Limited Computational Resources Favor Rationality *

Yuval Salant
The School of Computer Science and Engineering, and
The Center for the Study of Rationality
The Hebrew University, Jerusalem †

July 17, 2003

Abstract

A choice function is a rule that chooses a single alternative from every set of alternatives drawn from a finite ground set. A rationalizable choice function satisfies the *consistency* condition; i.e., if an alternative is chosen from a set A, then the same alternative is also chosen from every subset of A that contains it.

In this paper we study computational aspects of choice, through choice functions. We explore two simple models that demonstrate two important aspects of choice procedures: the ability to remember the past and the ability to perform complex computations. We show that a choice function is optimal in terms of the amount of memory and the computational power required for its computation if and only if the function is rationalizable. We also show that the computation of most other choice functions, including some "natural" ones, requires much more memory and computational power.

*I am indebted to Gil Kalai for his devoted guidance, constant support, and most valuable suggestions. I am grateful to Ariel Rubinstein and Ran Spiegler for the encouragement, productive discussions, and important comments. I thank Elchanan Ben-Porath, Michael Borns, Michal Maimaran, Zvika Neeman, Abraham Neyman, Motty Perry, Ron Siegel, and Menahem Yaari for fruitful conversations and helpful remarks.

†Email: yuvals@pob.huji.ac.il.

1 Introduction

Consider a basic model of choice where a decision-maker(DM) faces choices from sets of alternatives that are subsets of some finite ground set X. A *choice problem* (or a *choice set*) is a set $A \subseteq X$ of cardinality at least two. For every choice problem A, the task of the DM is to single out one element of A. In this model, *rational choice* means maximizing a utility function over the set of alternatives. An observable criterion of rationality is the *consistency* condition, often referred to as the principle of "Independence of Irrelevant Alternatives" (IIA): if a DM chooses some alternative from a choice problem A, then he will also choose the same alternative from every subset of A that contains that alternative.

The hypothesis of rational choice and its interpretation as utility maximization are under much scrutiny.[1] There are many well-documented violations of rational choice, and these violations have been given various reasons (see McFadden [8] and Rabin [13] for surveys). For example, Tversky and Shafir [20] found that subjects violate the IIA principle, and suggested that the tendency to defer choice, search for new alternatives, or choose a default option can be increased when the choice set is enlarged. One of the more common explanations for non-rational choice is that maximization is "hard" to carry out, and therefore it is traded for other mechanisms.

In this paper, we explicitly model computational aspects of choice, and examine the computational complexity of various choice processes. The main result of the paper, informally stated, is that rational choice is actually favored, rather than rejected, by computational considerations.

Specifically, we discuss two basic computational aspects of choice: the amount of memory and the computational power a DM has to invest in order to compute his choice. We believe that in many contexts a DM is unlikely to "work hard" and invest significant amounts of memory and computational power in choice processes. For example, consider the process of choosing a dish from a menu in a restaurant. As we scan the menu, we rarely remember every dish that appears on the menu, and we often do not use complex computations in this process. Instead, we use some approximation

[1] For an interesting discussion of the rational choice paradigm and arguments both in favor and against this paradigm, see Rubinstein [16], Chapter 1.

mechanism that allows us to choose a dish. Consequently, it is interesting to explore what types of choice procedures require modest amounts of memory and computational power.

We investigate this question in the context of choice functions. Given a finite set X of possible alternatives, a *choice function* c is a rule that chooses from every choice problem $A \subseteq X$ an element $c(A) \in A$. A *rationalizable* choice function obeys the IIA principle; i.e., for every choice problem $A \subseteq X$ and for every $A' \subseteq A$, if $c(A) \in A'$ then $c(A) = c(A')$. *Computing* a choice function refers to the process in which the elements of a choice problem are revealed to the DM sequentially in an arbitrary order. The DM has to choose one element of the sequence, according to his choice function. The DM's choice has to be *order independent*; that is, he has to choose the same element from a choice problem, regardless of the order in which it was introduced to him.

Our first model aims to explore the implications of limited memory on the type of choice functions that can be computed. According to this model, a DM can remember at most k alternatives simultaneously; i.e., the DM has k memory cells, where a memory cell is capable of storing at most one alternative at any given moment. If the DM has "remembered" k elements in his memory cells, and he now encounters a new element, the DM has to determine which k elements will be stored in the memory cells, and which element will be "forgotten." Forgetting an element implies that this element cannot be chosen later. After all the elements of the choice problem are revealed, the DM chooses one of the k elements that he remembers.

Our second model aims to incorporate computational complexity into the first model. We use automata for that purpose. Automata are commonly used in the literature of bounded rationality, in particular in the literature of repeated games (see Neyman [9, 10], and Rubinstein [14, 16]), in order to limit the computational power of agents, or the type of strategies they can employ. As in the literature of repeated games, we define the complexity of a choice function to be the number of states in the minimal automaton that computes it. The more states an automaton has, the more complex the automaton is, and likewise the choice function it computes.

The paper introduces two main results. Firstly, we show that rationality is optimal in terms of memory and computational power; i.e., the compu-

tation of a rationalizable choice function uses the least possible amount of computational resources.

Proposition 1. A choice function can be computed with a single memory cell if and only if it is rationalizable.

Proposition 2. A choice function can be computed by an automaton with $N-1$ states if and only if it is rationalizable. (No choice function can be computed by an automaton with less than $N-1$ states.)

Secondly, we argue that almost all other choice functions require considerably more computational resources.

Proposition 3. The computation of almost all choice functions requires at least $N(1-\varepsilon)$ memory cells, where $\varepsilon > 0$ is a small real number.

Proposition 4. Almost all choice functions can be computed only by automata with more than $\frac{2^N}{N^2}$ states.

As every choice function can be computed with $N-1$ memory cells, or by an automaton with 2^N states, Propositions 3 and 4 suggest that the computation of almost all choice functions necessitates resources that are very close to the maximal required resources. Moreover, Propositions 1–4 imply that it is much easier to compute a rationalizable choice function than an arbitrary choice function. As we demonstrate in the sequel, this "gap" also applies to a number of natural choice functions, which are of interest.

A note on the literature is in order. The line of research that seeks justifications for the rationality assumption is not a new one. Many economists would argue that violating the IIA principle cannot be economically viable, because it leaves the decision-maker vulnerable to the so-called "Dutch Books" (this argument is demonstrated by Yaari [23]). Kalai [4] shows that rational behavior is easy to learn in the PAC-learning model (PAC stands for Probably Approximately Correct; see Vidyasagar [21]). Rubinstein [15] argues that the frequent appearance of order relations in natural language can partly be explained by the fact that order relations are easy to describe. In this paper, we continue this line of research, and suggest computational justifications for the rationality assumption.

