Finding Bounded Rational Equilibria Part II: Alternative Lagrangians and Uncountable Move Spaces

David H. Wolpert*

November 4, 2004

Abstract

A long-running difficulty with conventional game theory has been how to modify it to accommodate the bounded rationality characterizing all real-world players. A recurring issue in statistical physics is how best to approximate joint probability distributions with decoupled (and therefore far more tractable) distributions. It has recently been shown that the same information theoretic mathematical structure, known as Probability Collectives (PC) underlies both issues. This relationship between statistical physics and game theory allows techniques and insights from the one field to be applied to the other. In particular, PC provides a formal model-independent definition of the degree of rationality of a player and of bounded rationality equilibria. This pair of papers extends previous work on PC by introducing new computational approaches to effectively find bounded rationality equilibria of common-interest (team) games.

1 INTRODUCTION

The fields of statistical physics, game theory, and distributed control/optimization share one fundamental characteristic: they are all concerned with how the probability distribution governing a distributed system is related to the functionals that it optimizes. This shared characteristic provides the basis for a mathematical language for translating many of the concepts of those fields into one another. This mathematical language is known as Probability Collectives (PC) [1, 2, 3, 4, 5, 6]. By allowing us to transfer theory and techniques between those fields, it provides a means of unifying them.

This pair of papers introduces computational techniques from PC for efficiently finding bounded rational equilibria of noncooperative games. The first paper starts with a review of PC and how to use it to formalize bounded rationality [7]. Also in that paper are a review of two of the previously explored

*D. Wolpert is with NASA Ames Research Center, Moffett Field, CA, 94035 dhw@ptolemy.arc.nasa.gov

techniques for finding bounded rational equilibria, Brouwer updating and Nearest Newton updating. After this that paper introduces iterative focusing, a new set of techniques for finding full rationality equilibria.

Due to space limitations, several other schemes for finding bounded rational equilibria could not be presented in that first paper. They are instead introduced in this second paper. This second paper also shows how to extend all of the approaches for finding equilibria (from both papers) to the case of uncountable move spaces of the players. Some issues that arise in practice when running these algorithms are also discussed here.

The version of Probability Collectives considered in this paper, involving product distributions, is called "Product Distribution" (PD) theory[1]. It's important to note that PD theory also has many applications in science beyond those considered in this paper. For example, see [3, 4, 8, 9, 10, 5, 6, 11] for work concerning distributed control and to distributed optimization. See also [12, 13, 10] for work showing, respectively, how to use PD theory to improve Metropolis-Hastings sampling, how to relate it to the mechanism design work in [14, 15, 16, 17], and how to extend it to continuous move spaces and time-extended strategies.

Throughout these papers δ functions are either Dirac or Kronecker as appropriate, integrals implicitly have a measure appropriate to the cardinality of the underlying space, and Θ is the Heaviside step function.

2 Variations of Previous Schemes and Practical Issues

In this section we first present some of the salient equations from [7] for completeness. We then show how to modify the Monte Carlo process used in parallel Brouwer updating to avoid the "thrashing" problem. Next we present some alternatives to Maxent Lagrangians for the case where the ultimate goal is finding $\text{argmin}_x G(x)$, i.e., when optimizing the game reduces to a minimization problem. We end with a discussion of issues that arise in practice.

2.1 Salient Equations

The "Maxent" or "*qp*" Lagrangian discussed in [7] is

$$\begin{aligned} \mathscr{L}(q) &\equiv \beta[E_q(G) - \epsilon] - S(q) \\ &= \beta[\int dx \prod_j q_j(x_j) G(x) - \epsilon] - S(q) \end{aligned} \tag{1}$$

having minimizing product distribution q given by

$$q_i(x_i) \propto e^{-E_{q_{-i}}(G|x_i)}. \tag{2}$$

Steepest descent of the Maxent Lagrangian forms the basis of the Nearest Newton algorithm. Direct application of the equations that minimize it form the

basis of the Brouwer update rules. The "pq" Lagrangian is instead minimized by the the product of the marginals of the Boltzmann distribution p^β.

These and other update rules are described in [7], and can all be written as multiplicative updating of q. The following is a list of the update ratios $r_{q,i}(x_i) \equiv q_i^{t+1}(x_i)/q_i^t(x_i)$ of some of those rules. In all of these F_G is a probability distribution over x that never increases between two x's if G does (e.g., a Boltzmann distribution in $G(x)$). In addition const is always a scalar that ensures the new distribution is properly normalized and α is a stepsize.[1]

Gradient descent of qp distance to F_G:

$$1 - \alpha[\frac{E_{q^t}(\ln[F_G] \mid x_i) + \ln(q_i^t(x_i))}{q_i^t(x_i)}] - \frac{\text{const}}{q_i^t(x_i)} \tag{3}$$

Nearest Newton descent of qp distance to F_G:

$$1 - \alpha[E_{q^t}(\ln[F_G] \mid x_i) + \ln(q_i^t(x_i))] - \text{const} \tag{4}$$

Brouwer updating for qp distance to F_G:

$$\text{const} \times \frac{e^{E_{q^t}(\ln[F_G] \mid x_i)}}{q_i^t(x_i)} \tag{5}$$

Importance sampling minimization of pq distance to $F_G(x)$:

$$\text{const} \times E_{q^t}(\frac{F_G}{q^t} \mid x_i) \tag{6}$$

Iterative focusing of $\tilde{q}$ with focusing function $F_G(x)$ using qp distance and gradient descent:

$$1 - \alpha\{\frac{E_{q^t}(\ln[F_G] \mid x_i) + \ln[\frac{q_i^t(x_i)}{\tilde{q}_i(x_i)}]}{q^t(x_i)}\} - \frac{\text{const}}{q^t(x_i)} \tag{7}$$

Iterative focusing of $\tilde{q}$ with focusing function $F_G(x)$ using qp distance and Nearest Newton:

$$1 - \alpha\{E_{q^t}(\ln[F_G] \mid x_i) + \ln[\frac{q_i^t(x_i)}{\tilde{q}_i(x_i)}]\} - \text{const} \tag{8}$$

[1] As a practical matter, both Nearest Newton and gradient-based updating have to be modified in a particular step if their step size is large enough so that they would otherwise take one off the unit simplex. This changes the update ratio for that step. See [9].

