Int. J. Human–Computer Studies (1998) **48**, 9–29

The potential for the evolution of co-operation among web agents

CRISTINA BICCHIERI

Department of Philosophy and Department of Social and Decision Sciences, Carnegie Mellon University, Pittsburgh, PA 15213, USA. email: cb36 + @andrew.cmu.edu

MARTHA E. POLLACK

Department of Computer Science and Intelligent Systems Program, University of Pittsburgh Pittsburgh, PA 15260, USA. email: pollack@cs.pitt.edu.

CARLO ROVELLI

Department of Physics, University of Pittsburgh, Pittsburgh, PA, 15260, USA. email: rovelli + @pitt.edu

IOANNIS TSAMARDINOS

Intelligent Systems Program, University of Pittsburgh, Pittsburgh, PA 15260, USA. email: tsamard@pogo.isp.pitt.edu

In building intelligent network agents, computer scientists may employ a variety of different design strategies, and their design decisions can have a significant effect on the ultimate nature of network interactions. Some agent designs are "co-operative", and populations of agents based on them would be able to interact smoothly, effectively utilizing network resources. In contrast, other agent designs can lead to ineffective and wasteful competition for network resources, resulting in massive bottlenecks and unacceptable access delays. We focus here on a particular design question, the *multiple-access problem*: if an agent seeking a piece of information knows of several sites that have, or might have, that information, how many queries should it issue, and when? We provide a formal analysis that demonstrates the viability of co-operative responses to this question under certain assumptions. We then discuss the limitations of this analysis and present the results of experiments done using a genetic-algorithms approach in which simulated network agents "evolve" co-operative strategies, under less restrictive assumptions than those made in the formal analysis.

1. Introduction

An emerging vision of future massive information networks has them populated by intelligent electronic agents who are capable of understanding a user's specification of information goals, and deciding upon and carrying out the tasks necessary to achieve those goals. These agents will have knowledge of how to locate, access and retrieve information from the network, thereby relieving users of the burden of having this knowledge themselves. Ideally, network agents will not be limited to tasks of pure information retrieval, but will also be capable of filtering email, scheduling meetings and so on, and they will be capable of learning their users' preferences about how such tasks

1071-5819/98/010009 + 21$25.00/0/hc970158

are to be performed. Simple intelligent network agents have already been built in early prototype form (e.g. Etzioni & Weld, 1994; Kautz, Selman, Coen, Ketchpel & Ramming, 1994; Lieberman, 1995; Perkowitz & Etzioni, 1995).

In building intelligent network agents, computer scientists may employ a variety of different design strategies, and their design decisions can have a significant effect on the ultimate nature of network interactions. Some agent designs are "co-operative", and populations of agents based on them would be able to interact smoothly, effectively utilizing network resources. In contrast, other agent designs can lead to ineffective and wasteful competition for network resources, resulting in massive bottlenecks and unacceptable access delays. More generally, populations of agents on a network may be comprised of agents that employ different interaction strategies—for example, some percentage of them may be fully co-operative, another group may be partially co-operative and the rest may be highly unco-operative. Given this, we ask the following questions:

(1) What is the nature of network interactions for different such populations? Under which conditions will particular design strategies be more effective than others?
(2) If agents are *adaptive*, i.e. capable of modifying their behavior in response to current network conditions, what will the population dynamics be? In particular, can adaptation of access strategy ever lead to co-operation, and, if so, under what circumstances?

In our research, we are addressing these questions using a variety of tools from artificial intelligence, distributed systems theory and nonco-operative and evolutionary game theory. Initially, we are focusing on the problem of information retrieval, in particular, on what we call the *multiple-access problem*: if an agent seeking a piece of information knows of several sites that have, or might have, that information, how many queries should it issue, and when? We have defined a set of information-access strategies that vary in the degree to which they are co-operative, and we are examining populations of agents that employ these strategies in varying proportions. The multiple-access problem is, of course, just one among many that must be addressed in the design of an intelligent network agent. Our focus on it is meant to be illustrative of the larger research enterprise, and to support the development of a model that can serve as the foundation for many related questions.

In this paper, we briefly define the multiple-access problem, and sketch a formal analysis of one version of it that uses techniques from game theory and queuing theory. We then discuss some limitations of this analysis, and present the results of experiments done using a genetic-algorithms approach in which simulated network agents "evolve" co-operative strategies, under less restrictive assumptions than those made in the formal analysis.

2. The multiple-access problem

2.1. PROBLEM DEFINITION

As is well known to users of the World Wide Web, the same piece of information is frequently located at multiple sites. For example, someone seeking a local weather forecast can typically retrieve that information from dozens of different sites. Usually,

a human web user will visit one of those sites at a time; if she encounters a delay in accessing the first site she attempts to visit, she may then try a second and so on. A similar situation arises when a web user performs a net search, using a tool like Lycos or Alta Vista, which take as input keywords of interest, and returns as output lists of sites that may contain relevant information. The usual strategy for the human user is to visit the listed sites one at a time, until the sought information is located.

In either scenario, the value of the information sought may be time-dependent, i.e. there may be some cost associated with a delay in getting the information. In some cases, e.g. for ordinary web searches, the cost may simply be the "opportunity cost" of tying up resources that could have been profitably used for other purposes. In other cases, the cost can be much more severe: examples include the use of the web by stock-traders to gain real-time information about financial markets, or its use by military commanders to gain real-time information about situation development. This possibility of significant costs due to delayed information access raises an interesting question, which we call *the multiple-access problem*: if an agent seeking a piece of information knows of several sites that have, or might have, that information, how many queries should it issue, and when? Should several queries be issued simultaneously, rather than in the sequential fashion that is typical today? This question becomes particularly relevant once artificial agents are tasked with the job of information retrieval, because such agents could easily spawn several sub-processes (subagents) to access multiple sites simultaneously—a task that is more burdensome for the human user.

If there is no extrinsic cost associated with accessing a site, a plan to simultaneously issue multiple queries might appear to be a good one, increasing the likelihood of getting the information quickly, without incurring any cost. However, if multiple agents form such plans, a cost may in fact be incurred, as the result may be a highly overloaded system, in which all agents, on average, do worse.

2.2. A SIMPLIFIED CASE

We begin by considering a simplified version of the problem, in which there are exactly two agents and two sites that the agents both know to have the information sought. We assume that if both agents attempt to access the same site, there is an equal chance that either one will get there first. We assume further that both agents have the same, very simple payoff function: the utility of being the first to access one or both of the sites is h while the utility of being second to access both is l, under the constraint that $h > l$. Finally, we assume that there is no extrinsic cost associated to the access (i.e. all agents can submit information requests "for free"), that all hosts are deemed equally reliable, and that all this information is common knowledge. With these assumptions, the decision that each agent must make can be modeled as a strategic-form game,† as shown in Figure 1. For example, if both agents adopt the strategy of accessing one site, then there is a 50% chance that they will go to different sites (in which case each will receive a reward of h), and there is a 50% chance that they will go to the same site. In the latter case, for each agent, there is a 50% chance of getting there first (and being rewarded h),

† For details of modeling games in strategic form, see Osborne and Rubinstein (1994). Note that, as is standard in the strategic-form notation, the first payoff in every cell belongs to the row agent, and the second to the column agent.

