoping to selfishly improve their utility by copying better performing ones. Occasionally with low probability a node applies a "mutation" function after copying another node. This involves changing the strategy randomly and changing the links randomly.

```
i ← this node
for each j in i.Links
Strats ← Strats ∪ j.Strategy
i.Utility ← payoff for i.Strategy given Strats
```

Figure 2: Weakest Link single game round pseudocode.

The utility measure is provided by the game that is being played between node neighbors. In this case the WL game. Periodically nodes play the WL game.

To play the WL game a node executes the function shown in figure 2. In our implementation WL rounds are initiated locally by single nodes and only the payoff of the node which initiated the round is affected by each single game, according to equation 1.

SLAC requires a utility value to be defined for each node. This is the average utility obtained from the game played since last reproduction. Mutating the strategy consists in choosing with uniform probability a random strategy between the possible ones. Mutating the links involves removing all existing links and replacing with a single link to randomly chosen node from the population. We mutated the strategy with probability m_s and the links with probability m_l. Our implementation of a weakest link round is described in figure 2

3.2.3 Some Results

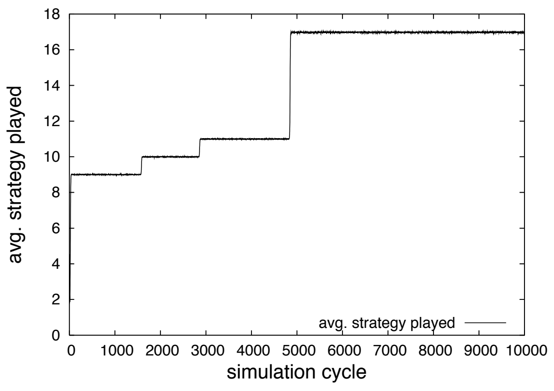
Our aim in these experiments was to discover how well SLAC is able to lead the network to a high social optimum hence we are interested in the equilibrium reached by the system, the time needed to reach it and the robustness of high social optimum equilibrium states to the invasion of nodes following lower exploitative strategies.

We found that via randomized change (mutation) the network, over time, climes a staircase of equilibria increasing in social utility - see figure 3 which shows a typical run. We also found this to be scalable and robust. Figure 4 shows how results scale. In fact we see a certain amount of reverse scaling which means we get better results from larger networks. Our intuition was that a “ratchet” effect was in operation where only a small number of strategies of higher social optimum would spread whereas lower strategies would be resisted creating robustness and an ever increasingly equilibrium over time leading to optimum. We tested our intuition by turning off mutation effects and manually inserting “seeds” into the system. These results can be seen in figure 5

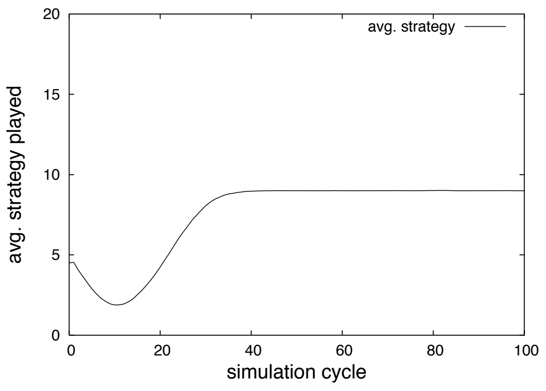
3.3 FirmNet

In our work, we bridge organizational theory with computer science by adopting the SLAC (Selfish Link-based Adaptation for Cooperation) protocol (Hales and Arteconi, 2006), developed to produce cooperation in P2P networks. We modify the SLAC protocol and produce three algorithms to mimic plausible individual decision-making routines of professionals in organizations.