Iterative focusing of $\tilde{q}$ with focusing function $F_G(x)$ using qp distance and Brouwer updating:

$$\text{const} \times e^{E_{q^t}(\ln[F_G] \mid x_i)} \times \frac{\tilde{q}(x_i)}{q_i^t(x_i)} \tag{9}$$

Iterative focusing of $\tilde{q}$ with focusing function $F_G(x)$ using pq distance:

$$\text{const} \times E_{\tilde{q}}(F_G \mid x_i) \times \frac{\tilde{q}(x_i)}{q_i^t(x_i)} \tag{10}$$

Note that some of these update ratios are themselves proper probability distributions, e.g., the Nearest Newton update ratio.

2.2 Modifications to the Monte Carlo Process of Parallel Brouwer

As described in [7], parallel Brouwer updating can be subject to "thrashing", in which each player's update confounds the updates of the other players. The simplest way to mitigate this is by not having each player i jump all the way from its current distribution q_i to the new one recommended by parallel Brouwer updating, q_i^*. Instead one can have each i only jump part way in the direction from q_i to q_i^*. (This in fact is what is done in practice.) This subsection presents an alternative approach.

To begin, note that we would not get any thrashing in parallel Brouwer if rather than the function $E_q(G \mid x_i)$, each agent i performed its update using $E_\pi(G \mid x_i)$ for some distribution π that is independent of q. The natural choice of π is exactly the distribution that q is designed to approximate well, namely the Boltzmann distribution [2].

To implement this modification, we need to have all agents i simultaneously estimate their associated functions $E_\pi(G \mid x_i)$ rather than $E_q(G \mid x_i)$. Precisely because q should approximate π well, we can do this using our Monte Carlo samples of q, simply by modifying how each agent uses those samples. The general idea is to use those samples of q as a proposal distribution for generating samples from π.

As an example, we can use the samples of q to estimate the integral $E_\pi(G \mid x_i)$ via importance sampling. To do this we write

$$E_\pi(G \mid x_i) = \frac{\int dx'_{-i} \, [\frac{\pi(x_i, x'_{-i})}{q(x_i, x'_{-i})} G(x_i, x'_{-i})] q(x_i, x'_{-i})}{\int dx'_{-i} \, [\frac{\pi(x_i, x'_{-i})}{q(x_i, x'_{-i})}] q(x_i, x'_{-i})},$$

and then sample q, using empirical averages across those samples to estimate both the quantity in the square brackets in the numerator of our integral and

[2] Note that in doing this, we change the equilibrium distribution from that of 2. Now it is given by $q_i(x_i) \propto e^{-\beta E_\pi(G \mid x_i)}$

the quantity in square brackets in the denominator. (Note that we only need to know π up to an overall normalization constant to do this.) Under the original sampling scheme, for each of its possible moves x_i, agent i forms the uniform average of the G values that arose when it made that move, and takes that average as its estimate of $E_q(G \mid x_i)$. Under the modified scheme, it would instead estimate the function $E_\pi(G \mid x_i)$ with a weighted average of those G values. The weights would be the associated values $\pi(x)/q(x)$.[3]

Another way to estimate $E_\pi(G \mid x_i)$ using samples generated from q would be via a Metropolis-Hastings random walk. Under this scheme q would be a proposal distribution, and the points it generates would be kept either if they raised $\pi(x)/q(x)$, or, if not, if the flip of an appropriately weighted coin comes up heads. At the end of the Monte Carlo block, each agent i would form the uniform averages over the kept points, thereby forming an estimate of its function $E_\pi(G \mid x_i)$ [4].

Integration of parallel Brouwer and the Metropolis-Hastings algorithm can be viewed other ways than as a modification to parallel Brouwer updating. In particular, it can be viewed as a modification to standard optimization via simulated annealing. The modification is that the proposal distribution is dynamically updated in an "intelligent" way, rather than (as in the conventional simulated annealing algorithm) being pre-fixed. This is the idea behind the Intelligent Coordinates algorithm [18].

In addition to mitigating the thrashing problem, the replacement of $E_q(G \mid x_i)$ with $E_\pi(G \mid x_i)$ sometimes results in new equilibria that better capture inter-agent dependencies in G. In particular, they will sometimes avoid the problem of spurious equilibria that can arise with a Lagrangian over product distributions, in which the equilibirum product distribution has high values for some x's that have poor G values.

As an example, say we have three agents all with binary move spaces, $\{0, 1\}$. Say that the $x_3 = 0$ plane of G values is given by $G(0,0,0) = 1, G(0,1,0) = G(1,0,0) = .5$, and $G(1,1,0) = 0$, while when $x_3 = 1$, $G(x) = .5$, regardless of x_1 and x_2. So we would like to have the equilibrium distribution be biased towards (1, 1, 0). However in one equilibrium of Eq. 2, agents 1 and 2 would have uniform distributions, giving a uniform distribution over their joint moves. This uniformity of $q(x_1, x_2)$ then means that agent 3 would not have any basis for choosing one or the other of its two moves: $E_q(G \mid x_3)$ would be independent of x_3. Accordingly, agent 3 would also choose a uniform distribution. The resultant product distribution would be uniform, and therefore would not be low over all x's with poor G values [5].

[3] Note that these weights can be communicated to all the agents by the same system that broadcasts G values to all the agents, if first all agents communicate q_i values to that system.

[4] Ref. [12] presents a detailed analysis of the use of samples of a product distribution to do Metropolis-Hastings sampling. That work does not directly concern the issue of optimization. Rather it concentrates on using Probability Collectives to improve the usual goal of the Metropolis-Hastings algorithm, namely sampling a provided probability distribution.

[5] Intuitively, the problem is that the move spaces of the agents do not factor the joint move space in a way that is "aligned" with G. See [13, 10] for a discussion of how to use semi-coordinate transformations of the move spaces to circumvent this problem.

