Open Collaboration for Innovation: Principles and Performance

Electronic Companion

Open Collaboration for Innovation: Principles and Performance

Electronic Companion

Contents

1. Model Summary and Main Assumptions

The overall context for the model is a population of agent types, where each agent must solve a set of tasks by accumulating sufficient resources required for each task. The numbers of agents (100) and tasks (100 per agent) were selected to provide a sufficient density of observations over a time frame as to reveal any treatment effects, consistent with similar approaches (e.g., Miller and Lin 2010). The resource limitation on agents ensured that transfer would be required in the model dynamics. This narrative augments the overall behavioral algorithm description appearing in the text.

- All simulations use a population of uniquely identified *agents*, $A = \{a_1,\ldots,a_{100}\}$.
    - 96% of the agents are assigned one of three primary cooperative types: Cooperator, Free-Rider, or Reciprocator.
    - A Cooperative type defines an agent's contribution profile (see below).
    - The remainder 4% behave as one of the three cooperative types by random draw.
- *Tasks* are goals to attain and each task is identified by its type, represented as an integer.
    - Task types are *resource-specific*, and resources for tasks are *type-specific*: resources for one task type cannot be applied to another task type.
- For every agent, a_i, there is a unique *resource vector* that retains a discrete set of resources for five task types and associated endowment level *r* for each resource ($r_1, r_2, \ldots, r_5$). Resources do not expire across periods. An agent can hold up to five, with FIFO replacement. Sensitivity analysis of resource vector size appear below.
    - At the start of each simulation, the resource vectors of all agents are initially populated via selection from random uniform distributions of tasks and resources.
    - As new resource types are accumulated, existing resource types are replaced on a first-in, first-out schedule. Rivalry computations are time insensitive and are engaged only during the transfer of a resource.
- For every agent, a_i, there is a *goal vector* that defines a set of tasks that the agent needs to attain ($g_1, g_2,\ldots, g_{100}$). Attainment of a task requires that the agent accumulates the required resource level for the task.
    - Agents work sequentially through their personal goal vector, one task at a time, asynchronously and incrementally.[1]
    - The attainment of any task requires 50 units of *resources*.

[1] As long as there is no dependency on the order of tasks in a diverse set of problem solvers, a property ensured here by the combination of resource constraints and random initializations, the overall results are insensitive to agents pursuing their individual tasks individually or in parallel (cf. Hong and Page 2004).

- Each simulation run requires 10,000 tasks to be solved (100 agents solving 100 tasks). See Sensitivity analysis of the number of agent tasks below.

- When an agent is faced with a task, it follows a *prescribed sequence of search*.
  - An agent first checks its *own resource vector* to see if it possesses resources for the task.
  - If it does not have any resources for the task, it checks a simple index that identifies all agents that possess resources for that task. The agent will attempt to transfer the needed resource from each. Search will terminate after the first successful transfer.
  - If the attempts at transfer fail or result in contributions of insufficient resources (i.e., accumulations < 50), the agent will then *generate the remaining resources* on its own at an accumulation rate defined by an exponential function reflecting a 10% per period accumulation rate (see Resource generation rate below).[2]
- An agent's *cooperative type* defines its contribution profile, based on Kurzban and Houser (2005).
  - At the end of each period, *p*, the average contribution level of the population, ε_p, is determined.
  - The Cooperative type of an agent determines what proportion of its endowment will be offered in a transfer. A *Cooperator agent* contributes at proportions that exceed 50% of its endowment, regardless of the contribution level of the population. The exact proportion level is determined for each transfer by a draw from a Gaussian distribution with a mean of 0.75 and a standard deviation of 0.125. For example, if a Cooperator agent's endowment for a requested task is 40 (resource units) and its proportion draw is 0.60, then it will contribute 24 resource units (0.6 x 40).
  - A *Free Rider agent* contributes at proportions that are less than 50% of its endowment, regardless of the contribution level of the population. The exact proportion level is determined for each transfer by a draw from a Gaussian distribution with a mean of 0.10 and a standard deviation of 0.05. The Free Rider proportions are purposely biased low (relative to the Cooperator Agents) to generate conservative results. For example, if a Free Rider agent's endowment for a requested task is 40 (resource units) and its proportion draw is 0.14, then it will contribute 5.6 resource units (0.14 x 40).
  - A *Reciprocator agent* behaves as either a Cooperator or a Free Rider, based on whether the average contribution levels of the population in the most recent two periods are non-decreasing ($\varepsilon_{p-1}/\varepsilon_{p-2} \geq 1$) or decreasing ($\varepsilon_{p-1}/\varepsilon_{p-2} < 1$), respectively. Therefore, Reciprocator

[2] This approximates a simple exponential function with a generic growth function where the mechanisms underlying the growth have the same influence on the additional time needed to finish the task, no matter how much of the task resources (if any) have been exchanged or the time (periods) already spent on generating them (Derman et al. 1973).

agents are specifically sensitive to (and privy to information about) the prior behavior of the population.

