Contents

List of Contributors

David Hales
University of Bologna
Dept. of Computer Science

Mura Anteo Zamboni 7,
40127 Bologna, Italy.
www.davidhales.com

1

Applying Evolutionary Approaches for Cooperation

A general method and a specific example

David Hales

Bologna University hales@cs.unibo.it

Summary. In this chapter we describe a simple general method by which existing evolutionary algorithms originating in the biological or social sciences can be translated into always-on protocols that adapt at run time. We then discuss how this approach has been applied to import a novel cooperation producing algorithm into a simulated peer-to-peer network. Finally we discuss possible applications and open issues.

1.1 Introduction

Increasingly, within biological and social sciences, models of behavior are expressed in the form of evolutionary algorithms. That is, individual entities such as cells, animals or human agents are represented as interacting, mutable and reproducing entities which are modeled computationally.

Computer simulation is used to specify and analyze such models because their behavior is complex and often produces emergent results that are not easily tractable analytically. Such models are typically co-evolutionary in nature in which the performance of individual entities is a result of some kind of interaction with other evolving entities in the population. This can be contrasted with evolutionary optimization algorithms, such as traditional Genetic Algorithms [5,13] which aim to optimize an *a priori* objective fitness function.

Such algorithms generally specify some rule by which entities interact, gaining some reward (often termed utility) and, then differentially reproduce based on utility. This differential reproduction process (the evolutionary bit) often requires that some entities "die" - they are removed from the population - and other entities produce "offspring" - they produce copies of themselves.

In the context of biological models the interpretation is clear: survival of the fittest and death to the weakest. In the context of sociological models the interpretation is less clear. The assumption here is that some imitation process is occurring that favors entities with high utility. Entities are seen as *behaviors* or *ideas* that can replicate horizontally, between peers within a

generation, in a population. Such culturally replicating entities are sometimes termed "memes" [4].

This latter cultural interpretation gives us a clue as to how evolutionary models can be accommodated within information systems composed of distributed processing entities sharing some communications network. Rather than requiring the entities themselves to die and reproduce - which is obviously not currently viable - we can implement the differential imitation of behaviors between entities. To put it more directly, computational entities can transmit executable codes to each other.

The target infrastructures we have in mind for the application of evolutionary algorithms are unstructured peer-to-peer (P2P) overlay networks. In a P2P overlay network there is a population of nodes, typically processes situated within a physical network, which maintain symbolic links to other nodes (often called their neighbors). P2P applications, like Skype[1] or BitTorrent[2] implement these to provide services. A valuable property of the overlay net abstraction is that rewiring nodes or changing the topology of the network is a logical process in which nodes simply drop, copy or exchange symbolic links. It is therefore feasible to maintain highly dynamic network topologies at the overlay layer. We make use of this property when we translate a novel tag-based algorithm into a P2P protocol (see section 1.5).

This chapter is structured as follows, firstly we provide a general method for translating evolutionary algorithms into P2P protocols (in section 1.2), then we discuss the basic problem of cooperation and formulate it as a Prisoner's Dilemma (sections 1.3 and 1.4). We then apply the general technique to the specific case of a novel tag-based cooperation algorithm (sections 1.5 and 1.6). Finally we conclude with a brief discussion of open issues (section 1.8).

1.2 From evolution to protocols

Translating evolutionary algorithms into parallel distributed P2P protocols is a relatively simple process. Generally this involves a parallel and asynchronous copying of application behaviors between pairs of processing entities based on a utility measure. In order to illustrate this translation process we give a set of pseudo-code template algorithms, starting with the kind of evolutionary algorithms given in biological and social simulation work and ending with an outline of a set of threads that could be the basis of a protocol design for a distributed P2P system.

