Decentralization of a Machine: Some Definitions

Pradeep Dubey*

9 February 2015

Abstract

We define some notions of the decentralization of a deterministic input-output machine. This opens the possibility for introducing game-theoretic elements – such as strategic players — *inside* the machine, as part of its design.

Key Words: input-output machine, informational complexity, decentralization, strategic players.

1 Introduction

A key feature of "decentralization" is that a complex system can be broken into smaller constituents, each of which functions on the basis of information that is much more *limited* than the total information prevalent in the system. Thus decentralization is particularly useful when information is costly to disseminate or assimilate. A prominent example (see, *e.g.,* [1]) is that of a free market economy, where each agent simply optimizes against prices, ignoring all his competitors; and yet an efficient trade is achieved collectively in equilibrium. Another example (see, *e.g.,* [2]) involves "local economic networks" in which each individual interacts with a small set of neighbors, oblivious of the rest of the participants, but the ramifications of his actions can be felt throughout the system.

The purpose of this note is to explore the possibility of decentralization, not in a traditional economic context, but in the design of "machines". We restrict attention to a machine f which maps finitely many inputs to outputs. A design for f consists of smaller machines α arranged in a hierarchy. Each α receives as input the outputs produced by a subset of machines lower down in the hierarchy. Based upon its

*Stony Brook Center for Game Theory; and Cowles Foundation for Research in Economics, Yale University

input, α in turn decides what output to produce, and which machines higher up in the hierarchy to transmit the output to. The design must, of course, implement f in the following sense: for every initial input sent into the bottom level of the hierarchy, the output at the top level is in accordance with f. Given any design, we consider the costs of internal communication between its machines, as well as their operating costs. An appropriate decentralization of the machine f is a design that minimizes total cost among all designs that can implement f.

We also point out how certain designs can have game-theoretic structures embedded in them.

2 The Design of a Machine

We restrict attention to a deterministic input-output machine.

A sequence of $0'$s and $1'$s will be called a 01-sequence, for short. W.l.o.g.[1] the inputs and outputs can be taken to be 01-sequences of length n. Denoting the set of such sequences by $Seq = \{0,1\}^n$, the *machine* is specified by a function

$$f : D \longrightarrow Seq$$

on a domain $D \subset Seq$; which decomposes into its component functions, or *elementary machines*

$$f_i : D \longrightarrow \{0,1\}$$

for $1 \leq i \leq n$.

Our aim is to build up the machine f using *smaller* elementary machines α. Each such α receives a 0 or a 1 from some subset of its predecessors, by way of input. *Conditional* on its input, α decides whether to produce 0 or 1 as output; and, furthermore, it decides which of its successors to transmit the output to. Thus, looking back from α, the input of α can be 01-sequences of different lengths, indexed by the subset of α's predecessors that transmitted their outputs to α.

To be more explicit, consider nonempty, disjoint, finite sets N_t in "time periods"[2] $t \in \Gamma = \{1, \ldots, T\}$. The initial set $N_1 = \{\eta_1, \ldots, \eta_n\}$ and the terminal set $N_T = \{\tau_1, \ldots, \tau_n\}$ are both of size n.. For $2 \leq t \leq T-1$, the intermediate sets N_t can be of arbitrary size.

There is a *directed graph* G on the node set $N = N_1 \cup \ldots \cup N_T$. If (α, β) is a directed edge from $\alpha \in N_l$ to $\beta \in N_t$, we require $l < t$; i.e, the arrow of time points

[1] See remarks 5 and 6

[2] "Time" is just a metaphor for arranging machines in layers that are totally ordered.

forward. For any node α, denote its predecessor set by

$$P_\alpha = \{\beta : (\beta, \alpha) \in G\}$$

and its successor set by

$$S_\alpha = \{\beta : (\alpha, \beta) \in G\}$$

Clearly P_α is empty for $\alpha \in N_1$; and S_α is empty for $\alpha \in N_T$. Furthermore P_α *may* be empty for some $\alpha \in N_T$. For each intermediate node $\alpha \in N_2 \cup \ldots \cup N_{T-1}$, we require both P_α and S_α to be nonempty.

Notation 1 *$N = N_1 \cup \ldots \cup N_T$ and $N_- = N_2 \cup \ldots \cup N_T$*

Each element $\alpha \in N_-$ corresponds to an elementary machine. Elements of N_1, in contrast, are "dummy" machines and will play the special role of "initializing", or starting off, the computational process.