- A *Random Agent* randomly adopts the contribution profile of one of the prior three types every time a contributory request occurs. Random agents are those whose behavior is too unstable to be categorized, found to constitute 4% in the general human population.
- Any agent can contribute no more than once a period. After contributing, it becomes unavailable for further requests as it concentrates on own tasks.
- While producing resources, agents are unavailable for requests.
- Agents will contribute only resources that are not currently in use. Agents will search for resources whenever they become necessary for completing the task. Agents do not stock and hoard resources if there is no current use for them.

Parameter	Explanation	Value(s)
Size of Collective	Number of agents	100
Agent Resources	Number of resources held by agent	5
Resource Distribution	Which 5 resources are initially assigned to each agent prior to a run	RND[1, 100]
Initial Resource Level	The initial resource levels for the 5 initial resources	RND[0, 50]
Goals	Number of goals per agent	100
Goal Resource Level	Resource level to achieve acquisition goal	100
Transfer Limit	Maximum transfers per agent per period	1
Cooperative Behavior	Donation level (of endowment) by Cooperators	G[0.75, 0.125] ††
Free Rider Behavior	Donation level (of endowment) by Free Riders	G[0.10, 0.05] ††
Resource Generation	Self-generation of resources function	Inc(x, 0.10) †††

† RND[i, j] = Random Uniform distribution between i and j (inclusive)
†† G[μ, σ] = Gaussian distribution with mean μ and standard deviation σ
††† Inc(x, 0.10) = Each period the current resource level, x, is increased by 10%

Table EC1: Specific model parameters, their role in the model and their values

2. Performance Measurement

The dependent variable in all of the experiments is collective performance, serving as a measure of system behavior. This is appropriate as our interest lies in understanding open collaboration as a system, rather the performance of agents (individuals or firms) that rely on it. Had we focused on performance of specific agents, the results could have been misleading. In open collaboration, as in a commons (Benkler 2002, 2006; Benkler and Nissenbaum 2006), high individual performance can achieved through non-cooperative strategies, such as free-riding and defection. Such behavior, which leads to high individual performance at the expenses of the collective, is the cause of the Tragedy of the Commons (Hardin 1968; Olson 1965). In contrast, the performance measure we chose accounts for cooperation, which is at the core of open collaboration. Non-cooperative strategies will not appear as improvement in performance. The use of a single measure of performance is common in models such as this (e.g., Dahan and Mendelson 2001; Terwiesch and Loch 2004; Terwiesch and Xu 2008).

The focus on the drivers of performance address some of the general concerns about using performance as the dependent variable in organizational research (March and Sutton 1997). Thanks to the methodology of agent-based modeling, performance can be interpreted in terms of specific causal (underlying algorithmic) components.

Once we focused on the measurement of collective performance, the remaining alternatives were largely similar, so we selected on the lucidity of presentation. One unsatisfying option was presenting absolute performance, such as the number of tasks completed in a period or the time required for a number of tasks to be completed. Such measures may be intuitive, but a relative measure is superior because it allows easy comparison across conditions. A critical feature of the inquiry is uncovering some of the drivers of open collaboration performance, so a good indicator of performance would convey the direction and magnitude of performance effects across conditions. After considering a few alternatives, it emerged that the proportion of tasks completed through cooperation is the most useful performance measure for our purposes. This measure incorporates the most general definition of performance: turning inputs into outputs. Completing tasks with more cooperation means using less resources, and thus this performance measure shows how efficiently (or cheaply, quickly, with less knowledge, etc.) a goal can be achieved. Better than an absolute measure of performance, this relative measure allows for easy comparison across different conditions without a change in scale. For it accounts for both output and input (effort, resources employed), it allows comparison across situations in which one or both change.

3. Sensitivity Analyses

3.1 Resource vector size

In the model, transfers between agents are based on the resources held by each. Therefore, the number of resources that an agent can retain (i.e., the resource vector) is important and was determined in the following manner. A sensitivity analysis on resource vector size was conducted by varying resource vector size with 100 agents seeking to solve 100 tasks, using three cases that would like lead to substantial performance variation:

i. The general human population, high Need Heterogeneity, and low Rivalry (the base case).
ii. Same as the base case, but incorporating low Need Heterogeneity.
iii. Same as the base case, but incorporating 100% Rivalry.

The results are shown in Figure EC1. As can be seen in the figure, when agents cannot retain any resources to exchange, performance is appropriately 0%, indicating that on average no exchanges occur between agents. The performance increases in a slightly non-linear fashion and reaches a maximum at 10. Furthermore, the distribution of resource vector sizes is largely insensitive to Need Heterogeneity and Rivalry, thus would not moderate these constructs. The performance of the base case using the selected resource vector size (5 resources) was, on the average, 70%, which was judged reasonable because it allows for both performance improvements and declines via other manipulations while avoiding ceiling or floor effects.