Figure 1.1 shows an outline of a typical co-evolutionary algorithm. We do not show here the particular way that entities interact to gain fitness (utility) or the specific reproduction method used. The reproduction phase may be

[1] http://www.skype.com
[2] http://www.bittorrent.com

```
Initialize some population P of N entities
loop for some number of generations
   entities in P interact in some way and obtain a fitness (utility)
   reproduce a new population P2 by replicating entities from
      P in proportional to fitness
   apply mutation to each entity in P2 with some low probability
   P = P2
end loop
```

Fig. 1.1. A generalized synchronous evolutionary algorithm. Similar to the kind of algorithms used in biological and social simulation work.

implemented in many ways. A common approach in biological models is to use so-called "Roulette Wheel" selection [5]. This is a probabilistic approach requiring access to all the finesses of the the entire population. A simpler approach, from the point of distributed implementation, is to use a Tournament Selection approach.

```
Initialize some population P of N entities
loop for some number of cycles
   select some entities in P to interact in some way and obtain utility
   loop for some number of reproductions
     select a random pair (i,j) of entities from P
     if Utility(i) > utility(j) then
       copy behavior of i to j
       apply mutation to j with low probability
       utility(j) = 0
     end if
   end loop
end loop
```

Fig. 1.2. A generalized asynchronous evolutionary algorithm using Tournament Selection method during reproduction.

Figure 1.2 shows the same general outline algorithm but with the reproduction phase expanded with a simple tournament selection approach. Some number of reproductions are performed in which random pairs of entities are selected, utilities are compared and the entity with the lower utility copies the behavior of the node with the higher utility - meaning the behavior of the higher utility node is effectively *reproduced*. After reproduction and with low probability some "mutation" is applied to the behavior, meaning that some randomized change is made in behavior.

The benefit of working with evolutionary algorithms is that, although algorithms are often presented as sequential and synchronous which aids simulation and analysis on a single machine, by their nature they should be easily

```
Active application thread for node i:
do forever:
  Engage in application level interaction with other nodes using Si
  Update utility value Ui

Active reproduction thread for node i:
do forever:
  wait(delta)
  j = selectRandomNode()
  receive(Uj, Sj) from node j
  if Uj > Ui then
    Si = Sj
    with low probability Mutate(Si)
    Ui = 0
  end if

Passive reproduction thread for node i:
do forever:
  send (Ui, Si) to requesting node j
```

Fig. 1.3. A generalized Tournament Selection approach represented as three concurrent threads assumed to be running in each node over a population of nodes forming a peer-to-peer system. Here Si representing an application behavior (or strategy) of node i and Ui represents the utility.

translatable into distributed implementations because evolutionary processes are fundamentally distributed. Sequential evolutionary algorithms are a simulation abstraction of a parallel process. Figure 1.3 gives a simple example of a set of threads that would need to be executed by each node in a peer-to-peer system such that it would implement the same tournament based selection.

1.3 Cooperation

It is well known that the maintenance of cooperation between entities within distributed open systems is a major issue for a successful protocol design. Consider, for example, a file-sharing system in which nodes in a network may download files without uploading. If all nodes behaved in this selfish way then no files would be shared at all and the network would have no value to any node. Another example might be the broadcasting of a message through the entire network where each node relays the message to its immediate neighbors. If a substantial number of nodes choose not to pass on the message then the broadcast would not be received by all nodes. A further example could involve the sharing of load and cooperative replication of content between servers responding to client queries. All these are particular manifestations

of so-called *Commons Tragedies* [12]. These are widely found in biology and human societies and are hence well studied in biological and social sciences.

Given the problem of designing cooperative protocols several general and inter-related approaches have been proposed including: incentives, mechanism design, micro-payments and evolutionary approaches. Each have their strengths and weaknesses, but here we will focus on a novel tag-based evolutionary approach which maintains cooperative behavior through a form of group-level selection [6, 16].

To test the novel approach we use a canonical game (the Prisoner's Dilemma or PD) which captures the dilemma of cooperation, reflected in the practical examples we have given. Within both biological and social sciences the PD game (or variants) are often used to test proposed cooperation producing mechanisms. Firstly we introduce the PD game and then the original tag-based evolutionary algorithm. We then translate this into a P2P protocol, following essentially the process we have discussed above. Finally we give some simulation results and discuss some of our on-going work applying the protocol to application domains within P2P.