Now consider the modified Monte Carlo process. In this new process agent 3 chooses its move based on $E_\pi(G \mid x_3)$. However since it is convex, π is most peaked for $x_3 = 0$, unlike the equilibrium q under Eq. 2. So the values of $E_\pi(G \mid x_3)$ now would distinguish between the two possible moves of x_3, biasing it towards the move 0. Similarly x_1 and x_2 would both be biased to equal 1. So our equilibrium distribution would be biased towards (1, 1, 0), which is exactly what we want.

Note that schemes like gradient descent always have the same equilibrium as that of Brouwer updating if one replaces $E_q(G \mid x_i)$ in thos schemes, whatever function one uses to replace $E_q(G \mid x_i)$ (so long as it is the same function in both schemes). Accordingly, replacement of $E_q(G \mid x_i)$ with $E_\pi(G \mid x_i)$ may be beneficial to steepest descent algorithms, in addition to parallel Brouwer.

2.3 Variants of Maxent Lagrangians

Consider the use of iterative update rules for the q_i in concert with Monte Carlo sampling of q. In such scenarios, at each stage of the iterative updating, for each of her moves x_i, each player i has an empirical estimate of the distribution $P(G \mid x_i)$ (and therefore of any distribution $P(f(G) \mid x_i)$ for invertible $f : \mathbb{R} \to \mathbb{R}$). Every player i uses her empirical estimate according to a pre-set algorithm — potentially varying from one player to the next — to determine how to update her distribution q_i. Our task as system designers is to choose those pre-set algorithms in such a way that the ultimate goal of the updating is achieved as quickly as possible.

In the update rules discussed above each empirical distribution is reduced to an expectation value which is then used to perform the update. While this need not be the case in general, update rules based on expectation values form a very rich set, including many rules not investigated previously. This subsection introduces some such novel update rules that are based on expectation values.

Both the *qp*-KL Lagrangian and *pq*-KL Lagrangians discussed above had the target distribution be a Boltzmann distribution over G. For high enough β, such a distribution is peaked near $\text{argmin}_x G(x)$. So sampling an accurate approximation to it should give an x with low G, if β is large enough. This is why one way to minimize G is to iteratively find a q that approximates the Boltzmann distribution, for higher and higher β.

However there are other target distributions that grow larger as G grows smaller e.g., logistic functions of G, step functions (i.e., Heaviside functions) of G, etc. So one set of alternatives to the Lagrangians discussed above is to choose some alternative target distribution(s), and for each one find the q minimizing *pq* or *qp* KL distance to it.

Return now to the Maxent Lagrangian. Say that after finding the q that minimizes the Lagrangian, we IID sample that q, K times. We then take the sample that has the smallest G value as our guess for the x that minimizes $G(x)$. For this to give a low x we don't need the mean of the distribution $q(G)$ to be low — what we need is for the bottom tail of that distribution to be low. This

suggests that in the $E(G)$ term of the Maxent Lagrangian we replace

$$q(x) \leftarrow q(x)\frac{\Theta[\kappa - \int dx'\ q(x')\Theta[G(x) - G(x')]]}{\kappa}.$$

This new multiplier of G is still a probability distribution over x. It equals 0 if $G(x)$ is in the worst $1 - \kappa$ percentile (according to distribution q) of G values, and κ^{-1} otherwise. So under this replacement the $E(G)$ term in the Lagrangian equals the average of G restricted to that lower κ'th percentile. For $\kappa = K^{-1}$, our new Lagrangian forces attention in setting q on that outlier likely to come out of the K-fold sampling of $q(G)$.

As usual, one can use gradient descent and Monte Carlo sampling to minimize this Lagrangian, taking care to account for q's now appearing twice in the integrand of the $E(G)$ term. Note that the Monte Carlo process includes sampling the probability distribution $\frac{\Theta[\kappa - \int dx'\ q(x')\Theta[G(x)-G(x')]]}{\kappa}$ as well as the q_i. This means that only those points in the best κ'th percentile are kept, and used for all Monte Carlo estimates. This may cause greater noise in the Monte Carlo sampling than would be the case for $\kappa = 1$.

As an example, say that for agent i, all of its moves have the same value of $E(G \mid x_i)$, and similarly for agent j, and say that G is optimal if agents i and j both make move 0. Then if we modify the updating so that agent i only considers the best values that arose when it made move 0, and similarly for agent j, then both will be steered to prefer to make move 0 to their alternatives. This will cause them to coordinate their moves in a way that improves the Lagrangian.

A similar modification is to replace G with $f(G)$ in the Maxent Lagrangian, for some concave nowhere-decreasing function $f(.)$. This would distort G to accentuate those x's with good values. Intuitively, this will have the effect of coordinating the updates of the separate q_i at the end of the block, in a way to help lower G. The price paid for this is that there will be more variance in the values of $f(G)$ returned by the Monte Carlo sampling than those of G, in general.

Note that if q is a local minimum of the Lagrangian for G, in general it will not be a local minimum for the Lagrangian of $f(G)$ (the gradient will no longer be zero under that replacement, in general). So we can replace G with $f(G)$ when we get stuck in a local minimum, and then return to G once q gets away from that local minimum. In this way we can break out of local minima, without facing the penalty of extra variance. Of course, none of these advantages in replacing G with $f(G)$ hold for algorithms that directly search for an x giving a good $G(x)$ value; x is a local minimum of $G(x) \Leftrightarrow x$ is a local minimum of $f(G(x))$.

An even simpler modification to the $E(G)$ term than those considered above is to replace $G(x)$ with $\Theta[G(x) - K]$. Under this replacement the $E(G)$ term becomes the probability that $G(x) > K$. So minimizing it will push q to x with lower G values. For this modified Lagrangian, the gradient descent update step

adds the following to each $q_i(x_i)$:

$$\alpha[\beta q(G < K \mid x_i) \quad + \quad \ln(q_i(x_i)) \; - \; \frac{\sum_{x'_i} \beta q(G < K \mid x'_i) + \ln(q_i(x'_i))}{\sum_{x'_i} 1}].$$

(11)

In gradient descent of the Maxent Lagrangian we must Monte Carlo estimate the expected value of a real number (G). In contrast, in gradient descent of this modified Lagrangian we Monte Carlo estimate the expected value of a single bit: whether G exceeds K. Accordingly, the noise in the Monte Carlo estimation for this modified Lagrangian is usually far smaller. In addition, just like in descent of the Maxent Lagrangian, the Monte Carlo estimation for Eq. 11 is well-suited to a distributed implementation.