2 Choice functions

Let $X = \{1,\ 2, \ldots,\ n\}$ be a finite set of alternatives (e.g., a set of possible dishes in a restaurant). Denote its cardinality by N. A *choice problem* (or a *choice set*) is a set $A \subseteq X$ of cardinality at least two. Let $S(X)$ be the collection of all possible choice problems; i.e., $S(X) = \{A \subseteq X : |A| \geq 2\}$. When confronted with a choice problem $A \in S(X)$, the DM chooses one element of A. This induces a choice function on $S(X)$. Formally,

Definition 2.1 *A* choice function $c : S(X) \to X$ *assigns to every set* $A \in S(X)$ *an element* $c(A) \in A$*, where* $c(A)$ *is the alternative chosen from the choice problem* A*.*

Throughout this paper we assume that the DM encounters the alternatives of a specific choice problem sequentially. When the DM chooses an alternative from a choice problem, his choice is *order independent*. That is, he chooses the same element from a set, regardless of the order in which the set was introduced to him. We are aware of the fact that choice may be *order dependent*, and we discuss several order-dependent procedures in the sequel.

The paper focuses on three choice functions: the *rationalizable* choice function, the *second-best* choice function, and the *median* choice function. In order to define these functions, let us assume (although this is not necessary) that there exists a strict order relation $\succ$ (i.e., a linear ordering) on the elements of X.

The rationalizable choice function chooses from every choice problem $A \in S(X)$ the maximal element according to $\succ$. An alternative definition states that a choice function is *rationalizable*, if it obeys the "Independence of Irrelevant Alternatives" axiom. That is, for every choice set A and for every $A' \subseteq A$, if $c(A) \in A'$ then $c(A) = c(A')$.

The second-best choice function chooses from every choice problem A the second-maximal element according to $\succ$. The second-best choice function may apply to situations where the DM employs considerations other

than $\succ$. For example, Sen [18] suggests that a person might choose the second-largest slice of cake in a party in order to appear polite, and McFadden [8] mentions that people tend to choose the second-lowest-priced wine in a restaurant, especially when a lower quality wine is offered at a price only slightly cheaper. For an axiomatic analysis of this function, see Baigent and Gaertner [1].

The median choice function chooses the *median* alternative according to $\succ$ from every choice problem. More precisely, the median choice function chooses the median element according to $\succ$ from sets of odd size. For sets of even size, the function chooses the higher element from the two median elements; e.g., if $1 \succ 2 \succ 3 \succ 4$ then $med(1,2,3,4) = 2$. A median choice function may be used to resolve a conflict between two opposing considerations that the DM weighs equally (such as risk and profit, or work hours and leisure). The median can also serve as a good approximation to the phenomenon of extremeness aversion described in Simonson and Tversky [19]. An axiomatic characterization of this function can be found in Gaertner and Xu [2].[2]

3 The memory cells model

A DM is often limited by his ability to remember the past, and this may affect his behavior. In the literature of repeated games, Lehrer [6] explores the implications of bounding the number of periods a player can recall from the past. In other contexts, Piccione and Rubinstein [12] investigate how economic agents with different abilities to recall past events interact, and Wilson [22] explains biases in inference making with a bounded memory model. In this section, we investigate the implications of limited memory in the context of choice functions. To this end, we use the memory cells model that was previously discussed in Kalai [4].

[2]Kalai, Rubinstein, and Spiegler [5] explore whether the choices of the second-best and the median choice functions can be rationalized by multiple rationales (i.e., as the maximization of multiple linear orderings). They find that these two functions can be rationalized only by a "large number" of orderings.

3.1 The model

Consider a situation in which the elements of a choice problem A are revealed to the DM one after the other in some arbitrary order. Suppose that the DM can remember at most k alternatives simultaneously; i.e., the DM has k memory cells, where a memory cell is capable of storing at most one alternative at any given moment. If the DM has stored the elements $x_1, x_2, \ldots, x_k$ in his memory cells, and now he encounters another element y, the DM has to determine which k elements among $x_1, x_2, \ldots, x_k, y$ will be stored in the memory cells and in what order. If some element is removed from the memory cells, the DM "forgets" it, and therefore this element cannot be chosen. After all the elements of the choice problem are revealed, the DM chooses one of the elements that appear in the memory cells by storing it in the first cell.

More precisely, the decision process of an agent with k memory cells can be described as follows. The DM has two mechanisms:

- A "hardware" of k memory cells. A memory cell can either be empty or contain an alternative of the choice problem. This hardware allows the DM to choose without any limitations from sets with at most k elements.

- A "software" that manages the content of cells. That is, a discrete function $f : X^{k+1} \longrightarrow X^k$, which is activated whenever all the cells are full and a new element appears.

In every stage of the decision process, a new alternative is revealed to the DM (the DM does not know in advance how many alternatives will be introduced to him). If there is an empty cell, the new alternative is stored in it (reordering the elements in the cells is always possible). If all the cells are full, f is activated. The *input* to f is a $(k+1)$-tuple, $(x_1, \ldots, x_k, y)$, where $(x_1, \ldots, x_k)$ are the k elements stored in the memory cells (by the order of their appearance in the cells), and y is a new element. The *output* of f is the k elements to be stored in the memory cells (by their order). As f only manages the content of the cells, the output of f is always a subset of the input to f, that is, $f(x_1, \ldots, x_k, y) \subset \{x_1, \ldots, x_k, y\}$. When the DM encounters the special character ϵ, which signals the end of the

choice problem, the DM moves the chosen alternative (which is stored in one of the cells) to the first cell.

This procedure should be *order independent*. That is, for every two sequences $(a_1, \ldots, a_m, \epsilon)$ and $(\pi(a_1, \ldots, a_m), \epsilon)$, where $a_1, \ldots, a_m$ are different alternatives and π is a permutation of $(a_1, \ldots, a_m)$, we require the choice from these two sequences to be the same. We also assume that an alternative appears at most once in a sequence of alternatives. This is not a necessary assumption, and most of the results continue to hold when this assumption is removed.

3.2 The optimality of rationalizable choice functions

A rationalizable choice function can be computed with a single memory cell. Indeed, suppose that the DM has an ordering $\succ$ on the alternatives. During the decision process, he can use the following simple algorithm:

- When the first element is revealed, store it in the memory cell.
- When a new element is revealed, compare it to the element in the memory cell, and store the higher element according to $\succ$ in the memory cell.

This strategy is *order independent*, can be implemented with a single memory cell, and computes a rationalizable choice function. One may wonder what other choice functions can be computed by a single memory cell. The next proposition answers this question.

Proposition 3.1 *A choice function can be computed by a single memory cell only if it is rationalizable.*

Proof. Let c be a choice function that can be computed by a single memory cell. In order to prove that c is rationalizable, we will show that c obeys the IIA principle.