1.4 The Prisoner's Dilemma and Variants

The two player single-round PrisonerÕs Dilemma (PD) game captures a situation in which there is a contradiction between collective and individual self-interest [3, 18]. 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 defection respectively. However, when individuals select different moves, different payoffs of temptation (T) and sucker (S) are awarded to the defector and the cooperator respectively (see figure 1.4a). 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, because it's the highest payoff, only one can attain it in a single game. No player wants S because it's the lowest payoff. No matter what the other player does, by selecting a D-move a player always gets a higher score than it would have obtained if it had selected C. D is therefore the dominant strategy – hence an ideally rational player would always choose D.

Therefore, the dilemma is that if both players select a cooperative (C) move they are jointly better off (getting R each) than if they both select D, but selfish players will select mutual defection, getting only P each, because of the individual incentive to select defection. We select this game as a minimal test that captures a range of possible application tasks in which nodes need to establish cooperation and trust with their neighbors but without central authority or external mechanisms that enforce it.

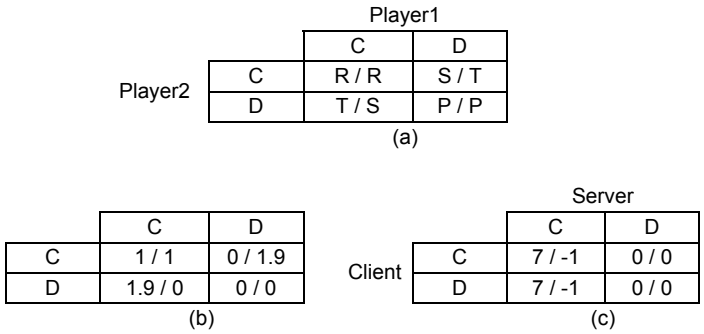


Fig. 1.4. Payoff matrix (a) shows the PD payoff structure, (b) shows the values used, (c) shows alternative values from an asymmetric generalized PD (GPD) where a single player determines payoff values. It is claimed this captures more realistically interactions between clients and servers [15].

Many variants of the PD are possible. For example, the Generalized PD (GPD) as used by Feldman et al [15] to model client / server interactions where payoffs are asymmetric (shown in figure 1.4c). Additionally, players may use probabilistic strategies where the move selected is determined by a real value indicating a probability to cooperate. Also, of course, payoffs can be varied, specifically the size of the temptation (the T payoff) inside the interval [2R..R][3].

1.5 Tag-based cooperation algorithm

Figure 1.5 shows outline pseudocode for a cooperation producing tag-based algorithm [6]. Agents play the PD in pairs. 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 uniformly 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.

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.

[3] When $T > 2R$ or $T < R$ then there is no longer a dilemma.

```
Loop some number of generations
  Loop for each agent (a) in the population
    Select a game partner agent (b) with same tag (if possible)
    Agents a and b invoke their strategies and get payoffs
  End loop
  Reproduce agents in proportion to their average payoff
  Apply mutation to tag and strategy of each reproduced agent with low
     probability
End loop
```

Fig. 1.5. The tag-based cooperation algorithm as given in [6].

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). There is no topological structure since agents are not situated in a space – such as a lattice or a ring – interaction is only structured using tag similarity and random selection.

The tag algorithm leads to very high levels of cooperation - even when the population of agents is initially set to all defect strategies. 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 payoff a C acting agent can get 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 acting agent is lost – the group becomes unsustainable 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).

A similar algorithm of tag-based cooperation was presented in [16]. Initial work on the tag ideas was proposed by John Holland (the "father" of genetic algorithms) [14].

1.6 The SLAC protocol

We translated the tag-based algorithm in the manor discussed perviously. This involved introducing a tournament selection process and application level behavior (in this case playing the PD with randomly selected neighbors) as a node level protocol composed of three threads. However, the strategy here, is not just the PD strategy of cooperate or defect, but also the tag. The tag needs to be a copyable feature that specifies the possible interaction partners of the node. Essentially it needs to specify some kind of group membership. The trick we used was to translate the tag into a neighbor list. That is, each node stores a list of it's immediate neighbors and this neighbor list (or view) determines which other nodes from the population can be selected for interaction - it performs the function of a tag. The combination of the PD strategy and the View (neighbor list) comprises the entire composite strategy. We called the new protocol SLAC (Selfish Link-based Adaption for Cooperation) because we no longer use tags as such but rather adapt links between nodes. Interestingly this approach can be closely compared to previously proposed link-based incentive schemes [19].