As was said, an input ν of $\alpha \in N_-$ consists of $0's$ and $1's$ indexed by the subset $V \subset P_\alpha$ of α's predecessors, from whence they came. Based upon ν, the elementary machine α first produces an output of 0 or 1; and then transmits this output to a subset $W(\nu) \subset S_\alpha$ of its successors

Notation 2 *.For any nonempty finite set A, let $SeqA$ denote the set of all* 01*-sequences whose elements are indexed by A (i.e., $SeqA$ is the set of all maps from A to $\{0,1\}$); and when A is the empty set, $SeqA$ denotes a singleton set, consisting of the "empty sequence" $\varnothing$ which signifies the absence of any input. Also denote $\mathcal{S}(A) = \cup\{SeqB : B \subset A\}$.*

Given $s = (s_1, \ldots, s_n) \in D$, the output of all $\alpha \in N$ is determined recursively as follows. First, the output of $\eta_i \in N_1$ is defined to be the component s_i of $s \in D$. Next, suppose the output of every $\beta \in \cup\{N_l : 1 \leq l \leq t\}$ has been determined. Consider $\alpha \in N_{t+1}$. Define $\mathcal{D}_\alpha \subset \mathcal{S}(P_\alpha)$ to be the set of 01-sequences, including possibly the empty sequence $\varnothing$, that can be received by α from P_α as we vary over all $s \in D$. The output of α is determined from its input in accordance with a given function, or *program,*

$$\pi_\alpha : \mathcal{D}_\alpha \longrightarrow \{0,1\};$$

and, for $v \in \mathcal{D}_\alpha$, the output $\pi_\alpha(v)$ is sent out to successor machines $\varphi_\alpha(v) \subset S_\alpha$ in accordance with a given *transmission rule*

$$\varphi_\alpha : \mathcal{D}_\alpha \longrightarrow Pow(S_\alpha)$$

where $Pow(S_\alpha)$ denotes the power set of S_α. If $\beta \in S_\alpha \setminus \varphi_\alpha(v)$, then it is understood that the empty sequence $\varnothing$ has come to β from α, *without* incurring any cost of transmission (i.e., without rendering the edge (α, β) "active" in the graph G. In particular, if $\varphi_\alpha(v)$ is the empty set, then every $\beta \in S_\alpha$ gets $\varnothing$ from α at no cost when $\alpha's$ input happens to be ν.

We assume that $S_\alpha = \cup\{\varphi_\alpha(v) : v \in \mathcal{D}_\alpha\}$ to rule out irrelevant elements from S_α.

Definition 3 $\mathcal{N} = \left\{G, \{\pi_\alpha, \varphi_\alpha\}_{\alpha \in G}\right\}$ *is called a* design. *We say that* $\mathcal{N}$ implements f *if*

$$f_i(s) = \text{ output of } \tau_i$$

for every $s = (s_1, \ldots, s_n) \in D$ *and* $\tau_i \in N_T$.

3 Informational Complexity

3.1 Fixed Costs

The edges of G correspond to routes for transmitting information inside the design $\mathcal{N} = \left\{G, \{\pi_\alpha, \varphi_\alpha\}_{\alpha \in G}\right\}$. Assuming that each route costs one unit of a "red" currency to build, the (red) *fixed cost* of the design is given by

$$c_F(N) = \#\text{ of edges of } G$$

Notice that, for any node α in G,

$$\#\text{ of edges of } G \text{ leading into } \alpha = \text{cardinality of } P_\alpha$$

is a measure of the extent of information I_α that is needed (as s varies over D) by α in order to execute its program π_α. By summing I_α over all the elementary machines α, we get $c_F(\mathcal{N})$. Thus $c_F(\mathcal{N})$ represents the (fixed costs) *informational complexity* of $\mathcal{N}$; and the smaller $c_F(\mathcal{N})$ is, the more decentralized $\mathcal{N}$ may be thought to be. This leads us to define

$$\kappa_F(f) = \min\{c_F(\mathcal{N}) : \text{the design } \mathcal{N} \text{ implements } f\}$$

For any given integer l, it would be interesting to catalogue machines f of *informational complexity* l, along with an accompanying design $\mathcal{N}$ that achieves $\kappa_F(f) = l$..