In all these variants it may make sense to replace the Heaviside function with a logistic function or an exponential. In addition, in all of them the annealing schedule for K can be set by periodically searching for the K that is (estimated to be) optimal, just as one searches for optimal coordinate systems [19, 1]. Alternatively, a simple heuristic is to have K at the end of each block be set so that some pre-fixed percentage of the sampled points in the block go into our calculation of how to update q.

Yet another possibility is to replace $E(G)$ with the κ'th percentile G value, i.e., with the K such that $\int dx' \, q(x')\Theta(G(x') - K) = \kappa$. (To evaluate the partial derivative of that K with respect a particular $q_i(x_i)$ one must use implicit differentiation.)

2.4 Heuristics for improving the update rules

There are a number of practical issues common to all the schemes elaborated above. The update rules given above are all completely distributed, in the sense that each agent's update at time t is independent of any other agents' update at that time. Typically at any t each agent i knows $q_i(t)$ exactly, and therefore knows $\ln[q_i(j)]$. However those update rules all involve conditional expectation values which often cannot be evaluated in closed form. As described above, one can circumvent this problem by having the expectation values be simultaneously estimated by all agents via repeated Monte Carlo sampling of q to produce a set of $(x, G(x))$ pairs. Those pairs are used by each agent i to estimate the expectation values it needs (e.g., $E(G \mid x_i = j)$), and therefore how to update its distribution.

Consider the case where we do need to use Monte Carlo to estimate conditional expected values of some $f(x)$, and x is high-dimensional. In this scenario block-wise Monte Carlo sampling to estimate conditional expectation values can be slow. The estimates typically have high variance, and therefore require large block size L to get an accurate estimate.

One set of ways to address this is to replace the team game with a non-team game, i.e., for each agent i have it estimate quantities based on a **private utility** g_i rather than G (e.g., based on $E(g_i \mid x_i = j)$ rather than $E(G \mid x_i = j)$

[6]. Each such private utility is chosen so that the Monte Carlo estimates have much lower variance than those based on G, without having any bias [1, 13].

As an example, say we are doing gradient descent of the Maxent Lagrangian. Replace the values of $G(x)$ recorded by agent i in the Monte Carlo process with the values of $g_i(x) = G(x) - D(x_{-i})$, where $D(x_{-i}) \propto \int dx'_i \; w(x'_i) G(x'_i, x_{-i})$ for weighting factors w_i determined by how frequently x'_i arose in the Monte Carlo process. This replacement speeds the convergence of the Monte Carlo process to accurate estimates of the true expectation values $E(G \mid x_i)$ [1]. Furthermore it can often be done with minimal communication overhead between the agents. Indeed, often it is easier to evaluate such a $g_i(x)$ than $G(x)$. The worst case is where $G(x'_i, x_{-i})$ must be explicitly re-evaluated for each of the possible x'_i. Even there though, those extra re-evaluations are often not a large extra expense. This is because they can be used to augment the Monte Carlo samples of values of $g_i(x^*_i)$ for $x^*_i \neq x_i$ as well as those for $x^*_i = x_i$.

Another useful technique is to allow samples from preceding blocks to be re-used. One does this by first "aging" that data to reflect the fact that it was formed under a different q_{-i} . For example, one can replace the empirical average for the most recent block k,

$$\hat{G}_{i,j}(k) \equiv \frac{\sum_{t=kL}^{kL+L} G(x^t) \delta_{x^t_i, j}}{\sum_{t=kL}^{kL+L} \delta_{x^t_i, j}},$$

with a weighted average of previous expected G's,

$$\frac{\sum_m \hat{G}_{i,j}(m) e^{-\kappa(k-m)}}{\sum_m e^{-\kappa(k-m)}}$$

for some appropriate aging constant κ.[7]

Typically such ageing allows L to be vastly reduced, and therefore the overall minimization of L to be greatly sped up. For such small L though, it may be that the most recent block has *no* samples of some move $x_i = j$. This would mean that $\hat{G}_{i,j}(k)$ is undefined. One crude way to avoid such problems is to simply force a set of samples of each such move if they don't occur of their own accord, being careful to have the x_{-i} formed by sampling q_{-i} when forming those forced samples.

There are numerous other techniques that are useful in practice. For example, typically one must use such techniques to decrease the step size in the descent rules (i.e., gradient descent and Nearest Newton) as one nears the border of $\mathcal{Q}$. Similarly, often the non-descent update rules (e.g., Brouwer) can be improved by making only a partial "step" at each iteration, i.e., by averaging the current q with the q given by the update rule as listed above, rather than by replacing it with that q.

[6]Formally, this means that each agent i has a separate Lagrangian, for example formed from the Maxent Lagrangian by substituting g_i for G. See [19] for the relation of this to bounded rational game theory.

[7]Not all preceding $\hat{G}_{i,j}(m)$ need to be stored to implement this; exponential ageing can be done online using 3 variables per (i, j) pair.

3 Empty bins, uncountable x

There are several circumstances in which naive empirical averaging of Monte Carlo samples to estimate update terms of the form $E(H \mid x_i)$ will not work. For example, consider the simplest situation, in which we have a finite number of agents and a finite move space for each agent. Even in this situation, if there are not enough Monte Carlo samples, it may be that for some potential move of some agent there are no instances in any of the Monte Carlo samples (in any of the blocks) in which that agent made that move. In that case, we cannot use empirical averaging to estimate the associated $E(H \mid x_i)$. As another example, say we have a large (but finite) number of Monte Carlo samples, but some agent has an uncountable number of potential moves. Then that agent will have no samples for almost all of its potential moves.

