Asynchronous Learning in Decentralized Environments: A Game Theoretic Approach

Eric J. Friedman*
School of Operations Research and Industrial Engineering,
Cornell University, Ithaca, NY 14853.

January 27, 2003

1 Introduction

Many of the chapters in this book consider collectives that are cooperative; all agents work together to achieve a common goal – maximizing the "world utility function." Often this is achieved by allowing agents to behave selfishly according to some "personal utility function," although this utility function is explicitly imposed by the designer so is not truly "selfish". In this chapter we consider the problems that arise when agents are truly selfish and their personal utility functions are intrinsic to their behavior. As designers we cannot directly alter these utility functions arbitrarily; all we are able to do is to adjust the ways in which the agents interact with each other and the system in order to achieve our own design goals. In game theory, this is the Mechanism Design Problem, and the design goal is denoted the "Social Choice Function" (SCF).[1]

Note that these design goals may be distinct from the agents' goals. For example, the most common SCFs to keep in mind are the Utilitarian (or maximizing) SCF, which simply maximizes the sum of all the agents' personal utility functions or the Egalitarian (or fair) SCF, which maximizes the value of the smallest personal utility function or some combination of these two, combining both maximization and fairness. However, the designer's SCF might be unrelated to the agents' personal utility functions. This commonly occurs in auctions, where the designer's goal is to maximize revenue.

Our main interest in this problem arises from problems on the Internet, in which agents may be either users or autonomous agents (bots). While, in the not so distant past agents could, with some level of accuracy, be assumed to be cooperative, this is no longer a reasonable assumption for a modern analysis of the Internet. This is clearly true at the user level, where users may behave selfishly, but is probably even necessary at lower levels such as network protocols, since the Internet is made up of many profit maximizing autonomous systems.[2]

For example, the current stability of the Internet can be attributed to the fact that most traffic uses the standard TCP protocol which reacts to congestion on the network by reducing its

*To appear in Collectives and the Design of Complex Systems, edited by K. Tumer and D. Wolpert, Springer-Verlag, 2003. Work supported in part by National Science Foundation Grant No. ANI–9730162. Email:friedman@orie.cornell.edu, www:http://www.orie.cornell.edu/∼friedman

[1]This chapter is meant to be self contained (but telescopic) at an informal level. For a more complete introduction to Mechanism design see [10, 14].

[2]See [4] for an interesting example of this relating to the BGP routing protocol.

transmission rate. This is necessary to avoid "congestive collapses" which plagued the Internet in the mid 1980s and lead to many crashes of significant fractions of the Internet. This is is no longer a significant problem due to the near universal adoption of congestion control.[3]

However, from a selfish point of view, congestion control is detrimental to an individual agent, since if everyone else is using it, then an agent can unilaterally increase their utility by disabling it on their own TCP connection. A single user disabling congestion control is obviously not a threat to the stability of the Internet; however if everyone did it then one could expect frequent congestive collapses, and everyone would suffer. Thus, the design of protocols, for which agents do not have strong incentives to circumvent, is, we believe, one of the fundamental tasks for the continued development and health of the Internet.

In this chapter we will provide an intuitive introduction to our work on this subject. The formal analysis can be found in [8] while details of the simulations are in [11] and the experiments are discussed in [9].

2 Mechanism Design

It is useful to formulate our problems according to the mechanism design paradigm. We assume that there is a set of feasible outcomes $p \in P$. For example, in a simplified version of the TCP example discussed above, p describes the complete state of the system, which includes every agent's transmission rate and delays. Clearly, by adjusting priorities in the network (such as in routers) one can feasibly redistribute delays; however, it is not possible to eliminate such delays completely and this restricts the choice of the set P.

Next we assume that agents have utilities over outcomes, $U(p) \in \mathcal{U}^n$, where $\mathcal{U}$ is the set of possible utility functions. For example, it would be natural to assume that agents' utilities are nondecreasing in their transmission rates and nonincreasing in their delays.

