

# DELIS

Dynamically Evolving, Large-scale Information Systems



## Emergent Group-Like Selection in a Peer-to-Peer Network

David Hales  
University of Bologna, Italy

[www.davidhales.com](http://www.davidhales.com)

This work partially supported by the EU  
within the 6th Framework Programme  
under contract 001907 (DELIS).



ECCS Conference  
Paris, Nov. 16<sup>th</sup>, 2005



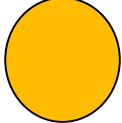
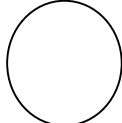


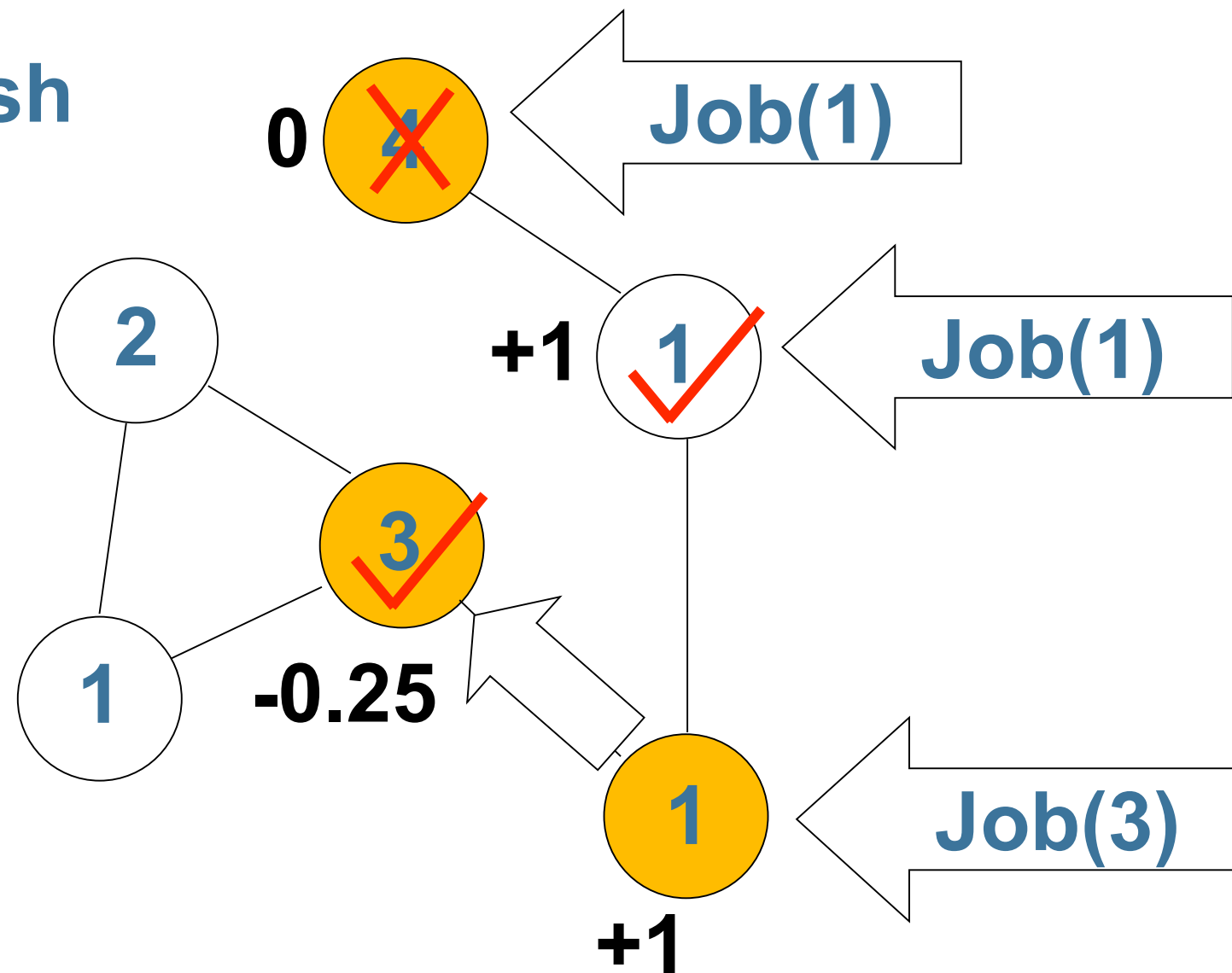
- Consider a P2P overlay network in which each node:
  - Offers a service (e.g. storage, processing etc.)
  - Receives jobs submitted by users
  - May ask neighbour nodes to help complete jobs
  - May complete jobs for neighbours
  - May move in the network by making and breaking links
  - Uses local information only
  - Behaves in a selfish way (boundedly rational)
  - May compare its performance to other nodes
  - May copy links, and behaviours of other nodes
- We want a scalable, robust, light-weight decentralised algorithm that self-organises network to maximise system level performance