FirmNet includes a number of nodes, which represent project managers. Project managers have relationships with clients, receive tasks to be completed, represent profit centres and can create teams to integrate different skills to perform a task. We assume an internal market with a job posting mechanism in which employees can search within an organization in which team they prefer to work. Incrementally modifying the picture, we describe three types of employees’ individual behaviour. In a first step, we represent agents as simply replicating links and decision-making of successful colleagues. In a second step, we assign to agents the ability to bargain their reward. Finally, we assume that agents also consider success of project managers to decide how to select their team. We use an agent-based model to simulate emerging network structures as a consequence of different employees’

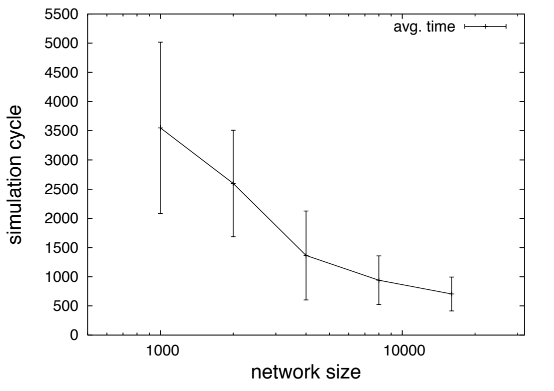


(a) Typical run

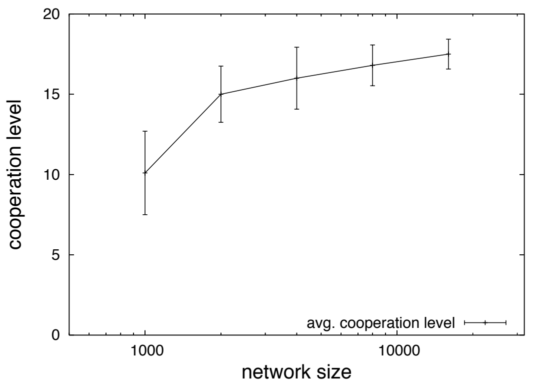


(b) Initial stage

Figure 3: A typical single run showing how the average strategy value changes in the network by cycle. The network has a fixed size of 4000 nodes. (a) shows the entire run to 10000 cycles, (b) shows the same run but over the first few cycles showing detail not visible in (a). Each strategy value is a possible equilibrium. The higher the value the higher the social utility of the system. Notice the staircase effect as the system climbs toward higher equilibrium over time. Chance mutation pushes the system to higher equilibrium because SLAC acts like a "social ratchet" accepting and spreading higher equilibria but resisting invasion by lower equilibrium strategies.



(a) Time for cooperation increase



(b) Cooperation level at 5000 cycles

Figure 4: Average time needed to increase cooperation level in different network sizes and cooperation level reached at 5000 cycles. 90% confidence intervals are shown. As can be seen larger networks reach higher cooperation levels more quickly. Ten runs were performed for each network size. The x-axis is on a log scale.

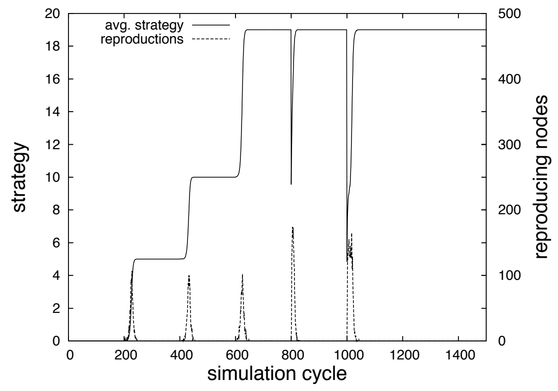


Figure 5: Experiment with no mutation and node injection (4000 node network). Initially all nodes are set to strategy 0 (minimum). Two nodes playing strategy 5 are injected at cycle 200. Two nodes playing strategy 10 are injected at cycle 400. Two nodes playing strategy 19 (maximum) are injected at cycle 600. To test robustness, 1000 nodes playing strategy 0 (minimum) at injected at cycle 800. 2000 nodes playing strategy 0 are injected at 3000. The lower dashed line shows the number of node movements (copying between nodes) made in each cycle. Here the "social ratchet" effect is clear: A single pair of nodes at higher strategy spread rapidly over the network but even a large number of lower strategy nodes are resisted.

individual decision-making and we assess the impact on firm performances and to wealth distribution within the organization.