3.1 Exploiting Supervised Learning

All of these problems can be addressed by exploiting the fact that we are working with a product distribution, in concert with the techniques from the field of supervised learning techniques (i.e., classification and regression)[20], which concern precisely the issue of estimating $E(H \mid x_i)$ from a finite set of Monte Carlo samples. As an example, consider the first problem case mentioned above, in which there a finite number of agents all with a finite number of potential moves, but we have too small a set of Monte Carlo samples to have samples of all moves for all agents. For this scenario each agent i must estimate $E(H \mid x_i)$ for all x_i using a "training set" of Monte-Carlo-generated (x_i, H) pairs that does not extend over all x_i. This is a standard problem in supervised learning [20]. Often it can be addressed by extrapolating from those x_i which did occur in the training set to infer estimates of $E(H \mid x_i)$ for the other x_i. Those estimates can then be used to form the updates.[8]

Similar techniques can be used even when the x_i are uncountable. Moreover, in general a supervised learning fit to the Monte Carlo data is parameterized by a finite set of numbers, and therefore for a finite number of agents those fits can be stored in a finite computer, regardless of the cardinality of the move spaces of the agents. However for uncountable move spaces we have the extra problem of how to store, update, and sample q, which is now a density function rather than a probability distribution.

Fortunately, given the regression $E(H \mid x_i)$, there are several ways to update and sample $q(x)$ without ever explicitly storing the values of $q(x)$ for all possible x. By using such sampling schemes in concert with the regression scheme, we can implement Monte Carlo updating for all three of the problematic scenarios described above. As outlined in this section, the key is to write the update rules in terms of multiplicative update ratios giving the new q in terms of the old one, as in the list presented above.

[8] In general, whenever it can be applied, such extrapolation should also be used to improve the estimates of $E(H \mid x_i)$ for those x_i values that do occur in the training set.

3.2 Uncountable x and finite parameterizations of q

For all of these update rules listed above, when x_i is a compact subset of a Euclidean space, one can still numerically perform the update in the conventional way if the associated probability density function is replaced by a (finite-dimensional) paramaterization of it. The simplest way to do that is, in essence, by binning x_i. This means that agent i now has a finite set of moves, one for each of its bins. The full density function is parameterized by the real numbers giving the probabilities agent i assigns to each of its bins, according to some pre-set rule. One example is where the probability density function has uniform density in each bin (as in Reimann integration). Another is where the density function is linearly increasing/decreasing across each bin, in such a way that the density function is everywhere continuous (as in the trapezoidal rule for integration). Formally, such binning schemes are semi-coordinate transformations [10, 6].

With such a scheme, one first applies supervised learning techniques to the Monte Carlo samples to determine the regression $E(H \mid x_i)$. For each bin j, having borders a_j and b_j, one then numerically computes two integrals:

$$\int_{a_j}^{b_j} dx_i \; q_i^t(x_i) E(H \mid x_i) \quad \text{and} \quad \int_{a_j}^{b_j} dx_i \; q_i^t(x_i).$$

The ratio of those two integrals determines the time-t expected value of H conditioned on x_i being in bin j. (For bins that are thin enough on the scale of variations in the regression and/or $q_i^t(x_i)$, these integrations can be replaced by simply evaluating the integrands at the centers of the bins.) This then gives the expected H conditioned on x_i being in bin j for all bins j. This is precisely what is needed to update of those bins' probabilities, according to whichever of the update rules listed above one is using.

Note that this scheme can be done even when the number of bins is far larger than the number of Monte Carlo samples. This contrasts with the case of estimating the conditional expectation value of H given bin j based only on averaging of all the Monte carlo samples that fall in that bin. Intuitively, using regression allows samples from neighboring bins to be used to help form the estimate.

While some binning schemes can be quite sophisticated, sometimes it would be advantageous to use a different parameterization. Often this can be done in a way that replaces the regression algorithm with a density estimation algorithm, using the usual Bayesian equivalence of regression and density estimation. For example, choose the masking function $F_G(x)$ in Eq. 10 to be $\Theta(K - G(x))$, Evaluating such an update based on a set of Monte Carlo samples can be done with conventional probability density estimation algorithms [20]. One simply collects the subset of the samples for which $G(x) < K$, and runs the density estimation algorithm on those points to estimate the density at x_i.

Intuitively, in this approach the Monte Carlo samples encode the probability density function q_i. For a smooth density estimator, this scheme will also ensure $q_i(x_i) \neq 0 \; \forall x_i$, thereby mitigating the problem that a statistical fluctuation of never picking x_i in some Monte Carlo block would guarantee it is never picked

in the future. Similar schemes can be used for non-step function choices of F_G. For example, one can use the value $F_G(x)$ for each x in the Monte Carlo sample as a weighting factor for that sample in a kernel density estimator.

3.3 Parameterless sampling

One problem with parametric schemes like these is that since q is given by a set of explicitly stored real numbers, one is always limited in how finely one can capture q by the finiteness of one's computer's memory. More importantly, if one has many parameters (to capture q with high accuracy), then updates can be computationally expensive, since each parameter has to be updated in each iteration. For example, with binning, one has to go through the update rule for each bin.

Alternative schemes use the regression $E(H \mid x_i)$ to apply any multiplicative update rule for uncountable x without any finite-dimensional parameterization of q. With such schemes, in each step the full density function given by an uncountable number of real numbers is implicitly updated (e.g., via gradient descent). However that density is never explicitly represented. Instead, all we ever explicitly do is sample it, potentially evaluating it at a finite number of points to do so. Intuitively, via our regression, the Monte Carlo samples themselves serve as our "parameterization" of $q(x)$.

Define $R_{q,i} \equiv \max_{x_i} r_{q,i}(x_i)$. Then for any $t > 1$ we can generate a sample from q_i^t if we can implement the following three-step procedure:

1) Sample from q_i^{t-1} to get a point x_i.