3.2 Number of agent tasks

The numbers of agents and tasks were selected to provide a sufficient density of observations over a time frame as to reveal any treatment effects consistent with similar approaches (e.g., Miller and Lin 2010). Figure EC2 shows the average performance under varying numbers of tasks per agent. The figure indicates that at least 30 tasks are required to achieve convergence to a stable performance level under the base case and extreme values of the two manipulations defined above (low Heterogeneity and high Rivalry). The additional tasks (70) were added to allow the model to produce stable performance levels for each manipulation and, as noted, sufficient density of observations.

3.3 Resource generation rate

The self-generation process is modeled as the accumulation of resources at a rate of 10% per period for a task. This approximates a simple exponential function with a generic growth function where the mechanisms underlying the growth have the same influence on the additional time needed to finish the task, no matter how much of the task resources have been exchanged (if any) or the time (periods) already spent on generating them (Derman et al. 1973). This was checked by a set of runs of the base case and the fit by the following function:

$$Resource\ level = R_1 e^{0.0953p}$$

where R_1 is the starting resource level held by the agent (acquired by exchange, or possessed initially by the agent), where $R_1 > 0$ (consequently, zero resource levels are initialized with 5 resource units reflecting an initial generation of 5%) and p is the current number of periods that have been dedicated to generating the resource. Figure EC3 illustrates the sensitivity of model at the 10% accumulation rate over various starting resource levels. For example, if an exchange resulted in 10% of the required resource (see point A in the figure), an agent will need to devote approximately 25 periods to accumulate 100% of the resource. However, if an exchange resulted in 50% of the resource (see point B in the figure), then an agent would only need approximately 8 or 9 periods to accumulate the necessary level.

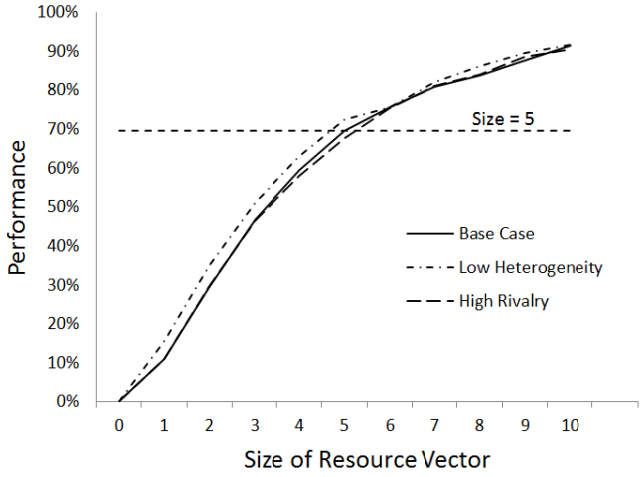

Figure EC1. Sensitivity analysis varying the size of the agents' resource vectors under different cases depicting its impact on collective performance

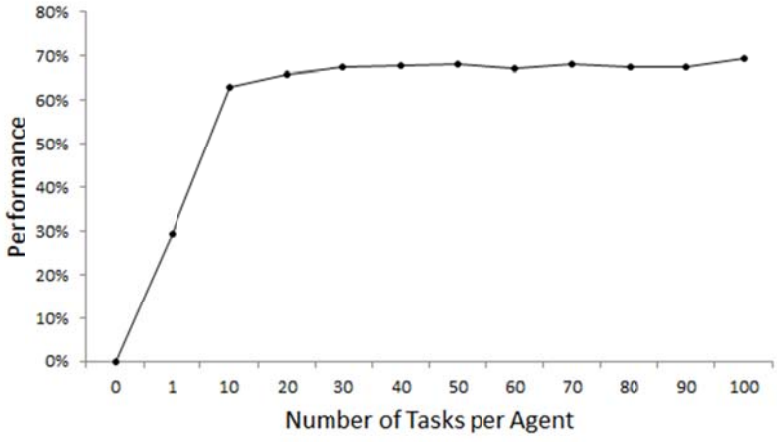

Figure EC2. Sensitivity analysis varying the number of tasks per agent under the base case parameters depicting its impact on performance