3.3.1 Model Specifics

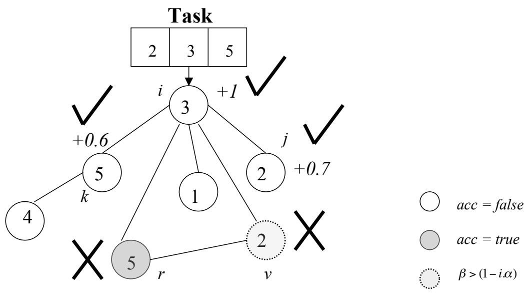
The FirmNet model contains two kind of agents: the *node-skill agents* (*NS*) and the *node-task agents* (*NT*). These agents are nodes in a Peer-to-Peer Network; they all hold a certain skill (S) that they use to perform some task. Agents NT, in addition, play the role of project managers, have direct contacts with clients and receive a certain Task to be completed. To complete a task, it is necessary to complete three jobs, each requiring a different skill, and NT agents need to form a team attracting NS agents, which hold the required skill. Thus, the difference between NT and NS agents is that NT agents arbitrage, on behalf of the firm, the relationship between skills and clients. Hence, the model simulates an organizational network in which teams arise having certain skills. The designed organization network is a Peer-to-Peer Network in which each node has a maximum number of links (network degree). Each link is bidirectional; a connection of a node a to another node b implies a connection of node b to node a. Links are undirected so the entire network can be considered as an undirected graph where each vertex is a node and each edge is a link.

The state variables of each node are: a *task flag* which indicates if the node has a task or not; a *skill type* (S) which is randomly initialized within a set of 5 elements $S \in \{1, 2, 3, 4, 5\}$ indicating the ability held by the single node; a *utility* (U), accumulated by each node after a certain task is completed; a list of neighbors (*local view*); a *commission* (α) which is the percentage that a task node is willing to pay to its employees; and an *accept threshold* (β) indicating the minimum amount of payoff a certain NS node wants to work on certain jobs.

The Skill is the only parameter which does not evolve: it is not copied during the reproduction phase (see later), but it may "mutate" (that is change) with a very small probability. The simulation

```
At each cycle if $N_i$ == NT :
    if ($N_i$.acc == false) && ($N_i$.task.contains($N_i$.skill))
    then
        $N_i$.acc = true; $N_i$.task.remove($N_i$.skill);
    if ($N_i$.task != empty) then for all $N_i$ neighbors j:
    if ($N_i$.task.contains($N_j$.skill) && ($N_j$.acc == false)
        && ($N_j$.benefit > $N_j$.beta)) then
        $N_j$.acc = true;
        $N_j$.task.remove($N_j$.skill);
end
```

(a) FirmNet algorithm



(b) Schematic diagram

Figure 6: Differences in average wealth for different degrees for different algorithms (a) and percentage of completed tasks (b). Here tasks are assigned every 20 cycles

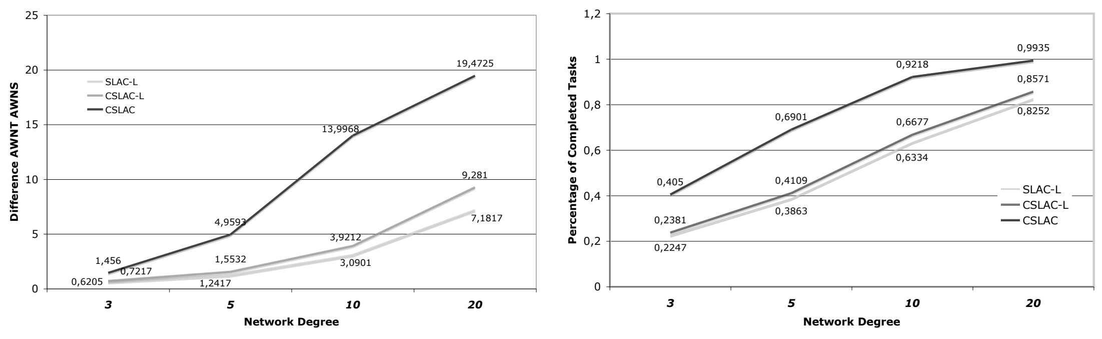
time is divided into cycles. At cycle 0, with probability 0.25, nodes receive a task to be completed. The tasks are produced selecting at random three values from a set of five elements ($J \in \{1, 2, 3, 4, 5\}$); the receiving nodes will then act as a *project manager* and will start looking for employees among their immediate neighbors (figure 6(a) shows the pseudo-code for the FirmNet model). Suppose node i has to complete a task composed of jobs $j_1 = 2$, $j_2 = 3$ and $j_3 = 5$, what will happen is shown in figure 6(b):