	Access one	Access both
Access one	$0.75h + 0.25l$, $0.75h + 0.25l$	$0.5h + 0.5l$, h
Access both	h, $0.5h + 0.5l$	$0.75h + 0.25l$, $0.75h + 0.25l$

FIGURE 1.

	Both access one	One accesses one One accesses both	Both access both
Access one	$7/12h + 1/3m + 1/12l$	$5/12h + 5/12m + 1/6l$	$1/3h + 1/3m + 1/3l$
Access both	$7/8h + 1/8m$	$2/3h + 1/3m$	$5/9h + 3/9m + 1/9l$

FIGURE 2.

and a 50% chance of getting there second (and being reward l). Thus, each agent's expected utility in this case is $0.5h + (0.5 \times 0.5h) + (0.5 \times 0.5l) = 0.75h + 0.25l$. Similar analyses can be given for the other cells.

What we see is that each agent has a dominant strategy, to access both sites, and if both agents choose their dominant strategies the result is a Nash equilibrium that is also Pareto-optimal. This means, in brief, that (i) each agent prefers to access both sites regardless of what the other does (it has a dominant strategy); (ii) when an agent believes that the other agent will access both sites, then it also prefers to access both sites itself (both accessing both sites is a Nash equilibrium) and (iii) there is no other combination of actions that leads to a better outcome for one agent without the other agent doing worse (both accessing both sites is Pareto-optimal). Thus, for the two-agent, two data-site case, under the assumptions we made above, there is no need for agents to explicitly co-operate with each other: by simply choosing their dominant strategies, they achieve a result that is as good as if they were co-operating.

This fact changes, however, as we introduce more agents. For example, in Figure 2, we show the strategic-form game for three agents and two sites, under the assumption that being the first to access either site results in a payoff of h, being second at one site and second or third at another results in a payoff of m, and being third at both sites results in a payoff of l, with the constraint that $h > m > l$. Note that Figure 2 lists the expected payoffs only for the row player. The expected payoffs for the other players are symmetrical. Each agent still has a dominant strategy, to access both sites, and the situation in which everyone accesses both sites is the unique Nash equilibrium of the game. To see this point more clearly, suppose that the values of h, m, and l are, respectively, 10, 5 and 1. We then obtain the matrix shown in Figure 3. It is clear in Figure 3 that regardless of what the other agents choose to do, the row agent will do best by accessing both sites. We again illustrate the calculations used in constructing these tables with a single cell, the upper-left, denoting the case in which all three agents choose to access one site. Then there is a 25% chance that they will all pick the same site, a 50% chance that the row agent and exactly one other agent will pick the same site, and a 25% chance that the row

	Both access one	One accesses one One accesses both	Both access both
Access one	7.58	6.41	5.33
Access both	9.16	7.58	7.33

FIGURE 3.

agent will pick a site uniquely, with the other agents both picking the other site. In the first case, the agent has a 1/3 chance of having its query responded to first, and receiving a reward of h, a 1/3 chance of having its query responded to second, and receiving a reward of m, and a 1/3 chance of having its query responded to third and receiving a reward of l. In the second case, it has a 1/2 chance of receiving h, and a 1/2 chance of receiving l. Finally, in the third case, it receives h for sure. Thus, in sum, when all three agents choose to access a single site, each agent's expected utility is $1/4 \times (1/3h + 1/3m + 1/3l) + 1/2 \times (1/2h + 1/2m) + 1/4h = 7/12h + 1/3m + 1/12l$. In performing similar calculations for other cells, note that we assume that an agent gets a reward for the information received only once: if it is first at one site and second at another, for example, it receives h, not $h + m$.

As noted above, the equilibrium for this example is the lower-right cell, denoting the case in which all three agents access both sites. This equilibrium, however, is deficient, i.e. Pareto-suboptimal, since all would do much better were they to access only one site. We thus have a classical Prisoner's Dilemma. Agents would stand to gain were they capable of co-operating with each other and submitting queries to one site only. In fact, as the number of agents increases relative to the number of databases, the spread increases between the co-operative, Pareto-optimal outcome, in which all agents access one database, and the defective outcome, in which they all access both databases. The more agents there are in the system, the greater would be the gain from co-operation.

But how could such co-operation arise? In a one-shot encounter, there is no reason for an agent to be co-operative: if one expected the other agents to access only one site, the rational choice would be to access both. The situation changes, however, if agents interact repeatedly with each other. Traditional game-theoretic models show that, when agents assess a sufficiently high probability of a future encounter, co-operative behavior can occur, since the gains from continuing co-operation far outweigh the temporary gains of defection. Such models, however, are not fully applicable to the case at hand. First, they assume that agents are not anonymous, but are able to identify one another, and thus to recognize and punish defectors. Second, these models assume that agents are fully rational, and that they are perfect calculators without any time or computational constraints. In fact, real agents (including web agents) are only boundedly rational: they have limited knowledge of their environment and of other players, and they are computationally limited.

3. The effect of limited knowledge

Perhaps surprisingly, the fact that web agents have limited knowledge can be used to demonstrate that, at least in an idealized case, co-operation may occur among agents

that are designed according to principles of rationality. The argument hinges on agents' lack of complete knowledge about the state of their environment.

Specifically, assume that there are m sites and n agents, with $n > m$, and that each agent can submit a query at any time. We denote the average frequency at which queries are sent as F, and assume that F is the same for all agents. We assume that agents' queries are independent of each other and are sent to sites in random fashion, so that on average queries are uniformly distributed among sites. That is, agents do not have preferences over sites, not do they cluster about particular sites for any other reason than statistical fluctuations. Further, each site has a response time of ΔT, which is the same for all sites. We have chosen to model sites as using a queuing, rather than a time-sliced approach to query management. At any time, a site can be either free or busy. If it is free and it receives a query, it becomes busy and stays busy for time ΔT, after which it sends an answer. If the site is busy when the query arrives, the query will wait in line until the site is free. Incoming queries will form a queue: for example, if there are five prior queries, the sixth query will be answered after $6\Delta T$. The maximum frequency at which a site replies is $1/\Delta T$.

As in the discussion above, we consider only the two most "extreme" strategies, full co-operation (C), in which each agent sends one query to a randomly selected site and waits for a response and defection (D), in which each agent sends queries to all sites at once. (To simplify the analysis, we assume that all sites have the information sought.) We denote by n_c the number of agents using strategy C (the co-operators), and by n_d the number of agents using strategy D (the defectors). Which strategy is more successful depends on the average waiting time experienced by agents employing either of them. We use T_c and T_d to represent the average waiting time for co-operators and defectors, respectively. Then n_cF is the total number of queries sent by co-operators in each time period, and n_dFm is the total number of queries sent by defectors in each time period. The average number of queries per period received by a site will thus be

$$\frac{n_cF + n_dFm}{m}.$$

We refer to the above number as F_{DB}, the frequency of queries received by a site. Its inverse $(1/F_{DB})$ is the average time between two queries at a site. There are then three possible situations that can arise:

Case (i): $1/F_{DB} > \Delta T$.