Assume to the contrary that c violates the IIA principle; i.e., there are sets $S \subseteq X$ and $S' \subseteq S$ such that $c(S) \in S'$ but $c(S') \neq c(S)$. Denote by $a = c(S)$ the element chosen from S. In that case, after all the elements

of S' (including a) are introduced to him, the DM must store $c(S')$ in the memory cell, because he may have to output his choice in the next stage. That is, he "forgot" a. If the elements of $S \setminus S'$ are revealed to the DM after the elements of S', he obviously cannot restore a in the memory cell, because a is not in $S \setminus S'$. Therefore, his choice from $S' \cup S \setminus S' = S$ is not a. Contradiction. □

3.3 Order-dependent choice using a single memory cell

The result of Proposition 3.1 is due to the *order-independence* assumption. When we remove this assumption, other choice procedures can be implemented with a single memory cell. In fact, a choice procedure c can be computed by a single memory cell if and only if c can be defined by a binary relation R_c that governs the function of the memory cell in the following way: if a is stored in the memory cell, and b appears next, then b is stored in the memory cell instead of a if $b \; R_c \; a$. Yet, as these procedures may yield different results depending on the order in which the alternatives appear, there is no unique choice from every set S, and therefore in most cases such choice procedures do not induce choice functions.

Consider, for example, a DM that experiences the *status-quo bias* (see Samuelson and Zeckhauser [17]); i.e., he favors the current status and the history relative to alternatives not yet experienced. We may think of this DM as attributing a "sentimental" value to objects in his possession, in addition to their "objective" utility. In other words, the DM has two functions:

- An "objective" utility function $u : X \longrightarrow R$.
- A "sentimental" value function $s : X \longrightarrow R_+$.

We define the *"sentimental" utility* of an alternative $a \in X$ to be $u(a)+s(a)$. The decision rule of the DM is as follows. If a is stored in the memory cell and b appears next, b is stored in the memory cell instead of a if the "objective" utility of b is greater than the "sentimental" utility of a, that is, $u(b) > u(a)+s(a)$. This procedure can be computed by a single memory cell. It chooses an alternative whose "objective" utility is greater than the "objective" utility of all the alternatives revealed *before* it, and whose "sentimental" utility

is greater than the "objective" utility of all the alternatives revealed *after* it. It is left to the reader to verify that this procedure does not necessarily induce a choice function.

Another interesting procedure, in this context, involves majority decisions. Consider a committee that has to choose the most suitable candidate for a job. The committee interviews the candidates in some order. After a new candidate is interviewed by the committee members, they use a majority vote to decide whether she is better or worse than the most suitable candidate so far. This choice procedure chooses the candidate that is "directly" better than everyone interviewed after her and "indirectly" better (that is, directly or through intermediate candidates) than everyone interviewed before her. The procedure can be implemented by a single memory cell; however it is order dependent.

3.4 Two memory cells

Let us assume order independence again. Two memory cells allow the DM to compute a richer collection of choice functions. We found two interesting types of such functions:

1. **"Choose one of the best two" procedure.** This procedure is based on an order relation $\succ$ and a binary relation R. Given a choice problem A, $\succ$ is used as a "filter" to choose the two most attractive alternatives in A. The choice between these two alternatives is made according to R. For example, from the set of all schools to which a prospective Ph.D. student is accepted, he uses the ranking of *www.usnews.com* to "filter" the two "best" schools, and then he visits these two schools in order to make a final decision based on personal impression. Note that the second-best choice function is a private case of the "Choose one of the best two" procedure, for the case where $a \; R \; b$ if and only if $b \succ a$.

2. **"Maximize two orderings" procedure.** This procedure is based on two order relations, $\succ_1$ and $\succ_2$, and a binary relation R. Given a choice problem A, the procedure computes the maximal elements according to $\succ_1$ and $\succ_2$, and chooses between them according to R. For example, consider the "frog's legs" procedure (see Luce and Raiffa [7]), where

the DM chooses a main course from a restaurant's menu according to $\succ_1$, when frog's legs appear on the menu; otherwise, he chooses according to $\succ_2$. Another interesting example is a choice function that is rationalizable up to one "mistake" on a set of two elements. That is, there exists an order relation $\succ$ such that the choice from all sets maximizes $\succ$, except for one set of size 2 (the "mistake") in which the minimal element according to $\succ$ is chosen.

Remark 3.2 *We conjecture that these two types characterize all choice functions that can be computed by two memory cells.*

Throughout this paper, we assume that the alternatives are revealed to the DM in an arbitrary order. Let us remove this assumption (for this paragraph only), and introduce an example, in which certain assumptions on the order of the alternatives dramatically reduce the number of memory cells needed for computing the choice. Consider a two-player zero-sum game where player 1 has $K = \{1, 2, \ldots, k\}$ pure strategies, player 2 has $M = \{1, 2, \ldots, m\}$ pure strategies, and the payoff function to player 1 is $g : K \times M \longrightarrow \mathbb{R}$. Assume that the players do not know the payoff function a priori, and that player 1 wants to compute a pure *maxmin* strategy according to the information he sees. More precisely, let $X = \{\ (i, g(i, j)) \mid i \in K \text{ and } j \in M\}$ be our ground set. The DM encounters a sequence of alternatives, $A \subseteq X$, and wants to compute a *maxmin* strategy on the elements of A; That is, he wants to compute a strategy i which obtains $max_{i\ :\ (i,g(i,\cdot))\in A}\ min_{j\ :\ (i,g(i,j))\in A}\ g(i, j)$. If the alternatives of A appear in an arbitrary order, then the DM will need a number of memory cells which is dependent on the number of his strategies. However, if the DM knows that the alternatives of the choice problem are grouped according to his strategies; i.e., all the alternatives with the same strategy of player 1 appear one after the other, then the DM can use only two memory cells regardless of the number of strategies. To see why, think of the first memory cell as minimizing over the current group of alternatives (with the same strategy of player 1), and of the second cell as maximizing over the minimizers computed by the first cell. An indication about whether the current group has finished and a new one started can be obtained by checking the first coordinate of an alternative. When the first alternative of a new group appears,

the second memory cell compares between its content and the content of the first cell and stores the maximal element. The first cell simply stores the new alternative. This example illustrates that choosing is easier when the choice problem is introduced in a structured and ordered way.

3.5 Arbitrary choice functions and memory cells

Any choice function can be computed with $N-1$ memory cells. Indeed, if we have $N-1$ memory cells, we can store $N-1$ elements in them, and therefore we can compute the choice from any set of size at most $N-1$. As for the choice from the whole set of alternatives X, we can store the first $N-1$ elements to be revealed in the memory cells, and when the N'th element appears, we can decide which element is chosen, store it in the first memory cell, and remove one arbitrary element.

Most of the choice functions we have investigated so far can be computed by a constant number of memory cells, which is independent of the number of alternatives. This suggests that they can be implemented by the DM without any concern about the number of alternatives. However, this is a rather exceptional phenomenon, as the computation of most choice functions requires a number of memory cells that is dependent on the number of alternatives N. A natural example of such a choice function is the median choice function.

