

Procedural Analysis of Choice Rules with Applications to Bounded Rationality*

Yuval Salant

Graduate School of Business, Stanford University

salant@stanford.edu

November 11, 2007

Abstract

I investigate the procedural complexity of a choice rule as measured by the size of the minimal automaton that implements it. I first identify simple behavioral properties that determine procedural complexity. I then show that any choice rule that is procedurally simpler than utility maximization is sensitive to presentation effects. Finally, I prove that choice rules that result from an optimal tradeoff between maximizing utility and minimizing procedural complexity are history-dependent satisficing procedures that exhibit primacy and recency effects, as well as a default tendency.

*I am indebted to Bob Wilson for his devoted guidance, constant support, and most valuable suggestions. I am grateful to Ariel Rubinstein and Jeremy Bulow for the encouragement, productive discussions, and important comments. I thank Gil Kalai, Ron Siegel, and Andy Skrzypacz for most insightful feedback in various stages of this project. I also thank Matt Jackson, Jon Levin, Michael Ostrovsky, Roy Radner, and Ilya Segal for helpful comments. This research is supported in part by the Leonard W. and Shirley R. Ely Fellowship of the Stanford Institute for Economic Policy Research.

1 Introduction

Economists have long recognized the need to explore cognitive and procedural aspects of decision making, such as limited computational power. Herbert Simon [21] suggested that procedural considerations may push decision makers towards various forms of “boundedly” rational behavior such as satisficing. Kahneman and Tversky [7] identified important behavioral biases and heuristics in decision making, and proposed decision models that accommodate them. The Bounded Rationality and Behavioral Economics literature has pursued the ideas of Simon and Kahneman and Tversky from theoretical, experimental, and empirical perspectives.¹ A common claim in this literature is that procedural and cognitive considerations lead decision makers to behave in ways that are inconsistent with utility maximization.

This paper studies this claim by analyzing a procedural framework of individual choice. I define the procedural complexity of a choice rule by the size of the minimal automaton that implements it, and compare the complexity of utility maximization to that of other behaviors. I find that any choice rule that is procedurally simpler than utility maximization is sensitive to presentation effects. Moreover, an optimal tradeoff between maximizing utility and minimizing complexity results in history-dependent satisficing procedures that exhibit primacy and recency effects, as well as a default tendency.

In the model of individual choice I investigate, the decision maker encounters the elements of a choice problem sequentially, and chooses one of them. This is the case, for example, in the online marketplace, where consumers choose among products listed in order. This is also the case in the labor market, where recruiters interview candidates sequentially, and in elections, where voters elect from ordered ballots. A *choice function from lists* (see Rubinstein and Salant [18]) summarizes individual behavior by assigning a chosen element to every non-empty list. A *rational choice function* assigns to every list the maximal element according to some utility function; a *satisficing choice function* chooses from every list the first element above some aspiration threshold, and if none exists, it chooses the last element. Choice functions from lists can describe a variety of cognitive and procedural effects such as primacy and recency. The model can also be easily modified to allow for situational cues such as a default alternative.

To explore procedural aspects of individual behavior, I model the decision-making process as an automaton.² Intuitively, an *automaton* is a machine with two main components: Hardware and Software. The hardware of the automaton is a finite set of information states, which includes two special states: an “initial” state and a “Stop” state. The software of the automaton includes a transition function and an output function. For every list, the automaton starts in the initial state.

¹See Rubinstein [17] for a discussion of models of bounded rationality, and Rabin [14] for a survey of developments in behavioral economics.

²The automaton model is one of the basic tools developed in computer science to investigate computational complexity (see Hopcroft and Ullman[6]). Rubinstein [15], Neyman [13], and Abreu and Rubinstein [1] adapt the automaton model to study procedural aspects in repeated games. Dow [4] and Wilson [23] study procedural models of inference-making, which may be thought of as variations of the automaton model.

It reads the elements of the list in order, and uses the transition function to determine the next state as a function of the current state and the element just read. When the Stop state is reached or the list ends, the output function determines the output of the automaton as a function of the last element read by the automaton, and the state of the automaton when reading that element.

This specification of an automaton differs from the one traditionally used in game theory. The introduction of the Stop state accommodates the possibility of making a choice before reading the entire list. Conditioning output on both the current state and the element just read enables distinguishing between “short-term” and “long-term” memory. Long-term memory refers to the states of the automaton – any information that is not used immediately must be “incorporated” in a state. Short-term memory refers to the ability of the automaton to output the element just seen without using a state for that purpose, as long as the automaton has not yet read the next element.

I define the procedural complexity of a choice rule as the amount of the long-term memory required for its implementation. More specifically, an automaton *implements* a choice function from lists if for every list, the automaton outputs the element that the function assigns to that list. The *procedural complexity* of a choice function is the minimal number of states (excluding the Stop state) required to implement the function.

The number of states may also be interpreted as a coarse measure of how much information processing is involved in making choices. In satisficing, for example, the only information relevant for making a choice is whether a satisfactory element has already appeared. Once it appears, the decision maker chooses it and moves to the Stop state. Thus, only one state is required for implementation (in addition to the Stop state). The complexity of rational choice, on the other hand, nearly equals the number of feasible alternatives because information processing depends on the identity of the best element seen so far.

My first objective is to identify properties of a choice function that determine its procedural complexity. The first such property is whether past information suffices to make a decision: a list L is *undecided* if it is empty or there is some continuation L' such that the element chosen from L is different from the one chosen from the concatenated list (L, L') . In satisficing, for example, any list that does not contain at least one satisfactory element is undecided. The second property is how past information affects future information processing: two lists L_1 and L_2 are *separable* if there exists a continuation L' such that the element chosen from (L_1, L') is different from the one chosen from (L_2, L') . In satisficing, for example, no two undecided lists are separable because the chosen element is always the first satisfactory element (or the last element) from the continuation.

Using these two properties, I define the *informational index* of a choice function as the cardinality of a maximal set of undecided lists, such that every two lists in the set are separable. The informational index constitutes a lower bound on the procedural complexity of a choice function. Indeed, if two undecided lists reach the same state of the automaton, then future information processing is identical, which implies they are not separable. Theorem 1 shows that the informational index of a choice function precisely equals the minimal number of states (excluding the Stop state) required

for implementation. This result adapts the Myhill-Nerode Theorem from automata theory (see Hopcroft and Ullman [6]) to the context of individual decision making.³

After identifying behavioral properties that determine procedural complexity, I discuss rational choice, or utility maximization, and its relation to other behaviors. First, because the complexity of rational choice nearly equals the cardinality of the outcome space, any situational cue that “simplifies” the outcome space reduces the burden of rational choice. Consider, for example, a situation in which a decision maker is endowed with a default alternative. From every list, he chooses the maximal element according to his utility function if its utility exceeds that of the default. Otherwise, he keeps the default. In this case, the number of states required for implementation is smaller than in rational choice without a default, because any alternative that is inferior to the default is ignored. If the default alternative benefits from a utility bonus in the spirit of the status-quo bias [8], maximization becomes even simpler.