- SLAC = Selfish Link-based Adaptation for Cooperation
- Demonstrated to be effective in P2P networks when:
  - Peers play the Prisoner's Dilemma with neighbours (ESOA'04)
  - Peers answer queries and share files (IEEE TSMC'05)
- But in these previous scenarios:
  - Nodes provided an identical service
  - Cooperation resulted from all nodes behaving identically
- This new problem requires specialists nodes to work together
- In order to maximise system level performance nodes need to do different things, not identical things
- This work therefore tests if SLAC can support inter-node specialisation



- To test this produced a simulation model called SkillWorld
- The population consists of  $N$  nodes (fixed)
- Each node has the following state:
  - A single skill from a set  $S \in \{1,2,3,4,5\}$  (the service provided)
  - An altruism flag  $A \in \{0,1\}$  (indicates if node helps neighbours)
  - A utility  $U \in \mathbb{R}$  (a performance measure)
  - Some set of links to other nodes (max. of 20)
- Each node asynchronously receives and attempts to complete jobs
  - Each job is marked with a single skill # (randomly chosen)
  - Job must be processed by a node with matching skill
  - If receiving node  $i$  has req. skill, job is completed  $U_i = U_i + 1$
  - If node  $i$  does not have req. skill it asks its neighbours
  - If a neighbour  $j$  is found with  $A = 1$  and matching skill then:
    - Job is completed,  $U_i = U_i + 1$ , but,  $U_j = U_j - 0.25$

