Why it is Better to be SLAC than Smart

Emma Norling and Bruce Edmonds
Centre for Policy Modelling
Manchester Metropolitan University
norling@acm.org, bruce@edmonds.name

Abstract
Tags have been shown to be a simple yet effective way of organising what are effectively selfish groups. In particular, Hales has shown that when individuals use the SLAC algorithm (and variant SLACER algorithm) for network structure, the system can maintain a high level of overall performance. Furthermore, the system is self-regulating, in that although groups of "parasites" may emerge, and for a short time prosper, they will eventually be expelled from their group and forced to change their ways – not by any explicit social process, just as an emergent property of the algorithm. This paper explores a further variant on the SLAC algorithm, in which agents attempted to be more "smart" about their network structure, by preferentially dropping links to neighbours who had not aided them. The surprising result is that a system of these agents very quickly devolves to minimal performance, instead of seeing an increase in performance. The revised algorithm is presented, together with a discussion of why these results occur.

1. Introduction

The SLAC and SLACER algorithms are simple copy and rewire algorithms that have been developed by Hales and colleagues to allow cooperative networks to develop between inherently selfish individuals (Hales 2004, Hales *et al*. 2005). With each individual using simple – and apparently naïve – rules for adjusting network connections, societies can evolve into predominantly cooperative groups. Although defectors may arise in these groups, and for some time prosper, the SLAC/SLACER algorithms effectively isolate selfish individuals over time, making it more profitable to for them to reform their ways and cooperate. Furthermore, recent work has shown that even when some individuals "cheat" and do not follow the rules of the algorithm, the algorithm is extremely robust (Arteconi & Hales 2005).

These algorithms have been demonstrated in a number of domains, from a simple prisoner's dilemma game (where agents randomly choose one of their neighbours to play against) to a peer-to-peer file-sharing domain (where a node forwards a request to one of its neighbours if it cannot handle it itself). This latter domain is a typical concrete application of such algorithms: domains in which there is no centralised mechanism for coordination but in which cooperation is essential. The SLAC and SLACER algorithms have been shown to quickly evolve to produce high levels (>90% success) of system level performance in such domains, even when seeded with 100% selfish individuals. These algorithms are summarised in Section 2, and some illustrative results are presented.

As will be seen in Section 2, the rules that both these algorithms use for network adjustment are extremely simple, and can result in an individual losing links to "good" neighbours. It appears to be intuitive that preferentially maintaining links to these good neighbours would aid an individual, and improve the overall performance of the system. Such a variation is introduced in Section 3, in which individuals keep track of which of their neighbours respond positively or negatively to requests for help, so that when they come to rewire, it is the unhelpful neighbours who are dropped. The simulation results demonstrate counter-intuitively that in fact this drastically degrades the performance of the system, and in fact all the individuals quickly become cheaters. As the analysis in Section 3.a shows, although this result feels counter-intuitive, the reasons are quite straightforward. The randomness of the rewiring in the SLAC/SLACER algorithms is actually a key to their success. The paper concludes first with brief outline of proposed future work, and finally with a summary of the findings of this study and its implications for future work in this area.

2. SLAC and SLACER

SLAC (Selfish Link-based Adaptation for Cooperation) uses the following scheme for network adaptation. At each time step, nodes are given a number of tasks to perform, for which they receive a reward for successful completion. The nature of the task will depend on the domain, but whatever the domain, they will require some cooperation. After all the tasks have been allocated, and performed where possible, the nodes go through a rewiring phase. In this phase, each node picks a random partner from the network. The node's own wealth

(rewards accumulated in the task performance phase) is compared to the partner's wealth. If the partner has greater wealth, the node drops its current links, then links to the partner *and* all the neighbours of the partner. As each link is made, if either node in the link exceeds some fixed maximum number of neighbours, one of the existing links is randomly selected and dropped. When linking to the partner, the node also copies the partner's behaviour – so if the partner was altruistic (would help other nodes), the node will also be altruistic, but if the partner is a cheat (will ask other nodes for help, but will not help others), the node will also be. Finally, with a small probability, random mutation is applied to the node's links, and with an even smaller probability, random mutation is applied to the node's behaviour.

SLACER (Selfish Link-based Adaptation for Cooperation Excluding Rewiring) is an extension of the SLAC algorithm. The only difference is that when a wealthier partner is found, the node *probabilistically* drops each of its existing links – in other words, some existing links may be retained.

a. The SkillWorld Task

SkillWorld is an artificial world that has been created to demonstrate the SLAC/SLACER algorithms (Hales & Arteconi 2005). In this world, each individual has a single skill, and can only process tasks that require that skill. If an individual receives a task that requires a different skill, it may ask its neighbours for help – if one of them does help, the individual who first received the task will get the reward, and the one who helped will incur some cost for performing the work.