Figure 1.6 shows the outline pseudocode for the SLAC protocol. Figure 1.7 shows the typical evolution of a SLAC network. Notice the quick formation of components and then the rapid spread of cooperation over the nodes. Figure 1.8 gives some results from computer simulations showing the time to attain high-levels of cooperation when starting from a random network with all nodes following defect strategies.

SLAC networks are highly robust because the evolutionary process will always push the network towards a cooperative state even if many nodes fail or leave the system or new nodes enter the system. This is one of the major benefits of using evolutionary approaches - they are inherently robust to noise. In fact, they rely on noise, via mutation, to function.

We have only given an overview of SLAC here, more detail can be found in previous publications [7, 9, 10].

1.7 Possible Applications

We have adapted the SLAC protocol for application in a number of simulated task domains[4]. In [9] we added probabilistic link copying producing fully connected cooperative networks following a small world topology. We then adapted and applied this for a broadcasting task, where randomly selected nodes need to spread a message to the entire network [2]. In [8] we applied SLAC to a task sharing scenario in which nodes receive jobs that require skills - this requires specialization within the group. We recently applied

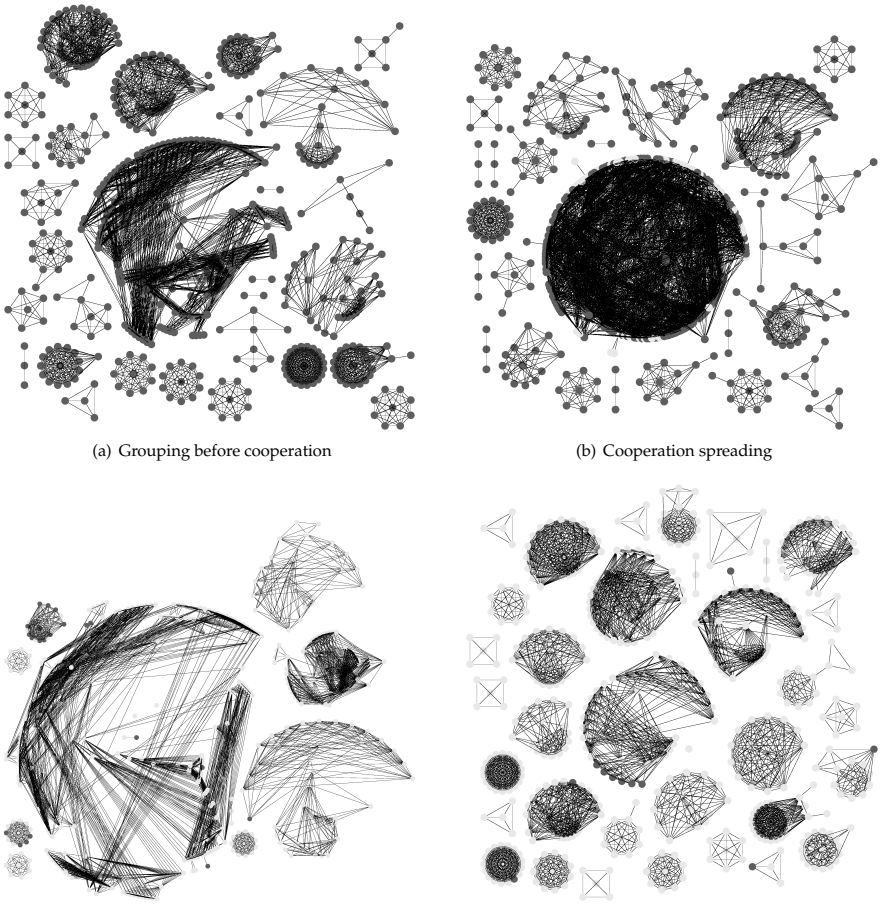
[4] The code for SLAC and related simulations can be found on the Peersim webpage: http://peersim.sourceforge.net

```
Active application thread for node i:
do forever:
  Select a random node from current neighbor list
  Play PD with neighbor and get payoff
  Update utility value Ui as payoff rolling average

Active reproduction thread for node i:
do forever:
  wait(delta)
  j = selectRandomNode()
  receive(Uj, Sj) from node j
  if Uj > Ui then
    Si = Sj
    with low probability Mutate(Si):
       mutation of PDstrategy = flip strategy
       mutation of View = drop all links and link to random node
    Ui = 0
  end if

Passive reproduction thread for node i:
do forever:
  send (Ui, Si) to requesting node j

where:

Si = {PDstrategy for node i, View for node i}
PDstrategy = {C | D}
View = list of immediate neighbor links (up to some max. = 20)
```