Proposition 3.3 *The computation of a median choice function requires* $\lceil \frac{N}{3} \rceil + 1$ *memory cells.*

Sketch of proof. Assume for simplicity that N is a multiple of 3. In order to see that $\frac{N}{3}$ memory cells are not enough, we can divide the N elements into three regions according to $\succ$:

$$\underbrace{x_1 \succ \ldots \succ x_{\frac{N}{3}}}_{\text{A}} \succ \underbrace{x_{\frac{N}{3}+1} \succ \ldots \succ x_{\frac{2N}{3}}}_{\text{B}} \succ \underbrace{x_{\frac{2N}{3}+1} \succ \ldots \succ x_N}_{\text{C}} .$$

Suppose that the $\frac{N}{3}$ elements of region B are revealed to the DM first, and that only $\frac{N}{3}$ memory cells are available. In that case, if $x_{\frac{N}{3}}$ is revealed next, then any of the $\frac{N}{3}+1$ elements revealed so far is still a possible choice. Yet,

we cannot store all of them in the memory cells as we have only $\frac{N}{3}$ cells. Therefore, the DM needs at least $\frac{N}{3}+1$ memory cells.

To see why $\frac{N}{3}+1$ cells are enough, note that when there are $\frac{N}{3}+1$ cells filled with elements and a new element is revealed, we have a total of $\frac{N}{3}+2$ elements. At least one of them (either the highest or the lowest according to $\succ$) cannot be chosen, because we cannot "pad" both sides of the $\frac{N}{3}+2$ elements with $\frac{N}{3}$ elements or more on every side as we have only N elements in total. Therefore, we can "forget" one of the $\frac{N}{3}+2$ elements. In Appendix B, we introduce an algorithm that uses this insight to compute the median choice function with $\frac{N}{3}+1$ cells. □

The following proposition suggests that most choice functions behave like the median example; i.e., almost all choice functions require many memory cells.

Proposition 3.4 *The proportion of choice functions whose computation requires at least $k = N - (1+\varepsilon)\log N$ memory cells, where $\varepsilon > 0$ is a small real number, tends to 1 as the number of alternatives N tends to infinity.*

Proof. It is enough to show that when we uniformly draw a choice function c from the collection of all choice functions, the probability that c needs at least k memory cells tends to 1 as N tends to infinity. In order to prove this, we fix a set S of k elements, uniformly draw a choice function c (by uniformly and independently choosing $c(A)$ from every choice set $A \supseteq S$), and show that with high probability all the elements of S are chosen in c by supersets of S. This implies that c must remember all the elements of S when it sees them (because the supersets of S choose every element of S), which in turn indicates that c needs at least $|S| = k$ memory cells.

Indeed, fix $S = \{1, 2, \ldots, k\}$ to be a choice set of k elements. S has 2^{N-k} supersets. We now show that for a uniformly drawn choice function, these supersets choose all the elements of S with a probability that tends to one as N tends to infinity. Let us fix an element $j \in S$, and denote by P_j the probability that j is "missed" (not chosen) by all the supersets of S. Then,

$$P_j = \prod_{i=0}^{N-k} \left(\frac{k+i-1}{k+i}\right)^{\binom{N-k}{i}} = \prod_{i=0}^{N-k} \left(1 - \frac{1}{k+i}\right)^{\binom{N-k}{i}}$$

because for every $0 \leq i \leq N-k$, we have $\binom{N-k}{i}$ supersets of S of size $k+i$, and the probability that such a superset "misses" j is $\frac{k+i-1}{k+i}$. Since $k+i \leq N$, we have

$$P_j \leq \prod_{i=0}^{N-k} (1-\frac{1}{N})^{\binom{N-k}{i}} = (1-\frac{1}{N})^{\sum_{i=0}^{N-k}\binom{N-k}{i}} = (1-\frac{1}{N})^{2^{N-k}}.$$

Let P denote the probability that at least one element of S was "missed" by the supersets of S. Then, by the union bound rule

$$P \leq \sum_{j=1}^{k} P_j = \sum_{j=1}^{k} (1-\frac{1}{N})^{2^{N-k}} = k(1-\frac{1}{N})^{2^{N-k}}.$$

Substituting k by $N-(1+\varepsilon)\log N$, and letting N tend to ∞, we have

$$P \leq (N-(1+\varepsilon)\log N)(1-\frac{1}{N})^{2^{(1+\varepsilon)\log N}} \leq N(1-\frac{1}{N})^{N^{1+\varepsilon}} \longrightarrow 0$$

because $(1-\frac{1}{N})^{N^{1+\varepsilon}} \longrightarrow e^{-N^{\varepsilon}}$. Therefore, the probability that all the elements of S are "hit" by a uniformly drawn choice function, which equals $1-P$, tends to 1 as N tends to infinity. Consequently, most choice functions must remember all the elements of S, and they therefore need at least $N-(1+\varepsilon)\log N$ memory cells. □

Proposition 3.4 and the median example suggest that most choice functions, including some "natural" ones, turn out to be impractical as the number of alternatives increases. Moreover, if the exact number of alternatives is unknown in advance or may grow with time, such choice functions are even less appealing.

4 The automata model

In the previous section, we dealt with memory limitations, but we did not impose additional computational restrictions on the DM. In this section, we would like to explore directly the implications of limited computational power on the nature of choice. We use automata for that purpose. An automaton is a "machine" that reads the alternatives, processes them (by

moving between states), and chooses one of them. The concept of using automata to model limited computational power is well established. Computer scientists use automata in complexity theory to model limited computational power (see Hopcroft and Ullman [3]). Game theorists have adopted this notion in order to impose limitations on the type of strategies players use (see Osborne and Rubinstein [11]). We also find automata attractive, as they are simple machines that can compute any choice function.

4.1 Preliminaries

The choice procedure of a DM may be visualized as follows. A sequence of alternatives is introduced to the DM. When the DM encounters the next element in the sequence, he examines the new information and integrates it with his knowledge base. If necessary, the DM also moves from one "state" (of mind) to another according to the new information. When the DM is notified by some means that the sequence is over, he chooses an alternative according to his state. An automaton can model this procedure. An automaton is a machine that reads the alternatives one after the other. When the automaton reads the next item, it checks its current state and the new input, and decides whether to change its state or not according to a predefined transition function. When the automaton reaches the end of the sequence, it outputs an alternative according to its state.

Formally, let $s = (s_1, s_2, \ldots, s_k)$ be a sequence of alternatives, where an alternative may appear more than once in the sequence.[3] Let ϵ be a special character that signals the end of the input sequence.

Definition 4.1 *A* finite automaton *is a four-tuple* $\langle M, q, f, g\rangle$ *where*

1. M *is a finite set of states.*
2. $q \in M$ *is the initial state.*
3. $f : M \longrightarrow X$ *is the output function of the automaton.* $f(m)$ *is the output of the automaton when it is in state* m *and it reads the character* ϵ*.*

[3]In the memory cells model we allowed an alternative to appear only once in the sequence. If we had allowed an alternative to appear more than once, only the result of Proposition 3.3 would have changed.

4. $g : M \times X \longrightarrow M$ *is the transition function of the automaton. When the automaton is in state* m *and it reads the input character* x*, it moves to the state* $g(m, x)$*.*

For every sequence $s = (s_1, s_2, \ldots, s_k)$, we define $\widehat{g}$ inductively by

1. $\widehat{g}(q, s_1) = g(q, s_1)$.
2. $\widehat{g}(q, s_1, \ldots s_i) = g(\widehat{g}(q, s_1, \ldots, s_{i-1}), s_i)$.