-  = altruist
-  = selfish



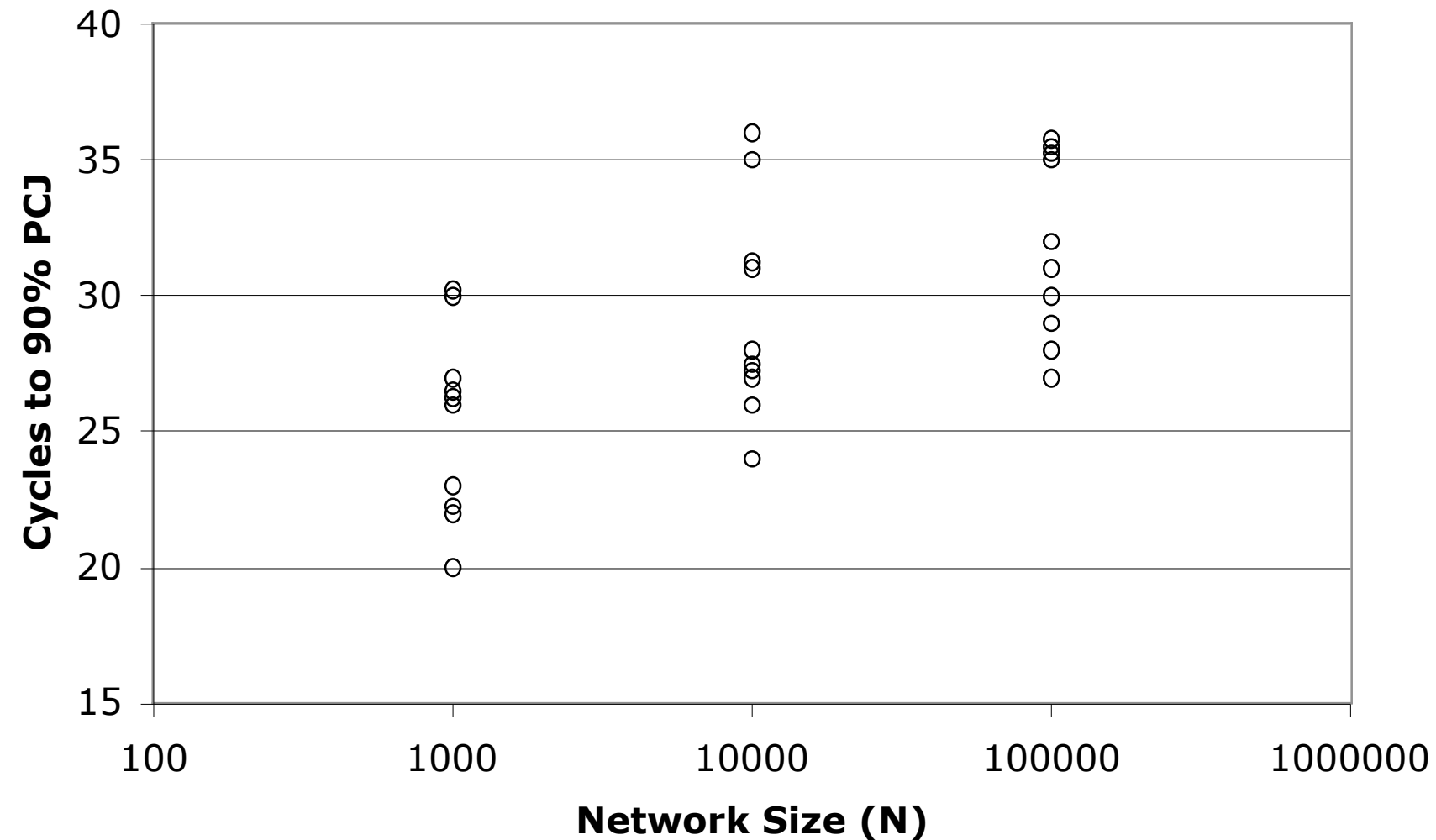


- SLAC follows a kind of evolutionary process
- Periodically each node:
  - Engages in application level activity producing utility (SkillWorld)
  - Compares its utility to another randomly chosen node
  - If the other node has higher utility then
  - Copy links and some behaviour of other node
  - With low probability “mutate” links and behaviour

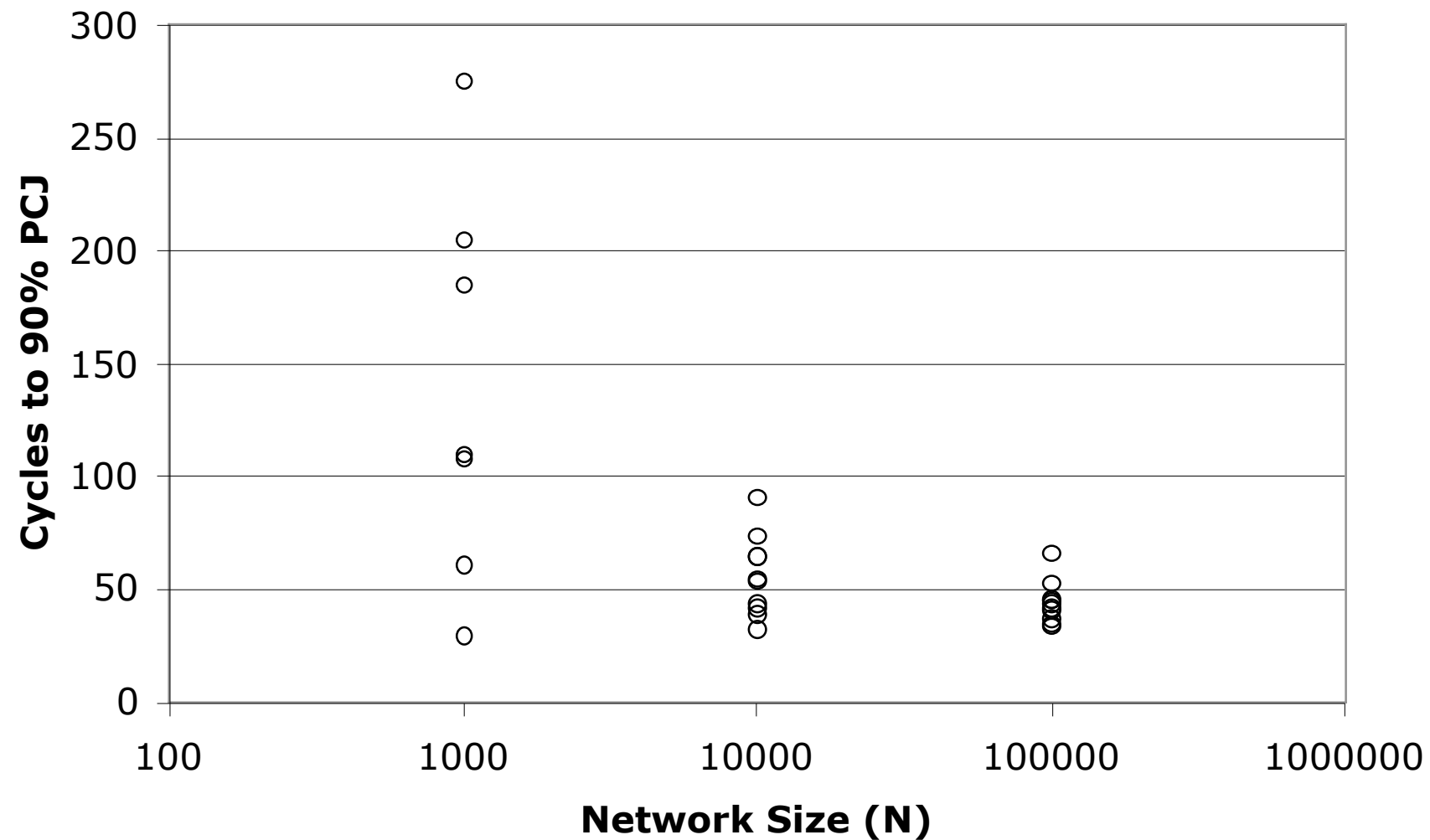
<p>Active thread:</p> <pre> do forever:   sleep for some short time period   i ← this node   with prob. P reproduce:     j ← SelectRandomNode()     j.GetState(i)     if i.Utility ≤ j.Utility       i ← CopyStatePartial(j)       Mutate(i) </pre>	<p>Passive thread:</p> <pre> do forever:   j ← this node   GetState(i):     Send j.Utility to i     Send j.Links to i     Send j.AltruismFlag to i </pre>
<p>Function CopyStatePartial(j):</p>	<p>Function Mutate(i):</p>
<pre> i.AltruismFlag ← j.AltruismFlag drop all links from i i.Links ← j.Links </pre>	<pre> with prob. M mutate i.AltruismFlag with prob. MR mutate i.Links:   drop all links from i   i.Links ← SelectRandomNode() </pre>