- node i will first check if itself is able and free (acceptance flag set to false) to work on one of such jobs; if so, it will set its acceptance flag to true and the job will be removed from the task list;
- node i will look among its immediate neighbors for nodes willing to work on the remaining jobs: if among them it finds some available node (say node j) with the right skill and a "β" value smaller then the margin i is willing to pay, j acceptance flag will be set to *true* and the job will be removed from the task list;
- if the task is now completed, payoff will be distributed to all the involved nodes; otherwise next cycle node i will execute the same algorithm starting from the previous point.

This means that NT agents have the duty of allocating the tasks to the NS. Payoffs are distributed only after the entire task is completed. In our model each job gives the same reward (1) and NT will ask the recruited neighbors for a commission α on such payoff. Suppose node i has $\alpha = 0.3$; this means that if node j will execute a certain job for NT, node i will get at least a 0.3 payoff ($i.U$ += 0.3) and node $j.U$ += 0.7. This happens in the case the node j accept threshold is $j.\beta = 0.7$. If for example $j.\beta = 0.6$ node i would get a 0.4 payoff ($i.U$ += 0.4) and $j.U$ += 0.6. If NT agents have two or more neighbors able to complete a certain task, they will select the one with the smallest β.

3.3.2 CSLAC, SLAC-L and CSLAC-L Variants

The CSLAC (Competitive-SLAC) algorithm is very similar to the original SLAC (see figure 1). In this version the rewiring is carried out identically to SLAC. The difference is in the copying of the strategies. The strategy of a node is considered to be the α and β values of the nodes. While in SLAC the losing node copies the winner nodes strategy without taking into account its values, here the losing node before coping β checks if some of the winners neighbors has its skill. If this is the case and the node also has a smaller β than its own, it will copy this value minus a 0.1 constant. The



(a) Wealth difference (b) Completed jobs

Figure 7: Differences in average wealth for different degrees for different algorithms (a) and percentage of completed tasks (b). Here tasks are assigned every 20 cycles

rationale behind this, is to create competition between NS agents. If a certain NS is going to join a new community, it will go in competition with the other nodes having its same skill by lowering it own accept threshold (β).

We compared the performance of CSLAC with other two variants of the SLAC algorithm: we called these SLAC-L and CSLAC-L. The "L" suffix stands for *limited*, meaning that in these versions, the SLAC and CSLAC algorithms are performed only by and among NS agents. We introduced these algorithms to to explore a situation in which professionals (NS nodes) and project managers (NT nodes) are embedded in separate communities and therefore consider only own peers wealth to form expectations concerning future payoffs.

3.3.3 Some Results

Simulation experiments highlight three key findings. First, the CSLAC algorithm is the most powerful algorithm, among the three used. Both under stable and dynamic environmental conditions, the CSLAC algorithm shows superior performances in terms of percentage of tasks completed and accumulated firm's wealth. Second, the algorithms have an impact of wealth redistribution within the organization. As the focal firm's performances increase, the average wealth of both NT and NS nodes increases but the discrepancy between NT and NS nodes' wealth becomes larger. Third, the performances of the algorithms depend on structural features of the network because both for low and high degree of the network (number of links that each node may have), the differences in performance of the algorithms tend to decrease. Some results can be seen in figure 7

3.4 Summary

In this section we have seen how variants of the SLAC algorithm can be applied to different domains requiring certain levels of group coordination to produce socially beneficial results. In the Weakest Link game it was shown how a kind of "social ratchet" mechanism operates to push the system towards social optimum. We also investigated the notion of "seeding" a network but injecting some small number of nodes which follow a more socially optimal strategy. In some sense this can be viewed as a kind of high-level run-time programming activity, where strategies might represent code replicating over the network to restructure interactions dynamically. In the FirmNet model, intuitions from organizational theory have been applied to produce variants of SLAC that involve some level of subcontracting between nodes. The results may have applicability to understanding human systems and engineering computer systems. In both cases the underlying mechanism results from the selection of productive groups or teams based on purely individual choices and selection methods. An individual node attempts to increase it's own payoff by copying others that outperform them. This emerges social structures that mediate interaction towards pro-social behavior.