In this notation, the automaton reaches state $\widehat{g}(q, s)$ after reading the sequence s, and outputs $f(\widehat{g}(q, s))$ after reading the sequence s followed by ϵ.

Definition 4.2 *An automaton* $A = \langle M, q, f, g \rangle$ *computes a choice function* c *if*

$$c(S) = f(\widehat{g}(q, s))$$

for every choice problem $S \in S(X)$*, and for every sequence of alternatives* $s = (s_1, s_2, \ldots, s_k)$*, such that* S *contains exactly all the different elements of* s*.*

It should be noted that this definition requires a certain amount of *robustness* from the automaton. The automaton cannot assume that it will see an alternative only once in a sequence. This is a significant restriction but we find it natural, as this is what happens in many decision problems. For example, when a DM scans the dishes on the menu in a restaurant, he may read certain parts of the menu more than once.

It is left to define the *complexity* of a choice function. The complexity of a choice function c is the minimal number M, such that there exists an automaton with M states that computes the function. More formally, let $M(\alpha)$ denote the number of states in an automaton α. Then,

Definition 4.3 *The complexity of a choice function* c *is*

$$comp(c) = \min\{\ M(\alpha) \mid \alpha \text{ is an automaton that computes } c\ \}.$$

4.2 A few facts about automata and choice functions

The following two simple propositions present an upper bound and a lower bound on the complexity of any choice function.

Proposition 4.4 *Any choice function can be computed by an automaton with 2^N states. That is, for every choice function c, $comp(c) \leq 2^N$.*

Proof. Let c be a choice function. Consider the automaton $\alpha = \langle M, q, f, g \rangle$, where:

1. $M = \{ S \mid S \subseteq X \}$.
2. $q = \emptyset$.
3. $g(S, x_i) = \begin{cases} S & \text{if } x_i \in S \\ S \cup \{x_i\} & \text{if } x_i \notin S. \end{cases}$
4. $f(S) = c(S)$.

The automaton α has 2^N states, and it computes c. Therefore,

$$comp(c) \leq 2^N.$$

□

Proposition 4.5 *No choice function can be computed by an automaton with less than $N-1$ states. That is, for every choice function c, $comp(c) \geq N-1$.*

Proof. Assume to the contrary that there is a choice function that can be computed by an automaton with $N-2$ states. As every state can output exactly one alternative, this implies that at least two alternatives, say x_1 and x_2, are not possible outputs of the automaton. Therefore, the automaton does not compute correctly the choice from the choice set (x_1, x_2). □

There is a significant gap between the upper bound and the lower bound presented above. The next proposition states that the complexity of most choice functions is close to the upper bound; i.e., the complexity of most choice functions is exponential in the number of alternatives N.

Proposition 4.6 *The proportion of choice functions that can be computed only by automata with more than $\frac{2^N}{N^2}$ states tends to 1 as N tends to infinity.*

Proof. The number of automata with k states is at most $kN^k k^{Nk}$, as this is the product of the number of possible initial states (k), the number of possible outputs (at most N^k), and the number of possible transitions (at most k^{Nk}). As any automaton can compute at most one choice function, these automata can compute at most $kN^k k^{Nk}$ different choice functions. Moreover, for every automaton with less than k states, we can construct an automaton with k states that simulates its action by adding "dummy" states with self-loops to the original automaton. Therefore, $kN^k k^{Nk}$ is an upper bound on the number of choice functions that can be computed by automata with k states or less.

The total number of choice functions is at least $N^{2^{N-1}}$ (see Proposition A.1 in Appendix A). Consequently, the proportion of choice functions that can be computed by automata with k or less states is at most

$$m = \frac{kN^k k^{Nk}}{N^{2^{N-1}}} \ .$$

Let us evaluate $\log m$ and show that when we substitute k by $\frac{2^N}{N^2}$ (we can actually substitute k by $\frac{2^N o(\log N)}{N^2}$), $\log m$ tends to $(-\infty)$ as N grows. From this we can conclude that m tends to 0 as N tends to ∞. Thus,

$$\begin{array}{lll} \log m & = & \log k + k\log N + Nk\log k - 2^{N-1}\log N \\ & \leq_{N\leq k} & (1+k+Nk)\log k - 2^{N-1}\log N \\ & \leq & k(N+2)\log k - 2^{N-1}\log N \\ & =_{k=\frac{2^N}{N^2}} & \frac{2^N}{N^2}(N+2)(N-2\log N) - 2^{N-1}\log N \\ & \leq & 2^N - 2^{N-1}\log N \\ & = & 2^{N-1}(2-\log N) \Longrightarrow_{N\to\infty} -\infty. \end{array}$$

□

4.3 The optimality of rationality

There is a substantial difference between the lower bound of $N-1$ states presented in Proposition 4.5, and the fact that most choice functions can be

computed only by automata with more than $\frac{2^N}{N^2}$ states. Consequently, choice functions that can be computed by automata with $O(N)$ states are much easier to compute than most choice functions. The question still remains, if there are such natural examples of choice functions. The next proposition suggests that rationalizable choice functions are optimal in this respect.

Proposition 4.7 *The complexity of a choice function is exactly $N-1$ if and only if it is rationalizable.*

Proof. First, we show that any rationalizable choice function can be computed by an automaton with $N-1$ states. Indeed, let c be a rationalizable choice function that maximizes the ordering

$$a_1 \succ a_2 \succ \ldots \succ a_{N-1} \succ a_N.$$

Consider the automaton $\alpha = \langle M, q, f, g \rangle$, where:

1. $M = \{a_1, a_2, \ldots, a_{N-1}\}$. That is, we identify the states with the elements of $X \setminus \{a_N\}$.
2. $q = a_{N-1}$.
3. $g(a_j, a_i) = \begin{cases} a_i & \text{if } a_i \succ a_j \\ a_j & \text{otherwise.} \end{cases}$
4. $f(a_j) = a_j$.

It is easy to verify that α computes the function c.

The complicated part is to show that a choice function c that can be computed by an automaton α with $N-1$ states is rationalizable. To do so, we prove in several steps that the construction of α must obey several restrictions, which imply that α induces a rationalizable choice function. Without loss of generality, we denote $X = \{1, 2, \ldots, N\}$ and assume that:

$$\left.\begin{array}{ll} c(1,2,\ldots,N) & = 1 \\ c(2,3,\ldots,N) & = 2 \\ \quad\vdots & \\ c(N-2,N-1,N) & = N-2 \\ c(N-1,N) & = N-1 \end{array}\right\} \tag{1}$$

Step 1. The $N-1$ states of α must have different outputs. Otherwise, there are two alternatives that the automaton (and therefore c) never chooses. Let us identify the states of α with their outputs. According to (1), the states of α are $1, 2, \ldots, N-2, N-1$.