- In each cycle,  $10N$  jobs are submitted to randomly selected nodes
- Each job is given a randomly selected skill requirement
- Nodes initialised with random skills and links (random network)
- Initial topology of network made little difference to results
- Compared initialisation of altruism flag randomly and all selfish
- Compared different network sizes  $N$
- Measured proportion of completed jobs (CJ) in each cycle
- Mutation values ( $M = 0.001$ ,  $MR = 0.01$ )
- If a node reaches its max. links (20) then a random link is discarded if a new link is required
- Utility for each node = CJ - total help cost
- Ran simulations until 90% of jobs completed

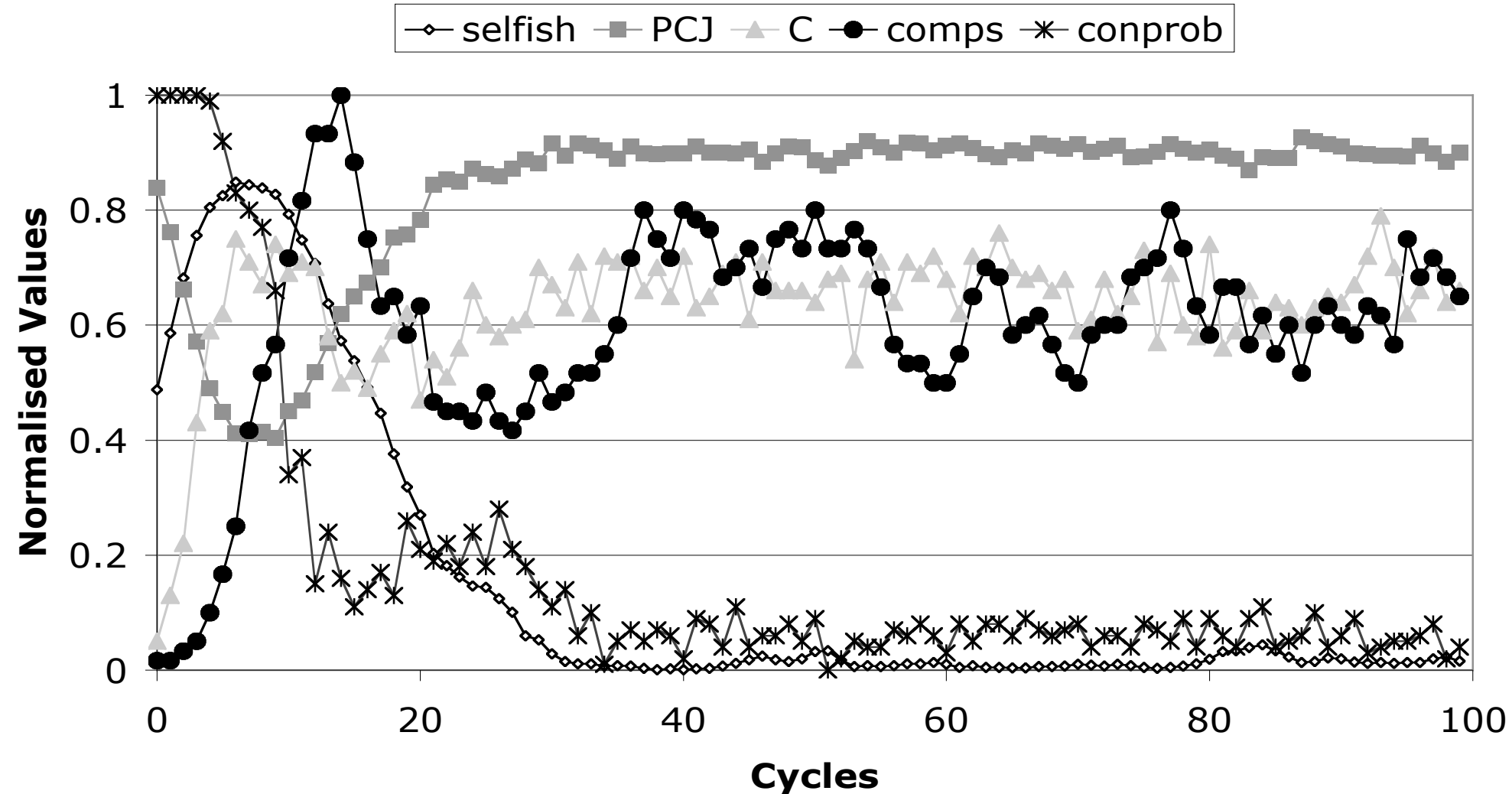




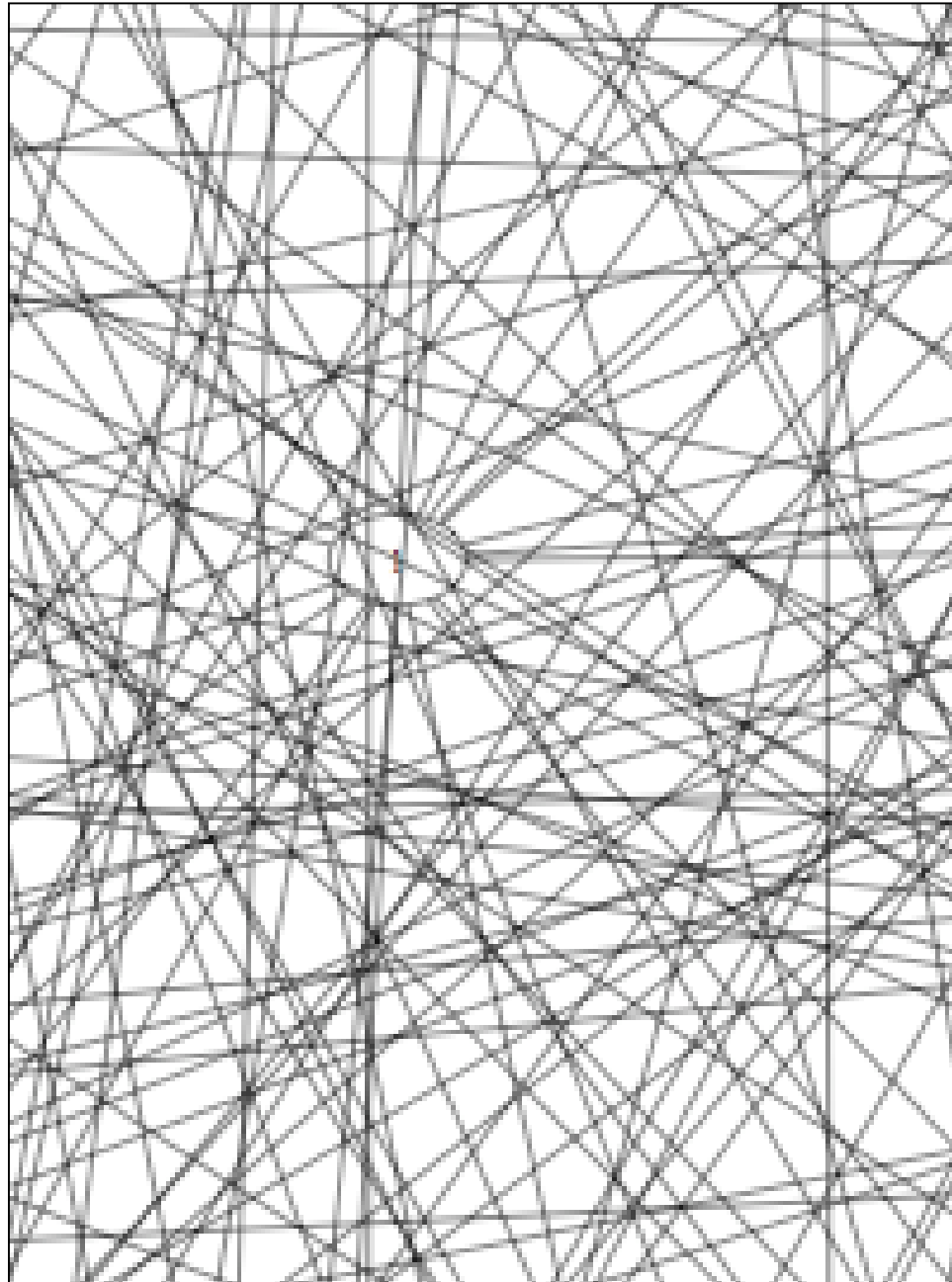
Number of cycles to high performance for different N  
When  $PCJ > 90\%$  over 90% of all jobs submitted to nodes are completed.  
*Nodes are initialised at random*