In case (i), the average delay between queries is greater than the time it takes for a site to respond to a query, and hence no queueing will occur. Instead, queries will be answered without delay. Thus, in this case, there is no particular advantage to either co-operation or defection because the average waiting time is the same for both types of agent, i.e., $T_c = T_d = \Delta T$.

Case (ii): $1/F_{DB} < \Delta T$.

In case (ii), queries arrive more rapidly than they can be processed and the system becomes increasingly overloaded. The waiting time for all queries increases steadily in time, and eventually becomes infinite. Thus, the average waiting time (averaged over

a very long time span) is infinite, and there is no particular advantage to either co-operation or defection. At least in the limit the entire system collapses, and no queries are answered.

Case (iii). $1/F_{DB} \prec\approx \Delta T$.

In case (iii), the average wait time between queries submitted is close to, but lower than, the response time at each site. We learn from queuing theory that in such a situation queues will form. Each queue has a length that fluctuates randomly in time. In particular, there are always time periods in which a queue has zero length. This follows from a simple consideration: if the queue had finite length at all times, a site would be answering queries at maximal frequency, and therefore it would be answering more queries than the queries it receives! More precisely, one can show that the fraction of time p during which a queue has zero length is $p = 1 - (F_{DB}\Delta T)^{-1}$.

Since sites are independent and are randomly chosen by the agents, the fluctuations in the length of their queues are independent as well. Therefore, at any moment the various sites will, in general, have queues of different length and, on average, there will be mp sites with zero length. In the limit of large m (and n), there will always be a finite number of sites having a queue of zero length.

It follows from this fact that defectors will achieve a better performance than co-operators. This can be shown as follows. If L is the average length of the queues, then the average waiting time for a co-operator (who sends a single query) is $T = L\,\Delta T$. A defector, on the other hand, sends one query to each of the m sites. It will receive m replies which, on average, will arrive within time $T = L\,\Delta T$. However, these replies will arrive well spread in time, as each queue is fluctuating. Now, the key issue is that the relevant time for the agent is not the response time averaged over the m queries sent, but it is the time of the first response obtained. We noticed before that in the limit of large m (and n), there will always be sites with a queue of zero length. Thus, in the limit the defector will obtain an answer without waiting, while the co-operator will have to wait a time T. Even away from this limit, however, it is clear that the wide fluctuations of the queues' length will strongly favor the defectors that try many sites at the same time.

Given this analysis, we can see that, if there is no cost associated with sending a query, then, at least under our current assumptions, agents should be designed to use a defector's strategy, since co-operation and defection are equivalent in cases (i) and (ii), while defecting pays in case (iii). However, consistent with the argument we have given in the previous section, we can easily show that overall the population performs better when all agents co-operate. To see this, recall that in case (iii), the frequency of queries received by any site is roughly equal to $1/\Delta T$; thus, we can derive the *critical submission frequency of queries sent by an agent* as follows:

$$1/F_{DB} \approx \Delta T,$$

$$F_{DB} \approx 1/\Delta T,$$

$$\frac{n_c F + n_d F m}{m} \approx 1/\Delta T,$$

$$F\left(\frac{n_c + n_d m}{m}\right) \approx 1/\Delta T,$$

$$F \approx \frac{m}{\Delta T(n_c + n_d m)}.$$

We use the symbols F_{crit} to denote this value, i.e.

$$F_{crit} \approx \frac{m}{\Delta T(n_c + n_d m)}.$$

F_{crit} corresponds to the critical value of F_{DB}, the frequency of queries received by any site. The critical submission frequency is the point at which the system shifts state: when F_{DB} is less than F_{crit}, the system is unloaded, but as it approaches and then exceeds F_{crit}, the system will shift to a loaded state, and the average waiting time will go to infinity. The exact value of F_{crit} will depend upon the proportion of agents who are co-operating and the proportion of agents who are defecting. If the agents are all co-operators, i.e. $n_d = 0$, the critical frequency is

$$F_{crit} \approx \frac{m}{\Delta T(n_c)},$$

whereas if we assume that the population only contains defectors, i.e. $n_c = 0$, the critical frequency becomes

$$F_{crit} \approx \frac{1}{\Delta T(n_d)}.$$

In a homogeneous population of co-operators, the critical frequency is higher by a factor of m. That is, the population of co-operators can submit queries m times as frequently as a homogeneous population of defectors before approaching the critical frequency and having the delays begin to increase towards infinity. This result is obvious, since a population of defectors submits m times as many queries per time period as a population of co-operators. Thus, a homogeneous population of co-operators will do much better than a homogeneous population of defectors. When the population is mixed, the critical frequency lies between the critical frequencies of the two homogeneous populations. Before the critical frequency is reached, however, defectors will do much better than co-operators. All agents will be queuing and waiting for an answer, but those who send more queries have a higher probability of getting a quicker answer. The population as a whole would do better if it consisted of co-operators, yet agents should, it appears, be designed to be defectors.

4. The co-operator's advantage

Suppose, however, that query submission has a cost. Minimally, there is the opportunity cost of using resources that might be employed elsewhere; moreover, each answered query might cost the agent a fixed amount of money, and so might connection time. Even if these costs are rather insignificant, they are enough to tilt the scale in favor of co-operation. This is because, if queries have a cost, then in cases (i) and (ii), it will pay to

be a co-operator rather than a defector: average waiting time is the same for both strategies, but the costs of defecting are higher.

As before, we assume that agents send queries to sites in a random way, and that queries are uniformly distributed in time with average frequency F. More precisely, they are distributed in time with a uniform probability distribution. Note, again, that assuming a uniform probability distribution in time of queries does not imply that queries are sent at constant time intervals from each other, or that precisely the same number of queries is sent in each unit of time. The number of queries generated in a given time interval will in general fluctuate from one time interval to another—the relative fluctuation will go to zero only in the limit of long time intervals. We also keep assuming that sites are homogeneous in that they all have the same response time ΔT, and they receive queries with the same frequency.

Once query submission is seen to have a cost, the only situation in which it pays to be a defector is case (iii). Yet agents do not know which situation currently prevails: since they do not know what percentage of agents are co-operating and defecting, they cannot compute the critical frequency. What we show below is that, under certain assumptions, it is extremely unlikely that the environment will be in a case (iii) configuration, and thus, agents should be designed to presume they are either in case (i) or (ii), and act co-operatively.