Fig. 1.6. The SLAC protocol. Each node in a P2P network runs the above threads. Here Si stores both PD strategy and the View (links to neighbors in the network) of node i and Ui represents the utility. We expand the mutation step to include the way mutation is applied to both PD strategy and View. The node View represents the function of the tag from the previous tag-based algorithm.

a modified form of SLAC to a distributed replica management domain [11]. However, these simulated application domains are still represented at a highly abstract level and further work is needed before implementations can be produced.

Although we have, what appears to be, a general mechanism for sustaining cooperation between nodes in an adaptive network there are several open issues that need to be addressed to produce a deployable protocol. The SLAC protocol creates an incentive for cooperative behavior because non-cooperative nodes become quickly surrounded by others of the same type. This relies on nodes communicating and copying both links and behaviors honestly. This mechanism can be subverted by nodes lying about strategies and links and other malicious behaviors. Interestingly, we performed some experiments with



(a) Grouping before cooperation

(b) Cooperation spreading

(c) Giant cooperative component breaks apart

(d) Cooperative groups are formed

Fig. 1.7. Evolution of a SLAC network with nodes playing the Prisoner Õs Dilemma. From an initially random topology composed of all nodes playing the defect strategy (dark shaded nodes), compo- nents quickly evolve, still containing all defect nodes (a). Then a large cooperative component emerges in which all nodes cooperate (b). Subsequently the large component begins to break apart as defect nodes invade the large cooperative component and make it less desirable for cooperative nodes (c). Finally an ecology of cooperative components dynamically persists as new components form and old components die (d). Note: the cooperative status of a node is indicated by a light shade.

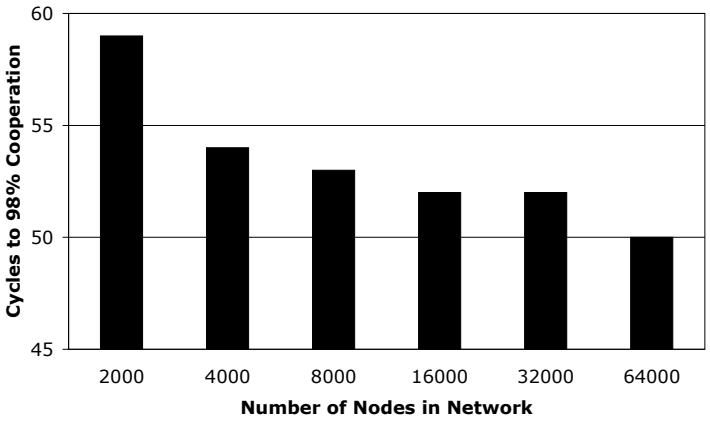


Fig. 1.8. Results of simulation experiments with the SLAC protocol playing the PD game. Each bar shows the time taken to attain high levels of cooperation. All nodes are initialized to play the defect strategy within a random topology. The results are averages over 10 independent runs, variances are low and not shown. Notice the slight reverse-scaling property which improves performance as network size is increased. Figure 1.7 shows snapshots from a typical single run.

certain classes of malicious behavior and found that in some cases this can improve performance [1]. However, we also found that a small number of highly malicious nodes can degrade performance of the entire network significantly.