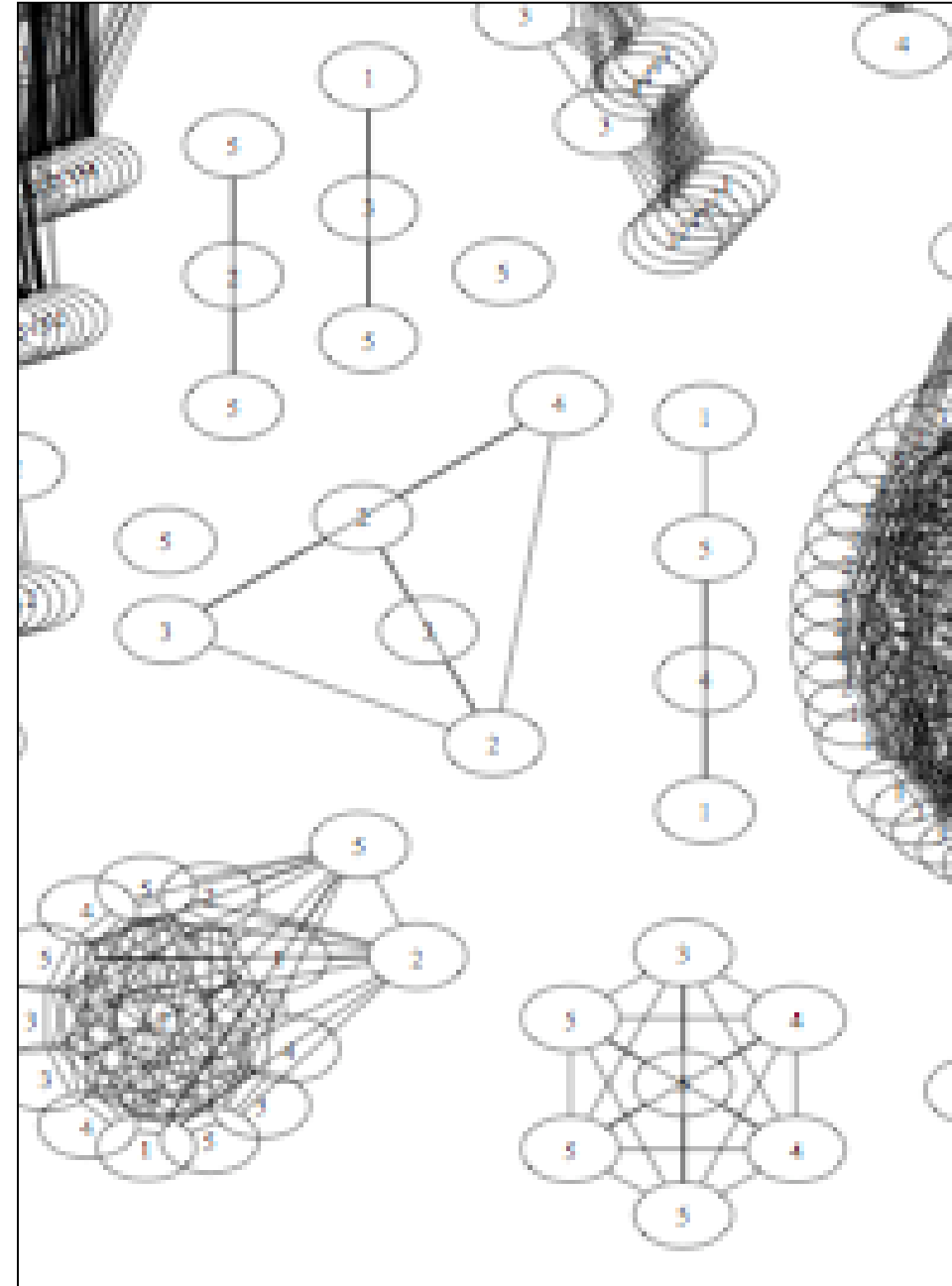
Number of cycles to high performance *when all nodes are initialised selfish*  
Note that there is a reverse scaling cost here.