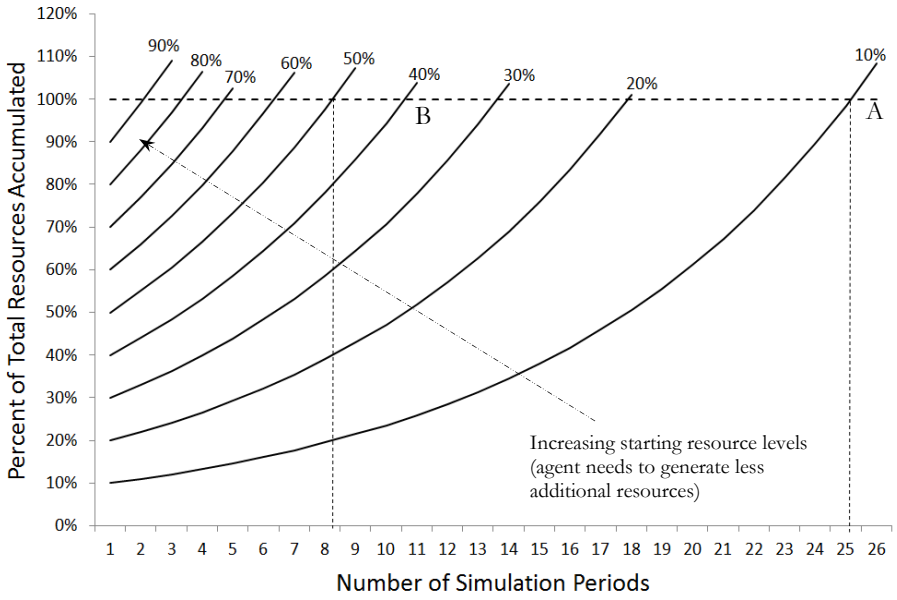

Figure EC3. Sensitivity analysis varying the initial resource percentage level (percentages indicated next to lines) and the number of periods required (under self-generation) to accumulate 100% of the resource.

4. Further Analyses

4.1 Contributions across Cooperative Types

To explicate the contribution dynamics, we analyzed how the contribution by cooperative types. Because of the sensitivity to Cooperators, we used the distribution identified in the callout in Figure 2 (5% Cooperators held constant, varying Free Riders and Reciprocators). A tipping point occurred when the Free Rider/Reciprocator ratio exceeded 50%. After this point, two things occur. First, the overall number of transfers drops. Second, the burden of self-generation of resources rises. As more Free Riders enter the population, they absorb more transfers while contributing disproportionately lesser amounts, lowering the average amount of contributions in the population. Consequently, Reciprocators lower their contributions to mimic Free Riders. As a result, more agents have to incur an increasing effort for self-generation of resources.

4.2 Causes of Non-Linearity under Low Need Heterogeneity

We find non-linear effect of rivalry under low Need Heterogeneity (Figure 3). What causes the non-linearity? Recall that non-rival resources are not affected by transfers or the level of Need Heterogeneity. As rivalry increases, the resources held by an agent deplete in use or transfer in proportion to the particular rivalry level. Consequently, when needs in the population become similar, more agents seek similar resources, but as

rivalry increases, less resources are available for (and after each) transfer. The availability of any rival resource declines exponentially over repeated exchanges, and repeated exchanges are at their highest when Need Heterogeneity as at its lowest.[3] Therefore, the nonlinear effect is most clearly revealed with 100% rival resources under Need Heterogeneity of 0% (Figure 3, edge β).

4.3 Tipping Point Behavior

For a stable ratio of Cooperators, varying the remaining population composition between Free Riders and Reciprocators leads to systematically declining performance as the percentage of Free Riders increases relative to Reciprocators (Figure 1, callout). To obtain an indicator of the nonlinear break in that decline, we performed a series of simple piecewise linear regressions for populations of Cooperators at 1%, 5%, 10% and 20%. The remaining populations had no significant drop in performance. For each data set, we calculated the tipping point via piecewise linear regression, then calculated the average performance before and after the tipping point. The tipping point was the ratio of Free Riders after which performance dropped as indicated by the estimated piecewise linear model. As can be seen, only after 60% to 70% of the remaining population becomes Free Riders does performance indicate a "bend" in declination. Also, as the ratio of Cooperators increases, the impact of the tipping point decreases (see Difference column in the table).

Cooperators	Free Rider Tipping Point	R^2*	Average Performance		Difference
			Before TP	After TP	
1%	70%	0.977	38.7%	13.80%	24.9%
5%	70%	0.988	50.53%	26.37%	24.26%
10%	60%	0.978	56.0%	41.9%	14.1%
20%	60%	0.955	63.5%	56.6%	6.9%

Table EC1: Estimated tipping point in selected Figure 1 populations

* Variance accounted for by piecewise linear model fit.

[3] Specifically, for a rivalry level r and given an initial level k of a resource held by an agent, the level of the resource remaining after x exchanges is given as: ke^{-rx}.

4.4 Performance Means at Selected Parameter Values

	Rivalry	Need Heterogeneity	Mean Performance
Edge γ	100%	100%	68.53%
	50%	100%	73.17%
	0%	100%	71.49%
Edge δ	0%	100%	71.49%
	0%	50%	71.67%
	0%	0%	76.82%
Edge β	0%	0%	76.82%
	50%	0%	37.11%
	100%	0%	34.53%
Edge α	100%	0%	34.53%
	100%	50%	56.45%
	100%	100%	68.53%

Table EC2: Performance means at selected parameter values at the edges of Figure 3. The third point of an edge is the first point of the next edge.

5. Flowchart of Agent Behavioral Algorithm

The agent behavioral algorithm is visualized in a flowchart depicting the sequence of processes in the model.