Step 2. Let us examine state i. Since $c(i, i+1, \ldots, N) = i$, the input sequence $(i, i+1, \ldots, N)$ must result in the automaton reaching state i. For every additional input $i \leq j \leq N$, the automaton must remain in state i (otherwise, its output is changed when it should not be). Therefore, we conclude that $g(i, j) = i$ whenever $i \leq j \leq N$. Moreover, for $i > 1$, if the input sequence $(i, i+1, \ldots, N)$ is followed by the alternative $i-1$, the automaton must move from state i (where it was before $i-1$ appeared) to state $i-1$, because $i-1$ is the new output. This implies that $g(i, i-1) = i-1$. It is left to examine the transitions $g(i, k)$ for $k < i-1$.

Step 3. There are no backward transitions in α; i.e., it is impossible to have $g(i, k) = j$ for $k < i-1$ and $j > i$. To see why, assume to the contrary that such a transition is possible. Consider the sequence $(i, i+1, \ldots, N)$, which results in the automaton reaching state i. If the next input is k, the automaton must move to state $j > i$. If we inserted ϵ now, the automaton would output j. However, if the sequence continues with the elements $(j-1, \ldots, i, \epsilon)$, then according to step 2, we would reach state i again and the output would be i. Yet, in both cases the exact same elements appear and therefore the automaton should output the same element! Consequently, there are no backward transitions in α. This implies that the initial state is $N-1$; otherwise, since there are no backward transitions, the automaton would not output the correct choice for $(N-1, N)$.

Step 4. Let $k < i-1$, and suppose that $g(i, k) = j$. We have already showed in step 3 that $j \leq i$. We now show that $j = k$. Indeed, consider the sequence:

$$S = (\underbrace{N, N-1, \ldots, i+1, i}_{S1}, \underbrace{k}_{S2}, \underbrace{i-1, i-2, \ldots, k+1}_{S3}) \ .$$

According to Assumption (1) on the choices of c, this sequence should result in the automaton reaching state k. However,

- If $j > k$, then the subsequence $S1$ would bring us to state i. The subsequence $S2$ would bring us to state j. Since $j > k$, according to step 2, subsequence $S3$ would bring us to state $k+1$. Yet, we should reach state k.
- If $j < k$, then the subsequence $S1$ followed by $S2$ would bring us to state j. Since $j < k$, subsequence $S3$ would leave us in state j, although we should reach state k.

Consequently, we must have $g(i,k) = k$, for $k < i-1$.

To conclude, according to steps 1-4 the automaton $\alpha = \langle M, q, f, g\rangle$ must be defined as follows:

1. $M = \{1, 2, \ldots, N-1\}$.
2. $q = N-1$.
3. $g(j,i) = \begin{cases} i & \text{if } i < j \\ j & \text{otherwise.} \end{cases}$
4. $f(j) = j$.

The construction of the automaton α implies that α induces a choice function that maximizes the ordering $1 \succ 2 \succ \ldots \succ N-1 \succ N$. In other words, c is rationalizable. □

4.4 The second-best and the median

Throughout this paper, we have paid special attention to the median and the second-best choice functions. In this subsection, we explore the complexity of these two choice functions. We show that the complexity of the second-best choice function is quadratic in the number of alternatives N, and that the complexity of the median choice function is exponential in N.

Proposition 4.8 *The complexity of a second-best choice function c is*

$$\binom{N}{2} \leq comp(c) \leq \binom{N+1}{2} + 1.$$

Proof. The lower bound is obtained by showing that for every pair of alternatives (x_i, x_j) where $x_i \succ x_j$, the automaton that computes the function must reach a different state. Indeed, consider the two pairs:

1. (a_i, a_k) where $a_i \succ a_k$.
2. (a_j, a_m) where $a_j \succ a_m$.

Then,

- if $k \neq m$, the two pairs must reach different states because the choice from the two pairs is different.
- If $k = m$, then it must be that $i \neq j$. Assume to the contrary that both pairs reach the same state. Consequently, if we input the highest element (according to $\succ$) after the pairs, they will reach the same state (with the same output). Yet, their choice should be different.

The upper bound is obtained by an explicit construction. Assume that our second-best choice function, denoted by c, chooses the second maximal element according to the ordering $a_1 \succ a_2 \succ \ldots \succ a_N$. Let $\alpha = \langle M, q, f, g \rangle$ where:

1. $M = \{(a_i, a_j) \mid a_i \succ a_j \text{ or } i = j\} \cup \{q_0\}$ where the first element of every state is the highest element seen so far, and the second element is the second-highest element.
2. $q = q_0$.
3. $f(a_i, a_j) = a_j$ because we output the second-highest element.
4. $g(q_0, a_i) = (a_i, a_i)$.
5. $g((a_i, a_j), a_k) = \begin{cases} (a_k, a_i) & \text{if } a_k \succ a_i \\ (a_i, a_k) & \text{if } a_i \succ a_k \succ a_j, \text{ or } a_i \succ a_k \text{ and } i = j \\ (a_i, a_j) & \text{otherwise.} \end{cases}$

α computes our choice function c. Therefore,

$$comp(c) \leq |M| = \binom{N}{2} + N + 1 = \binom{N+1}{2} + 1.$$

□

Remark 4.9 *The construction of the automaton in Proposition 4.8 can be used to compute any "choose one of the best two" function, with the necessary adjustments of the output function f.*

Proposition 4.10 *The complexity of a median choice function is at least* $\frac{2^N}{N^2}$.

Proof. Let α be an automaton that computes a median choice function *med* (which is based on the ordering $\succ$). We show that at most N^2 choice problems can reach the same state of α. As there are $2^N - N - 1$ choice problems, this implies that α must have at least $\frac{2^N-N-1}{N^2}$ states.

Indeed, let us fix a choice problem $A = \{\ a_1, a_2, \ldots, a_k, \ldots, a_{2k}\ \}$ with an even number of alternatives $2k$, where $med(A) = a_k$. Denote by q_A the state of the automaton after reading the elements of A. Assume that the automaton reaches the same state q_A after reading the elements of another set B. The conditions that B has to satisfy are:

(1) $med(B) = med(A) = a_k$. Otherwise, if the next character were ϵ, q_A would have to output two different elements.

(2) B has at most one element that is *higher* than a_k according to $\succ$, and does not appear in A. Otherwise, suppose that B has two elements x_1, x_2 that are higher than a_k, and that do not appear in A. If the automaton reaches q_A after seeing the elements of A or the elements of B, then it must continue to the same state if the next elements are (x_1, x_2). That is, the automaton would output the same element after seeing (A, x_1, x_2) and (B, x_1, x_2). Yet, the medians of these two sequences are different, because $med(A, x_1, x_2) \neq med(A) =_{(1)} med(B) = med(B, x_1, x_2)$.

(3) B has no element x that is *lower* than a_k and does not appear in A. Otherwise, the automaton would reach the same state after seeing the sequences (A, x) and (B, x). Yet, the medians of these two sequences are different, because $med(A, x) \neq med(A) =_{(1)} med(B) = med(B, x)$.