Second, although utility maximization is as complicated as the outcome space, it is simplest among all choice rules that are robust to presentation effects.⁴ Theorem 2 shows that rational choice functions are *uniquely* simplest among all choice functions that are order-independent, i.e., functions that choose the same element from every two lists that are permutations of one another. As an immediate implication, rational choice functions are also uniquely simplest among all choice functions that are repetition-independent, i.e., functions that choose the same element from a list L and the lists (L, a) and (a, L) , where a is an element of L .

Utility maximization continues to be easiest to implement among all choice functions that are order-independent (alternatively, repetition-independent) for two-element lists, but possibly order-dependent (alternatively, repetition-dependent) for larger lists. Therefore, *any choice function simpler than rational choice is order-dependent for some two-element list, and repetition-dependent for some two-element list.*

Third, because the complexity of rational choice nearly equals the cardinality of the outcome space, rational choice is complicated when the outcome space is large. This motivates designing choice procedures that are “close” to utility maximization but are easier to implement. For example, in an organization, a manager may replace utility maximization with a simpler choice rule if a non-sophisticated subordinate is to implement that rule. I investigate this *choice design* problem under the assumption that the objective is to minimize the maximal utility loss, or regret, associated with the resulting choice rule. Theorem 3 establishes that under mild restrictions the generically unique solution to this choice design problem is a history-dependent satisficing procedure in which the threshold for making a choice adjusts according to the identity of the elements that appear in the list.

³See Kalai and Stanford [9] for a related analysis in the context of repeated games.

⁴Several previous papers have pointed out that rational choice is “simple”. Campbell [3] and Bandyopadhyay [2] show that intuitive procedural properties characterize choice correspondences that can be represented as the maximization of a binary relation. Rubinstein [16] suggests that the frequent appearance of order relations in natural language can partly be explained by the fact that order relations are easy to describe. Kalai [10] shows that rational behavior is easy to learn in the Probably-Approximately-Correct learning model.

This history-dependent satisficing procedure exhibits the following presentation effects: (i) Primacy effect – moving an element, except maybe the last, towards the beginning of the list improves the likelihood it is chosen, (ii) Recency effect – moving an element that is not chosen to the last position in the list may cause it to be chosen, and (iii) Default tendency – in the presence of a default, the decision maker ignores all the elements in some utility range above the default, except the last element in the list. Thus, particular kinds of presentation effects, which are considered “biases” in some contexts, are actually “optimal” when taking procedural costs into account.⁵

2 The model of choice from lists

The model of choice I investigate is based on Rubinstein and Salant [18]. Let X be a finite set of N elements. A *list* L is a finite sequence of elements from X . A given element may not appear, appear once, or appear multiple times in a given list. A *choice function from lists* assigns to every non-empty list an element from the list, interpreted as the chosen element.

Choice functions from lists can describe a variety of cognitive and procedural effects. These include primacy effect (individuals may examine the first few alternatives in a list more attentively), recency effect (decision makers may recall the last few alternatives in a list more vividly), reference-point (the first element in a list may serve as a reference point to which subsequent alternatives are compared), “contrast” effect (an element may stand out relative to its neighbors in the list), and “saliency” effect (an element that appears multiple times in a list may draw special attention.)

The model of choice from lists extends the classical model of choice from sets, or frame-independent choice. A choice function from lists is *frame-independent* if it chooses the same element from every two lists that induce the same set of alternatives, i.e., if neither the ordering of the elements nor their frequency affect choice.

The following examples demonstrate the flexibility of the model.

Example 1. Satisficing (Simon [21]). The decision maker has in mind a value function v over X and an aspiration threshold v^* . He chooses the first element x in the list with value $v(x) > v^*$ and, if there is no such element, he chooses the last element. Clearly, satisficing is frame-dependent.

Example 2. Rational choice. The decision maker has in mind a strict preference relation (i.e., complete, asymmetric and transitive) \succ over X . He chooses the \succ -maximal element from every list. A rational choice function is frame-independent.

Example 3. Maximizing with a bias. The decision maker has in mind a utility function u and a “bias” function b from X to R_+ . He evaluates the elements of a list in order. He begins by designating the first element as a “favorite”. When reaching the i ’th element, a_i , the decision maker replaces the current favorite y with a_i if $u(a_i) > u(y) + b(y)$, i.e., he gives a bonus to the current favorite. When the list ends, the decision maker chooses the current favorite. The bonus

⁵Wilson [23] makes a similar argument in the context of inference-making with limited memory.

$b(y)$ may be interpreted as an endowment effect [8].

Example 4. Contrast. The decision maker classifies the elements of X as “conventional” or “non-conventional”. He chooses the first conventional element that appears after two consecutive non-conventional elements. If there is no such element, the last element is chosen.

Example 5. Choosing the most popular element. The decision maker chooses from every list the element that appears the largest number of times in the list. If there is more than one such element, the decision maker chooses the one among them that appears first on the list.

3 Procedural framework

To investigate a choice function’s procedural complexity, I consider an automaton implementing it. In this section, I adapt the standard automaton model to the context of individual decision making. After defining the notion of an automaton, I demonstrate through examples how an automaton makes choices. I then introduce a measure of procedural complexity, and relate it to descriptive properties of the corresponding choice function.

3.1 Automaton

An *automaton* is a finite-state machine $M = \langle Q, q_0, Stop, g, f \rangle$. The finite set Q is interpreted as a set of information states or the hardware of the automaton. The initial state $q_0 \in Q$ is the state in which the automaton starts processing the elements of a list. The state $Stop \notin Q$ enables the automaton to stop and make a choice before the list ends. The transition function $g : Q \times X \rightarrow Q \cup \{Stop\}$ and the output function $f : Q \times X \rightarrow X$ are the software of the automaton. If the automaton is in state q and it encounters the element x it moves to state $g(q, x)$. If $g(q, x) = Stop$ then $f(q, x)$ is the element chosen by the automaton when reading the element x in state q . Otherwise, $f(q, x)$ is the *tentative* choice of the automaton.

For every list $L = (a_1, \dots, a_k)$, the automaton M starts in the initial state q_0 and reads the elements of the list in order. As a function of the current state q (which is initially q_0) and the current element a_i , the automaton moves to state $g(q, a_i)$. If $g(q, a_i) = Stop$, then the automaton stops and chooses $f(q, a_i)$. Otherwise, information processing continues until the automaton stops or the list ends. In the latter case, the automaton chooses $f(q, a_k)$ where a_k is the last element in the list, and q the state of M when reading a_k .

Formally, given an automaton M , define $M(q, (a_i, L'))$ as the element chosen by M if $q \neq Stop$ is the current state of M , and M is about to process the list (a_i, L') . That is,

- (1) $M(q, (a_i, L')) = f(q, a_i)$ if $g(q, a_i) = Stop$ or if L' is empty. Otherwise,
- (2) $M(q, (a_i, L')) = M(g(q, a_i), L')$.

Thus, $M(q_0, L)$ is the element the automaton M chooses from the list L .

Implementation. An automaton M implements a choice function C if $M(q_0, L) = C(L)$ for every list L .