3.2 Variable Costs

After the fixed cost has been incurred to set up a design $\mathcal{N} = \left\{G, \{\pi_\alpha, \varphi_\alpha\}_{\alpha \in G}\right\}$ to implement f, there is a variable cost of operating $\mathcal{N}$. Suppose it costs one unit of a "blue" currency to transmit an "information bit" from one machine to another in $\mathcal{N}$., i.e., to send output 1 or 0 from α to β via the directed edge $(\alpha, \beta) \in G$. For ease of notation, suppose further that all inputs $s \in D$ arrive at $\mathcal{N}$ (i.e., at the initial nodes N_1 in $\mathcal{N}$) with the same frequency[3].

For any $s \in D$, let $\sigma_s(G)$ denote the number of edges in G that are rendered active when the initial input $s \in D$ is fed into $\mathcal{N}$. The (blue) *variable cost* of communication in $\mathcal{N}$ is then

$$c_V(\mathcal{N}) = \sum_{s \in D} \sigma_s(G)$$

We shall refer to the pair $c_F(\mathcal{N})$, $c_V(\mathcal{N})$ as the *full informational complexity* of $\mathcal{N}$.

4 Programming Complexity

Fix a design $\mathcal{N} = \left\{G, \{\pi_\alpha, \varphi_\alpha\}_{\alpha \in G}\right\}$ that implements $f : D \longrightarrow Seq$. For any initial input $s \in D$ into; $\mathcal{N}$, denote by $s_\alpha \in \mathcal{D}_\alpha$ the input that is received by α. Let us think of an "algorithm" Γ_α for π_α which inspects $s_\alpha \in \mathcal{D}_\alpha$ term by term (according to some rule intrinsic to Γ_α) and determines $\pi_\alpha(s_\alpha)$ in $\beta_{\Gamma_\alpha}(s_\alpha)$ inspections. (If s_α is the empty sequence $\varnothing$, then $\beta_{\Gamma_\alpha}(s_\alpha)$ is taken to be 0, since the output $\pi_\alpha(\varnothing)$ is predetermined independent of the initial input s, and so requires zero inspections of s.) From our point of view, the algorithm for π_α is *completely characterized* by the integers $\{\beta_{\Gamma_\alpha}(s_\alpha) : s \in D\}$. We may therefore assume that there is a *finite* set $\mathfrak{T}_\alpha$ of algorithms (reflecting the current state of knowledge) for π_α from which to choose.in the minimization problem displayed below.

The *programming complexity* (or, "green cost") of the design $\mathcal{N}$ is given by (assuming again, for simplicity, that all $s \in D$ are equally likely)

$$c_p(\mathcal{N}) = \min\left\{\sum_{\alpha \in G}\sum_{s \in D} \beta_{\Gamma_\alpha}(s_\alpha) : \Gamma_\alpha \in \mathfrak{T}_\alpha\right\}$$

It is worth noting that the minimizer in the above display will choose for π_α not an algorithm that does best in the worst case on $\mathcal{D}_\alpha$, but rather *on average,* putting

[3]Otherwise, consider $c_V(\mathcal{A}) = \sum_{s \in D} \pi_s \sigma_s(G)$ where π_s is the probability of the occurence of s.

more weight on those sequences s_α in $\mathcal{D}_\alpha$ that occur frequently as inputs at α when s varies over D. Also note that c_P is in the spirit of a *variable* cost.

One might ask how to define an analogue of fixed (or, set-up) costs for programs. We are just being speculative, but here it would seem necessary to take into account the fact that one might be able to go from one program π_α to another π'_α by a modification costing m "yellow" dollars; so that if π_α cost k such dollars to set up, then one can set up both π_α and π'_α for $k+m$ of those dollars. Thus the partial order in which the programs are set up will be relevant to the total fixed cost of setting up all the programs. However we do not have any clear ideas regarding the fundamental cost or effort of setting up a program and leave this issue for future consideration.

