Heterogeneity, Root-finding, and Decentralization

James D Thomas [*]
Dept of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
jthomas+@cs.cmu.edu

Katia Sycara
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
katia+@ri.cmu.edu

Abstract

Increasing interest in agent-based systems both in AI (in terms of multiagent systems and distributed artificial intelligence) and in computational approaches to the social sciences calls for a greater understanding of their computational properties. Often, problems that are difficult for agent-based systems because of issues of coordination or delayed information could be solved using a centralized root-finding algorthim. This paper proposes a method that allows for the approximation of such centralized root-finding solutions by systems of decentralized agents. By making each agent's responses to payoff signals heterogeneous, proper coordination emerges among the agents without the need for communication or negotiation. We apply this technique to both Newton's method and Gallager's algorithm for multicommodity flow. In empirical simulations, the method is shown to produce results comparable to the original algorithm.

1 Introduction

The computational process of root finding underpins many systems of interest to distributed computation. Root finding algorithms take an equation $f(x)$ and find the x such that $f(x) = 0$, usually by incrementally adjusting x closer and closer to the true root. In economics, the existence of general equilibrium [1] depends on finding an allocation such that the excess demand function is zero. In many resource allocation problems, for example multicommodity flow [2], the optimal allocation is one that equalizes the marginal costs of the resources (the derivative of total costs with respect to the usage of each individual resource); this is accomplished by pairwise subtracting the marginal costs of resources from each other, and somehow finding the allocation that sets all these functions to zero, insuring equality in the appropriate terms.

The properties of root finding algorithms are well understood. Numerical methods such as Newton's method have been developed for their quick convergence [7], and more application oriented methods such as Gallager's algorithm for multicommodity flow [2] have proven convergence results in specific settings. Unfortunately, these techniques are difficult to apply in decentralized settings where decision making is spread across autonomous agents instead of under central control. Multicommodity flow algorithms depend on conceptualizing flow as passive, divisible entity under centralized control instead of as an aggregation of autonomous agents.

An example that highlights this difficulty is the computational ecosystem model developed by Huberman, Hogg, and their colleagues at Xerox Parc [6],[4], a simple model of distributed resource allocation, where individual agents choose between two resources on the basis of which resource has a higher payoff. Equilibrium is found when the payoffs from the two resources are equal. If the agents were under central control, this would be a simple root finding problem; however, the fact that all the agents make their decisions independently of each other produces unstable oscillations. The essential problem is one of coordination; it is difficult to get the number of agents using a given resource to adjust incrementally, since once they see that the other resource is paying more, they *all* want to switch to it. This leads to instability and failure to converge.

Huberman & Hogg [4] first explored the idea of exploiting agent heterogeneity to stabilize the system. Thomas & Sycara [9] expanded on that insight. By slightly simplifying the model, they were able to prove that while some forms of agent heterogeneity guaranteed stability at the cost of converging to a sub-optimal equilibrium, heterogeneity in switching costs (ie, giving each agent a unique, fixed threshold that the payoff difference between the resources must exceed before the agent will switch) was shown to produce guaranteed convergence to the true equilibrium of the system. The aggregate behavior of the resulting system looked very much like a root finding algorithm.

This paper takes this line of research one step further by reversing the design flow: we develop a technique that starts with a traditional root finding algorithm, and allows it to be implemented in a system of decentralized agents.

The importance of this technique is twofold. First, when we must solve a problem with a system of agents, it allows us to design a system that exactly mimics the behavior of existing serial algorithms. Second, it allows us to view many problems in a new light. In addition to the computational ecosystem model [6], the approach potentially offers new insight into the aggregate behavior of any system that models the decisions of agents that must make 'all or nothing' de-

[*]Thanks to Matt Glickman, Bryan Routledge, John Miller, Onn Shehory, and Somesh Jha for helpful discussion. Sponsored by NSF grant IRI-9612131 and ONR grant N-00014-96-1-1222

cisions in the face of payoff signals, both in social science inspired models such as the computational Tiebout models [5] and traditional computer science problems like load balancing [8].

The organization of the paper is as follows. First, we introduce the technique by applying it to Newton's method, a canonical root finding method. We then take Gallager's solution to the multicommodity flow problem [2] and show how the technique allows us to mimic his algorithm.

2 Decentralizing Root Finding

Root finding algorithms [7] are a core part of the numerical computation toolbox; they are used to find the roots of unknown functions. In the example of multicommodity flow discussed above, in order to minimize global shipping times we need to find flow levels that equalize the marginal congestion per unit of flow between competing links. Sometimes the equations can be solved analytically; often they are too complicated and the solutions must be estimated numerically. This is done by creating a new function formed by subtracting one marginal cost from the other; if we can find the root of this equation, the x such that $f(x) = 0$, we know that the two marginal cost terms are equivalent.