As discussed in the Introduction, we allow for a wide range of world utilities, which are commonly called the social choice functions, $\mathcal{F}$: $F|\mathcal{U}^n \to P$. Symbolically the utilitarian SCF is given by maximizing the utilitarian objective function[4], $\sum_i U_i(F(U))$ while the egalitarian objective function is (essentially) given by $\min_i U_i(F(U))$.[5]

Now, the system we design which determines the interactions among the agents is known as a Mechanism. Abstractly a mechanism consists of message spaces, A_i and a mechanism function $M|A \to P$. We interpret this as a two stage process in which the agents each choose a message to send $a_i \in A_i$ and then the mechanism chooses the outcome $M(a)$. For this mechanism we get an induced utility for message vectors, $G_i(a) = U_i(M(a))$. This is the utility to the agent for choosing message a_i when the other agents choose a_{-i} and thus the vector or messages is $a = (a_i, a_{-i})$.

We can now interpret this as a game in which agents are attempting to choose a_i so as to maximize their own utility. In this view the forward problem is that of finding the solution concept, $S(U)$, for this game, that is the set of vectors a which could arise when a group of agents play this game, while the reverse problem is that of choosing A and $M(\cdot)$ to maximize the social objective function.

[3] One notable exception to this is streaming media, for which reliable congestion controls have only recently been developed and are expected to be adopted in the near future [5].

[4] Note, the definition of social choice function is standard, while the "social objective function" is not in the literature on mechanism design.

[5] Actually, the egalitarian social choice function is usually defined in terms of a lexicographic ordering of the sorted vector of U_i's and in the continuous setting can not be computed as the maximum of a continuous objective function.

3 Game Theoretic Analysis: the forward problem

In this section we discuss the so-called "forward problem" or that of finding the relevant solution concept. In the standard mechanism design literature the solution concept is typically assumed to be either the Nash equilibrium of the game or the dominant strategy outcome (both defined below). Our claim, based on theoretical analyses, simulations and experiments with human subjects, is that these are not adequate in the noisy, asynchronous setting of the Internet where information about the underlying network and the behavior of other agents is extremely limited. In the following we will provide a simplified outline of this argument.

First consider the well known prisoner's dilemma[6]:

	C	D
C	1,1	-1,2
D	2,-1	0,0

This table represents the game (mechanism). Both the row agent and the column agent can choose actions C (cooperate) or D (defect) which determine the vector of payoffs. For example, if both agents cooperate, then both receive 1 unit of utility while if one cooperates and the other defects, then the defector gets 2 while the cooperator only gets -1.

It is well known that the rational outcome of this game is for both agents to defect, since defecting is always optimal (gets more utility) no matter what strategy your opponent chooses. In the language of mechanism design, we say that D strictly dominates C. Thus, we can construct a solution concept for which $S^{Dom}(U)$ only contains strategies which are not dominated and for the prisoners' dilemma, this set would contain a single strategy vector (D, D).

Next consider a slightly more complicated game, the prisoner and the altruist, in which the column agent develops a conscience. Now, if she defects but the other agent doesn't she feels guilty which reduces her utility in this outcome:

	C	D
C	1,1	-1,0
D	2,-1	0,0

In this game one can easily check that $S^{Dom}(U) = \{(D, D), (D, C)\}$; however, since the row agent never chooses C it is irrational for the column agent to choose C since she really faces the following game:

	C	D
D	2,-1	0,0

In this "reduced game" C is strictly dominated by D. Thus, we consider the solution concept induced by "iterated dominance" in which we iteratively remove dominated strategies. This procedure is well defined (and independent of the order of removals) and yields $S^{ItDom}(U) = \{(D, D)\}$.

The solution concept S^{ItDom} is quite common in mechanism design and is larger than most others typically studied, such as the set of Nash equilibria, e.g. $S^{ItDom} \supset S^{Nash}$. When learners know their own utility function, it is easy to see that most reasonable models of learning will converge to S^{ItDom} as shown quite generally in [15].