2) Toss a coin with probability of heads

$$\frac{r_{q^{t-1},i}(x_i)}{R_{q^{t-1},i}}. \tag{12}$$

3) If the coin came up heads, keep our x_i as the desired sample of q_i^t. Otherwise return to (1).[9]

Note that this scheme will also work if we can only evaluate the values $r_{q^{t-1},i}(x_i)$ up to an overall proportionality constant, so long as $R_{q^{t-1},i}$ is redefined to include that constant. Similarly the scheme will work so long as $R_{q^{t-1},i}$ in step (2) is replaced by any fixed quantity that is bounded below by the actual $R_{q^{t-1},i}$. So in practice we can set that value in step (2) to some small factor multiplied by the maximal value of over some set of values x_i' of $r_{q^{t-1},i}(x_i')$. Accordingly we can sample q^t if we can sample q^{t-1}, can evaluate $A \times r_{q^{t-1},i}(x_i)$

[9] Proof: Since this three-step sub-sampling scheme is a stochastic process, it generates x_i's according to some distribution $\pi(x_i)$. So to prove that $\pi = q_i^t$, it suffices to note that for any two values x_i, x_i', $\frac{\pi(x_i)}{\pi(x_i')} = \frac{q_i^t(x_i)}{q_i^t(x_i')}$. **QED**

for any particular x_i and fixed (though perhaps unknown) constant A, and can evaluate a lower bound on $A \times R_{q^{t-1},i}$.

Performing this three-step procedure for all agents will give us a sample of the joint distribution q^t. We then add that joint sample to the training set and form a new regression (to be able to calculate $r_{q^t,i}(x_i)$). We then use that new regression to find a lower bound on $R_{q^t,i}$. This allows us to repeat the three steps, and thereby form the next update to q. Generalizing, if we set q^1 to some easily sampled distribution (e.g., the uniform distribution), and can always perform the stipulated regressions, then with our three-step procedure we have an iterative algorithm for sampling $q^t(x) = \prod_i q_i^t(x)$ for all t. Then, at the end of the run, we use the final joint samples as guesses for the solution x to our optimization problem.

Say we are at iteration t, having formed samples of all of the $q_i^{t'}$ for $t' < t$ via the three-step procedure, and therefore having been able to evaluate $R_{q^{t'},i}$ and $r_{q^{t'},i}(x_i)$ for any $x_i, t' < t$. To employ the precise scheme outlined above to sample q_i^t we would first sample q_i^1, and then send that sample through t successive stochastic keep/reject steps. The probability of a rejection at each step in that chain is given by how small the ratio $\frac{r_{q^{t-1},i}(x_i)}{R_{q^{t-1},i}}$ is for typical x_i generated by sampling q_i^1. For large enough t, even if the rejection probability for each step in the chain is small, the probability of a rejection somewhere along such a chain — followed by starting all over with a new sample of q_i^1 — may be quite high. Accordingly, this three-step procedure might take a long time to actually generate the desired sample of q_i^t.

As an alternative, note that by hypothesis we can evaluate $r_{q^{t'},i}(x_i)\ \forall x_i,\ t' < t$, up to a t'-dependent overall proportionality constant, which without loss of generality we set to 1. So write

$$q_i^t(x_i) \quad = \quad q_i^1(x_i) \prod_{t'=1}^{t-1} r_{q^{t'},i}(x_i)$$

As long as we are sure that the product on the righthand side is finite and never negative, we can employ a modified version of our sub-sampling procedure. To do this define

$$c_i(\{q^{t'} : t' < t\}, x_i) \quad \equiv \quad \prod_{t'=1}^{t-1} r_{q^{t'},i}(x_i). \tag{13}$$

Assuming we can evaluate $r_{q^{t'},i}(x_i)\ \forall, x_i, t' < t$, we can evaluate $c_i(\{q^{t'} : t' < t\}, x_i)\ \forall x_i$. Next define

$$C_i(\{q^{t'} : t' < t\}) \quad \equiv \quad \max_{x_i} c_i(\{q^{t'} : t' < t\}, x_i). \tag{14}$$

In analogy to the earlier case, we can form an estimate of a (conservative lower bound) on $C_i(\{q^{t'} : t' < t\})$ by evaluating $c_i(\{q^{t'} : t' < t\}, x_i)$ for many x_i.

As before, the first step of our procedure is to sample q_i^1 to produce a suggested sample of q_i^t. We then accept that suggested sample with probability

$$\frac{c_i(\{q^{t'} : t' < t\}, x_i)}{C_i(\{q^{t'} : t' < t\})},$$

resampling q_i^1 if we reject the suggested sample. This gives us our desired sample of $q_i^t(x_i)$. Doing this for all i then gives our sample of $q^t(x)$.

Exactly as before, such a sample of q^t can be combined with our previous Monte Carlo samples to provide a training set for a supervised learning algorithm that forms a regression $E_{q^t}(H \mid x_i)$. We can use that to evaluate $r_{q^t,i}(x_i)$ for any x_i, up to an overall proportionality constant. So we can evaluate the product $c_i(\{q^{t'} : t' < t+1\}, x_i)$ for a large number of x_i, and thereby estimate (a lower bound on) $C_i(\{q^{t'} : t' < t\})$. This then allows us to generate a sample of the next distribution q^{t+1} by using subsampling. So we again have an iterative algorithm. However this one avoids the need for more than one keep/reject step in forming the sample of q^t for any t. (The price paid for this is a more expensive numerical evaluation of the associated max.)

3.4 Including density estimation

A remaining potential difficulty is that as q_i^t gets more and more peaked, we might get a lot of rejections when we subsample, since the ratio $\frac{c_i(\{q^{t'}:t'<t\},x_i)}{C_i(\{q^{t'}:t'<t\})}$ will be very small for almost every point formed by sampling q_i^1. More generally, if we are only generating candidate x_i by examining points generated by sampling q_i^1, then we won't have reduced the overall computational burden in finding x with low G values compared to the simple process of sampling q_i^1 without any subsequent subsampling.