3.2 Examples

In what follows, I use transition diagrams to demonstrate how an automaton operates. The circles in the diagram correspond to the states of the automaton. The left-most circle is the initial state. Edges represent transitions: a character x on an edge from state q to state q' indicates that given that the automaton is in state q and it reads the character x , the automaton moves to state q' . The mapping $x \rightarrow y$ adjacent to state q implies that $f(q, x) = y$.

Assume $X = \{1, 2, 3, 4\}$.

Satisficing. The two-state automaton in Figure 1 implements a satisficing procedure in which 3 and 4 are the satisfactory elements.

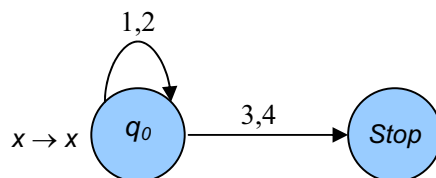


Figure 1: Satisficing

Indeed, when the automaton is in state q_0 and the element x appears, there are two possibilities:

- (1) If $x \in \{3, 4\}$ then $g(q_0, x) = Stop$, and the automaton chooses $f(q_0, x) = x$.
- (2) If $x \in \{1, 2\}$ then $g(q_0, x) = q_0$, and $f(q_0, x) = x$ is chosen *only if* x is the last element in the list.

Thus, from the list $(4, 1, 2)$ the automaton chooses 4 because $g(q_0, 4) = Stop$, and from the list $(2, 1)$ it chooses 1 because 1 is the last element in the list and the automaton did not stop before seeing it.

Maximizing with a bias. Let $u(1) = 0$, and $u(x) = x$ otherwise. Set the bonus associated with each element at $b = 1.5$. The three-state automaton in Figure 2 implements a “maximizing with a bias” procedure based on these primitives.

Contrast. The four-state automaton in Figure 3 implements a contrast procedure in which the elements 2 and 3 are “conventional”, and 1 and 4 are “non-conventional”.

Rational choice. The four-state diagram in Figure 4 (in which self-loops are excluded) implements a procedure that maximizes the preference relation $4 \succ 3 \succ 2 \succ 1$.

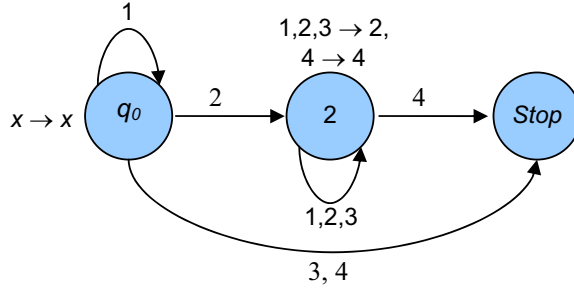


Figure 2: Maximizing with a bias

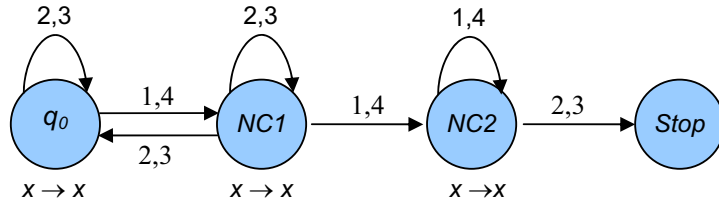


Figure 3: Non-conventional elements highlight the attractiveness of conventional ones

3.3 Procedural complexity

The *procedural complexity* of a choice function from lists C is the minimal integer K such that there exists an automaton with K states (excluding *Stop*) that implements C . If no finite automaton implements C , then the complexity of C is infinite.

This complexity measure has several interpretations in addition to the long-term memory and the information processing interpretations discussed in the introduction. First, states may represent “states of mind” of the decision maker. In satisficing, for example, the decision maker is either unsatisfied (in the initial state) or satisfied (in the *Stop* state), and in the contrast example, he becomes more convinced to make a conventional choice the more non-conventional options he sees. The number of states is also indicative of the size of the automaton.⁶ As such, it may measure the difficulty of procedurally describing or communicating a choice rule. Finally, the number of states corresponds to the difficulty of learning a choice rule: the amount of data required to learn a choice function of procedural complexity K is bounded below and above by linear functions of K and $K \log K$ respectively (see the appendix for details on this point.)

Of course, there are many other complexity measures of interest. In the basic search model, for example, the source of procedural complexity is the “variable” cost associated with searching and

⁶When fixing the set X , the size of the transition and output functions is linear in the number of states.

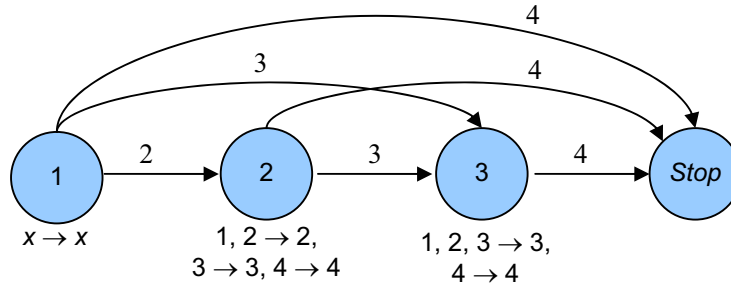


Figure 4: Rational choice

evaluating the next element in the list rather than the “fixed” cost of implementing a given choice rule. This variable cost may be thought of as corresponding to the cost of actual transitions of the automaton rather than the number of states it has. In the context of repeated games, Lipman and Srivastava [12] measure complexity by the responsiveness of an automaton to changes in the history of the game, and Eliaz [5] studies the complexity of the transition function. Analyzing these and other complexity measures is beyond the scope of the current paper.

3.4 Simplest choice functions

Consider an automaton M with one non-stopping state q_0 that implements a choice function C . Then, $f(q_0, x)$ equals x , or else M does not choose correctly from some one-element list. Classifying an element x as “satisfactory” if $g(q_0, x) = \text{Stop}$, or “non-satisfactory” if $g(q_0, x) = q_0$, we have that C is consistent with choosing the first satisfactory element from every list, or the last element in the list if no satisfactory element exists. Thus,

Observation 1 *A choice function has minimal complexity if and only if it is a satisficing procedure.*

3.5 Informational index

One way to determine the complexity of a choice function is to build an automaton that implements the function and prove that implementation is impossible with fewer states. I now show how to determine complexity without constructing the automaton explicitly, using only simple descriptive properties of the choice function.

For two lists L_1 and L_2 , let (L_1, L_2) be the concatenated list that consists of the elements of L_1 followed by the elements of L_2 . Given a choice function C , a list L is C -undecided if L is empty or if there is a continuation L' such that $C(L, L') \neq C(L)$. Otherwise, L is C -decided. Intuitively, a list is C -undecided if after seeing that list, the decision maker is still uncertain about his choice. Two lists L_1 and L_2 are C -separable if there exists a non-empty list L such that $C(L_1, L) \neq C(L_2, L)$.

Thus, two lists are C -separable if after seeing one of them, future choices may be different than after seeing the other.

Informational index. The *informational index* of a choice function C is the cardinality of a maximal set of C -undecided lists such that every two lists in the set are C -separable.