Note that if A is a set of odd size, then condition (1) remains the same and conditions (2)-(3) are translated to:

- Condition (2): B has at most one element that is *lower* than a_k according to $\succ$, and does not appear in A.
- Condition (3): B has no element x that is *higher* than a_k, and does not appear in A.

Applying conditions (1)-(3) to B (and symmetrically to A) implies that B must be one of the following:

- A set of $2k-1$ elements which is a subset of A (there are $2k-1$ such sets). Otherwise, there are at least two elements of A that do not appear in B.
- A set of $2k$ elements with one element that does not appear in A and that is higher than a_k (there are at most $2k(N-2k)$ such sets).
- A set of $2k+1$ elements, which is a superset of A (there are at most $N-2k$ such sets).

Therefore, at most $1+2k-1+2k(N-2k)+N-2k \leq N^2$ sets can reach the same state of α. □

4.5 The connection between memory cells and automata

The memory cells model and the automata model are related. Any choice function that can be computed by k memory cells can be computed by an automaton with $(N+1)^k$ states. To see why, think of the automaton as imitating the action of the memory cells. In order to imitate the action of the memory cells, the automaton would need a state for every possible configuration of the memory cells, and a transition function that moves from one state to another according to the respective change in the configuration of the memory cells. More formally, given a choice function c that can be computed by k memory cells, consider the automaton $\alpha = \langle M, q, f, g \rangle$ where:

1. $M = \{a = (a_1, \ldots, a_k) \in (X \cup \{e\})^k\}$, where e stands for an empty memory cell.
2. $q = (e, \ldots, e)$, because before the computation starts, the memory cells are empty.

3. $f(a_1, \ldots, a_k) \in \{a_1, \ldots, a_k\}$ is the element that will be moved to the first memory cell (i.e., will be chosen), if the special character ϵ appears next.

4. $g((a_1, \ldots, a_k), x) = (b_1, \ldots, b_k)$ *if and only if* the configuration of the memory cells changes from $(a_1, \ldots, a_k)$ to $(b_1, \ldots, b_k)$ when the alternative x appears.

The automaton α computes c, and therefore $comp(c) \leq (N+1)^k$.

In the other direction, we are unable to define a connection between the number of states in an automaton that computes a choice function, and the number of memory cells needed to compute the same function.

5 A short discussion

This paper has investigated, through the model of choice functions, two computational aspects of choice: the ability to remember the past and the ability to perform complex computations. Obviously, our goal is not to cover all the computational considerations of choice, but rather to focus on a few aspects, which we believe to be important. Still, we want to point out two aspects, which are not discussed in the paper, and deserve special attention.

Firstly, the paper does not deal with the question of how difficult it is to formulate our underlying preferences. A choice function can be viewed as a set of rules that govern our choices. Formulating these rules is preliminary to the actual computation of choices from choice sets. It involves understanding the general problem we are facing, identifying the elements of the set X, and shaping our beliefs and tastes. While formulating these rules may be simple in some cases (e.g., when the alternatives are dishes in a restaurant that are ranked only according to their price), this process may be very difficult in other cases (e.g., when it is difficult to identify the alternatives, or when we rank them according to several criteria).

Secondly, our analysis deals with the computation of complete choice functions. Sometimes, we wish to compute partial choice procedures that are neither defined for all the subsets of the ground set X, nor induce choice functions. For example, consider a procedure that for a given sequence of

alternatives searches for the first element with a utility that is greater than some threshold θ. For sequences with no such element, the procedure is either not defined or fails to output a relevant choice. One might argue that this choice procedure is not more difficult to compute than a rationalizable choice function. Indeed, if we use the tools introduced in the paper, then this procedure can be computed by an automaton with at most $k+1$ states, where $k \leq N$ is the number of elements in X with utility higher than θ, and we add an additional initial state which outputs a "fail" signal. In the memory cells model, the procedure needs one memory cell since it requires no history.

With these points in mind, the results of the paper can be summarized as follows. Computationally speaking, a choice function that maximizes a utility function is optimal. Almost all other functions, including some natural ones, do not "survive" the computational test. These results suggest that rational behavior (in the sense of maximizing a utility function) can be derived from the assumption that decision-makers have either limited memory or limited computational power.

Appendix

A A lower bound on the number of choice functions

Denote the number of choice functions on N alternatives by $f(N)$. Then,

$$f(N) = 2^{\binom{N}{2}}\ 3^{\binom{N}{3}}\ \cdots\ \left(\frac{N}{2}\right)^{\binom{N}{\frac{N}{2}}}\ \cdots\ N^{\binom{N}{N}} = \prod_{k=2}^{N} k^{\binom{N}{k}}$$

because for every $2 \leq k \leq N$ there are $\binom{N}{k}$ choice sets of size k on which choice functions can differ, and there are k possible choices from every such choice set.

Proposition A.1 *Let $N \geq 5$. Then, $f(N) > N^{2^{N-1}}$.*

Proof. By induction on the number of alternatives N.
For $N = 5$, the inequality holds. Let us assume that $f(N) > N^{2^{N-1}}$, and prove that $f(N+1) > (N+1)^{2^N}$. Indeed, using the combinatorial identity $\binom{N+1}{k} = \binom{N}{k} + \binom{N}{k-1}$, we have:

$$\begin{aligned}
f(N+1) = &\ \textstyle\prod_{k=2}^{N+1} k^{\binom{N+1}{k}} = \prod_{k=2}^{N+1} k^{\binom{N}{k}+\binom{N}{k-1}} \\
= &\ \textstyle\prod_{k=2}^{N} k^{\binom{N}{k}} \prod_{k=2}^{N+1} k^{\binom{N}{k-1}} = f(N) \prod_{k=2}^{N+1} k^{\binom{N}{k-1}} \\
= &\ \textstyle 2^N f(N) \prod_{k=3}^{N+1} k^{\binom{N}{k-1}} = 2^N f(N) \prod_{k=2}^{N} (k+1)^{\binom{N}{k}} \\
= &\ \textstyle 2^N f^2(N) \frac{1}{f(N)} \prod_{k=2}^{N} (k+1)^{\binom{N}{k}} = 2^N f^2(N) \prod_{k=2}^{N} (\frac{k+1}{k})^{\binom{N}{k}} \\
\geq &\ \textstyle 2^N f^2(N) (\frac{N+1}{N})^{\sum_{k=2}^{N}\binom{N}{k}} = 2^N f^2(N) (\frac{N+1}{N})^{2^N-N-1} \\
\geq_{2>\frac{N+1}{N}} &\ \textstyle 2^{N-1} f^2(N) (\frac{N+1}{N})^{2^N-N} \\
\geq_{\text{i.a.}} &\ \textstyle 2^{N-1} N^{2^N} (\frac{N+1}{N})^{2^N-N} = (N+1)^{2^N} 2^{N-1} (\frac{N}{N+1})^N \\
\geq &\ (N+1)^{2^N}
\end{aligned}$$

where the last inequality is derived from the inequality $(\frac{N}{N+1})^N \geq \frac{1}{e}$. □

B Computing the median with $\frac{N}{3}+1$ memory cells

It is possible to compute the median with $\frac{N}{3}+1$ cells due to the following insight: when there are $\frac{N}{3}+1$ memory cells filled with elements and a new element appears, we have a total of $\frac{N}{3}+2$ elements. At least one of them (either the highest or the lowest according to $\succ$) cannot be chosen, because we cannot "pad" both sides of the $\frac{N}{3}+2$ elements with $\frac{N}{3}$ elements or more on every side, as we have only N elements in total.