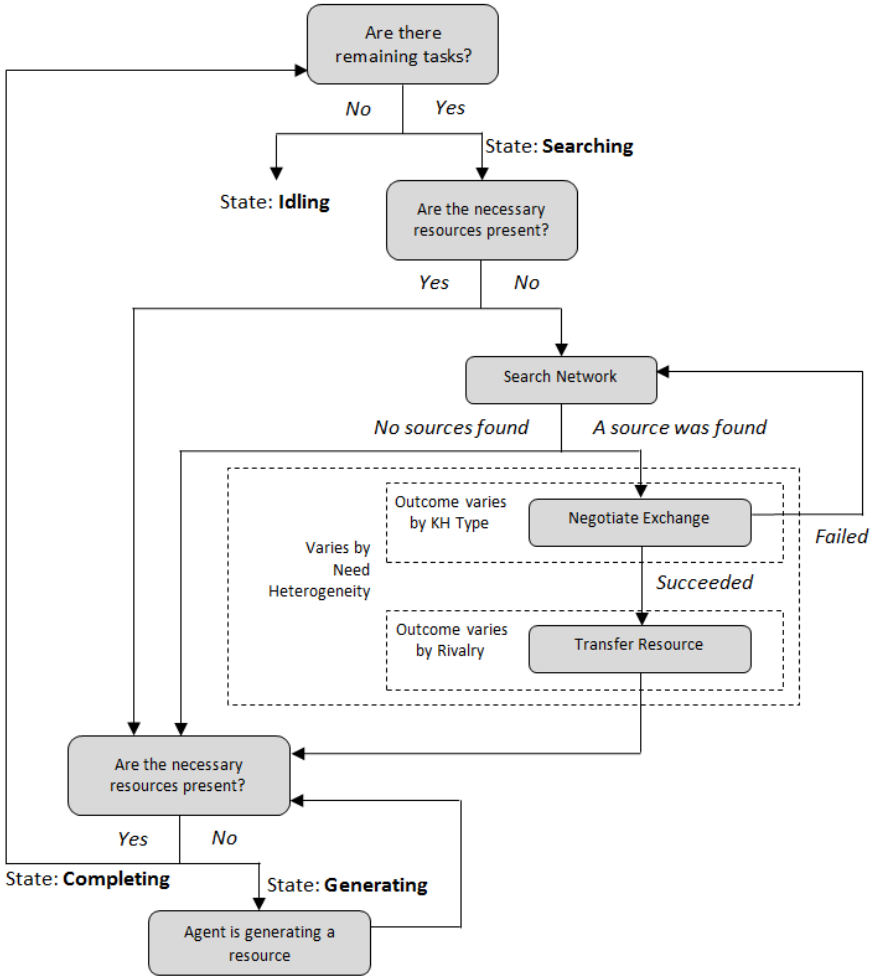

Figure EC4. Flowchart of Agent Behavioral Algorithm

6. Example of Agent Behavioral Algorithm

What an agent does in any given time step depends on the agent's behavioral state. Four behavioral states are defined:

a. **Searching** for a resource from other agents

b. **Self-Producing** a resource if the quantity obtained through search is not sufficient

c. **Completing** a task when sufficient resources have been accumulated (through exchange, generating its own resource, or both), then addressing the next task.

d. **Idling**, when the agent has no more tasks to complete.

Agent	Agent State	Resource Sought	Agent Activity within a Period
33	Self-producing Self-producing	X	• Agent 33 is self-production of Resource X from prior periods. • Agent 33 accumulates an additional amount of Resource X (as defined by the growth function), but less than the required 50 units. • Agent 33 retains the same behavioral state; the simulation advances to the next randomly selected agent.
2	Searching Completing	D	• Agent 2 has 30 units of Resource D and is searching for the additional 20 units. • Agent 2 finds that Agent 53 has the needed resource and makes a request. Agent 53 is a Free-Rider and agrees to provide 5 units (and thus loses 5 resource units, due to 100% Rivalry). • Agent 2 sees that Agent 71 has the needed resource and makes a request. Agent 71 is a Cooperator and contributes 15 of its 50 resource units (and thus loses 15 resource units). • Agent 2 now has accumulated the necessary 50 resource units so the task requiring Resource D can now be completed. • Agent 2 shifts to a new behavioral state; the simulation advances to the next randomly selected agent.

71	Searching Self-producing	D	• Beginning on a new task, Agent 71 searches for the required Resource D. It first searches its own inventory and finds that it has 35 resource units, because it transferred 15 units to Agent 2. • Agent 71 sees that Agent 33 has the needed resource and makes a request. Agent 33 does not respond because it is in self-producing state. • Agent 71 sees that Agent 55 has the needed resource and makes a request. Agent 55 is a Reciprocator and agrees to provide 10 of its 22 resource units (and thus loses 10 resource units). • No more agents have the resource, so Agent 71 must self-produce the remaining units. • Agent 2 shifts to a new behavioral state; the simulation advances to the next randomly selected agent.
14	Completing Searching	Y W	• Agent 14 had accumulated the necessary 50 resource units so the task requiring Resource Y is now completed. Due to the rivalry conditions, completion of the task also eliminates that resource. • Agent 14 beings a new task requiring Resource W. • Agent 14 shifts to a new behavioral state; the simulation advances to the next randomly selected agent.
83	Completing Idling	Z	• Agent 83 now has accumulated the necessary 50 resource units so the task requiring Resource Z can now be completed. There are no more tasks for Agent 83. • Agent 83 shifts to a new behavioral state; simulation advances to the next randomly selected agent.