SLAC, SLACER and the SkillWorld task have been re-implemented using Java and Repast. The parameters and their default values are summarised in Table 1 – the values used match those used in previously reported results from the original implementation (Hales 2004, Hales & Arteconi 2005). This implementation was then extended further with a supposedly "smart" network adaptation algorithm, as described in Section 3.

Table 1 Parameters used in re-implementation of SLAC/SLACER with SkillWorld

Parameter	Use	Default value
N	Number of individuals	2000
D	Max links per individual	20
M	Link mutation rate	0.01
MR	Behaviour mutation rate	0.001
J	Average number of tasks per individual per time step	10
R	Reward for successful completion	1.0
C	Cost of performing job for other	0.25
R	Initial proportion of altruists in population	0.5
S	Number of skills/task types	5

b. Simulation Results and Discussion

The simulation results from this re-implementation mirrored those of the original, giving confidence that the algorithm was correctly encoded. The aim of this re-implementation was to give a basis for the extension, so it was important to ensure that the original was followed as closely as possible. These results are included here also serve to emphasize those achieved by the so-called "smart" algorithm that is presented in Section 3. In both the SLAC (Figure 1) and SLACER (Figure 2 and Figure 3) cases, it can be seen that the proportion of altruists quickly grows, as does the proportion of jobs that can be completed.

The difference is that in SLACER, after the initial growth of altruists, there is subsequently a decline – the degree of which is relative to the proportion of links that are probabilistically retained – which is in turn reflected in the percentage of jobs completed. This is because of the network structures produced by the two algorithms. In the case of SLAC, isolated clusters of nodes form, which can grow to be quite large but rarely encompass the entire population. This is because nodes drop *all* existing links when they link to a new node. This also means that sub-clusters can break away from a larger group, and in this way cheaters can be expelled from a cluster. Because a small cluster with a high proportion of cheaters will not do well (because most neighbours will not help), the nodes will find better partners in other clusters, and because they adopt the behaviour of their partner, they will become altruistic again. However in SLACER, nodes will usually maintain some of their existing links when they adapt their network. This means that there is usually (after the initial network structuring) only one component, and cheaters are unlikely to be expelled from the network. In this case, a cheater will only change its behaviour when other cheaters largely surround it. Furthermore, because it is likely to retain links to at least some of the cheaters, these cheaters will then benefit from its new altruism, increasing their wealth and making it even more likely that they will attract more cheaters. As the number of links being retained increases, the

likelihood of remaining connected to a cheater increases, thus the lower average performance in Figure 3 over Figure 2.

Why then bother with SLACER? Put simply, because some applications *require* a connected network. Furthermore, studies that have looked at other domains have found that the degradation in performance is not so severe as it is for SkillWorld – it seems that this domain is particularly sensitive, with a link dropping rate of < 0.91 causing the average performance to fall below 90%.