To support this analysis, we introduce the notion of average waiting time T which, using standard queuing theory can be computed as a function of the frequency at which queries are received at each site (F_{DB}) (Schwartz, 1987, p. 57, formula 2-63):

$$T = \Delta T \frac{2 - F_{DB}\,\Delta T}{2 - 2F_{DB}\,\Delta T}.$$

When T is "essentially zero", the environment is in an unsaturated state [case (i)]. We say "essentially zero" because no response in a network is ever truly instantaneous: there is always some delay. We therefore introduce the notion of an agent's minimum significant waiting time—this is the smallest unit of time that an agent will count. In other words, if an agent's minimum significant waiting time is t, whenever it gets a response in less than t, it will consider that response to be essentially immediate: it will say that there was no wait. Given the usual delays in network transmission, it is reasonable to consider t as being on the order of one second. Any waiting time less than one second can be taken to cost zero, whereas higher waiting times have a proportionally higher cost. Note that the time scale of t is typically much higher than the response time of each site, ΔT; this, we will assume, is on the order of microseconds.

When the average waiting time is lower than t, both co-operators and defectors get an answer in less than one second on average; therefore, there is no advantage to being a defector. And, in fact, given our assumption that all queries have a cost associated with them, there is a disadvantage to being a defector, because a defecting agent will submit more queries, thereby incurring a higher cost, without getting the benefit of quicker response.

To be a defector is better only when the average waiting time is larger than t (and less than infinity; we will return to this below.) To compute the values of F_{DB} for which the average waiting time T is larger than t, we solve the equation

$$t < \Delta T \frac{2 - F_{DB}\,\Delta T}{2 - 2F_{DB}\,\Delta T}$$

with respect to F_{DB}. The solution is

$$F_{DB} > \frac{1}{\Delta T}\frac{2t/\Delta T - 2}{2t/\Delta T - 1} = \frac{1}{\Delta T}\frac{1 - \Delta T/t}{1 - \Delta T/2t}.$$

For $t \gg \Delta T$, this can be written as

$$F_{DB} > \left(1 - \frac{\Delta T}{2t}\right)\frac{1}{\Delta T}.$$

Recall now that $1/\Delta T$ is the critical frequency of receiving queries for a site. If the frequency F_{DB} is higher than $1/\Delta T$, then the average waiting time is infinite. In this case there is no advantage in being a defector either. In fact, the same comment that was made earlier still applies: the defector is at a disadvantage, given the extra costs associated with submitting more queries. Therefore, the region in which there are non-negligible and finite waiting times, and thus in which being a defector pays, is the region

$$\frac{1}{\Delta T} > F_{DB} > \left(1 - \frac{\Delta T}{2t}\right)\frac{1}{\Delta T}.$$

We denote this region as the critical region. If $t \gg \Delta T$, then this critical region is very small. For instance, assuming $t = 1$ s and $\Delta T = 1$ μs, as above, we have that the critical region is

$$\frac{1}{\Delta T} > F_{DB} > 0.9999995\,\frac{1}{\Delta T}.$$

Namely, the critical region covers only 1 part in 2 millions of the range of the frequencies below the critical frequency, and it is only within this region that it pays to be a defector! The intuitive meaning of this result is that if network agents interact with the system at time scales much larger than the answering time ΔT, then the system is almost always either completely congested or free. Thus, since each agent does not know which portion of the region it is currently in, it is distinctly to its disadvantage to be designed to be a defector. This is illustrated in Figure 4, which graphs the average waiting time T, as a function of F_{DB}, using the formula defined above. For simplicity, we take $\Delta T = 1$, so the function becomes $T = (2 - F_{DB}/2 - 2F_{DB})$. The regions corresponding to cases (i), (ii) and (iii) above are shown, under the assumption that the minimum significant waiting time $t = 100\,\Delta T$. As is readily apparent in the figure, under such assumptions, an agent should not assume it is in region (iii), the only region in which defecting pays.

5. Towards less restrictive assumptions

5.1. GENERALIZED ACCESS STRATEGIES

Anyone who uses the World Wide Web knows that in fact finite delays are extremely common — more common than zero or infinite delays. Our formal analysis depends on a number of very strong assumptions and simplifications. In particular, we assumed that the average submission rate does not fluctuate (although, as explained above, the precise number of queries submitted during any time interval may vary). In practice, the

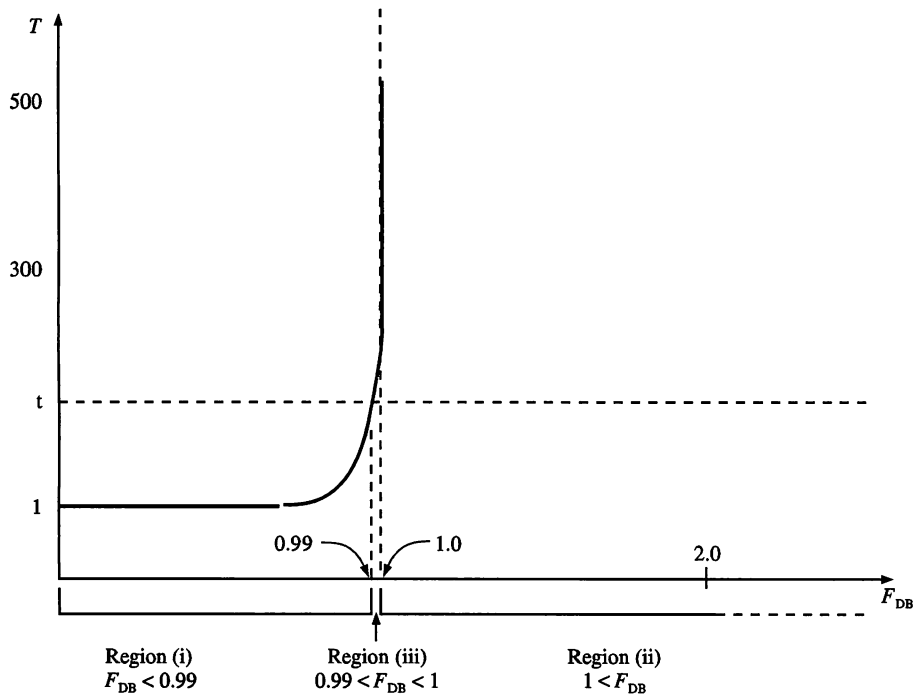

FIGURE 4.

submission rate itself varies, and queries are not distributed uniformly to all sites, but rather tend to cluster to particular sites. The average number of queries arriving at any site during a fixed time interval will vary—and as it does, the system will fluctuate between case (i) and case (ii) situations, and finite queues will form.

In the case in which queries are not uniformly distributed, it makes sense to consider a wider range of strategies for information access, i.e. not to restrict the analysis to the extremes of full co-operation, in which agents submit an information request to only one site, and full defection, in which they immediately submit information requests to all potential sites. Agents have a much larger (in fact, infinite) set of possible access strategies, the space of which we can illustrate with a few examples, for which we provide pseudo-code in Figure 5.