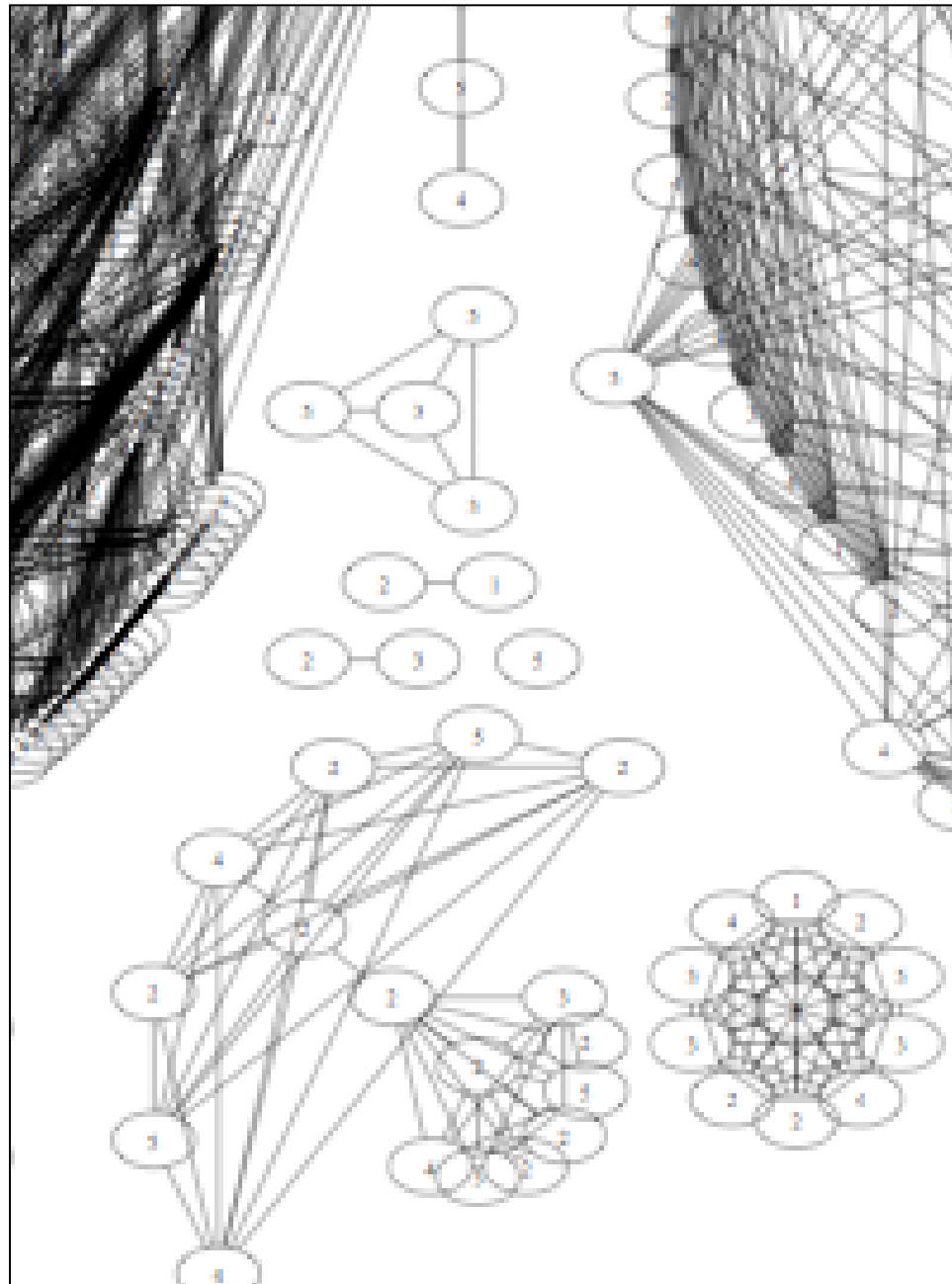
Typical single run (N=1000) from random initialisation.  
selfish = proportion non-altruists, C = clustering coefficient,  
comps = components in the population (normalised by dividing by 60),  
conprob = average probability that a route exists between any two nodes.