Table EC3: An example of agent actions within a single period. Conditions: 100% Rival resource, 0% Heterogeneous Needs.

7. Pseudocode Simulation Algorithm

The simulation is written in VisualBasic.net. The core algorithm for it, reflecting the overall program logic, is presented in pseudocode below. The key procedures are listed in **bold**. This is a discrete-event model with pseudo-parallelism where for each time-step, every agent is given time to generate the action appropriate for their current behavioral state. An agent's use of its time may or may not involve another agent, such as requesting a resource from another agent or simply generating more of a resource on its own. As noted, the order of agent engagement each period is determined randomly.

```
; ---------------------
; There are three main object categories: agent, resource, resource access structure.
;
Object agent     ;all simulations use 100 agents
     type = [cooperator, reciprocator, free_rider, random]   ;the agent's cooperative strategy
     state = [searching, generating, task_done, idle]        ;the agent's state of activity
     tasks = array (set of 100 tasks the agent must acquire)      ;what resources the agent is seeking
     next_task = integer    ; identifier of the next goal – the next task the agent is trying to acquire
     resources = array (set of 5 task resources the agent currently possesses)
     level = integer        ; current level of resource for the goal agent is seeking
     exchange_limit = [true, false]   ; agents can donate only once per period
     …
end.
;
; ---------------------
; For a given experiment condition, selected rivalry is constant for all resources
Object resource
     type = integer                  ; unique identifier for a resource
     rivalry = real of [0..1]        ; weighted availability of resource to donor after transfer
     level = 50    ; resource level required (constant for all tasks)
end.
;
; ---------------------
; Globally accessible structure that indicates which agent has which resource (and level)
; serves as a global index to quickly locate agents with resources
Object resource_access_structure
     map = array (set of all n resource types in model x set of all 100 agents)
     map.agents(type) = [list of agents that possess resources for a particular type]
 end.
;
; ---------------------
; There are two control components: Experiment_Loop and Run_Replication
Procedure Experiment_Loop    ;overall control for handling the full experiment
     experiment_done = false
     while experiment_done = false do
          Initialize_Experiment     ; handles all population type distributions and parameter settings
          replications = 100        ; for each cell, 100 replications are run
          for reps = 1 to replications do     ;run replications for each experimental condition
```

```
            Initialize_Replication                  ; randomize problem(s) & initial agent resources
            Run_Replication      ; run until all agents are idle (dormant)
            Save_Data            ; retain simulation data
        next reps
    endwhile
end.




;
; --------------------
Procedure Run_Replication      ; for each replication, each agent is given 100 random tasks to achieve
    repeat
     for agent = 1 to 100          ; pseudo-parallelism, time given to each agent
        this_agent = GetRandomAgent (.state ≠ idle)       ; no replacement and not idle
     ; actions depend on the current state of the agent, so check it…
     select
        case: this_agent.state = searching   ; agent has a task, looking to exchange
            Seek_Exchange (this_agent)
        case: this_agent.state = generating  ; agent needs to generate its own resource
            this_agent.level = this_agent.level * 1.1          ; 10% per period generation rate
            if this_agent.level ≥ resource.type.level then    ; this is actually 50 for all types
              this_agent.state = task_done           ; agent has sufficient resources
            else
              this_agent.state = generating   ; agent needs to generate more of this resource
            endif
        case: this_agent.state = task_done  ; sufficient resources acquired for this task
            this_agent.next_task = next (this_agent.tasks) ; get next task, null if none left
            if this_agent. next_task = null then this_agent.state = idle
             else
               if [agent already has this resource] then
                    if this_agent.level ≥ resource.type.level then this_agent.state = task_done
                         else this_agent.state = generating          ; needs to generate more resource
                    endif
                 else    this_agent.state = searching                ; agent needs to search for an exchange
               endif
            endif
     endselect
     next agent
    ; increment period counter, p, calculate data for period (p) including eta = average_donation(p)
    until this.agent = null; replication is done when all agents have completed all 100 of their tasks
end.
;
; ---------------------
; There are two exchange processing procedures: Seek_Exchange, Attempt_Exchange
Procedure Seek_Exchange (this_agent)
    ; agents will look through a list and attempt an exchange. If an exchange occurs, it will not attempt
    ; any other exchanges
    agent_list = GetAgents (resource_access_structure, this_agent.next_task)          ; who has the resource?
    success = false
    while agent_list ≠ null and success ≠ true do  ;work through the list and negotiate exchanges
```

```
        donor_agent = next(agent_list)                ; get next agent in list
        if donor_agent.exhange_limit = false then     ; exchange limit not reached for this agent
            Attempt_Exchange (this_agent, donor_agent, this_agent.next_task)    ; contact the agent
        endif
    endwhile    ; search ends if a successful exchange occurs or all attempts have failed
    if success = true then
        if this_agent.level ≥ resource.type.level then    ; agent acquired sufficient resources for this task
            this_agent.state = task_done
        else
            this_agent.state = generating   ; not quite there, agent needs to generate more resources
        endif
    else  this_agent.state = generating       ; no exchange succeeded, agent generates its own resources
    endif
end.
;
; ---------------------
Procedure Attempt_Exchange (this_agent, donor_agent, task)
    ; types are based on the functional definitions proposed by Kurzban and Houser
    ; and vary by the proportion of their own endowment that they are likely to donate
    select
        case: donor_agent.type = cooperative
            proportion = G[.75,.125]       ; Gaussian with mean = .75 and std. dev. of .125
            donation = proportion * (donor_agent.resources (task))
            if [exchange is made] then
                success = true
                if resource.rivalry > 0 then
                    [reduce donor_agent's resource for this task proportionally]
                endif
            endif
        case: donor_agent.type = free_rider
            proportion = G[.10,.05]        ; Gaussian with mean = .10 and std. dev. of .05
            donation = proportion * (donor_agent.resources (task))
            if [exchange is made] then
                success = true
                if resource.rivalry > 0 then
                    [reduce donor_agent's resource for this task proportionally]]
                endif
            endif
        case: donor_agent.type = reciprocator
            ; eta (p) is the average donation level per agent for period p.
            ; assume the current period is p
            if (eta(p-1)/eta(p-2) ≥ 1 then              ; donation levels are non-decreasing
                [agent acts as a cooperator]
            else                                        ; donation levels are decreasing
                [agent acts as a free-rider]
            endif
        case: donor_agent.type = random
            type = RND[cooperator, free-rider, reciprocator]          ;uniform distribution selection
            [agent acts as type]
    endselect
end.
```