The most co-operative strategy involves accessing only a single site at a time. In one version of this strategy ("unconditional co-operation"), the agent picks a single site to which to issue a query, and then waits indefinitely for a response. In another ("impatient co-operation"), the agent cycles through the sites known to have the information in question, waiting for some fixed time period before retracting the query and issuing a new query to a different site. Somewhat less co-operative is the "conditional co-operation" strategy, in which the agent begins by issuing a query only to one site, but, if it fails to receive a response within some fixed time, issues another query to a second site without retracting the first query and so on. As time passes, this agent will have queries

Unconditional Co-operation

```
i := pick-db;
issue-query-to (i);
while data not
  retrieved
    wait.
```

Impatient co-operation

```
i := pick-db;
issue-query-to (i);
t := 0;
while data not
  retrieved
  begin
  while t < T
    if data retrieved
    then
    exit
    else t := t + 1;
  retract-query-from (i);
  i := pick-db;
  issue-query-to (i);
  t := 0;
  end.
```

Conditional co-operation

```
i := pick-db;
issue-query-to (i);
t := 0;
while data not retrieved
  begin
  while t < T
  if data retrieved then
        exit
    else t := t + 1;
  if not all db's queried
then
      begin
      i := pick-db;
      issue-query-to (i);
      end;
t := 0;
end.
```

Grim co-operation

```
i := pick-db;
issue-query-to (i)
t := 0;
while t < T
  if data retrieved then
      exit
        else t := t + 1;
for all other db's j
  issue-query-to (j);
while data not retrieved
  wait.
```

Defection

```
for all db's j
  issue-query-to (j);
while data not
  retrieved
    wait.
```

FIGURE 5.

outstanding at more and more sites. A somewhat more extreme version of this approach is "grim co-operation", in which the agent begins by being co-operative, issuing a query only to one site, but if an answer is not received within a fixed time period, it immediately issues queries to all remaining known sites. The least co-operative strategy is "defection", which involves immediately issuing requests to all sites known to have the required information.

Given a set of such strategies, one would like to be able to analyze the performance of populations of agents using them. However, a clean formal analysis, using the tools of game theory, is quite difficult. In part, this is due to the complex behavior that results from the larger space of strategies, and in part, to other limitations of traditional game-theoretic tools, including difficulties in modeling asynchronous interactions and situations in which an agent must infer the strategies used by other agents. Similarly, there are difficulties in directly applying traditional queuing theory to the more complex case, because it makes at least two strong assumptions that do not apply to the problem we are considering: (i) agents know the size of all queues, and (ii) agents enter (or, in our problem, submit a query to) a single queue at a time. We are currently working to extend existing techniques, so that we can develop formal models of the general case, in which clustering of queries can occur, and in which it therefore makes sense for agent designers to consider a much wider range of access strategies. In the interim, we are approaching the general case experimentally, applying a genetic-algorithms approach in a simulator we designed, to explore the possibility that co-operation could evolve among web agents.

5.2. A WEB-AGENT SIMULATION SYSTEM

We designed and built a simple discrete-event simulator to model the web-agent scenario. Our simulator is written in the C++ programming language, using the

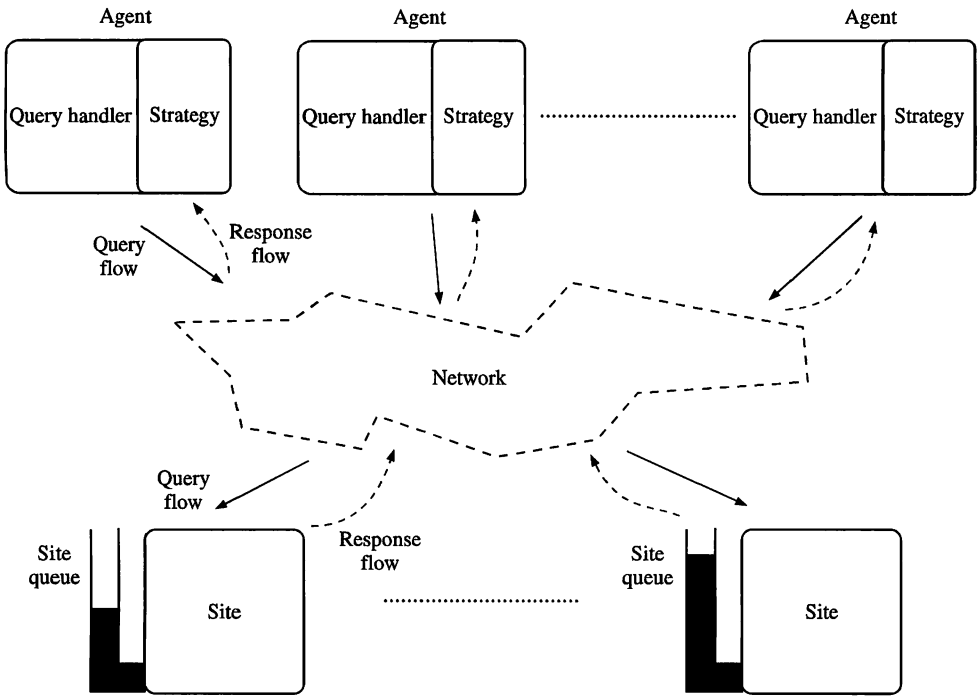

FIGURE 6. The simulator.

C+ +Sim library, and is installed on a Sun Sparc running SunOS. Its object-oriented design allows easy modification, so that users can readily add new strategies with which they would like to experiment. The architecture of the simulator is depicted in Figure 6.

As can be seen, the simulator has three key kinds of object: agents, sites and the network.† Agents submit queries to particular sites, delays in the network may occur, and eventually the queries are received by the sites where they enter queues to await processing. After a processing delay, responses are then returned to the agents, who accumulate a "score": the more quickly the response was received, the greater the score awarded. After receipt of a response to a query from one site, all copies of that query at other sites are deleted. Currently, we assume that all sites have the same information: we are thus modeling a situation more like that of seeking the local weather than that of using a net search. The simulator collects various data during each experiment; an example of the output produced is shown in Figure 7.

† Only a simplified version of the network object has so far been implemented.

```
Snapshot at time : 150

Simulation duration                          : 150
Load (mean of exponential distribution)      :  3.00
Site wait at least                           :  0.10 stu
Sites additionally wait on average           :  0.40 stu
Reward in performance function is            : 10.00
Threshold in performance function is         :  1.20
Number of agents                             :  30
        30 of type 'GA strategy'
Number of sites                              :  10
        Total queries generated              : 1431
        Average score per agent              :  46.8054
        Average score per queries generated  :  0.981246
        Total queries answered               : 1274
        Average queue length                 :  73.4427
```

FIGURE 7. Typical output of the simulator.

The simulator permits the user to set a number of parameters for each experiment. These include the following.

(1) *Simulation parameters*
   (a) The duration of the simulation, which is measured in simulation time units (STUs).
   (b) The number of agents.
   (c) The number of sites.