The convergence properties of traditional root-finding methods are well studied, and we will not discuss them here. The goal of this paper is to to mimic existing algorithms in a decentralized format, and so the behavior of the traditional algorithm is approximated (modulo the granularity forced on us by a finite number of discrete agents) by the decentralized system.

Our methodology will be as follows. We will describe a root finding method, and then show its equivalence to an idealized decentralized system with an infinite continuum of agents. Normally, root finding works over arbitrary domains, but in order to deal with the infinite number of agents, we normalize the domain to the interval $[0,1]$. In our continuum of agents, we let the agents be either 'on' or 'off', and denote the proportion of agents turned 'on' at time t by $x(t)$. The 'on' agents receive a payoff equal to $f(x(t))$, and the 'off' agents receive a payoff of zero.

2.1 Newton's Method

Newton's method depends on knowing the derivative of the function in question; there are other algorithms that deal with cases where such information is unavailable. The equations developed here generalize easily to other techniques with little difficulty.

Newton's method starts at a point $x(t)$. It calculates the derivative $f'(x(t))$, and extends the tangent line from $x(t)$, and lets $x(t+1)$ equal the point where the tangent line crosses the x-axis. In situations where Newton's method is well behaved, it converges quadratically.

The basic equation describing Newton's method is:

$$x(t+1) \quad = \quad x(t) - f(x(t))/f'(x(t))$$

The term $f(x(t))/f'(x(t))$ represents the distance from $x(t)$ to the point of zero crossing of tangent line. Let us denote it by $d(x(t))$. Now we show how to get a decentralized system to mimic this equation. We start with the math, and try to gain intuition and explanation later. Our approach will be to start with the simplest agent system we can think of and 'fill in the gaps' between its behavior and the behavior we want. We start with agents that make the simplest decisions possible: if they see that $d(x(t)) < 0$, they know that x needs to increase, and all the agents currently off $(1 - x(t))$ all switch on; if $d(x(t)) > 0$, all the on agents switch off. This gives us the following equations (we assume without loss of generality that $f(x)$ is monotonically decreasing):

$$\begin{array}{rclcll} x(t+1) &=& x(t) + (1 - x(t)) &=& 1 & d(x(t)) < 0 \\ x(t+1) &=& x(t) & & & d(x(t)) = 0 \\ x(t+1) &=& x(t) - x(t) &=& 0 & d(x(t)) > 0 \end{array}$$

Clearly, this won't work; the system just switches back and forth between $x(t) = 1$ and $x(t) = 0$. The key is to limit the proportion of agents who switch. Understanding how heterogeneity accomplishes this is the core of the technique. If the agents are identical, and all see the same value of $f(x)$, it is unclear how they could coordinate their actions so that the right number of agents switched. But by forcing the agents to have heterogeneous responses to the same signals (or by sending heterogeneous signals to identical agents), we can force exactly the right number to switch. We will calculate how to do this by introducing a term, $P_{x(t)}$, which stands for the proportion of agents that move each turn. We can now rewrite the above equations as:

$$\begin{array}{rcll} x(t+1) &=& x(t) + (1 - x(t)) * P_{x(t)} & d(x(t)) < 0 \\ x(t+1) &=& x(t) & d(x(t)) = 0 \\ x(t+1) &=& x(t) - x(t) * P_{x(t)} & d(x(t)) > 0 \end{array}$$

For the $d(x(t)) < 0$ case, since we want $(1 - (x(t))) * P_{x(t)} = -d(x(t))$, setting $P_{x(t)}$ equal to $-d(x(t))/((1-x(t))$ will give the correct behavior; for $d(x(t)) > 0$, $P_{x(t)} = d(x(t))/x(t)$. Given these values for $P_{x(t)}$, we have (in the $f(x(t)) > 0$, $d(x(t)) < 0$ case):

$$\begin{array}{rcl} x(t+1) &=& x(t) + (1 - x(t)) * P_{x(t)} \\ &=& x(t) + (1 - x(t)) * -d(x(t))/(1 - x(t)) \\ &=& x(t) - d(x(t)) \end{array}$$

which is exactly the behavior Newton's method requires. The $d(x(t)) > 0$ case follows almost identically.