8. Appearances of "Open Source" in the Academic Literature, 1995-2010

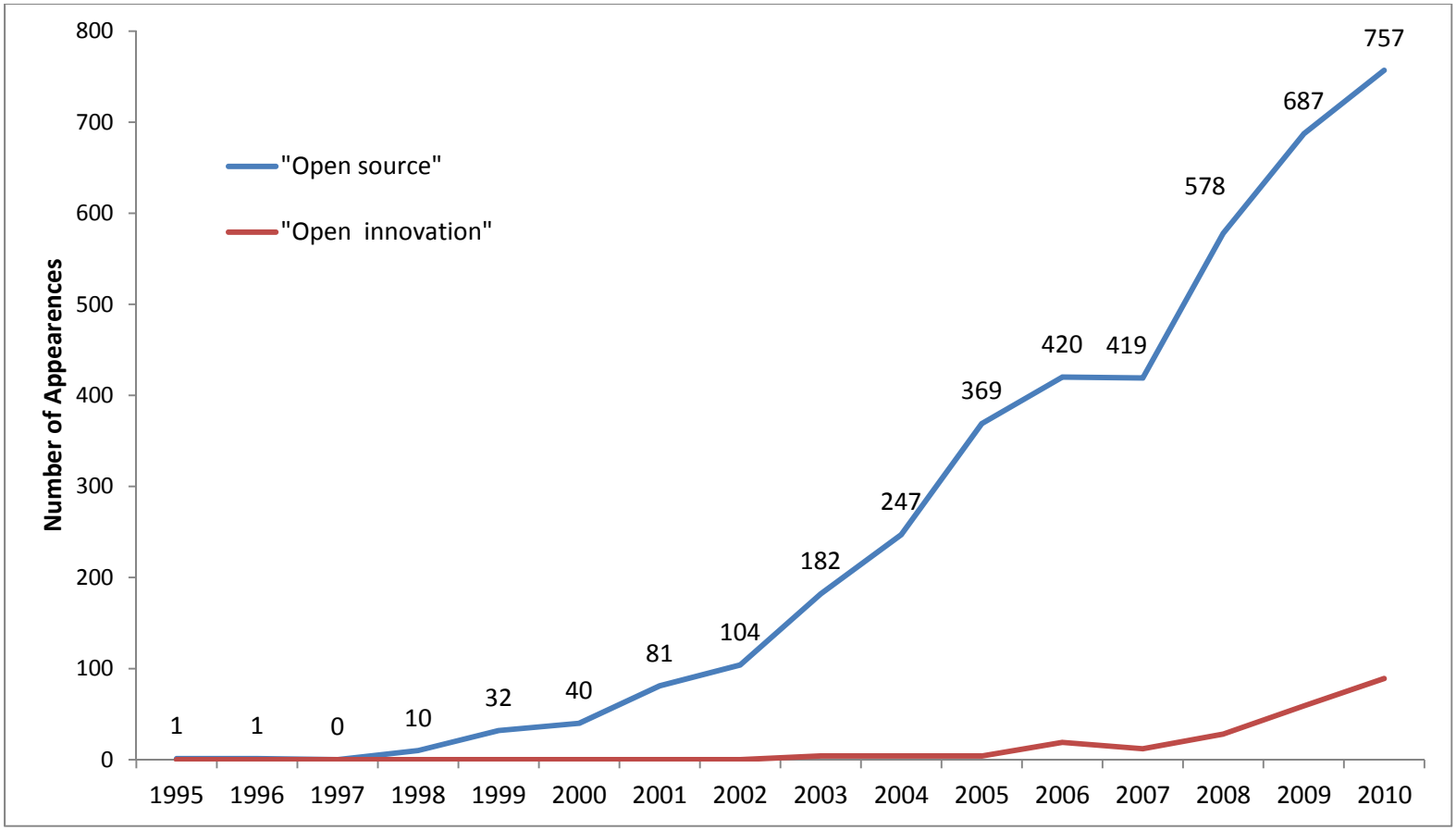

Figure EC5. Appearances of the term "open source" in the academic literature, 1995-2010. The term "open innovation", which has come to acquire a different meaning (Chesbrough 2003), is included for comparison only. The citation count measured appearances of the two words consecutively, as a term. We found a similar pattern when counting appearance of the two words in the same paragraph not consecutively. The count was carried out in April 2011, using the Web of Science database, which provides access to over 10,000 leading journals of science, technology, social sciences, arts and humanities and over 100,000 book-based and journal conference proceedings.

9. Glossary

Seeker – an agent searching for a resource

Source – an agent possessing a resource

Contribution – the successful acquisition of a resource by a seeker from a source

Cooperative Type – An agent's behavioral tendencies concerning cooperation. Can take the values of Cooperator, Reciprocator and Free Riders (Kurzban and Houser 2005).

Self-production – when an agent produces a resource itself, as opposed to acquiring it from others

Simple index – ("who has what") directs the search by identifying agents that possess the desired resource.

Resources – are used in a production of a task. A resource can be thought of as a fundamental building block (e.g., apple seed is a resource for growing apple) or the result of the combination of other resources, as in self-production (e.g., a preexisting module is a resource in writing software).

Tasks – achieved by the employment of resources. Tasks can be goods or services, individual or collective.

Participant – an individual working in an open collaborative setting.