[6]We consider the prisoner's dilemma for simplicity, but note that there are many applications of the following ideas. See [8] for a detailed bibliography and [6] for a recent application to routing and TCP

The intuition behind this result is quite straightforward. Initially, agents would never play dominated strategies. After a reasonable period of time agents notice that their opponents are not playing these strategies and thus stop playing strategies that are dominated with respect to this smaller set of opposing strategies. This process iterates until play is within S^{ItDom}. Note that knowledge of one's own payoffs and observation of other agents' strategies is crucial to this argument.[7]

3.1 Decentralized Learning with Limited Information

In many settings an individual may not know their own utility function explicitly. For example, when I adjust the controls on my TV set (such as brightness, contrast, etc.) I am unable to optimize the picture without trial. Additionally, on the Internet I do not know the effect of an increase in transmission rate on congestion related delays, since I do not know the details of the transport layer of the network. Interestingly, even if agents do not know their own utility functions, but are "reasonable learners"[8] and play is synchronous, such that every agent simultaneously updates their strategy at discrete intervals, then play (approximately) converges to S^{ItDom} as shown in [8].[9] The intuition behind this result is based on the idea that dominated strategies have a lower expected payoff under any probability distribution of opponents' strategies and that if an agent "experiments" sufficiently often and randomly then she will be able to detect dominated strategies.

3.2 Asynchronous Play

Interestingly, when play is asynchronous, (as is typical on the Internet and in many other "real-time" settings) play need not converge to S^{ItDom} and thus we should consider a weaker (larger) solution concept if we want to guarantee that our mechanism will work as desired.

To gain some intuition, consider the prisoner and the altruist, but this time assume that the prisoner gets to choose first and then the altruist chooses a strategy *based on the knowledge of the strategy chosen by the prisoner*. In this case, if the prisoner chooses D then so will the altruist, while if the prisoner chooses C then the altruist will also choose C and this will result in a higher payoff to the prisoner. Thus the prisoner will choose C and the outcome will be (C, C) which is not in S^{ItDom}. This is known as the Stackelberg outcome [20].

Although, we are not actually considering Stackelberg's model as described above, we claim that such outcomes can naturally arise in asynchronous learning. First we describe a simple model of asynchronous play, motivated by the Internet.

Consider an agent trying to decide the rate at which to transmit data over a network. The agent's choice of how often to update their transmission rate should depend on both the round trip time between their computer and the one they are communicating with as well as the amount of "noise" in the congestion that they observe. This is because the only information that the agent receives is packet acknowledgements (ACKs) or lack of them. These ACKs are only received after a packet reaches the destination successfully and the ACK travels back to the origin. Thus, new information is received with a delay of at least the round trip time. In addition, the variance of the round trip times can be quite large and thus it would be sensible to average some reasonable number of round trip times before significantly altering the transmission rate.

[7] Note that this is the author's interpretation of Milgrom and Robert's results and differs somewhat from the interpretation given in their paper [15].

[8] The technical definition of a reasonable learner is fairly complicated [8], but most adaptive learning algorithms, such as those used on the Internet, are reasonable in this sense.

[9] Note that convergence is in the sense of PAC learning [19].

Thus, we model agents as maintaining a specific strategy for a fixed period of time, and then choosing a new strategy based on some average of the performance in the previous period. There is no reason to assume that these periods are synchronized or even similar between different agents. For example the round trip time can range from several milliseconds on a local network to 1/2 a second (or longer) when transmitting internationally.

Now consider a situation in which a very slow agent is in a game with a much faster agent. In this case, it is as if the slower agent were the Stackelberg leader, since during a single period of the slow agent, the fast agent will converge to and mostly play the best response to the slow agent's strategy. Then the slow agent will learn the action that yields the highest payoff, *based on her opponents optimal responses*. It is straightforward to see that this will yield the Stackelberg outcome.

Thus, for asynchronous games our solution concept S^{Async} should at least contain all Stackelberg outcomes S^{Stack} and thus will not be contained within S^{ItDom}.

In [8] we propose a Stackelberg-solution-concept which is constructed from any synchronous solution concept which takes into account all possible "extremely asynchronous games" and constructs a minimal solution concept such that no smaller solution concept would be useful. For example, if we believed that synchronous play converged to Nash equilibria (which we don't!) then for asynchronous play the smallest reasonable solution concept would contain all (generalized) Stackelberg equilibria.

