Decentralized Designs for Deterministic Machines

Working Paper 17-07

PRADEEP DUBEY

September 9, 2017

Stony Brook University

Decentralized Designs for Deterministic Machines*

Pradeep Dubey†

9 September 2017

Abstract

We define the notion of the decentralization of a deterministic input-output machine. This opens the possibility for introducing game-theoretic structures *inside* the machine, as part of its design.

Key Words: input-output machine, complexity of communication, decentralization, quota games.

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 efficient trade is achieved collectively in equilibrium. Another example (see, *e.g.,* [3]) 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 deterministic machine f which maps finitely many inputs to outputs. A design for f consists of smaller machines α in a hierarchical array. Each α receives, as

*This is an elaboration of a special case of the model in [2]

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

input, the outputs produced by a subset of machines lower down in the hierarchy. Based upon its input, α in turn decides what output to produce, and then transmits this output to a subset higher up in the hierarchy. 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 complexity of communication between its constituent machines (or, equivalently, the aggregate input across all its constituents). A decentralized design for f is one that achieves minimal complexity.

Our purpose here is to give a *definition* of decentralized designs, and show that they exist in fairly general circumstances. However, the question of the computation of these designs, or even of some of their qualitative features (for different categories of machines), is left for future exploration.

The special focus of this note is on designs that are quota-based and have resonance with simple (voting) games.

2 A Canonical Case: Quota-based Designs

For any positive integer m, denote $M = \{1, \ldots, m\}$ and $\mathcal{Q}^M = \{0,1\}^M$,i.e., $\mathcal{Q}^M$ is the set of all sequences of $0's$ and $1's$ whose components are indexed by the elements of M (or, equivalently, the set of all maps from M to $\{0,1\}$). Consider a deterministic input-output *machine* specified by a function

$$f : \mathcal{Q}^N \longrightarrow \mathcal{Q}^K$$

where $N = \{1, \ldots, n\}$ and $K = \{1, \ldots, k\}$. Then f decomposes into its component functions, or *elementary machines*

$$f_i : \mathcal{Q}^N \longrightarrow \{0,1\}$$

for $i \in K$.

Our aim is to distribute the computation of f amongst an array of *smaller* elementary machines. First, each component s_l of input $s = (s_1, \ldots, s_n) \in Q^N$ is sent into an initial "dummy" machine $l \in N$, whose role is to merely transmit s_l to elementary machines α inside the array. *Conditional* on all inputs received from its "predecessors", α is programmed to produce 0 or 1 as output and to transmit this output to its "successors". The process iterates a finite number of times, until we reach "terminal" machines, indexed by K. It is required, of course, that f be implemented by the process, i.e., the output of $i \in K$ be $f_i(s)$ for all $s \in Q^N$ and all $i \in K$.

To be precise, consider nonempty, disjoint, finite sets V_t in "time periods"[1] $t = 1, \ldots, T$. The initial set $V_1 = N$ and the terminal set $V_T = K$. For $2 \leq t \leq T-1$, the intermediate sets V_t can be of arbitrary size.

Denote $V = V_1 \cup \ldots \cup V_T$. There is a *directed graph* $G = (V, E)$ with vertex set V and edge set $E \subset V \times V$. If $(\alpha, \beta) \in E$ is a (directed) edge from $\alpha \in V_l$ to $\beta \in V_t$, we require $l < t$; i.e, the arrow of time points forward. For any vertex $\alpha \in V$, 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 $\alpha \in V_1$; and S_α is empty for $\alpha \in V_T$. All other P_α and S_α are required to be nonempty.

Notation 1 $V_- = V_2 \cup \ldots \cup V_T$

Each vertex $\alpha \in V_-$ corresponds to an elementary machine whose operation will be described below.

In contrast, vertices in V_1 are "dummy" machines and play the role of "initializing" the computational process. For any $s = (s_l)_{l \in N} \in \mathcal{Q}^N$, the input as well as output of $l \in V_1 = N$ is s_l; thus the simple purpose of $l \in V_1$ is simply to receive the data s_l from s and to move s_l on as input to all[2] its successors S_l.