1.8 Discussion and Conclusion

We aimed in this chapter to practically illustrate how to take an evolutionary algorithm and translate it into a P2P network protocol in general. We also presented a specific example and gave some results. We have briefly discussed subsequent work in which we have adapted and applied the developed protocol to simulated task domains.

There are a number of open issues that need to be addressed before practical implementations of these protocols can be deployed. These mainly related to malicious behavior. Currently the protocols require some degree of honesty in the nodes and this can not be assumed in open systems. In some recent work we have proposed not to use utility and link reporting between nodes but rather to rely on a binary node satisfaction function and randomized linking. In this case each node needs to set a desired utility level which it attempts to obtain by changing links randomly [11].

Finally, in the context of wireless systems, the current proposed protocols rely on the ability to randomly sample from the entire population of nodes. This would appear highly implausible or costly in the real world. However,

recently evolutionary models from biology have shown that this may not be a requirement of such link based approaches. It has been shown, for example, that it is only necessary for nodes in a network to have access to their neighbors neighbors (two hops) in order to support cooperative evolution [17].

1.9 Acknowledgments

Thanks go to Stefano Arteconi, from the Dept. of Computer Science, University of Bologna, for the picture shown in figure 1.7 and also for the Peersim implementations of SLAC.

References

1. S. Arteconi and D. Hales. Greedy cheating liars and the fools who believe them. Technical Report UBLCS-2005-21, University of Bologna, Dept. of Computer Science, December 2005.
2. S. Arteconi and D. Hales. Broadcasting at the critcial threshold. Technical Report UBLCS-2006-22, University of Bologna, Dept. of Computer Science, October 2006.
3. R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.
4. R. Dawkins. *The Selfish Gene: Second Edition*. Oxford University Press, 1989.
5. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
6. D. Hales. Cooperation without Space or Memory: Tags, Groups and the Prisoner's Dilemma. In *Multi-Agent-Based Simulation, Lecture Notes in Artificial Intelligence 1979, Springer*, 2004.
7. D. Hales. From Selfish Nodes to Cooperative Networks: Emergent Link-based Incentives in Peer-to-Peer Networks. In *proceedings of The Fourth IEEE International Conference on Peer-to-Peer Computing (p2p2004), 25-27 August 2004, Zurich, Switzerland. IEEE Computer Society Press*, August 2004.
8. D. Hales. Emergent group-level selection in a peer-to-peer network. *Complexus*, 2006(3):108–118, 2006.
9. D. Hales and S. Arteconi. Slacer: A self-organizing protocol for coordination in p2p networks. *IEEE Intelligent Systems*, 21(2):39–35, 2006.
10. D. Hales and B. Edmonds. Applying a socially-inspired technique (tags) to improve cooperation in p2p networks. *Transactions in Systems, Man and Cybernetics - Part A: Systems and Humans*, 35(3):385–395, 2005.
11. D. Hales, A. Marcozzi, and G. Cortese. Towards cooperative, self-organised replica management. Technical Report UBLCS-2007-02, University of Bologna, Dept. of Computer Science, January 2007.
12. G. Hardin. The tragedy of the commons. *Science*, 162:1243–1248, 1968.
13. J. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
14. J. Holland. The effect of labels (tags) on social interactions. Working Paper 93-10-064, Santa Fe Institute, 1993.
15. J. Chuang M. Feldman, K. Lai and I. Stoica. Robust Incentive Techniques for Peer-to-Peer Networks. In *ACM Conference on Electronic Commerce, ACM Press*, 2004.
16. R. L. Riolo, M. D. Cohen, and R. Axelrod. Evolution of cooperation without reciprocity. *Nature*, 414:441–443, 2001.
17. F. C. Santos, J. M. Pacheco, and T. Lenaerts. Evolution of cooperation without reciprocity. *PLoS Comput. Biol.*, 2(10), 2006.
18. J. M. Smith. *Evolution and the Theory of Games*. Cambridge University Press, 1982.
19. 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.

Index