The informational index may be thought of as measuring how much information a choice function uses in making decisions because for every two undecided lists, the function behaves differently if and only if they are separable. Consider, for example, the contrast procedure. In this procedure, a list is undecided if and only if it contains no three consecutive elements, such that the first two are non-conventional and the third one is conventional. Among undecided lists, the procedure distinguishes between lists ending with (i) a conventional element, (ii) a non-conventional element preceded by a conventional one, and (iii) the remaining lists. Thus, the informational index is 3.

The informational index of a choice function constitutes a lower bound on the number of states required for implementation. To see this, assume to the contrary that the informational index of a choice function C is K and there exists an automaton M with fewer than K non-stopping states that implements C . Consider a collection of K lists which are C -undecided and C -separable. Because the lists are undecided, M must reach a non-stopping state after processing any one of them. Because M has fewer than K such states, two of these lists L_1 and L_2 reach the same non-stopping state. But then M chooses the same element from (L_1, L) and (L_2, L) for every continuation L , in contradiction to L_1 and L_2 being C -separable.

The following result states that the informational index of a choice function C is *identical* to the number of states in the minimal automaton that implements C . This result adapts the Myhill-Nerode Theorem [6] from the computer science literature and a related result by Kalai and Stanford [9] from the repeated games literature to the context of individual decision making.

Theorem 1 *The informational index of a choice function from lists C is $K < \infty$ if and only if the minimal automaton implementing C has exactly K states.*

Proof. Assume the informational index of a choice function C is $K < \infty$. As illustrated above, any automaton implementing C has at least K states. I now construct an automaton M with exactly K states that implements C .

Define the binary relation \sim_C over the collection of C -undecided lists as follows: $L_1 \sim_C L_2$ if $C(L_1, L) = C(L_2, L)$ for every non-empty list L . Clearly, the relation \sim_C is an equivalence relation with K equivalence classes. Denote the equivalence class of a list L by $[L]_{\sim_C}$.

The states of M correspond to the K equivalence classes of \sim_C . The initial state is the equivalence class of the empty list. The transition function maps a state q and an element $a \in X$ to another state as follows: take any $L \in q$ and move to state $[(L, a)]_{\sim_C}$ if the concatenated list (L, a) is undecided; if (L, a) is decided, move to the Stop state. Let $f(q, a) = C(L, a)$ for some list $L \in q$.

The functions f and g are well-defined functions. Indeed, assume $L_1 \sim_C L_2$. Then $C(L_1, a) = C(L_2, a)$ for every element a and thus f is well-defined. In addition, because L_1 and L_2 are undecided and $C(L_1, L) = C(L_2, L)$ for every non-empty list L , the list (L_1, a) is undecided if and only if (L_2, a) is undecided. If both are undecided, then $(L_1, a) \sim_C (L_2, a)$. Thus g is well-defined.

It remains to show that M implements C . I do so by induction on the length of the list. For every one-element list $L = (a)$, $M(q_0, L) = M(q_0, (a)) = f(q_0, a) = a = C(a)$. Consider now a list $L = (L', a)$ where L' is non-empty. If L' is undecided, then by construction M reaches state $[L']_{\sim_C}$ after reading L' , so $M(q_0, (L', a)) = M([L']_{\sim_C}, a)$. Since a is the last element in the list, $M([L']_{\sim_C}, a) = f([L']_{\sim_C}, a)$. By the definition of f , $f([L']_{\sim_C}, a) = C(L', a) = C(L)$. Thus, $M(q_0, L) = C(L)$. If L' is decided, denote by L'' the shortest beginning of L' that is decided. By definition of the function $M(\cdot)$, $M(q_0, L) = M(q_0, L'')$, by the induction assumption $M(q_0, L'') = C(L'')$, and by the fact that L'' is decided $C(L'') = C(L)$. Thus, $M(q_0, L) = C(L)$. ■

By Theorem 1, some choice functions cannot be implemented by finite automata. Consider, for example, the “choosing the most popular element” procedure. If $a, b \in X$, $a \neq b$, then $\{L_k = (a, \underbrace{b, \dots, b}_k)\}_{k \geq 0}$ is an infinite set of undecided and pairwise separable lists. Thus, the informational index is infinite, so this function cannot be implemented by a finite automaton.

4 Rational choice and other behaviors

In rational choice, information processing depends on the identity of the best alternative seen so far. Hence, the minimal number of states required for implementation nearly equals the number of feasible alternatives. After establishing this point in observation 2, I demonstrate how situational cues simplify the difficulty of maximizing. I then prove that any choice function that is procedurally simpler than rational choice must be frame-dependent even for some two-element list.

Observation 2 *The procedural complexity of a rational choice function is $N-1$, where $N = |X|$.*

Proof. Let C be a rational choice function that maximizes the preference relation \succ , x_{\max} be the \succ -maximal element in X , and x_{\min} be the \succ -minimal element in X . By Theorem 1, it suffices to show that the informational index of C is $N - 1$. Consider the collection of one-element lists $\{(x) \mid x \in X \setminus \{x_{\max}\}\}$. These lists are C -undecided (consider the continuation (x_{\max})) and pairwise C -separable (consider the continuation (x_{\min}) .) Thus, the index of C is at least $N - 1$.

Consider any other C -undecided list L . If L is empty then it is not separable from (x_{\min}) . If L is C -undecided and non-empty, then it does not contain x_{\max} , and in particular $C(L) \in X \setminus \{x_{\max}\}$. In addition, L is not C -separable from the one-element list $(C(L))$, because for every continuation the \succ -maximal element among $C(L)$ and the \succ -best element in the continuation is chosen. Thus, there is no larger collection of C -undecided lists that are pairwise C -separable, implying that the index is $N - 1$. ■

It is straightforward to construct an automaton with $N - 1$ states that implements rational choice. Indeed, let x_{\max} be \succ -maximal in X and x_{\min} be \succ -minimal in X . Denote the set of states by $Q = X \setminus \{x_{\max}\}$ and the initial state by $q_0 = x_{\min}$, so that every state is associated with a different element in X . When encountering an element $x \neq x_{\max}$ in state q , move to the state that corresponds to the \succ -better element among q and x . When encountering $x = x_{\max}$, move to the *Stop* state. Finally, output the \succ -better element among q and x .

4.1 Rational choice and situational cues

Situational cues may reduce the burden of maximization as the following examples demonstrate.

1. Restricted domain of alternatives. The decision maker is sometimes able to partially affect the identity of the alternatives that appear in choice problems. For example, many websites enable consumers to choose products only in a certain price range. In such cases, the domain of feasible alternatives shrinks from X to $\bar{X} \subset X$, so maximization becomes easier.

2. Restricted domain of choice problems. The decision maker may also be able to affect the ordering of the alternatives. For example, when making an online purchase, the decision maker can often sort the items according to specific attributes, such as price or popularity. The resulting ordering may be correlated with the decision maker's preferences, and may therefore simplify maximization. For example, if the decision maker's preference relation is single-peaked with respect to price, and listing according to price is possible, then maximizing becomes simpler: once the decision maker "passes his peak" he chooses the \succ -maximal alternative between the \succ -best alternative before the peak and the first one after the peak. That is, no states are required for processing elements that appear after the peak. Of course, if pre-sorting generates an ordering that is identical to the individual's preference relation, making choices becomes trivial.