We can address this problem by periodically using a density estimation algorithm to produce an estimate of the current distribution, an estimate that is easy to sample. We don't use that estimate directly instead of q_i^t however, since it won't exaclty equal q_i^t in general. Instead, we use it as a proposal distribution in importance sampling from q_i^t. More precisely, at time step t, say we run a density estimation algorithm on our Monte Carlo samples to form a density $\hat{q}_i^t(x_i)$ that both can be easily sampled and with high probability is a good approximation to q_i^t. Write

$$q_i^t(x_i) \quad \propto \quad \hat{q}_i^t(x_i)\, d_i(\{q^{t'} : t' < t\}, \hat{q}_i^t, x_i) \tag{15}$$

where

$$d_i(\{q^{t'} : t' < t\}, \hat{q}_i^t, x_i) \quad \equiv \quad \frac{q_i^1(x_i) c_i(\{q^{t'} : t' < t\}, x_i)}{\hat{q}_i^t(x_i)}. \tag{16}$$

Then define

$$D_i(\{q^{t'} : t' < t\}, \hat{q}_i^t, i) \quad \equiv \quad \max_{x_i} d_i(\{q^{t'} : t' < t\}, \hat{q}_i^t, x_i). \tag{17}$$

As usual, without loss of generality we can ignore any overall proportionality constants in the evaluations of $\hat{q}_i^t(x_i)$ and/or $c_i(\{q^{t'} : t' < t\}, x_i)$ (so long as the same constants appear in the evaluation of $D_i(\{q^{t'} : t' < t\}, \hat{q}_i^t, i)$), and can replace the constant $D_i(\{q^{t'} : t' < t\}, \hat{q}_i^T, i)$ with a lower bound on it.

In the first step of the new version of our three-step procedure we generate a sample of $\hat{q}_i^t(x_i)$. (In the original three-step procedure the analogous step was to sample $q^1(x_i)$.) We then keep that sample with probability

$$\frac{d_i(\{q^{t'} : t' < t\}, \hat{q}_i^t, x_i)}{D_i(\{q^{t'} : t' < t\}, \hat{q}_i^t, i)},$$

forming a new sample if the suggested sample is rejected. In this way we can exactly sample the density function $q_i^t(x_i)$. Moreover, assuming our density estimate is reasonably accurate, and that our lower bound on $D_i(\{q^{t'} : t' < t\}, \hat{q}_i^t, i)$ is not too much lower than the actual value, the ratio giving our acceptance frequency will not be too small.

In practice, we may want to exploit algorithms that combine the generation of $\hat{q}_i^t$ from the training set and the sampling of that distribution. As an illustration, say x_i is the set of real numbers between 0.0 and 1.0, and write the cumulative distribution function of q_i^t as $CDF_{q_i^t}$. Then one way to form a sample of $q_i^t(x_i)$ is to generate a point ξ_i by uniformly sampling [0.0, 1.0], and then return the value $[CDF_{q_i^t}]^{-1}(\xi_i)$. This suggests an algorithm in which we first use our training set of Monte Carlo samples to form $\hat{CDF}_{q_i^t}$, an estimate of $CDF_{q_i^t}$. We then sample [0.0, 1.0] uniformly to produce ξ_i, and return the value $[\hat{CDF}_{q_i^t}]^{-1}(\xi_i)$.

As an example of a rough, fast way to do this, say there are N separate x_i values in our training set, the set of those values being written as $\{x_i^j\}$. Define $I(x_i^j)$ as the interval of all real numbers that are closer to x_i^j than to any other training set element. Also define the function $\text{int}(x_i)$ as the greatest integer below x_i. Then our algorithm for sampling (an estimate of) $q_i^t(x_i)$ would consist of the following steps:

A) Sample [0.0, 1.0] uniformly to generate ξ_i.

B) Sample uniformly from within the interval $I(x_i^{\text{int}(N\xi_i)+1})$.

Similar schemes can be used when x_i is more than one-dimensional, for example by substituting Voronoi cells about training set x_i values for intervals about them.

Say we are able to sample $q_i^t(x_i)$ exactly, either by using subsampling of points generated from q_i^1 or by using a density estimate $\hat{q}_i^t$. Then we can use the original three-step procedure recounted above to sample $q_i^{t+1}(x_i) = q_i^t(x_i) r_{q^t,i}(x)$. Similarly, to sample $q_i^T(x_i)$ for subsequent $T > t+1$, we can use the modified version of the three-step procedure based on a product of $r_{q^{t'},i}(x_i)$'s. In the current context, this means we sample $q_i^t(x_i)$, and then

keep/reject those samples according to the ratios

$$\frac{c_i(\{q^{t'} : t < t' < T\}, x_i)}{C_i(\{q^{t'} : t < t' < T\})}.$$

Once T is so much larger than t that we start getting a lot of rejections, we can rerun our density estimation algorithm.

3.5 Performing the needed evaluations

Say we are at the t'th iteration, and assume we already have a full Monte Carlo sample of q^{t-1}. We need to be able to evaluate $r_{q^{t-1},i}(x_i)$ to form a sample of q^t using our three-step procedure. We can do this for any of the update rules listed above, so long as can calculate $\ln(q_i^{t-1}(x_i))$, $E_{q^{t-1}}(\ln(F_G) \mid x_i)$, $E_{q^{t-1}}(\ln(F_G))$, $S(q_i^{t-1})$, $\int dx_i \; E_{q^{t-1}}(\ln(F_G) \mid x_i)$, and $\int dx_i \; \ln(q_i^{t-1}(x_i))$. The first two of these depend on x_i, and the last four are averages over all x_i.[10]

We can perform our calculations as follows:

i) Assume that we can evaluate $q_i^{t-1}(x_i) \; \forall x_i$, just as we are able to sample q^{t-1}. That takes care of the first term.

ii) As usual, to estimate $E_{q^{t-1}}(\ln(F_G) \mid x_i)$, apply any handy supervised learning algorithm (e.g., Gaussian nearest neighbor averaging) to the training set of $(x_i, \ln(F_G(x)))$ pairs given by the Monte Carlo samples of q^{t-1}.

iii) Given our supervised learning algorithm, use numerical integration to estimate $\int dx_i \; E_{q^{t-1}}(\ln(F_G) \mid x_i)$. In practice, if the integrand is quite peaked, it may make sense to assist the integration by first using a density estimation algorithm to form an easily sampled estimate of $q_i^t(x_i)$. Numerical integration via importance sampling can then be used with that estimate as the proposal distribution to do the integration. Assuming the peaks of $q_i^t(x_i)$ roughly matches those of $E_{q^{t-1}}(\ln(F_G) \mid x_i)$, such integration should be relatively efficient.