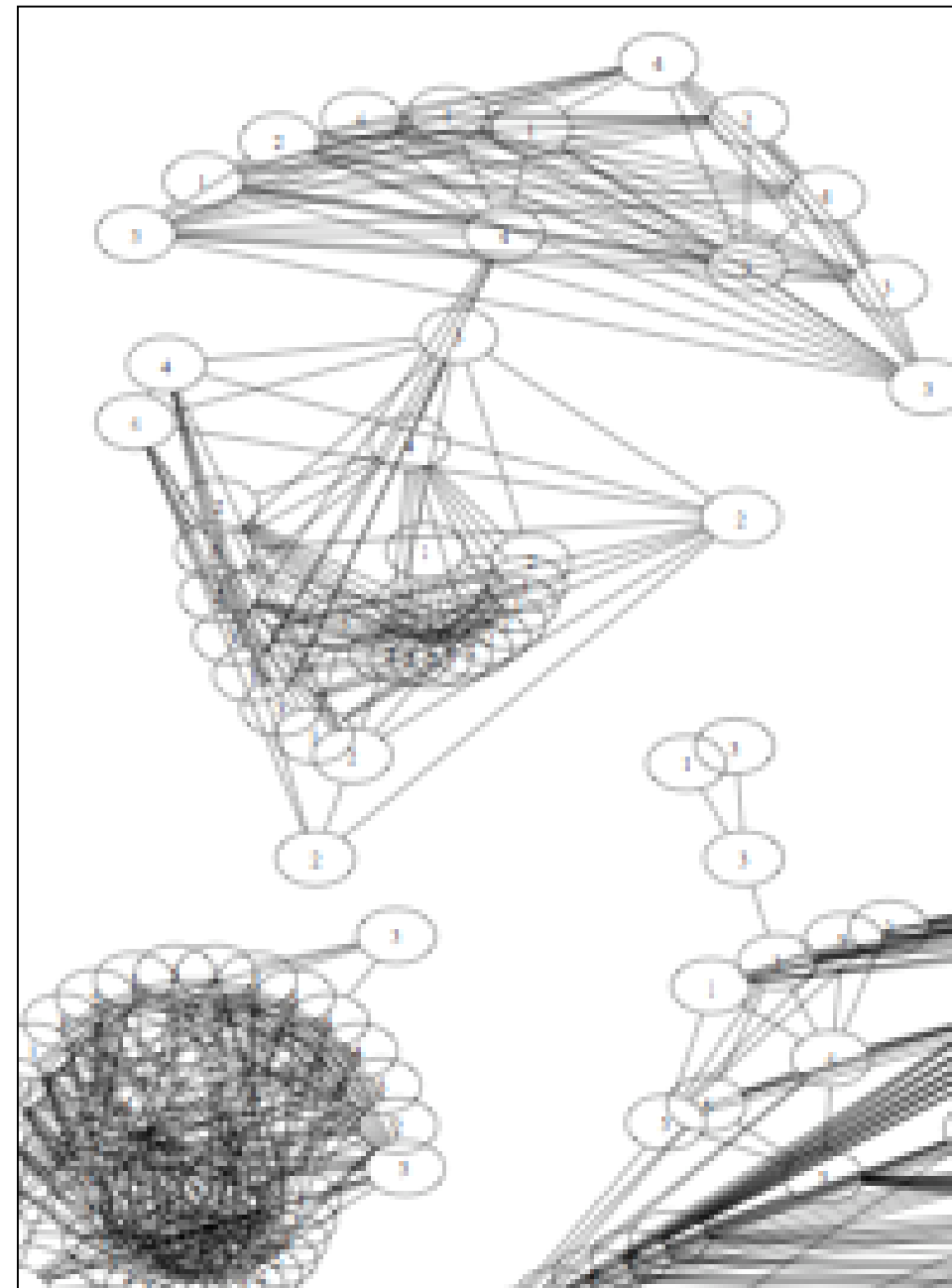
**Cycle 0**



**Cycle 10**



**Cycle 20**



**Cycle 30**



- From individual bounded utility maximising behaviour (SLAC)
- Altruistic “tribes” emerge with internal specialisation
- Tribes that do well - collectively - tend to recruit new nodes
- Tribes that perform badly - collectively - tend to lose nodes
- Hence productive tribes prosper, defective tribes “die”
- This is a kind of “tribe selection” via recruitment and retention
- By giving nodes the ability to choose their tribes a kind of tribe level evolution happens - evolution at the next level

- But SLAC produces extreme tribalism with disconnected components => SLACER (on-going work)
- SLAC assumes honest passing of info and utility comparison => Greedy Cheating Liars (on-going work)
- The SkillWorld task is an “easy” test => more realistic scenarios
- System performance does not attain more than about 93%
- If Skill mutated then can adjust to different job task loadings
- But if Skill is copied like the AltruismFlag then fails to converge, yet in similar scenarios it does
- Tribe recruitment is the key idea => (on-going work)
- Future work could drop utilities and move to satisficing where aspiration level is a kind of “required service level”



# DELIS

## Conclusions



Dynamically Evolving, Large-scale Information Systems

David Hales, University of Bologna, [www.davidhales.com](http://www.davidhales.com)





 Dynamically Evolving, Large-scale Information Systems

David Hales, University of Bologna, [www.davidhales.com](http://www.davidhales.com)