Note that this leads to a counter-intuitive result for learning in these settings, since for many of the most common problems on the Internet, such as adjusting transmission rate with FIFO queuing, the slower agent may get a higher payoff than the fast one! Thus, less sophisticated (slower) algorithms may be preferred for game theoretic reasons.[10]

3.3 Guaranteed Convergence

The results in the previous section provide lower bounds for the relevant solution concept in asynchronous decentralized environments. While we don't know the relevant solution concept, S^{Async} precisely, we can prove an upper bound on this set, i.e. a set S^O such that $S^O \supset S^{AsyncDec}$.

Consider the payoff matrix for the prisoner in the prisoner and the altruist game:

	C	D
C	1	-1
D	2	0

In an asynchronous decentralized setting, the reason the prisoner may not realize that C is dominated is because based on her information there is no reason that the payoff matrix couldn't be:

	C	D
C	-1	1
D	2	0

This is the matrix obtained by swapping the payoffs when she plays C and in this game C is not dominated.

[10]Note that it is quite common in game theory for less sophisticated agents to outplay extremely sophisticated ones. One particularly interesting and important example is the well known tit-for-tat strategy which outperformed many significantly sophisticated strategies in computational tournaments [2].

This could not arise if all the payoffs for playing D were larger than all the payoffs for playing C. Since then for any permutation of the payoffs D would dominate C. In general we call such a strategy "overwhelmed". Since agents will clearly learn not to play overwhelmed strategies an iteration of this argument shows that they will converge to the set S^{ItOver} which is obtained by the iterated elimination of overwhelmed strategies. In [8] we prove that reasonable learners will (approximately) converge to S^{ItOver} and thus we have shown that $S^{Stack} \subseteq S^{Async} \subseteq S^{ItOver}$.

Notice that for both the prisoners' dilemma and the prisoner and the altruist game the set S^{ItOver} contains all of the strategies and thus provides an uninteresting result for the forward problem in asynchronous settings. Indeed, for typical games this is true. For example, in standard network congestion models no strategies are overwhelmed.

However, as we discuss below, our main interest here is whether we can construct mechanisms which have nice properties, such as having guaranteed convergence, i.e. $|S^{ItOver}(U)| = 1$ for all $U \in \mathcal{U}^n$. One interesting example of such a mechanism arises on a network in which all servers use the "fair-share" protocol [18, 7], as discussed below.

3.4 Numerical and Experimental Support

Since our goal in this analysis is to develop useful tools to apply to the Internet and other decentralized environments we should validate our results, since a theorem is only as good as the axioms on which it is based.

One validation is based on a set of numerical simulations. This work, [11], studied a wide variety of common learning algorithms in a variety of settings. It showed rapid convergence to S^{ItOver} but slow or non-convergence to S^{ItDom} and provided clear evidence of convergence to Stackelberg outcomes in highly asynchronous settings. However, it did raise issues about when asynchrony is relevant; for example, under small amounts of asynchrony play often converged to S^{ItDom}.

A second validation is based on a set of experiments with human subjects. In that work, [9], we studied the convergence of people adjusting a data-rate slider on a web browser. The data clearly shows nonconvergence to S^{ItDom} and approximate convergence to S^{ItOver} for a small number of agents (0-5). It also showed (approximate) Stackelberg behavior. However, for a larger number of agents (8) play did not even converge to S^{ItOver}. Thus, when designing mechanisms for real people in decentralized settings, one needs to be very careful!

4 Mechanism Design: the inverse problem

Although finding the correct solution concept, solving the forward problem, is interesting in its own right, the inverse problem, that of constructing a mechanism that "implements" the social choice function is of great practical importance, especially in engineering settings. For example, our main interest in the forward problem for networks is to assist in the designs of network protocols that lead to stable and efficient networks.

Clearly, this problem depends crucially on the solution concept. For example, implementation for both S^{Nash} and S^{ItDom} has been well studied and the condition under which a social choice function can be implemented in these solution concepts is well understood. In particular, many important social choice functions can be implemented and almost any social choice function can be approximately implemented to any degree of accuracy [12] for either of these.[11]

[11] However, even for these solution concepts the mechanisms that implement various social choice functions seem quite fragile and unrealistic.