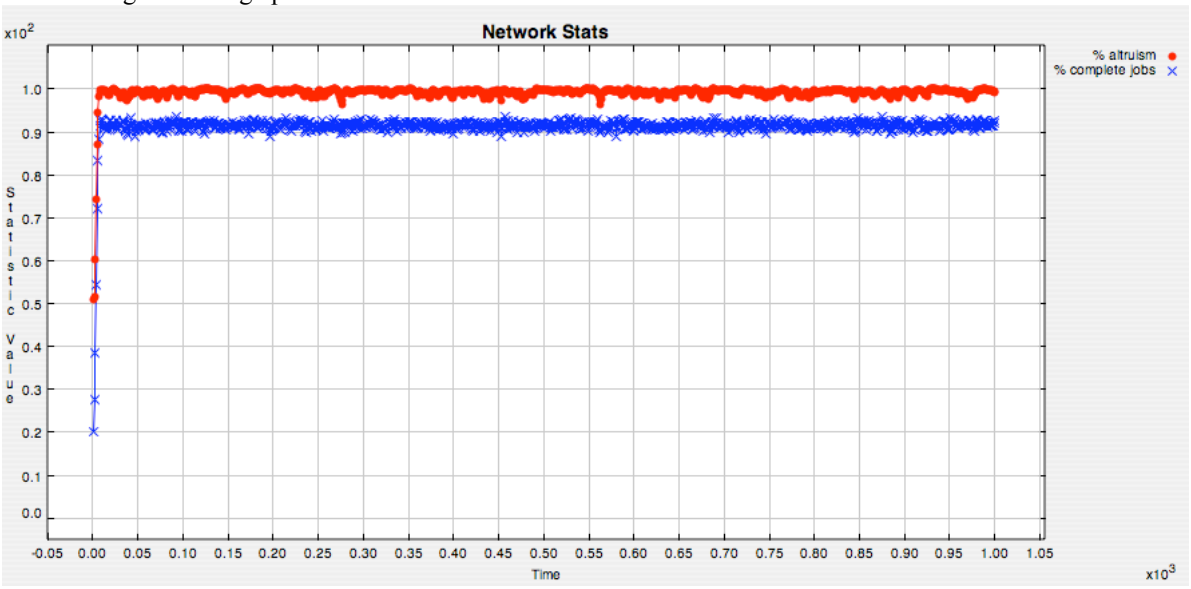

Figure 1 Typical run of the SLAC algorithm in SkillWorld.

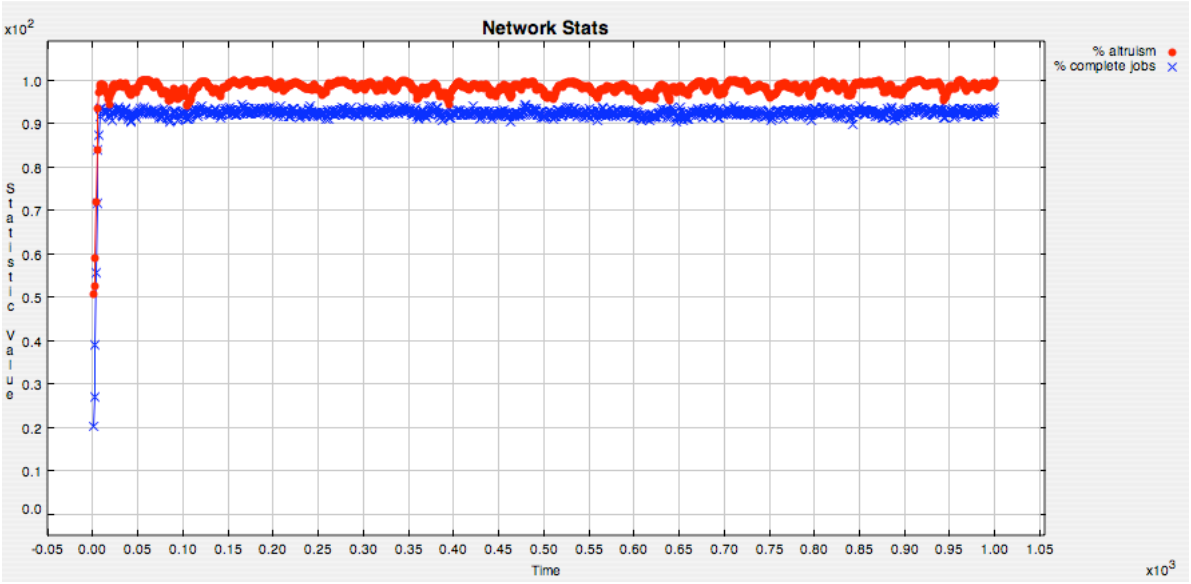

Figure 2 Typical run of the SLACER (with rate of link dropping, W = 0.95) algorithm in SkillWorld.