5 Decentralization of a Machine

For non-negative numbers (weights) x, y, z with $x + y + z = 1$, define the combined complexity (cost) of $\mathcal{N}$ by

$$c(\mathcal{N}) = xc_F(\mathcal{N}) + yc_V(\mathcal{N}) + zc_P(\mathcal{N})$$

Definition 4 *For any given c, the* complexity *of a machine f is*

$$\kappa(f) = \ min\{c(\mathcal{N}) : \textit{the design } \mathcal{N} \textit{ implements } f\};$$

and the concommitant decentralization *of f is a design which achieves the minimum in the above display.*

To check that the definition of complexity makes sense, and that the minimum is achieved by at least one design, it suffices to display a design that implements f. But this is obvious. Let there be just two time periods, so that

$$N = N_1 \cup N_2 = \{\eta_1, \ldots, \eta_n\} \cup \{\tau_1, \ldots, \tau_n\}$$

Let G consist of all n^2 edges (η_i, τ_j), where $1 \leq i \leq n$ and $1 \leq j \leq n$; further let $\pi_{\eta_j} = f_j$ and $\varphi_{\eta_j}(v) = N_2$.

6 Remarks

Remark 5 *In place of $\{0, 1\}$, one may consider an arbitrary finite set S of symbols. The entire foregoing (and, forthcoming) discussion holds mutatis mutandis. The machine is now given by $f : D \longrightarrow S^n$ where $D \subset S^n$; and $\mathcal{S}(P_\alpha)$ means the set of sequences in S indexed by elements of subsets of P_α; and each π_α maps the relevant domain $\mathcal{D}_\alpha \subset \mathcal{S}(P_\alpha)$ into S.*

Remark 6 *Any machine M which maps a finite set of inputs into outputs can be put into the format $f : D \longrightarrow Seq$ for some $D \subset Seq$ by coding, i.e, a one-to-one map of the inputs and outputs into* 01- *sequences of suitable length n. Many different functions f can now represent M, depending on the coding. Thus it makes sense to define $\kappa_X(M) = \min\{\kappa_X(f) : f \text{ represents } M\}$; this also defines "optimal" coding.*

Remark 7 *It would be interesting to explore the connection between the algebraic structure of f and the geometrical structure of its decentralized design(s).*

Remark 8 *If an output of $\alpha \in N_l$ is transmitted to $\beta \in N_t$ for some $t > l$, one might argue that the output has to be put in storage for fot $t - l$ periods (at α or at β or between the two). In this case storage costs, which we have ignored, might have to be factored into the total cost.*

Remark 9 *Our notion of informational complexity is based on the worst case scenario, since we try to implement $f(s)$ for* all *$s \in D$. We could instead consider approximate implementation, e.g., for 95% of the sequences in D. This might reduce the complexity by a huge amount by getting rid of the few bad elements which were very costly to handle.*

Remark 10 *There is no immediate tension between informational complexity and programming complexity. Indeed as informational complexity reduces, the length of inputs will fall for several $\mathcal{D}_\alpha$, which in turn will tend to make the programs π_α less complex. Inspite of this apparent complementarity, there may be trade-offs between the two complexities. We leave this for future inquiry.*

7 Game-theoretic Design

Suppose $\varphi_\alpha(v) = S_\alpha$ for all v and consider the general context of remark 5. Each elementary machine α in a design may be regarded as a "player" in a game who is choosing a best reply to the actions of its rivals in P_α that it receives by way of input. If two elementary machines α and β are identical (possibly after a relabeling of strategies and players), then α and β may be regarded as the same player, *provided* π_α and π_β are both derivable from the same "payoff function". Thus introducing players into an design $\mathcal{N}$ places considerable structure on $\mathcal{N}$. From the perspective of this paper, there is no game given apriori, but only the machine f. We are at liberty to invent players, along with their strategies and payoff functions — indeed to invent the whole game — provided it leads to an efficacious design for implementing

f. It is evident that many kinds of game-theoretic structures fit into what we have called "design." For instance, consider a finite normal form game. Given an n-tuple of (pure) strategies, let each of the n players make a best reply to the n-tuple. This yields a new n -tuple. Iterate the procedure k times. This conforms to a design. In fact we could allow differential time lags of information, some players becoming aware of their rivals' revisions earlier than others, or even each player observing his different rivals with varying time lags. This ,too, is a design.

One might wonder which kinds of machines f are amenable to game-theoretic designs, but this is a topic for future exploration.

References

[1] Debreu, G. (1959). Theory of Value. *Yale University Press.*

[2] Jackson, M.O. (2010). Social and Economic Networks. *Princeton University Press.*