(2) *Agent parameters*
   (a) The submission-rate distribution, representing the delay between submission of subsequent distinct queries. This is an exponential probability distribution, for which the user specifies the mean. It is important to distinguish between two things: (i) the delay between submission of "distinct" queries, and (ii) the delay between submission of multiple copies of the same query (for the less-than-fully-co-operative strategies). This parameter controls only the former. The latter is controlled by a subparameter of the query-submission strategy, described immediately below.
   (b) The query-submission strategy. The user specifies the strategy to be used, along with defining parameters, such as the time to wait before submitting copies of the query to additional sites. Four of the strategies already introduced (Unconditional Co-operation, Conditional Co-operation, Grim Co-operation, and Defection) can be specified, as can variations of them, as described below. We have not yet implemented a query-retraction mechanism, except after successful receipt of the requested information, and thus the Impatient Co-operation strategy cannot yet be specified.
   (c) The performance function, which defines the score that the agent will be awarded after receiving an answer to the query. So far, we have implemented this as a time-dependent exponential decay, as shown in Figure 8.

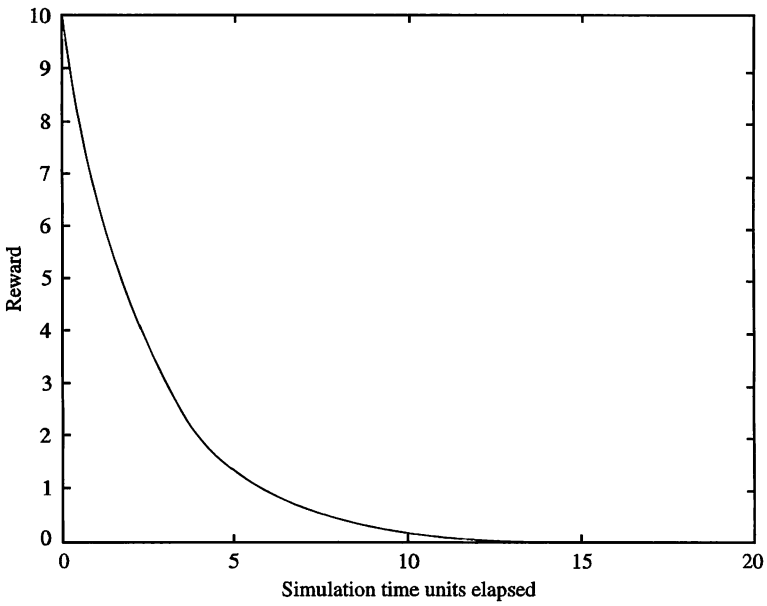

FIGURE 8. The exponential decay performance function.

(3) *Site parameter*
   (a) The processing-time function, which represents the delay at a site due to processing time. As currently implemented, the user specifies two values: a constant minimum processing delay, and the mean of an exponential probability distribution representing additional delay. The total processing time for any query at site S is $y + x$ STUs, where y is the constant minimum delay and x is a value drawn from the distribution.

5.3. EVOLVING ACCESS STRATEGIES

Initially, we conducted exploratory studies aimed at finding "optimal" strategies for populations of simulated agents to use, given certain kinds of environments (e.g. number of agents, number of sites, average delay, etc.) We were unable to find obvious regularities, and hence decided to adopt a genetic-algorithms approach (Holland, 1975), allowing the optimal strategies to evolve. Genetic algorithms (GAs) are algorithms that control search in a given search space by applying Darwinian ideas of natural selection. A fitness function is defined, and the GAs attempt to find a point in the search space that optimizes the fitness function. The connection with Darwinian theory comes from the fact that this class of algorithms begins with an initial population of individuals that represent points in the search space and has them "breed" offspring that share their characteristics. Mimicking natural selection, the probability that an individual's characteristics will survive in future generations is determined by its level of fitness. In

addition, GAs often employ mechanisms analogous to biological mutation and crossover to introduce some random element into the search, and thus avoid getting stuck at local maxima.

For our studies, the search space consisted of query-submission strategies that could be employed by network agents, and the fitness function was the average reward received by all agents. We represent a strategy as a list of up to five rules, where each rule is a pair consisting of an integer j and a real number r. The first value (j) dictates the number of sites to which the query should be sent, and the second value (r) dictates the delay, in STUs, before the agent should transmit the query to additional sites. Once all the rules in the strategy have been applied, they are re-applied, starting from the first rule again, until all the sites have received queries or a response has been obtained. For example, an agent using the strategy $\langle(5, 3.0), (2, 4.0)\rangle$ would first submit a query to five sites, then wait for 3.0 STUs; if a response had not been received, it would next submit to two more sites, and wait 4.0 additional STUs; if a response had still not been received, it would submit to five more sites and so on. Assuming that the number of sites is s, the strategies defined earlier can be encoded as follows.

- Unconditional Co-operation: $\langle(1, \infty)\rangle$.
- Conditional Co-operation: $\langle(1, T)\rangle$, where T is the time to wait until sending the query to an additional site.
- Grim Co-operation: $\langle(1, T), (s, \infty)\rangle$, where, again, T is the time to wait until broadcasting to all sites.
- Defection: $\langle(s, \infty)\rangle$.

As should be clear, a much wider variety of strategies can be represented as well.

Mutation of a strategy is achieved in this model in three ways. The first possibility is to randomly select a rule in the strategy, and change either its j value (by adding or subtracting 1) or its r value (by drawing a number from a Gaussian distribution with mean 0, and adding this number to the previous value for r). The second possibility for mutation is to delete a randomly selected rule from the strategy. The third possibility is to add a new, randomly generated rule to the end of the strategy. We also used uniform crossover in generating offspring: this means that each rule in an offspring is chosen equiprobably from one of its parents. These processes are depicted in Figure 9.

The experiments were conducted as follows. We began with 50 randomly generated strategies, and evaluated each over a period of 150 STUs for a population of 30 agents and 10 sites, which were uniform, in that all the agents and sites had the same parameter settings. The GA then produced a new population of 50 offspring. The strategies in generation $i + 1$ were formed by picking strategies from generation i, with probability proportional to their fitness value, and then applying the crossover and mutation operators with a fixed probability. This process was repeated for 100 generations, at the end of which we selected the strategies that had the highest overall fitness value.

We ran experiments using three different values for agent submission-rate distribution: in one ("high load"), the mean of the distribution was 1.0, in the second ("medium load") it was 3.0 and in the third ("low load") it was 9.0. Recall that the submission-rate distribution defines the period between submission of distinct queries. Thus, in the high-load settings, an average of 30 distinct queries were being submitted each STU (30 agents each submit a distinct query on average every 1.0 STU). The total number of actual