The key question now is how to understand and implement these $P_{x(t)}$ terms, which represent the proportion of agents who need to switch, in a system of decentralized agents. How we do this depends on the design of the particular system we are building. If we can control the agent design, one way to accomplish this is to give each agent a different 'switching threshold' – a parameter drawn from a uniform distribution over the interval $[0,1]$. We could then globally broadcast the $P_{x(t)}$, and an agent would only switch if the $P_{x(t)}$ term exceeded the switching threshold. Since each agent has a different threshold, some will switch and some won't. For an arbitrary agent, the probability of switching is equal to $P_{x(t)}$ being less than some number randomly drawn from a uniform distribution over $[0,1]$, or simply $P_{x(t)}$ itself. If we aggregate this behavior over many agents, each with its own switching threshold, the expected proportion of agents who switch is exactly $P_{x(t)}$.

Alternately, the values for $f(x)$ and $f'(x)$ could be broadcast, and the agents could perform the calculations themselves. There is something unsatisfying in that this method still requires the computation of the $P_{x(t)}$ terms. This is unavoidable, if the exact properties of Newton's method are to be duplicated; however, in the next example, we will see that we can make simplifying assumptions that sacrifice some performance for gain in simplicity and intuitive resonance.

Note that this technique can be easily extended to other root-finding methods. For the secant method, which replaces the $f'(x)$ term with a linear interpolation $(x(t)-x(t-1))/(f(x(t))-f(x(t-1)))$, the calculations follow identically.

3 Decentralizing Gallager's Multicommodity Flow Algorithm

Multicommodity flow is a problem that lends itself well to interesting decentralization approaches. A multicommodity flow problem consists of a set of locations connected by directed links. There are batches of cargo that must be shipped between the locations. The problem is to route the cargo so that total shipping time is minimized; since the links grow congested with overuse, this often means splitting each batch of cargo over different routes.

Gallager [2] described an algorithm that distributed the computation across locations. Essentially his solution estimates the marginal cost (where cost meant total shipping time across the whole system) of transport along each link and incrementally adjusts flows across competing links until the marginal costs were equalized. Wellman [10] took that one step further by his application of the Walras algorithm, a distributed optimization method based on general equilibrium theory. Wellman created agents whose economic interests incorporated the goals and constraints of the problem, and the marginal cost estimates and flow allocations emerge out of the equilibrium produced by adjustments in supply and demand.

This paper explores a different kind of decentralization, that of letting the flow consist of discrete, autonomous agents. We will assume (as Gallager did) that the marginal costs are already estimated – this contrasts to Wellman [10], where the estimation of costs emerged out of the market equilibrium. We will use Gallager's algorithm, and show how to implement it with discrete agents.

Gallager's algorithm is as follows: let $m_{i,j}(k)$ be the marginal cost (in terms of global shipping time) to send a unit of flow destined for node k between node i and node j. Let $t_i(k)$ represent the total flow destined for k passing through node i, and $x_{i,j}(k)$ represent the proportion the total flow out of node i destined for node k that passes over the link from i to j. Finally, let j_{min} be the j such that $m_{i,j}(k)$ is minimized, i.e., the link from i to j with the lowest marginal cost per unit of flow. The basic idea is to increase flow along this link, and decrease it along other competing links.
This change, denoted $\Delta x_{i,j}$ is:

$$\begin{array}{rcll}\Delta x_{i,j} & = & -\eta * (m_{i,j}(k) - m_{i,j_{min}}(k))/t_i(k) & j \neq j_{min} \\ \Delta x_{i,j} & = & \sum_{j \neq j_{min}} \eta * (m_{i,j}(k) - m_{i,j_{min}}(k))/t_i(k) & j = j_{min}\end{array}$$

Thus, the proportion of agents that need to switch for each $i, j (j \neq j_{min})$ link is simply $\eta * (m_{i,j}(k) - m_{i,j_{min}})/t_i(k)$. The similarity to the root finding case should be clear. Again, we can assign each agent a switching threshold uniformly drawn from $[0, 1]$, and broadcast the signal $\eta * (m_{i,j}(k) - m_{i,j_{min}})/t_i(k)$. If agents only switch when their switching threshold is less than the signal, then the expected number of agents who switch will be correct, and the system as a whole will mimic Gallager's algorithm exactly. We could simplify this algorithm by replacing $t_i(k)$ with some large constant T, and multiplying the switching thresholds by T/η; this would slow convergence somewhat, since the step size would be smaller. Under this scheme, the only thing needed to be broadcast to the agents are the marginal costs of each link. This gives us a natural economic interpretation: think in terms of pricing the links by their marginal costs and think of the switching thresholds as heterogeneous transaction costs. Then, agents see the price differences to ship goods along the different links, and they only switch if the price difference exceeds their unique, individual transaction cost.