3. Default alternative. Assume the decision maker is endowed with a default alternative δ . When choosing from a list, the decision maker either chooses an element from the list or keeps the default. In order to incorporate the default into the automaton model, the range of the output function f should be modified from X to $X \cup \{\delta\}$. Rational choice in the presence of a default implies that the decision maker chooses from every list the \succ -maximal element x if $x \succ \delta$, and δ otherwise. As long as there are elements in the set X that are \succ -inferior to δ , the complexity of this procedure is smaller than that of rational choice without a default, because the number of "relevant" choices shrinks. Moreover, if the default alternative benefits from a utility bonus in addition to its intrinsic value (as in the status-quo bias [20]), then the set of elements that are \succ -superior to δ may shrink further, and the resulting choice function is even simpler.

In addition to situational cues, internal mechanisms may also simplify maximization. In the maximizing with a bias example, the decision maker gives a bonus $b(x)$ to an element x in addition

to its utility $u(x)$. In this case, an element x for which $u(x) + b(x) \geq u(y)$ for every other element y , does not “require” a state. Indeed, once x is seen, it can either be outputted, if its utility is higher than the utility plus the bonus of the current favorite, or ignored in any other case. Thus, if bonuses are high enough, maximizing with a bias requires fewer states than rational choice.⁷

4.2 Rational choice and frame dependence

I now show that rational choice is procedurally simplest among all choice functions that satisfy certain consistency properties. The first consistency property I investigate is order-independence.

Order-independence for pairs. A choice function C is *order-independent for pairs* if for every two elements $a, b \in X$, $C(a, b) = C(b, a)$.

Observation 3 *Any choice function that is order-independent for pairs is at least as complicated as rational choice.*

Proof. Let C be order-independent for pairs. Denote by $X^* \subseteq X$ the set of all elements $x \in X$ for which there exist elements $w(x), l(x) \in X \setminus \{x\}$ such that $C(x, w(x)) = w(x)$ and $C(x, l(x)) = x$. The cardinality of the set X^* is $\geq N - 2$. Otherwise, there are two elements x_1 and x_2 that are both “winners” ($C(x_i, x) = x_i$ for every $x \in X$) or both “losers” ($C(x_i, x) = x$ for every $x \in X$), which is impossible. Indeed, if x_1 and x_2 are winners, then $C(x_1, x_2) = x_1$ and $C(x_2, x_1) = x_2$, in contradiction to order-independence for pairs. Similarly, there cannot be two losers.

For every $x \in X^*$, the one-element list (x) is undecided since $C(x, w(x)) = w(x) \neq C(x)$. The list (x) is separable from the empty list because $C(x, l(x)) = x \neq C(l(x))$. For every two distinct elements $x, y \in X^*$, the lists (x) and (y) are separable because $C(x, l(x)) = x \neq C(y, l(x))$. Thus, the collection of one-element lists (x) , where $x \in X^*$, and the empty list are undecided and separable. Therefore, the complexity of C is at least $N - 1$. ■

If a choice function is order-independent for more than just two-element lists, a stronger result can be obtained. Theorem 2 shows that if a choice function is order-independent for *every* list and cannot be represented as the maximization of a strict preference relation, then it is *strictly* more complicated than rational choice.

Order-independence. A choice function C is *order-independent* if for every list (a_1, a_2, \dots, a_k) and every permutation σ of $\{1, \dots, k\}$, $C(a_1, a_2, \dots, a_k) = C(a_{\sigma(1)}, a_{\sigma(2)}, \dots, a_{\sigma(k)})$.

A choice function C is *rationalizable* if there exists a strict preference relation \succ such that for every list L , $C(L)$ is the \succ -maximal element in L . Otherwise, C is non-rationalizable.

⁷Exceptions include cases in which for every element x there is $y \in X$ such that $u(x) + b(x) < u(y)$, and in addition $u(x_{\min}) + b(x_{\min}) > u(z)$ where x_{\min} is the u -minimal element in X and $z \in X \setminus \{x_{\min}\}$.

Theorem 2 *Any non-rationalizable order-independent choice function is strictly more complicated than rational choice.*

Proof. Let C be order-independent. By observation 3, the procedural complexity of C is at least $N - 1$. Assume it is exactly $N - 1$. I first state several implications of order independence and having complexity of $N - 1$. I then use these implications to prove that if C is not rationalizable, then complexity is at least N .

Fact 1. There is an element w such that $C(w, x) = w$ for every $x \in X$. Similarly, there is an element l such that $C(l, x) = x$ for every $x \in X$.

Proof. I prove the first part. The proof of the second part is analogous. Assume to the contrary that for every element x there exists an element $w(x) \neq x$ that “beats” x , i.e., $C(x, w(x)) = w(x)$. Because of order-independence, there is at most one element y such that $C(y, z) = z$ for every $z \in X$. Hence, for every element $x \in X$ except at most one, there exists an element $l(x)$ such that $C(x, l(x)) = x$. Therefore, there is a set $X^* \subseteq X$ of cardinality at least $N - 1$ such that if $x \in X^*$, then there are elements $w(x), l(x) \in X \setminus \{x\}$ satisfying $C(x, w(x)) = w(x)$ and $C(x, l(x)) = x$. Applying the second part of the proof of observation 3 to this case, complexity is at least N .

Let $Y = \{(x) \mid x \in X \setminus \{w\}\}$. By fact 1 and order-independence, Y is a collection undecided and pairwise separable lists. Because complexity is $N - 1$, every other list is either decided or not separable from some $(x) \in Y$. This implies:

Fact 2. $C(L) = C(l, L)$ for every list L .

Proof. The empty list is undecided by definition, and hence not separable from some $(x) \in Y$. Therefore, $l = C(l) = C(x, l) = x$ and thus $x = l$. That is, the empty list is not separable from (l) implying that $C(L) = C(l, L)$ for every list L .

Fact 3. $C(L) = w$ if w is an element of L .

Proof. Otherwise, by order-independence the one-element list (w) is undecided, and hence not separable from some $(x) \in Y$. But then $w = C(w, l) = C(x, l) = x$ in contradiction to the fact that $x \neq w$.

Fact 4. $C(a, L) = C(L)$ for every list L and for every element $a \in L$.

Proof. Assume to the contrary there is an element a and a list L that includes a such that $C(a, L) \neq C(L)$. By facts 2 and 3, $a \notin \{w, l\}$. Consider the list (a, a) . It is undecided because $C(a, a, w) = w$, and hence it is not separable from some $(x) \in Y$. Thus, $x = C(x, l) = C(a, a, l) = a$ where the last equality is derived from fact 2 and order independence. Hence, (a, a) is not separable from (a) . Denote by L' a sublist of L in which one instance of a is omitted. Because C is order-independent, $C(L) = C(a, L')$ and $C(a, L) = C(a, a, L')$. Because (a) and (a, a) are not separable, $C(a, L') = C(a, a, L')$ implying that $C(L) = C(a, L)$ which is a contradiction.

To conclude the proof of the theorem, I now show that if C is not rationalizable, complexity

is at least N . Denote by $S(L)$ the set of distinct elements that appear in the list L . By fact 4 and order-independence, C must satisfy $C(L) = C(L')$ if $S(L) = S(L')$. Thus, without loss of generality, we can assume that C is defined over sets.