Given our assumed ability to evaluate $q_i^{t-1}(x_i) \; \forall x_i$, we can similarly use our supervised learning algorithm and numerical integration to estimate $E_{q^{t-1}}(\ln(F_G))$, again using density estimation if need be. Alternatively, we can estimate $E_{q^{t-1}}(\ln(F_G))$ simply by averaging the values in the training set of $\ln(F_G)$.[11]

iv) Similarly, we can use numerical integration to estimate $\int dx_i \; \ln(q_i^{t-1}(x_i))$. and/or $S(q_i^{t-1})$. A simpler approach to estimating the entropy, analogous to the

[10] Those last four arise in calculating the additive const term in one or the other of the update rules, and where needed implicitly assume *a priori* bounds on their integrals. Note that any multiplicative const terms are irrelevant, since as described above they cancel out in the subsampling.

[11] It probably makes most sense to do this if our supervised learning algorithm is one for which we're *a priori* guaranteed that such averaging gives the same answer as numerical integration.

averaging process of step (iii), is to simply estimate the entropy as the empirical average of the values of $\ln[q_i(x_i)]$ over the Monte Carlo samples.

Doing all this, we can evaluate every term that arises in our subsampling procedure. This allows us to sample q^t. For the next iteration, this ability to sample q^t is used again, this time to do part (1) of the subsampling procedure for generating samples of q^{t+1}. To perform parts (2) and (3) we need to evaluate the update ratio, and therefore must be able to perform (some subset of) steps (i) through (iv) above. Since by hypothesis we can evaluate $q^{t-1}(x)$ for any x, and can evaluate the update ratio $r_{q^{t-1},i}(x_i)$ (up to irrelevant proportionality constants) for any such x_i, we can evaluate $q^t(x)$ for any x Therefore we can perform step (i). We can also perform steps (ii) through (iv) using the Monte Carlo samples of q^t. Therefore we can perform parts (2) and (3) of our subsampling procedure. So we can generate a sample of q^{t+1}.

4 Conclusion

A long-running difficulty with conventional game theory has been how to modify it to accommodate the bounded rationality characterizing all real-world players. A recurring issue in statistical physics is how best to approximate joint probability distributions with decoupled (and therefore far more tractable) distributions. It has recently been shown that the same information theoretic mathematical structure, PC, underlies both issues. This structure provides a formal model-independent definition of the degree of rationality of a player and of bounded rationality equilibria. This pair of papers extends previous work on PC by introducing new computational approaches to effectively find bounded rationality equilibria of common-interest (team) games.

References

[1] D. H. Wolpert, "Factoring a canonical ensemble," 2003, cond-mat/0307630.

[2] ——, "Bounded rationality game theory and information theory," 2004, submitted.

[3] W. Macready, S. Bieniawski, and D. Wolpert, "Adaptive multi-agent systems for constrained optimization," 2004, technical report IC-04-123.

[4] C. F. Lee and D. H. Wolpert, "Product distribution theory for control of multi-agent systems," in *Proceedings of AAMAS 04*, 2004.

[5] S. Bieniawski and D. H. Wolpert, "Adaptive, distributed control of constrained multi-agent systems," in *Proceedings of AAMAS04*, 2004.

[6] S. Bieniawski, D. H. Wolpert, and I. Kroo, "Discrete, continuous, and constrained optimization using collectives," in *Proceedings of 10th*

AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Albany, New York, 2004, in press.

[7] D. H. Wolpert, "Finding bounded rational equilibria part i: Iterative focusing," in *Proceedings of the International Society of Dynamic Games Conference, 2004*, 2004, in press.

[8] S. Airiau and D. H. Wolpert, "Product distribution theory and semi-coordinate transformations," 2004, submitted to AAMAS 04.

[9] D. H. Wolpert and S. Bieniawski, "Distributed control by lagrangian steepest descent," in *Proceedings of CDC04*, 2004.

[10] ——, "Adaptive distributed control: beyond single-instant categorical variables," in *Proceedings of MSRAS04*, A. S. et al, Ed. Springer Verlag, 2004.

[11] N. Antoine, S. Bieniawski, I. Kroo, and D. H. Wolpert, "Fleet assignment using collective intelligence," in *Proceedings of 42nd Aerospace Sciences Meeting*, 2004, aIAA-2004-0622.

[12] D. H. Wolpert and C. F. Lee, "Adaptive metropolis hastings sampling using product distributions," in *Proceedings of ICCS 04*, 2004.

[13] D. H. Wolpert, "What information theory says about best response, binding contracts, and collective intelligence," in *Proceedings of WEHIA04*, A. N. et al, Ed. Springer Verlag, 2004.

[14] D. H. Wolpert, K. Tumer, and J. Frank, "Using collective intelligence to route internet traffic," in *Advances in Neural Information Processing Systems - 11.* MIT Press, 1999, pp. 952–958.

[15] D. H. Wolpert and K. Tumer, "Optimal payoff functions for members of collectives," *Advances in Complex Systems*, vol. 4, no. 2/3, pp. 265–279, 2001.

[16] ——, "Collective intelligence, data routing and braess' paradox," *Journal of Artificial Intelligence Research*, 2002.

[17] D. H. Wolpert, "Theory of collective intelligence," in *Collectives and the Design of Complex Systems*, K. Tumer and D. H. Wolpert, Eds. New York: Springer, 2003.

[18] D. Wolpert, K. Tumer, and E. Bandari, "Intelligent coordinates for search," 2002, submitted.

[19] D. H. Wolpert, "Information theory — the bridge connecting bounded rational game theory and statistical physics," in *Complex Engineering Systems*, A. M. D. Braha and Y. Bar-Yam, Eds., 2004.

[20] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd ed.)*. Wiley and Sons, 2000.