As for the machines $\alpha \in V_-$, an input μ consists of $0's$ and $1's$ indexed by the set P_α of α's predecessors, from whence they came. Based upon μ, the elementary machine α first produces an output of 0 or 1; and then transmits this output to the set S_α of its successors. The output of $\alpha \in V_-$ is determined from its input in accordance with a given function, or *program*[3],

$$\pi_\alpha : \{0,1\}^{P_\alpha} \longrightarrow \{0,1\}$$

Notation 2 *Let* Π *denote a class of "available" programs (which includes the identity map from* $\{0,1\}$ *to itself*[4]*).*

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

[2] More generally, the subset of S_α to which the output of α is transmitted, may also be allowed to depend on the input string at α (see [2] for more details).

[3] Technically speaking, a program is *a priori* given as a map from $\{0,1\}^M$ to $\{0,1\}$ where $M = \{1, \ldots, m\}$ for some integer m. It can be applied at α by first setting up a bijection between M and P_α (the predecessor set of α), and thus allows for many variations, one for each bijection. All these variations are understood to be equally available.

[4] Recall that the identity map is applied at all the initial vertices in V_1.

(Note that the length of the input-string may vary across the different programs[5] in Π, but the output of all programs is of length one, since it is either 0 or 1.)

Definition 3 $\mathcal{N} = \left\{G, \{\pi_\alpha\}_{\alpha \in V_-}\right\}$ *is called a* Π-design for f if $\pi_\alpha \in \Pi$ *for all* $\alpha \in V$; *and if, furthermore,*

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

for every $s = (s_1, \ldots, s_n) \in Q^N$ *and every* $i \in K = V_T$. *Also,* $\mathcal{N}$ *is just called a* design *if it is a* Π*-design for some* Π.

Let us consider the scenario where Π is a resource which has been acquired ("paid for") already; in other words, we have free access to "black boxes", each of which may operate according to any program in Π of our choosing. The complexity of a design is then just the complexity of communication between the black boxes. Now notice that, in the design $\mathcal{N} = \left\{G, \{\pi_\alpha\}_{\alpha \in V_-}\right\}$ for $f : \mathcal{Q}^N \longrightarrow \mathcal{Q}^K$, where $G = (V, E)$, the black boxes are represented by the vertices in V which transmit information along the directed edges E. Once an arbitrary input $s \in Q^N$ is sent into the initial layer $V_1 = N$, exactly one bit of information flows on each edge of E, until we wind up with $f(s) = (f_i(s))_{i \in K}$ at the terminal layer $V_T = K$. This motivates the definitions below:

Definition 4 *The* complexity $\chi(f, \mathcal{N})$ *of communication in a design* $\mathcal{N} = \left\{G, \{\pi_\alpha\}_{\alpha \in V}\right\}$ *for* f *is the number of edges of its graph* G.

Definition 5 *The complexity of communication for* f *modulo* Π *is*

$$\chi(f, \Pi) = \min\{\chi(f, \mathcal{M}) : \mathcal{M} \text{ is a } \Pi\text{-design for } f\}$$

Definition 6 *A* Π*-design* $\mathcal{N}$ *for* f *is said to be* decentralized *if* $\chi(f, \mathcal{N}) = \chi(f, \Pi)$

Remark 7 *It is evident from the definitions that if there exists a* Π*-design for* f, *then there also exists a decentralized* Π*-design for* f.

[5] W.l.o.g the length of the input strings may be assumed to have a finite upper bound (see, e.g., Remarks 8 and 10 below).

2.1 Monotonic Maps

Let $s, t \in \mathcal{Q}^N$, and write $s \geq t$ if $s_l \geq t_l$ for every component $l \in N$. Consider the case when, for every $i \in K$, the component map f_i is *monotonic*, i.e., $f_i(s) \geq f_i(t)$ if $s \geq t$; and, to avoid trivialities, assume $f_i(0, \ldots, 0) = 0$ and $f_i(1, \ldots, 1) = 1$.

The following analogy from cooperative game theory will be useful. Each f_i can be considered a *simple (or, voting) game* v_i on the player set N as follows: an input $s = (s_l)_{l \in N} \in \mathcal{Q}^N$ is identified with the coalition $T(s) = \{l \in N : s_l = 1\} \subset N$; and $T(s)$ is called *winning* in v_i if $f_i(s) = 1$, and *losing* if $f_i(s) = 0$.