Hence, if C is not rationalizable, then it violates the standard Independence of Irrelevant Alternatives property. That is, there are sets $A \subset B$ such that $C(L_B) \in A$ but $C(L_A) \neq C(L_B)$, where L_S is some listing of the elements of the set S . The list L_A is undecided because $C(L_A) \neq C(L_A, L_{B \setminus A})$, and hence is not separable from some $(x) \in Y$. In particular, $C(x, L_{B \setminus A}) = C(L_A, L_{B \setminus A}) = C(L_B)$, and thus because $C(L_B) \notin B \setminus A$ we obtain that $x = C(L_B)$. Thus, L_A is not separable from $C(L_B)$, and in particular, $C(L_A, l) = C(C(L_B), l)$. By fact 2 and order-independence, $C(L_A, l) = C(L_A)$ and $C(C(L_B), l) = C(L_B)$ and thus $C(L_A) = C(L_B)$ in contradiction to $C(L_A) \neq C(L_B)$. ■

The second consistency property I explore is robustness to the repetition of elements in a list.

Repetition-independence. A choice function C is *repetition-independent* if $C(L) = C(L, x) = C(x, L)$ for every list L and every element x in L . If this condition holds only for two-element lists, then C is *repetition-independent for pairs*.

The following observation is an immediate implication of observation 3 and theorem 2.

Observation 4 *Any choice function C that is repetition-independent for pairs is at least as complicated as rational choice. If C is repetition-independent and non-rationalizable, then it is strictly more complicated than rational choice.*

Indeed, assume C is repetition-independent for pairs. Then, for every two elements a and b , $C(a, b) = C(a, b, a) = C(b, a)$. Thus, C is order-independent for pairs, and the first part of observation 4 follows. Similarly, if C is repetition-independent then it must be order-independent. Otherwise, there exist lists L_1 and L_2 that are permutations of one another such that $C(L_1) \neq C(L_2)$. By repetition independence, however, $C(L_1) = C(L_1, L_2) = C(L_2)$, which is a contradiction.

Thus, observations 3 and 4 imply:

Observation 5 *Any choice function C that is less complicated than rational choice must be order-dependent and repetition-dependent for pairs.*

In rational choice, there is a *maximal* element that is chosen from every list it appears in, and a *minimal* element that is never chosen if other elements are available. Consider a choice function C , not necessarily rationalizable, that satisfies this property. Then, C is at least as complicated as rational choice. Indeed, the collection of $N - 1$ one-element lists excluding the list that consists of the maximal element is a collection of undecided lists (the maximal element is chosen if it appears later) that are pairwise separable (the minimal element “separates” them). Hence, the complexity of C is at least $N - 1$. The following property characterizes all the choice functions with a minimal element and a maximal elements that have the same complexity as rational choice.

Markovian choice. A choice function C is *markovian* if $C(L, a) = C(C(L), a)$ for every list L and for every element a .

That is, a choice function is markovian if for every list L , the element $C(L)$ is a sufficient statistic for making a choice for any possible future information. Rational choice and maximizing with a bias are clearly markovian.

Observation 6 *Let C be a choice function with a minimal element and a maximal element. The procedural complexity of C is exactly $N - 1$ if and only if C is markovian.*

Proof. Let w be the C -maximal element and l the C -minimal element. Assume the complexity of C is $N - 1$ and let $Y = \{(x) \mid x \in X \setminus \{w\}\}$. The set Y is a collection of undecided and pairwise separable lists. Because complexity is $N - 1$, any undecided list L is not separable from some $(z) \in Y$. Because $C(L) = C(L, l) = C(z, l) = z$, L is not separable from $C(L)$. In particular, $C(C(L), a) = C(L, a)$ for every element a .

In the other direction, assume C is markovian. Construct an automaton with $N - 1$ states that implements C as follows. Let $Q = X \setminus \{w\}$, $q_0 = l$, $g(q, x) = f(q, x) = C(q, x)$ for every $x \neq w$, $g(q, w) = \text{Stop}$ and $f(q, w) = w$. It is straight forward to see that this automaton implements C , and thus the complexity of C is at most $N - 1$. ■

5 Choice design

In rational choice, information processing depends on the identity of the best element seen so far, so the complexity of rational choice nearly equals the cardinality of the outcome space. Consequently, when the outcome space is large, rational choice is complicated. This motivates designing choice procedures that are “close” to utility maximization but are easier to implement, which is the topic of this section.

The first step is to define a measure of “closeness” between utility maximization and a given choice function C . I focus on a measure that is independent of the details of the lists-generating process. Let u be a utility function and M an automaton that implements C . Given a list L , let $u(L)$ be the utility of the u -maximal element in L , and let $u(M(L)) \equiv u(M(q_0, L))$ be the utility of the element that M chooses from L . I define the regret of M with respect to u by

$$\text{regret}_u(M) = \max_{L \in \mathcal{L}} \{u(L) - u(M(L))\},$$

where \mathcal{L} is the set of all non-empty lists. That is, $\text{regret}_u(M)$ is the maximal utility loss associated with using M to make choices rather than maximizing u .

Using $\text{regret}_u(M)$ as a measure of closeness between u and M , the objective of the following choice design problem is to minimize regret subject to the cognitive limitation of having only K

available states:

$$(\text{MinReg}(K)) : \min_{M : |M|=K} \text{regret}_u(M)$$

where $|M|$ denotes the number of states in M . Clearly, $\text{MinReg}(K)$ has a solution because the number of automata with K states is finite.

I now characterize the automata that solve $\text{MinReg}(K)$ and the choice procedures they implement. Let x_{\max} be the u -maximal element in X , and let $u_{\max} = u(x_{\max})$. Define x_{\min} and u_{\min} similarly.

Claim 1 *The unique automaton with one state that solves $\text{MinReg}(1)$ implements a satisficing procedure with aspiration threshold $t_0 = \frac{u_{\min} + u_{\max}}{2}$.*⁸

Proof. Let y be the u -smallest element such that $u_y \geq t_0$ and z the u -largest element such that $u_z \leq t_0$. Then the regret associated with a satisficing procedure with threshold t_0 is $V_1 = \max\{u_{\max} - u_y, u_z - u_{\min}\}$.

Let M be an automaton with one state that solves $\text{MinReg}(1)$. Then, $f(q_0, x) = x$ for any element x . If there exists an element x such that $u_x > t_0$ but $g(q_0, x) = q_0$ then the regret associated with M is at least $u_x - u_{\min}$ (e.g. for the list (x, x_{\min})), which is larger than V_1 . Indeed, $u_x > u_z$ and thus $u_x - u_{\min} > u_z - u_{\min}$. Moreover, $u_x \geq u_y$ and thus $u_x - u_{\min} \geq u_y - u_{\min} \geq u_{\max} - u_y$, where at least one of the two inequalities is strict because either $u_x > u_y$ or $u_y > t_0$. Similarly, if $u_x < t_0$ but $g(q_0, x) = \text{Stop}$, then $\text{regret}_u(M)$ is at least $u_{\max} - u_x$ (consider the list (x, x_{\max})) that is larger than V_1 . Thus, the above satisficing procedure is the unique solution to $\text{MinReg}(1)$. ■

For $K > 1$, more than one automaton solves $\text{MinReg}(K)$. I first identify the value of the solution V_K in Claim 2. I then discuss problems associated with some of the automata that achieve V_K , and present corresponding refinements. Theorem 3 identifies the unique automaton that solves $\text{MinReg}(K)$ subject to these refinements.