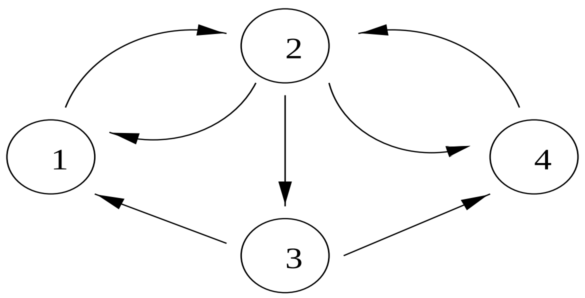

Figure 1: Simple multicommodity flow network

We tested out both of these approximations against Gallager's original algorithm by applying them to a simple multicommodity flow problem and comparing it with the results obtained using Gallager's original algorithm.

We used the network shown in figure 1 (following [3],[10]):
There are 100 units of goods to be shipped from $4 \to 1$, and 100 from $1 \to 4$. The only tricky thing about this network is that at node 2, each flow must split itself between two possible links; and since they both share the $2 \to 1$ link in common, their decisions by necessity effect the other. We used the following cost functions:

$$c_{1,2}(x) = c_{2,1}(x) = c_{2,4}(x) = c_{4,2}(x) = x^2 + 20x$$
$$c_{3,1}(x) = c_{2,3}(x) = c_{3,4}(x) = 2x^2 + 5x$$

Empirically, both of our decentralized systems mimic the properties of Gallager's algorithm well. We set the problem up so that the optimum flows were all integers, so that the granularity introduced by having discrete agents doesn't affect our evaluation of our system's performance (although that graininess is an issue of real concern). Setting $\eta = .1$, Gallager's algorithm settled into the optimal solution after six turns. We ran both of our systems over fifty trials. In the first case, where we attempted to exactly mimic Gallager's system, in every trial the system came within 1% of the true solution in seven turns and .1% in fourteen; for the simplified case, setting $T = 100$, every trial was within 1% in six turns and .1% in fourteen, producing near identical performance. Of course, more detailed empirical investigation is needed, but the preliminary results are promising.

4 Conclusions and future work

We have presented a method for implementing traditional root-finding algorithms in systems composed of discrete, autonomous agents. The key to the technique is understanding what proportion of the agents needs to shift from one resource to another, and setting up heterogeneity (either in the agents or the signals sent to them) that produces that shift. We have applied the technique to Newton's method and Gallager's algorithm for solving multicommodity flow problems. We also showed that with a few simplifications the system derived from Gallager's algorithm has a natural economic interpretation.

There are obvious open empirical questions; since any implementation of the ideas here is necessarily an approximation, understanding the effects of issues such as noise and number of agents can only be done through simulation. In addition, the heterogeneity terms can be implemented in a number of different ways, each undoubtedly with its own set of advantages and disadvantages.

But in the longer term, the understanding of the importance of heterogeneity in agent design shows promise. In computer science, we can use the technique to help us with the design of distributed algorithms by allowing us to lean on existing serial techniques. In computational modelling in social sciences, we have another tool for analyzing the aggregate behavior of heterogeneous agents.

References

[1] Michael D. Whinston Andreu Mas-Colell and Jerry R. Green. *Microeconomic Theory.* Oxford Press, 1995.

[2] Robert G. Gallager. A mimimum delay routing algorithm using distributed computation. *IEEE Trans. Communication*, 25(1):73–85, January 1977.

[3] P. T. Harker. Multiple equilibrium behaviors on networks. *Transportation Science*, 22:39–46, 1988.

[4] Tad Hogg and Bernardo A. Huberman. Controlling chaos in distributed systems. *IEEE Trans. on Systems, Man and Cybernetics*, 21(6):1325–1332, November/December 1991.

[5] John H. Miller Ken Kollman and Scott E. Page. Political institutions and sorting in a tiebout model. *To Appear in American Economic Review*, 1998.

[6] J. O. Kephart, Tad Hogg, and Bernardo A. Huberman. Dynamics of computational ecosystems. *Physical Review A*, 40, 1989.

[7] William H. Press, Saul A. Tukolsky, William T. Vertterling, and Brian P. Flannery. *Numerical Recipies in C.* Cambridge University Press, 1992.

[8] Andrea Schaerf, Yoav Shoham, and Moshe Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, (2):475–500, 1995.

[9] James Thomas and Katia Sycara. Heterogeneity, stability, and efficiency in distributedsystems. Technical report, Carnegie Mellon University, 1997.

[10] Michael Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 22:39–46, 1988.