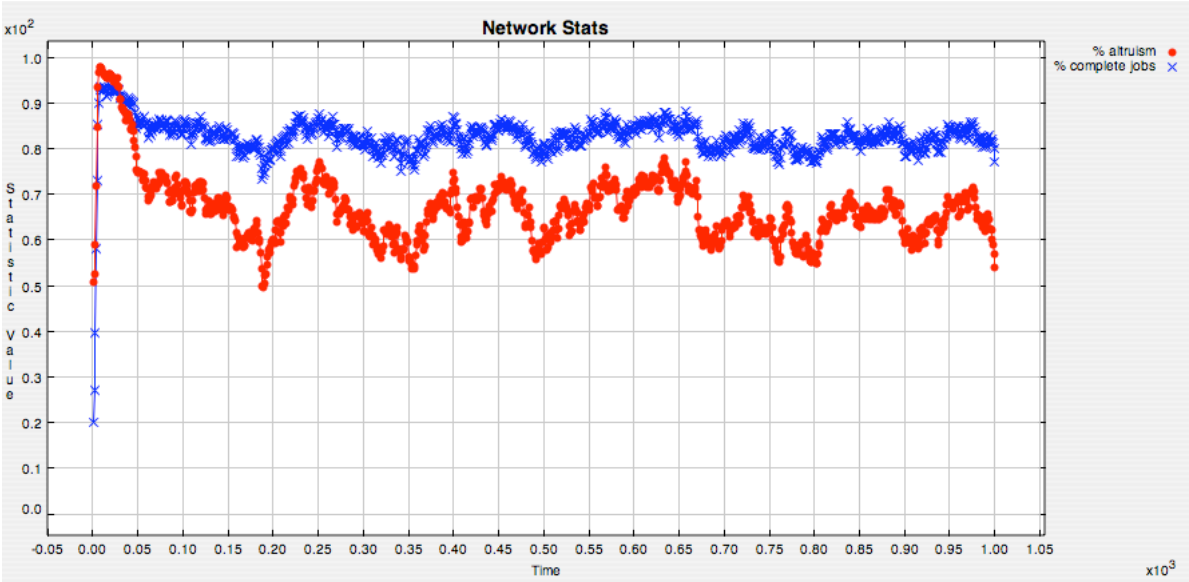

Figure 3 Typical run of the SLACER (with rate of link dropping, W = 0.90) algorithm SkillWorld.

3. The "Smart" Algorithm

In the SLAC and SLACER algorithms, agents discard links with no regard to how useful they have been in the past. In the SLAC algorithm, this is an effective way of isolating cheats, but it leads to a disconnected network. In the SLACER algorithm, links are probabilistically retained, which maintains the connectedness of the network. However intuitively it would seem that an individual could build a better network by retaining those links that had been useful in the past. This would maintain connectedness, as in SLACER, but hopefully achieve a higher overall performance. It was this intuitive feeling that motivated the "smart" algorithm presented here.
To keep track of their helpful (and unhelpful) neighbours, individuals put weights on their links to their neighbours. Prior to a time step, these were set to 0, then when an individual asked a neighbour for help, it would increment the weight if it received help, or decrement it otherwise. To prevent the nodes being penalised when they were *unable* to help (they didn't have the required skill), nodes were also able to see their neighbours' skills, and they would only request help from those able to provide it.
In the link adaptation stage, a node did not drop links before copying its partner's links. Instead, it copied all the links, or added the partner's links' weight where a link already existed. When all the partner's links had been added, the node would prune its links back to the maximum number allowed, dropping those with the lowest weight first.

a. Simulation Results and Discussion

The simulation results of this algorithm at first appeared to have a bug: rather than improving the performance of the system, the outcome was disastrous (see Figure 4). Further analysis revealed the reason for this counter-intuitive result.
Nodes preferentially link to other nodes that are wealthy. However when there are few cheats in a reasonably well-connected network these cheats *will* be wealthier than altruists, because they will be able to find helpers for tasks but not incur the cost of helping others. Hence if an altruistic node compares itself to a cheating node at this stage of the evolution of the network, they will link. The problem is, at the same time, many of the cheating node's neighbours will have been dropping their links to it, so the altruistic node will copy the cheating nature, but many of the links that gave the cheating node its wealth will be gone, so it will not gain these benefits. This problem quickly snowballs, so that the few altruists who are left (or nodes which randomly mutate to become altruists) do poorly – and change to being cheats themselves – because all the cheaters take advantage of them. The result is that the percentage of jobs completed quickly drops to 20%, because *all* the nodes are cheats and so only the jobs that are matched with nodes with the appropriate skill are completed.