4 Conclusion

The work we have outlined in this workpackage aims towards the production of predictable and efficient distributed networks in uncertain and dynamic environments. Two main approaches have been presented, one focusing on the enforcement of rational action and the other on the evolution of strategies (or routes) via an adaptive process. These two approaches are not exclusive and could possibly be combined in certain circumstances. For example, where the evolutionary / adaptive approach actually converges to rational behavior, an authority approach could add stability against malicious attacks from nodes behaving irrationally and hence not in the correct adaptive way. Where the two approaches differ is when irrational behavior is actually selected by evolution especially when rational behavior is suboptimal at the system level (e.g, in social dilemmas or when pure altruism is required).

References

[1] Dolev, S., Schiller, E. M. and Spirakis, P. G. (2006) Game Authority: for Robust Distributed Selfish-Computer Systems. [DELIS-TR-0395]

[2] Fischer, S., Kammenhuber, N., and Feldmann, A. (2006) REPLEX — Dynamic traffic engineering based on Wardrop routing policies. In *Proc. 2nd Conference on Future Networking Technologies (CoNext)*, pages. 6–17, Lisboa, Portugal, December 2006. pp. 6–17. [DELIS-TR-0427]

[3] Fischer, S., Räcke, H. and Vöcking, B. (2006) Fast convergence to Wardrop equilibria by adaptive sampling methods. In *Proc. 38th Ann. ACM. Symp. on Theory of Comput. (STOC)*, pages 653–662, Seattle, WA, USA, May 2006. ACM. [DELIS-TR-0424]

[4] Hales, D. and Arteconi, S. (2006) SLACER: A Self-Organizing Protocol for Coordination in P2P Networks. *IEEE Intelligent Systems* 21(2):29-35. [DELIS-TR-0368]

[5] Hales, D. and Babaoglu, O. (2006) Towards Automatic Social Bootstrapping of Peer-to-Peer Protocols. *ACM SIGOPS Operating Systems Review (Special Issue on Self-Organizing Systems)* 40(3). [DELIS-TR-0371]

[6] Hales, D. and Edmonds, B. (2005) Applying a socially-inspired technique (tags) to improve cooperation in P2P Networks. *IEEE Transactions in Systems, Man and Cybernetics - Part A: Systems and Humans, 35(3):385-395.* [DELIS-TR-0111]

[7] Holland, J. (1993) The Effect of Lables (Tags) on Social Interactions. *SFI Working Paper 93-10-064.* Santa Fe Institute.

[8] Jelasity, M. and M. van Steen (2002), "Large-Scale Newscast Computing on the Internet". *Technical Report IR-503, Vrije Universiteit Amsterdam.*

[9] Marcozzi, A.; Hales, D. (2006) Emergent Social Rationality in a Peer-to-Peer System. *Technical Report UBLCS-2006-23, University of Bologna, Dept. of Computer Science.* [DELIS-TR-0372]

[10] Mollona, E. and Hales, D. (2006) Knowledge-Based Jobs and the Boundaries of Firms. *Journal of Computational Economics 27(1):35-62.* [DELIS-TR-0230]

[11] Mollona, E.; Marcozzi, A. (2006) FirmNet: The Scope of Firms and the Allocation of Task in a Knowledge-Based Economy. *Technical Report UBLCS-2006-26, University of Bologna, Dept. of Computer Science.* [DELIS-TR-0375]

[12] Riolo, R.; Cohen, M.; Axelrod, R. (2001) Evolution of cooperation without reciprocity. *Nature 414, pp. 441-443.*

[13] Rossi, G., Arteconi, S., Hales, D. (2006) Evolving Networks for Social Optima in the "Weakest Link Game". *Technical Report UBLCS-2006-21, University of Bologna, Dept. of Computer Science.* [DELIS-TR-0369]