However, in the asynchronous decentralized setting the problem appears to be much more difficult, since the solution concepts are much weaker. Although our work here is preliminary, the following results demonstrate the difficulty of implementing social choice functions in these settings.

Given a social choice function, $F|U \to P$ we consider the so-called direct mechanism, in which agents simply reveal their type, or utility function U_i, and the mechanism chooses the outcome defined by the social choice function, i.e., $A_i = \mathcal{U}_i$ and $G_i(a) = F(a)$. Clearly, in a direct mechanism an agent may have incentives to misstate their type. For example, in a salary mechanism, typically the firm states that it is on the verge of bankruptcy, while the employee usually has a large and underfed family.

We say that a direct mechanism is strictly strategyproof, if reporting truthfully is always a dominant strategy for every agent. It is proven in [8] that for any solution concept $S^{AsynDec}$ which contains S^{Stack} only social choice functions which are strictly strategyproof can be implemented in $S^{AsynDec}$. Thus, in an asynchronous decentralized setting the restriction on implementable social choice functions is quite severe. Furthermore, under reasonable assumptions on the set $\mathcal{U}$ there are more severe restrictions.

In fact, while we don't know the precise restrictions for implementing under $S^{AsynDec}$ we have proven that in order for a mechanism to implement a social choice function under S^{ItOver} the social choice function must be strictly coalitionally strategyproof. That implies that not only is it in the best interest of every agent to report their utility function truthfully in the direct mechanism, but even for groups of colluding agents their best strategy is for all to report truthfully. If this were also true for implementation under $S^{AsynDec}$ then this constraint of the set of implementable social choice functions would be extremely severe.

It is possible that one can approximately implement a significantly larger class of social choice functions to a high degree of accuracy; however, we do not know of any results of this type for asynchronous decentralized systems. Some interesting results in this direction have been shown by several authors [16, 13, 1]

5 Example: Sharing a congested link

In this section we consider an important example: a group of users sharing a congested data network. For simplicity, we will focus on a simple network with a single link, but most of our analysis generalizes directly to arbitrary network structures. This model was first studied in [17] and our presentation follows that paper.

Consider n users, sharing a single congested data link. Each user has a utility function $U_i \in \mathcal{U}$ which is a function of $p_i = (l_i, d_i)$ where l_i is the transmission rate and d_i is the average delay faced by user i's packets. The set $\mathcal{U}$ is the set of all concave functions which are nondecreasing in l_i and nonincreasing in d_i.

Clearly $P \subset \Re_+^{2n}$, but the precise characterization of P is complex and depends critically on the technology. For example, if we assume that users' packets are generated according to a Poisson process and that the transmission times (or packet sizes) follows an exponential distribution, then the set P is characterized by the following set of constraints :

$$\forall S \subset \{1, 2, \ldots, n\}, \quad \sum_{i \in S} l_i d_i \geq C(\sum_{i \in S} l_i)$$

where $C(x) = x/(\mu - x)$ and μ is the capacity of the link.

First we consider the forward problem for some well known protocols. For example, the most common mechanism arises from first-in-first-out (FIFO) queuing, in which the packets are served

in order of their arrival. Under this protocol we get a game in which users choose their transmission rate, l_i, and obtain a delay from the system of $d_i = C(\sum_i l_i)/l_i$.

While FIFO is commonly used, it is easy to see for this mechanism that $|S^{Stack}| \neq 1$ for most choices of utility functions by considering the first order conditions. Thus, the outcome need not be unique under FIFO. In fact, when people play this game, play does not seem to converge [9].

A second interesting protocol is the fair-share mechanism introduced in [18], which is closely related to fair-queuing [3] a well known protocol. Under fair share we can recursively compute the delays, as follows. First, reorder the agents so that $l_1 \leq l_2 \leq \cdots \leq l_n$. Then define $v_i = C(l_1 + \cdots + l_i + (n-i)l_i)/(n-i+1)$. The fair-share mechanism is defined by $d_i = \sum_{j=1}^{i} v_j/l_i$. The key characteristic of this mechanism is the fact that a user's delay is not affected by small changes in transmission rates of users with higher transmission rates.