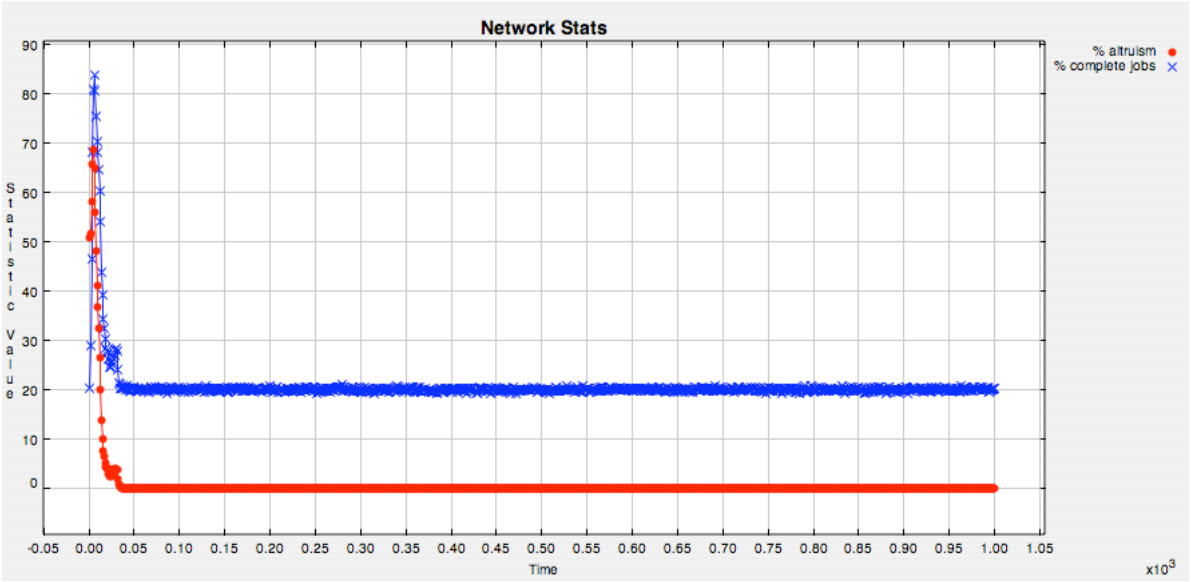

Figure 4 Typical run of the "smart" algorithm on the SkillWorld task.

4. Future Work

As presented in Section 2.a, SkillWorld requires an agent to complete only a single task (either by processing it themselves or by finding a willing helper) in order to receive a reward. In many ways this has parallels with peer-to-peer type applications, where for example the goal is to obtain a particular file. For this reason we continue to have an interest in this test domain, using it to try to discover alternative algorithms that can maintain connected networks *and* maintain high levels of overall performance.

However we are also interested in a variation on this domain, in which *several* skills might be required in order to receive a reward. Each agent would still have only a single skill, and would need to rely on a number of neighbours (or possibly neighbours of neighbours) in order to complete a job. A job would be specified as a series of tasks (which may or may not be ordered), and all of these tasks must be completed. Only then would the agent receive the reward for that job. As in SkillWorld, a cost would be attached to performing a task for someone, and as in SkillWorld, the requesting agent could choose whether or not to share their reward with their collaborators.

This variant has some parallels with a supply chain. While industrial supply chains would perhaps not benefit from these types of algorithms – because other factors including loyalty and reputation are an important part of such systems – they could perhaps be useful in the formation of more dynamic and transitory supply relationships, such as those required to put together a holiday package. Our future work will thus be using a variant of SkillWorld to explore these issues.

5. Conclusion

The SLAC and SLACER algorithms use very simple mechanisms for network adaptation, which can cause nodes to lose "useful" neighbours. However the attempt here to introduce a "smart" algorithm that would preferentially drop "bad" neighbours has failed in a spectacular way. The analysis of this failure has shown that the primary reason for this is that cheats on average will do better than altruists using this approach. If nodes preferentially link to high scoring nodes, the chances of this being a cheat are high (because these are the higher scoring nodes). At the same time, nodes are preferentially dropping links within their network that have cheated, so the links gained from linking to a cheat are likely to be of limited use.

The randomness of SLACER, or the complete rewiring of SLAC ensure that an agent linking to a cheat will usually gain at least some useful neighbours in the process, so that even though the newly-linked node will be a bad neighbour, they may well still be able to use the other new links productively. The potential loss of other useful neighbours in the re-wiring process is thus offset by this gain. It is possible that a *smarter* "smart" algorithm could be devised to produce higher levels of overall productivity while maintaining a fully- (or at least largely-) connected network, but the question that would then need to be considered would be the cost of such an algorithm – one of the beauties of the SLAC/SLACER algorithms is their simplicity and ease of implementation.

6. Acknowledgements

Many thanks are due to David Hales, who developed the SLAC and SLACER algorithms and has been involved in numerous discussions surrounding this topic. Thanks too to the participants of the Cooperative and Selfish Systems workshop in May 2006 for their comments. This work has been undertaken as part of the NANIA project, funded under the EPSRC's Novel Computation Initiative.

7. References

Arteconi, S. & Hales, D. (2005) Greedy Cheating Liars and the Fools Who Believe Them. Dec. 2005, Technical Report UBLCS-2005-21

Hales, D. (2004) 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.

Hales, D. & Arteconi, S. (2005) Friends for Free: Self-Organizing Artificial Social Networks for Trust and Cooperation. Sept 2005, http://arxiv.org/abs/cs.MA/0509037

Hales, D.; Arteconi, S.; Babaoglu, O. (2005) SLACER: randomness to cooperation in peer-to-peer networks. Proceedings of the Workshop on Stochasticity in Distributed Systems (STODIS'05) including in the Proceedings of IEEE CollaborateCom Conference, Dec. 19, 2005. San Jose, CA.

Hales, D. & 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.