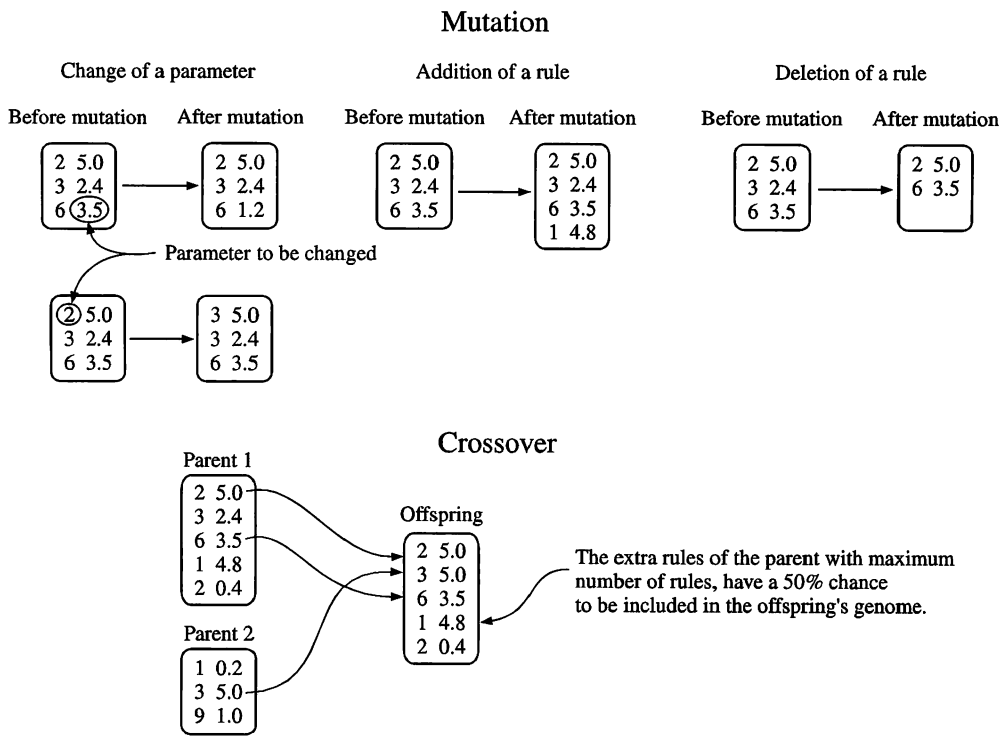

FIGURE 9. The GA's operators.

queries (copies of queries) going through the network depends upon the strategies being used by the agents. For example, if all agents are unconditionally co-operative, the total number of queries will average 30/STU, while if they are defectors, the total number will be 300/STU, because each agent sends 10 copies of its message. Similar calculations apply to the medium and low load settings (averages of 10 and 3.3 distinct queries per STU, respectively).

For the site-processing time function, we set y to 0.1 and x to 0.4, so each site, on average, takes 0.5 STUs to process a query. Thus, the total capacity of the 10 sites is an average of 20 queries per STU. Obviously, then, saturation will occur in the high-load situation even when all the agents are using co-operative strategies. The low-load situation does not result in saturation when the agents are co-operative, but can when they use less co-operative strategies.

For the agent performance function, we used the function shown above in Figure 8. Because all sites were assumed to be the same with respect both to processing speed and quality of information returned, agents made random selections about where to submit queries.

Our results are summarized in Figures 10–12, which plot the average fitness of each generation for each of the three loads, and in Figure 13, which lists the five best strategies

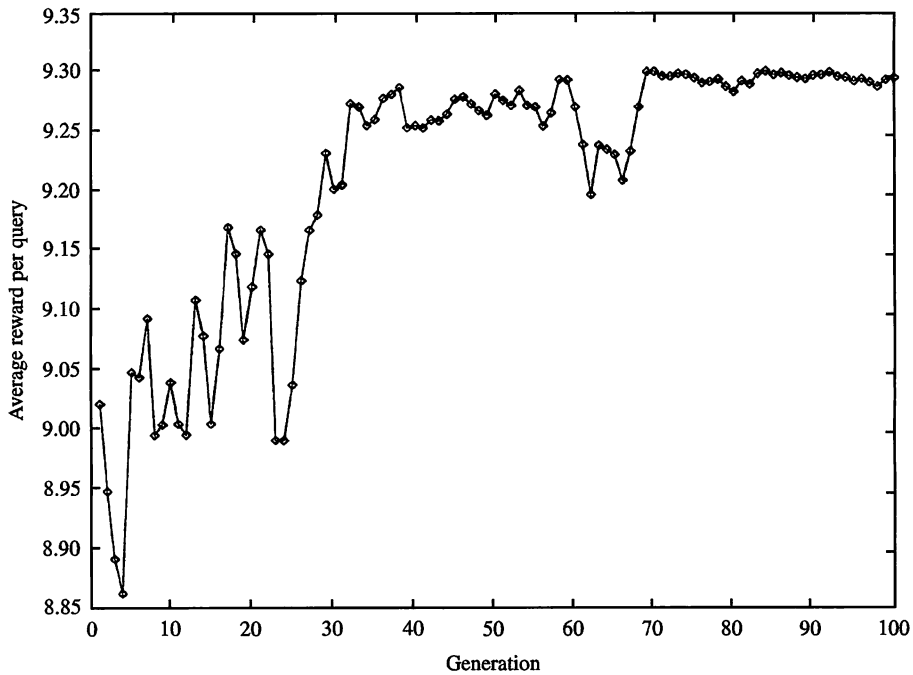

FIGURE 10. The GA average fitness for low load setting.

evolved for each load. Notice that in all cases, the agents evolve strategies that perform increasingly well, although not necessarily strategies that are increasingly co-operative. In the low-load situation, populations of agents that perform well tend to send multiple queries to sites: in all of the top five strategies, the agents begin by sending copies of their query to eight out of the 10 sites.† By trying a lot of sites they increase the probability of receiving an answer from at least one of them quickly, while the low load ensures that all this traffic will not overload the sites. Relatively unco-operative behavior therefore results in a good overall performance when there is a plethora of resources. In contrast, when the load is high, populations of agents with highly co-operative behavior do best. In the high-load settings, what evolved were strategies that send two copies of the same query. Not surprisingly, for medium-load situations, the strategies that we evolved are in between: the agents send the same copy of a query to half of the sites available to them.

† Notice that in all cases, the delay before sending to additional sites is quite long: for instance, in the high-load case, the top five strategies all involve a delay of 99.3 STUs before submitting additional queries. Given that an entire experiment was 150 STUs, we see that in general the first term in each rule dominates: what matters most is the number of sites to which the queries are initially sent.