2.2 Quota-Based Designs for Monotonic Maps

Consider the class Π^* of *quota-based* programs, under which a machine simply counts the number of $1's$ (or, $0's$) in its input string and outputs 1 (or, 0) if, and only if, this number exceeds some prescribed "quota".

A Π^*-design for $f : \mathcal{Q}^N \longrightarrow \mathcal{Q}^K$ will be also called quota-based.

Let us exhibit a "*naive*" quota-based design for every f_i ,where $i \in K$, and then collate these into a naive quota-based design for f. In the naive design, each quota will in fact be maximal, i.e., correspond to the *unanimity rule*.

The vertices V of the underlying graph $G = (V, E)$ consist of three disoint tiers: $V = V_1 \cup V_2 \cup V_3$. Here $V_1 = N = \{1, \ldots, n\}$ and $V_3 = \{i\}$. The middle tier N_2 has as its elements the collection $\mathcal{W}_i$ of *minimal winning coalitions of* v_i, namely coalitions that are winning in v_i and all of whose strict subsets are losing in v_i.

Given any input string $s \in \mathcal{Q}^N$, its component s_l enters the vertex $l \in N = V_1$ as input, and from there it is transmitted (unaltered) to all vertices $T \in \mathcal{W}_i = V_2$ such that $l \in T$. Next, the vertex T outputs 1 if *all* the components of its input string[6] are 1, otherwise it outputs 0 (i.e., the quota at T is just the cardinality of T, and T operates via the "unanimity voting rule"). The output from each $T \in \mathcal{W}_i$ is next transmitted to the unique element i of the singleton set V_3. The output of this final vertex i is 0 if *all* the components of its input string[7] are 0; otherwise the output is 1. The transmission of information is along the directed edges of the graph G; and so – to be pedantic – the edge set E of G is given by $E = \{(l, T) ; l \in V_1, l \in T \in V_2\} \cup \{(T, i) : T \in V_2\}$.

It is clear that that this design implements f_i, i.e., for any input $s \in \mathcal{Q}^N$ at V_1, the output at $i \in V_3$ is $f_i(s)$. Now we simply collate the designs of all f_i, for

[6] these components are indexed by T

[7] Recall that these components are indexed by $\mathcal{W}_i$. Note moreover that, looking at $1's$ in the input string at T_3 (instead of $0's$), we may equivalently consider a quota of one (instead of the unanimity rule), i.e., the output is 1 if the number of $1's$ in the input string is at least one.

$1 \leq i \leq k$, to obtain a design for f. The bottom tier of vertices is inchanged and remains $N = \{1, \ldots, n\}$. The middle tier is $\mathcal{W}_1 \cup \ldots \cup \mathcal{W}_k$ and the final tier is $\{1, \ldots, k\}$. Transmissions take place along directed edges from bottom to middle and middle to top, as described above.

However this naive design for f , based on unanimity rules, may be far from minimal . By way of example, suppose that v_i corresponds to a simple majority game, i.e., a string is mapped to 1 precisely when half or more of its components are 1. Then it would be much better to construct a design for f_i as follows: map all the outputs of N_1 to a single centralized vertex in N_2 and let the quota at this vertex be the smallest integer that is greater than, or equal to, $n/2$.

Thus the naive design only assures us that there *exists* a decentralized quota-based design for f, but provides little clue as to what it looks like or how to compute it from f. This seems like a topic that might merit investigation.

Remark 8 *The collection $\mathcal{C}$ of minimal winning coalitions of a simple game on player-set $N = \{1, \ldots, n\}$ form a "clutter", i.e., if $S, T \in \mathcal{C}$ and $S \subset T$ then $S = T$. By [4], the cardinality of $\mathcal{C}$ is bounded above by $g(n) = n!/(m!(n-m)!)$ where m is the smallest integer above $n/2$. Thus, by way of a gross overestimate, the edges of a naive quota-based design are no more than $g(n)nk$. This then also serves as a bound for $\chi(f, \Pi^*)$ for an arbitrary $f : \mathcal{Q}^N \longrightarrow \mathcal{Q}^K$, so long as f is monotonic.*

2.3 General maps

In the general case, when the f_i are not monotonic, replace $\mathcal{W}_i$ by the set of *all* winning coalitions of the game v_i that corresponds to f_i, and repeat the above construction to obtain a quota-based design. (This design works, of course, in the monotonic case as well; but we get a much sparser design by restricting to minimal winning coalitions, in the monotonic case.)