Let us divide the N elements of the ground set X into three regions according to $\succ$: $\underbrace{a_1 \succ \ldots \succ a_{\frac{N}{3}}}_{\text{A}} \succ \underbrace{a_{\frac{N}{3}+1} \succ \ldots \succ a_{\frac{2N}{3}}}_{\text{B}} \succ \underbrace{a_{\frac{2N}{3}+1} \succ \ldots \succ a_N}_{\text{C}}$,
and describe an algorithm that uses the above insight to implement the median choice function with $\frac{N}{3}+1$ cells.
When a new element y appears, the algorithm does the following:

Case 1: There are still empty memory cells.

- Place the median in the first cell in the following way:
  1. If we are in an "even" stage, i.e. the number of elements revealed so far (including y) is even, store the two medians up to this stage in the first two cells (with the higher according to $\succ$ in the first cell).
  2. If the number of elements revealed so far (including y) is odd, store the median in the first cell.
- Arrange the rest of the elements according to $\succ$ in the remaining memory cells. That is, put the maximal element in the first available cell, the second maximal in the second available cell, etc.

Case 2: There are no empty memory cells. Denote the elements in the cells (according to their location) by $x_1, x_2, \ldots, x_{\frac{N}{3}+1}$, and suppose that y appears next. Assume that the median up to the current stage (before y appeared) is x_1. Moreover, if the previous stage was even, then x_2 is the second median. Note that after y is revealed there are $\frac{N}{3}+2$ elements, and at least one of them (as mentioned above) cannot be chosen.

- Execute exactly one of the following options:
  1. If we are in an even stage and $y \succ x_1$, then move x_1 to the second cell, and store the element, which is the closest element to x_1 from above (according to $\succ$), in the first cell.
  2. If we are in an even stage and $x_1 \succ y$, then leave x_1 in the first cell, and store the element, which is the closest element to x_1 from below (according to $\succ$), in the second cell.
  3. If we are in an odd stage and $y \succ x_1$, then leave x_1 in the first cell.
  4. If we are in an odd stage and $x_2 \succ y$, then move x_2 to the first cell.
  5. If we are in an odd stage and $x_1 \succ y \succ x_2$, then store y in the first cell.
- "Forget" an element that cannot be chosen in the following way:[4]
  1. If the minimal element according to $\succ$ is from region C, forget it.
  2. Otherwise, the maximal element is from region A, so forget it.
- Arrange the rest of the elements in the remaining cells according to $\succ$.

Note that the algorithm can distinguish between even stages and odd stages by comparing the elements in the first and second cells. If the element in the first cell is higher (according to $\succ$) than the element in the second cell, then the previous stage was even. Otherwise, the previous stage was odd. In even stages, the algorithm always stores the two median elements in the first two cells. In odd stages, the algorithm always stores the median in the first cell. As the algorithm always stores the median element up to the current stage in the first cell, we are guaranteed that when the sequence of alternatives is over, the median element will indeed be stored in the first memory cell. □

[4] In this step of the algorithm, we assume that the DM can identify the relative location of elements according to $\succ$. This assumption is consistent with the model, as we do not limit the power of the DM in any way except for memory. Removing this assumption will imply that the DM needs even more than $\frac{N}{3} + 1$ memory cells to compute the median.

References

[1] B. Baigent and W. Gaertner, *Never choose the uniquely largest: A characterization*, Economic Theory **8** (1996), 239–249.

[2] W. Gaertner and Y. Xu, *On rationalizability of choice functions: A characterization of the median*, Social Choice and Welfare **16** (1999), 629–638.

[3] J.E. Hopcroft and J.D. Ullman, *Introduction to automata theory: Languages and computation*, Addison Wesley, Cambridge, Massachusetts, 1979.

[4] G. Kalai, *Learnability and rationality of choice*, Discussion paper no. 261, Center for the Study of Rationality, Hebrew University (2001).

[5] G. Kalai, A. Rubinstein, and R. Spiegler, *Comments on rationalizing choice functions which violate rationality*, Econometrica **70** (2002), 2481–2488.

[6] E. Lehrer, *Repeated games with stationary bounded recall strategies*, Journal of Economic Theory **46** (1988), 130–144.

[7] R. D. Luce and H. Raiffa, *Games and decisions: Introduction and critical survey*, New York: Wiley, 1957.

[8] D. McFadden, *Rationality for economists?*, Journal of Risk and Uncertainty **19** (1999), 73–105.

[9] A. Neyman, *Bounded complexity justifies cooperation in the finitely repeated prisoner's dilemma*, Economic Letters **19** (1985), 227–229.

[10] _____, *Finitely repeated games with finite automata*, Mathematics of Operations Research **23** (1998), no. 3, 513–552.

[11] M. Osborne and A. Rubinstein, *A course in game theory*, The MIT Press, Cambridge, Massachusetts, 1994.

[12] M. Piccione and A. Rubinstein, *Modeling the economic interation of agents with diverse abilities to recognize equilibrium patterns*, Journal of European Economic Association (forthcoming).

[13] M. Rabin, *Psychology and economics*, Journal of Economic Literature **XXXVI** (1998), 11–46.

[14] A. Rubinstein, *Finite automata play the repeated prisoners' dilemma*, Journal of Economic Theory **39** (1986), 83–96.

[15] ———, *Why are certain properties of binary relations relatively more common in natural language*, Econometrica **64** (1996), 343–356.

[16] ———, *Modeling bounded rationality*, Zeuthen Lecture Book Series, The MIT Press, Cambridge, Massachusetts, 1998.

[17] W. Samuelson and R. Zeckhauser, *Status quo bias in decision making*, Journal of Risk and Uncertainty **1** (1988), 7–59.

[18] A. Sen, *Internal consistency of choice*, Econometrica **61** (1993), no. 3, 495–521.

[19] I. Simonson and A. Tversky, *Choice in context: Tradeoff contrast and extremeness aversion*, Journal of Marketing Research **29** (1992), 281–295.

[20] A. Tversky and E. Shafir, *Choice under conflict: The dynamics of deffered choice*, Psychological Science **3** (1992), no. 6, 358–361.

[21] M. Vidyasagar, *A theory of learning and generalization*, Communications and control engineering series, Springer, London, 1997.

[22] A. Wilson, *Bounded memory and biases in information processing*, NAJ Economics **5:3** (2002).

[23] M. E. Yaari, *On the role of "dutch books" in the theory of choice under risk*, Nancy L. Schwartz Memorial Lecture (1985).