The value of the solution to $\text{MinReg}(K)$ is determined by the elements $\{a_i\}_{i=1}^{K-1}$, defined recursively as follows. The element a_1 is the “closest” element in terms of utility to the threshold t_0 :

$$a_1 = \arg \min_{x \in X} \{|t_0 - u(x)|\}.$$

If there is more than one such element, a_1 is the one with the highest utility.

For $i \geq 2$, a_i is the u -closest element to t_0 among the elements in the set $X \setminus \{a_1, \dots, a_{i-1}\}$:

$$a_i = \arg \min_{x \in X \setminus \{a_1, \dots, a_{i-1}\}} \{|t_0 - u(x)|\}.$$

If there is more than one such element, a_i is the one with the highest utility.

Relabel the elements a_1, \dots, a_{K-1} according to utility so that $u(a_1) > \dots > u(a_{K-1})$, and define $u_i = u(a_i)$. Clearly, the relabeled elements are adjacent in terms of utility, i.e., if there is $x \in X$

⁸If there exists an element x such that $u(x) = t_0$, then there is an additional automaton that solves $\text{MinReg}(1)$, which outputs x rather than ignoring it.

such that $u_i > u(x) > u_{i+2}$ then $x = a_{i+1}$. Let y be the u -smallest element such that $u_y > u_1$ and z the u -largest element such that $u_z < u_{k-1}$. Claim 2 identifies the value of the solution to $\text{MinReg}(K)$, and is proved in the appendix.

Claim 2 *The value of the solution to $\text{MinReg}(K)$ is $V_K = \max\{u_{\max} - u_y, u_z - u_{\min}\}$.*

Let M be an automaton that solves $\text{MinReg}(K)$ and let $q_i = g(q_0, a_i)$ be the state of M after reading a_i in the initial state. Claim 4 in the appendix provides a partial characterization of M . Specifically, claim 4 shows that

(i) M must “remember” a_i : q_i is neither the *Stop* state nor the initial state, if $g(q, x) = q_i$ then either $q = q_i$ or $x = a_i$, and $f(q_i, x) \in \{a_i, x\}$; and

(ii) M “satisfices” with respect to the threshold t_0 in the initial state: for any element $x \neq a_i$, $g(q_0, x) = \text{Stop}$ if $u(x) > t_0$ and $g(q_0, x) = q_0$ otherwise.

Consider, for example, the problem $\text{MinReg}(2)$ solved over $X = \{x_0, x_1, x_2, x_3\}$, with $u(x_0) = 0$, $u(x_1) = 1.5$, $u(x_2) = 2$, and $u(x_3) = 3$. Then, the u -closest element to $t_0 = 1.5$ is x_1 , implying that $a_1 = x_1$ and that the value of the solution is $V_2 = 1$. Figure 5 summarizes the properties any automaton that solves $\text{MinReg}(2)$ must satisfy. For example, when reading the element x_3 in state q_1 , the automaton must move to the *Stop* state, or else the automaton chooses x_0 or x_1 from the list (x_1, x_3, x_0) implying a regret of at least $3 - 1 = 2 > V_2$.

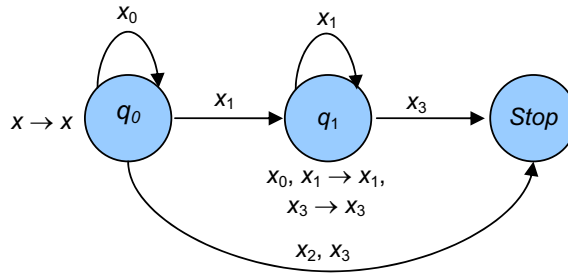


Figure 5: $\text{MinReg}(2)$

Note that the figure does not specify the element $f(q_1, x_2)$ outputted when reading the element x_2 in state q_1 , because $f(q_1, x_2)$ varies across automata that solve $\text{MinReg}(2)$. In particular, $f(q_1, x_2)$ may equal x_1 even though $u(x_2) > u(x_1)$. Changing $f(q_1, x_2)$ to x_2 strictly improves the performance of the automaton for some lists while not affecting performance for the remaining lists. This is clearly a desirable property:

u-Efficiency. An automaton M satisfies u -Efficiency if there exists no other automaton M' , $|M'| = |M|$, such that $u(M'(L)) \geq u(M(L))$ for every list L , and $u(M'(L')) > u(M(L'))$ for at least one list L' .

In addition, Figure 5 does not specify the state $g(q_1, x_2)$ because $g(q_1, x_2)$ varies across automata that solve $\text{MinReg}(2)$. Specifically, $g(q_1, x_2)$ may equal *Stop*. In this case, conditional on seeing

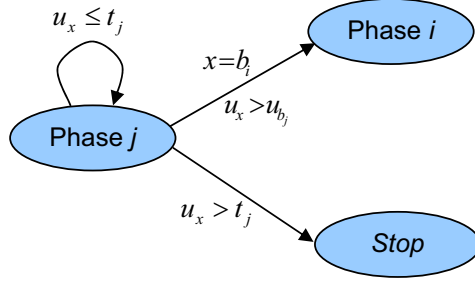


Figure 6: History-dependent satisficing

the one-element list (x_1) , the regret associated with all possible continuations of (x_1) , denoted by $\text{regret}_u(M \mid (x_1))$, is at least 1. If, however, $g(q_1, x_2) = x_1$ then $\text{regret}_u(M \mid (x_1))$ is at most 0.5. Minimizing this “conditional” regret is clearly a desirable property. Formally, let $\text{regret}_u(M \mid L) = \max_{L' \in \mathcal{L}} \{u(L, L') - u(M(L, L'))\}$.

regret-Efficiency. An automaton M satisfies *regret-Efficiency* if there exists no other automaton M' , $|M'| = |M|$, such that $\text{regret}_u(M' \mid L) \leq \text{regret}_u(M \mid L)$ for every list L that is M - and M' -undecided, with at least one strict inequality.

I now describe the unique automaton that solves $\text{MinReg}(K)$ and satisfies u - and *regret-Efficiency*.

History-dependent satisficing. A *history-dependent satisficing* procedure with K phases is characterized by K aspiration elements b_0, \dots, b_{K-1} and K corresponding thresholds t_0, \dots, t_{K-1} . For every list, the procedure starts in phase 0. In phase j , $1 \leq j \leq K - 1$, the next element x in the list is processed as follows (see Figure 6 for an illustration):

- (i) If x is the last element in the list, choose the u -maximal element among b_j and x . Otherwise,
- (ii) if $x = b_i$ and $u_x > u_{b_j}$, switch to phase i . Otherwise,
- (iii) satisfice: if $u_x > t_j$ choose x and stop, and if $u_x \leq t_j$ ignore x and stay in phase j .