3 Unrestricted Designs

It is evident that the complexity of communication for f diminishes as the set of available programs is enhanced, i.e., $\chi(f, \Pi) \leq \chi(f, \Pi')$ if $\Pi \supset \Pi'$. But there is a limit to the dimunition.

Definition 9 *The* quintessential complexity of communication *for f is*

$$\chi(f) = \chi(f, \widetilde{\Pi})$$

where $\widetilde{\Pi}$ *denotes the set of all possible programs (equivalently:* $\chi(f) = \min\{\chi(f,\Pi) : \Pi$ *is arbitrary*$\}$*).*

Remark 10 *For* $f : \mathcal{Q}^N \longrightarrow \mathcal{Q}^K$ *consider the design whose graph has vertex set* $V = N \cup K$ *and edge set* $E = \{(l,i) : l \in N$ *and* $i \in K\}$*. And let the program at* $i \in K$ *be simply the component function* f_i *of* f*. It is immediate that this design implements* f *and that its complexity of communication is* nk *where* n *and* k *are the cardinalities of* N *and* K *respectively. It follows that*

$$\chi(f) \leq nk$$

This in turn implies that w.l.o.g. we might as well restrict attention to programs in $\widetilde{\Pi}$ *whose input strings have length no more than* nk*.*

4 Remarks

Remark 11 ***(Complexity of Programming)*** *We have ignored the "programming complexity" at each vertex, i.e., the complexity of its input-output map (which is why we referred to the vertex as a "black box"). However, in the special context of quota-based programs, it might make sense to define the complexity of a vertex to be simply*[8] *its quota* q*. Indeed, by way of a concrete physical model, think of the transmission of* 1 *(or,* 0*) as sending an electical impulse (or, no impulse). Imagine that each vertex is equipped with an internal clock plus a counter. Then a vertex in* V_t *will need to count the impulses it has received by time* t *in order to decide whether or not to put out an impulse; but it can stop counting once the total reaches* q*. One may therefore define the programming complexity of a design to be the sum of the quotas of all the vertices, and take this into account along with the complexity of communication, in order to define efficient designs. For further elaboration of this theme, see [2].*

Remark 12 ***(Influence of Inputs and Banzhaf Indices)*** *Consider* f *for which each component map* f_i *is monotonic. One might well ask how "influential" is input component* l*? To be precise: if the component* s_l *is unilaterally altered (from* 1 *to* 0*, or* 0 *to* 1*) in every* $s \in \mathcal{Q}^n$*, how many components of the output would get changed? Let this number be denoted* $h_l(s)$ *for each* $s \in \mathcal{Q}^n$*. Then (implicitly regarding every* $s \in \mathcal{Q}^n$ *as equally likely) we may define* $\sum_{s \in \mathcal{Q}^n} h_l(s)$ *to be the influence of input*

[8] There is a related literature on "circuit complexity", which defines the complexity of an arbitrary program via Boolean "gates" (namely: and, or, not), as was pointed out to me by Yakov Babichenko.

l on f. It is easy to see that the vector $\left(\sum_{s\in\mathcal{Q}^n} h_l(s)\right)_{l=1,\ldots,l}$ *is a scalar multiple (the scalar being* 2^n *or* 2^{n-1}*, depending on the convention of counting) of the vector* $\left(\sum_{1\leq i\leq k} \beta_l(v_i)\right)_{l=1,\ldots,l}$ *,where* β_l *is the* ***Banzhaf index*** *of player l in the simple game* v_i *that corresponds to* f_i*.*

Remark 13 ***(Noncooperative Games)*** *We have incorporated some aspects of cooperative game theory, namely simple games, into the machine. In [2], we point out how non-cooperative games may also have a bearing.*

References

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

[2] Dubey, P. (2015). Decentralization of a Machine: Some Definitions, *Working Paper, Department of Economics, Stony Brook University.*

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

[4] Sperner, E. (1928) Ein Satz uber Untermengen einen endlichen Menge. *Math. Z. 27, 544-548.*