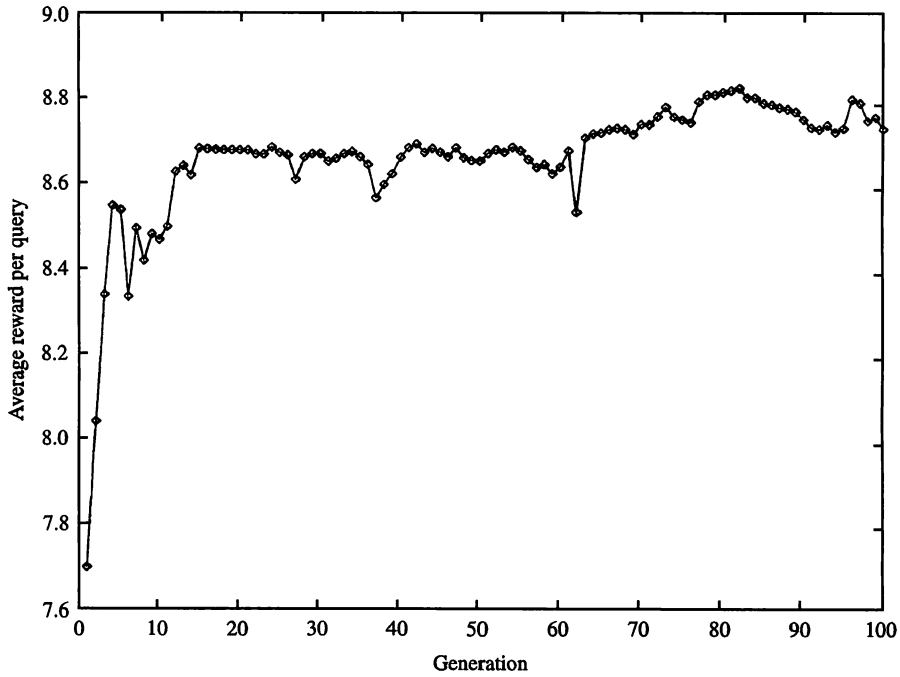

FIGURE 11. The GA average fitness for medium-load setting.

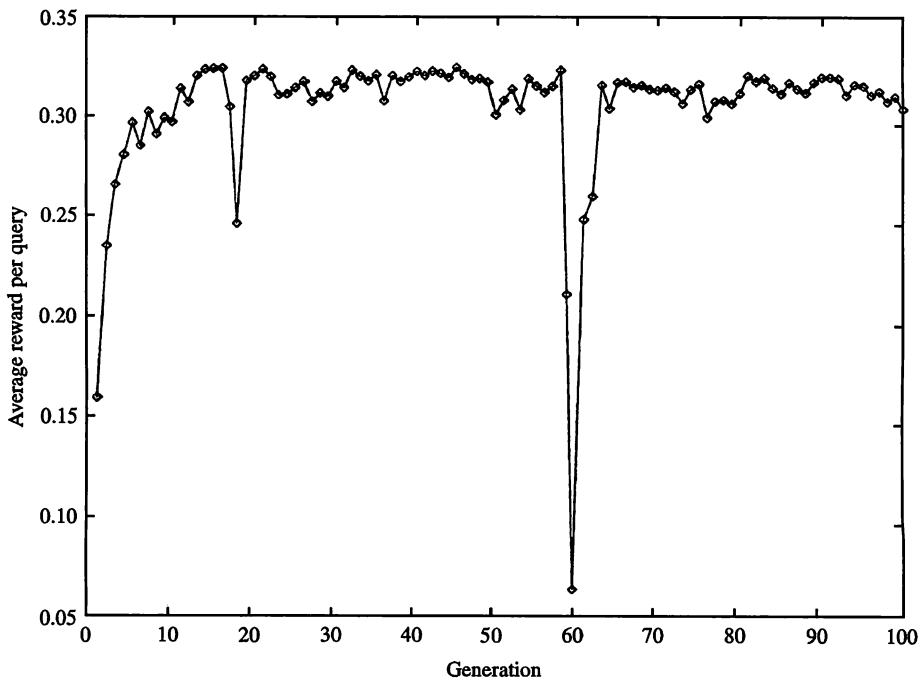

FIGURE 12. The GA average fitness for high-load setting.

Low load

Rule set				
(8, 91.9)	(6, 36.1)	(7, 38.3)	(5, 148.6)	(8, 92.7)
(8, 87.9)	(7, 38.3)	(6, 42.1)	(5, 148.6)	(5, 74.9)
(8, 87.9)	(7, 38.3)	(6, 38.3)	(5, 148.6)	(5, 74.9)
(8, 91.6)	(6, 63.8)	(5, 38.3)	(5, 148.6)	
(8, 82.8)	(6, 36.1)	(7, 38.3)	(5, 148.6)	

Medium load

Rule set				
(5, 99.0)	(2, 5.1)	(2, 9.8)	(4, 72.1)	(8, 93.2)
(5, 95.1)	(3, 9.8)	(5, 10.8)	(6, 68.5)	(1, 55.2)
(5, 125.5)	(2, 6.3)	(1, 9.8)	(6, 68.5)	(4, 120.9)
(5, 125.5)	(1, 14.3)	(3, 9.8)	(6, 68.5)	(5, 120.9)
(5, 125.5)	(1, 9.7)	(2, 9.8)	(6, 68.5)	(4, 120.9)

High load

Rule set				
(2, 99.3)	(8, 2.4)	(6, 60.6)	(6, 111.8)	(9, 79.4)
(2, 99.3)	(8, 2.4)	(1, 13.1)	(9, 106.1)	(9, 79.4)
(2, 99.3)	(8, 2.4)	(1, 13.1)	(8, 78.0)	(9, 78.0)
(2, 99.3)	(7, 37.8)	(8, 78.0)	(9, 78.0)	(8, 42.0)
(2, 99.3)	(7, 37.8)	(8, 78.0)	(8, 78.0)	(8, 45.0)

FIGURE 13. Top five strategies for each load setting.

6. Conclusions

We have shown with a simple model how network agents can profitably be designed to be co-operative. However, the assumptions made for this analysis were extremely strong, e.g. queries are uniformly distributed in time and over sites. Relaxing these assumptions to develop a formal analysis of a more realistic environment is the topic of our ongoing work. For example, we plan to model network agents that engage in clustering behavior, in the sense that small groups tend to send queries to the same sites. Our formal analyses have been supplemented with experimental work, using a simulation system we described above. We presented preliminary results from a set of genetic-algorithm experiments that show a connection between the quantity of resources available (in this case, how heavily loaded the network is) and the level of co-operation that emerges: the scarcer the resources, the greater the amount of co-operation that results.

Principal support for this work has come from the Office of Naval Research (Contract N00014-95-1-1161). Pollack has received additional support from the Rome Laboratory (RL) of the Air Force Materiel Command and the Advanced Research Projects Agency (Contract F30602-93-C-0038), and from an NSF Young Investigator's Award (IRI-9258392).

References

ETZIONI, O. & WELD, D. (1994). A Softbot-based interface to the Internet. *Communications of the ACM*, **37**, 72–76.

HOLLAND, J. (1975). *Adaptation in Natural and Artificial Systems*, Michigan: University of Michigan Press.

KAUTZ, H., SELMAN, B., COEN, M., KETCHPEL, S. & RAMMING, C. (1994). An experiment in the design of software agents. *Proceedings of the 12th National Conference on Artificial Intelligence*, pp. 438–443.

LIEBERMAN, H. (1995). Letizia: an agent that assists web browsing. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 924–929.

OSBORNE, M. J. & RUBINSTEIN, A. (1994). *A Course in Game Theory*, Cambridge, MA: MIT Press.

PERKOWITZ, M. & ETZIONI, O. (1995). Category translation: learning to understand information on the Internet. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 930–936.

SCHWARTZ, M. (1987). *Telecommunication Networks*. Reading, MA: Addison-Wesley.