Interestingly, for any realization of utility functions and any number of users, $|S^{ItOver}| = 1$ and thus play is guaranteed to converge (to the Nash equilibrium) under the fair share mechanism. Thus, networks with fair-share mechanisms are more stable than those using FIFO. This is seen empirically in [9].

Lastly, we note that the mechanism design problem (or inverse problem) is easy to solve for the standard social choice functions when the solution concept is S^{Nash} or S^{ItDom} and we are interested in approximate optimization. (See [17] for the former and [12] for the later.) However, for the case of S^{Async} there does not exist a mechanism which implements the utilitarian social choice function. This and other negative results are proven in [8].

One important open question is whether one can circumvent these negative results using ideas such as pricing or by providing additional information which would allow users to learn more efficiently. (See [8] for further discussion.) Another interesting approach is to design mechanisms that are within constant factor of optimality, such as in [16, 1].

These are among many open questions still to be resolved in the application of mechanism design to decentralized settings, such as the Internet

6 Acknowledgments

This paper is based directly on joint work with Scott Shenker, Mike Shor, Amy Greenwald, Adam Landsberg and Barry Sopher and indirectly on conversations with many others.

References

[1] A. Archer and É. Tardos. Truthful mechanisms for one-parameter agents. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, 2001.

[2] R. Axelrod. *The evolution of cooperation.* Basic Books, New York, 1984.

[3] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. *Journal of Internetworking*, 1(1):3–26, January 1990.

[4] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A bgp-based mechanism for lowest-cost routing. In *Proceedings of the 2002 ACM Symposium on Principles of Distributed Computing*, 2002.

[5] S. Floyd, M. Handley, J. Padhye, , and J. Widmer. Equation-based congestion control for unicast applications. In *Proc. ACM Sigcomm 2000*, 2000.

[6] E. Friedman. Selfish routing on data networks isn't too bad: Genericity, tcp and ospf. mimeo, Cornell University, 2002.

[7] E. J. Friedman. Strategic properties of heterogeneous serial cost sharing. Mathematical Social Sciences (forthcoming), 2000.

[8] E. J. Friedman and S. Shenker. Learning and implementation in the Internet. mimeo, available from (www.orie.cornell.edu/∼friedman), 2002.

[9] E. J. Friedman, M. Shor, S. Shenker, and B. Sopher. Asynchronous learning with limited information: An experimental analysis. mimeo, available from (www.orie.cornell.edu/∼friedman), 2001.

[10] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, Massachusetts, 1991.

[11] A. Greenwald, E. Friedman, and S. Shenker. Learning in network contexts: Experimental results from simulations. *Games and Economic Behavior*, 35(1):80–123, 1999.

[12] Matthew Jackson. A crash course in implementation theory. forthcoming in Social Choice and Welfare, 2001.

[13] K. Jain and V. Vazirani. Applications of approximation algorithms to cooperative games. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 364–372, 2001.

[14] A. Mas-Colell, M. Whinston, and J. Green. *Microeconomic Theory*. Oxford University Press, Oxford, 1995.

[15] P. Milgrom and J. Roberts. Rationalizability, learning and equilibrium in games with strategic complementarities. *Econometrica*, 58:1255–1278, 1990.

[16] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing*, pages 129–140, 1999.

[17] S. Shenker. Efficient network allocations with selfish users. In P. J. B. King, I. Mitrani, and R. J. Pooley, editors, *Performance '90*, pages 279–285. North-Holland, New York, 1990.

[18] S. Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. *IEEE/ACM Transactions on Networking*, 3:819–831, 1995.

[19] L. Valiant. A theory of the learnable. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, Washington, D.C., 1984.

[20] H. von Stackelberg. *Marktform und Gleichgewicht*. Springer-Verlag, 1934. English translation, entitled *The Theory of the Market Economy*, published in 1952 by Oxford University Press.