Population Composition – The mix of different cooperative types in a given population. For instance, in the *general human population*) 13% of individuals can be classified as Cooperators, 63% as Reciprocators, 20% as Free Riders, and remaining 4% are inconsistent.

10. References

Benkler, Y. 2002. Coase's Penguin, or, Linux and *the Nature of the Firm. Yale Law Journal* **112** 369-446.

Benkler, Y. 2006. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, New Haven, Conn., Ch. 1.

Benkler, Y., H. Nissenbaum. 2006. Commons-Based Peer Production and Virtue. *J. Polit. Philos.* **14**(4) 394–419.

Chesbrough, H. 2003. *Open Innovation: The New Imperative for Creating and Profiting from Technology*. Harvard Business School Press, Boston.

Dahan, E., H. Mendelson. 2001. An Extreme-Value Model of Concept Testing. *Manag. Sci.* **47**(1) 102-16.

Derman, C., L. Gleser, I. Olkin. 1973. *A Guide to Probability Theory and Application*. Holt, Rinehart and Winston, New York.

Hardin, G. 1968. The Tragedy of the Commons. *Science*(162) 1243-8.

Hong, L., S. E. Page. 2004. Groups of Diverse Problem Solvers Can Outperform Groups of High-Ability Problem Solvers. *Proc. Natl. Acad. Sci. U.S.A.* **101**(46) 16385-9.

Kurzban, R. O., D. Houser. 2005. An Experimental Investigation of Cooperative Types in Human Groups: A Complement to Evolutionary Theory and Simulations. *Proc. Natl. Acad. Sci. U.S.A.* **102**(5) 1803-7.

March, J. G., R. I. Sutton. 1997. Organizational Performance as a Dependent Variable. *Organ. Sci.* **8**(6) 698-705.

Miller, K. D., S.-J. Lin. 2010. Different Truths in Different Worlds. *Organ. Sci.* **21**(1) 97-114.

Olson, M. 1965. *The Logic of Collective Action: Public Goods and the Theory of Groups*. Harvard University Press, Cambridge, Massachusetts.

Terwiesch, C., C. H. Loch. 2004. Collaborative Prototyping and the Pricing of Custom-Designed Products. *Manag. Sci.* **50**(2) 145-58.

Terwiesch, C., Y. Xu. 2008. Innovation Contests, Open Innovation, and Multiagent Problem Solving *Manag. Sci.* **54**(9) 1529-43.