Thus, in a history-dependent satisficing procedure with K phases, there are K elements according to which the decision maker updates his aspiration threshold. The decision maker satisfices with respect to the current aspiration threshold. The following result is proved in the appendix.

Theorem 3 *There exists an automaton that solves $\text{MinReg}(K)$ subject to u - and regret-Efficiency, and it is generically unique. This automaton implements a history dependent satisficing procedure with K phases characterized by the elements $b_0 = x_{\min}, b_1 = a_1, \dots, b_{K-1} = a_{K-1}$ and the thresholds $t_i = \frac{u_i + u_{\max}}{2}$.⁹*

⁹There are two non-generic cases. First, if there is an element x such that $u(x) = t_i$ for some i , a solution can output x in phase i rather than ignoring it. Second, if there are two solutions to the problem $\arg \min_{x \in X \setminus \{a_1, \dots, a_{K-2}\}} \{|t_0 - u(x)|\}$, then a_{K-1} can be either of them.

Note that for $K = 1$, the history-dependent satisficing procedure described in Theorem 3 is the satisficing procedure of Claim 1.

5.1 Properties of the solution

The unique automaton M that solves $\text{MinReg}(K)$ subject to u - and *regret*-Efficiency exhibits several procedural effects. First, because aspiration thresholds increase along the list and because M chooses the first element above the current threshold, there is a *primacy* effect. That is, moving any element (except maybe the last element in the list) toward the beginning of the list improves the likelihood that this element is chosen.

There is also a mild *recency* effect: moving an element x that is not chosen to the last position in a list sometimes turns it to the chosen element. This is clearly true if the list does not contain any aspiration elements. This is also true for some lists that contain aspiration elements. Indeed, consider a list L and let a_i be the u -maximal aspiration element in L . Assume that (i) M does not move to the *Stop* state before L ends, (ii) L contains an element x with utility $u_i < u_x < t_i$, and (iii) the last element of L has utility $< u_i$. Then, because M does not move to the *Stop* state before the list ends, M chooses the u -maximal element among a_i and the last element in the list. Therefore, the element a_i is chosen from L , and x is chosen from the list L' in which x appears last and the remaining elements are ordered as in L .

When the decision maker is endowed with a default alternative δ with utility u_δ , he also exhibits a *default tendency*: any element in some utility range above u_δ is ignored by the decision maker unless it is the last element in the list. To see this, let \bar{X} be the collection of elements in X that are u -superior to δ , and let $K < |\bar{X}|$. Solving $\text{MinReg}(K)$ over the space $X \cup \{\delta\}$ yields a history-dependent satisficing procedure in which $a_0 = \delta$ and $t_0 = \frac{u_\delta + u_{\max}}{2}$. In particular, the decision maker ignores all the elements in the utility range $[u_\delta, t_0]$, which are not aspiration elements, except maybe the last among them.

Finally, M also generates predictions regarding the time it takes to make a choice. Denote by $t_M(L) = t(L)$ the number of elements M reads from the list L before moving to the *Stop* state. If processing each element takes one unit of time, then $t_M(L)$ measures the time until M makes a choice from L .

The identity of the aspiration elements that appear in a list L significantly affects $t_M(L)$. To see this, let the *impression* of an undecided list L be the utility value of the u -maximal element among those in $\{a_1, \dots, a_{K-1}\}$ that appear in L , or u_{\min} if L does not contain any element a_i . Then, the higher the impression of an undecided list L the larger the incremental time for making a choice, $t(L, L') - t(L)$, for any possible continuation L' of L . This is because a higher impression implies a higher threshold. The converse is also true: if two lists L_1 and L_2 are undecided and there is a list L for which $t(L_1, L) - t(L_1) > t(L_2, L) - t(L_2)$, then the impression of L_1 is higher than that of L_2 . Thus, the incremental time for making a choice after L_1 is *always* weakly larger than after L_2 .

5.2 Costly states

In $\text{MinReg}(K)$, the number of states is determined exogenously. It is straightforward to extend the analysis to a situation in which an explicit cost c is associated with each state:

$$(\text{MinReg}) \quad \min_M \{ \text{regret}_u(M) + c|M| \} \quad \text{s.t. } u\text{-Efficiency and } \text{regret-Efficiency}$$

In this case, the utility loss associated with an automaton M comes from two sources: the procedural cost corresponding to the number of states in M , and the cost associated with regret. Additional states impose a procedural cost but may reduce regret.

Clearly, only automata with $\leq N - 1$ states solve MinReg . Indeed, $\text{regret}_u()$ is bounded below by zero and it is possible to reduce regret to zero by an automaton with $N - 1$ states that implements utility maximization. Any additional states increase procedural costs but cannot reduce regret further. Moreover, by Theorem 3, for any fixed $1 \leq K \leq N - 1$, the unique automaton that solves $\text{MinReg}(K)$ subject to u - and regret -Efficiency is a K -phase history-dependent satisficing procedure. Thus, there are $N - 1$ candidate automata to solve MinReg . By Claim 2 the value of the solution to MinReg is $\min_{1 \leq K \leq N-1} \{V_K + cK\}$. Generically, there is a unique K that solves this problem, and hence a unique history-dependent satisficing procedure that solves MinReg .

Example. Evenly-spread utilities. Let $X = \{1, 2, \dots, N\}$ and assume $u(x) = x$. For simplicity, assume that N is an even number. Then, the unique solution to MinReg (even without the refinements) is full utility maximization if $c < 1/2$, and satisficing with the threshold $t_0 = \frac{u_{\min} + u_{\max}}{2}$ if $c > 1/2$. To see this, note that the regret associated with a one-state or a two-state optimal automaton is $V_1 = V_2 = N/2 - 1$, and that similarly $V_{2m-1} = V_{2m} = N/2 - m$. Therefore, if it is profitable to enlarge the number of states from one to three, it is also profitable to continue doing so until the automaton has $N - 1$ states. This argument applies to any cost structure in which the cost per-state is weakly decreasing.

Endogenizing the number of states generates additional predictions regarding the time it takes to make choices.

Improved costs. Assume two decision makers share the same utility function u but differ on procedural costs. Let c_i be the cost per-state of decision maker i .

Claim 3 *Let $c_1 \geq c_2$ and let M_i be the unique automaton that solves MinReg when the cost per-state is c_i . Then, $t_{M_1}(L) \leq t_{M_2}(L)$ for every list L .*

Proof. Denote by b_i the marginal benefit from moving from an optimal automaton with $i - 1$ states (that corresponds to the elements a_0, \dots, a_{i-1}) to an optimal automaton with i states. Then, the unique solution to MinReg is an automaton of size k if and only if $f(m) = V_1 - \sum_{j=2}^m b_j + mc_i$ is minimized at k . If costs reduce from c_1 to c_2 , then the only change in $f(m)$ is that costs decrease by $m(c_1 - c_2)$, which is increasing in m . Thus, the optimal automaton under c_2 would have weakly

more states and would nest (in the graph theoretic sense) the optimal automaton under c_1 . Hence, for every beginning L , either M_1 and M_2 reach the same state, or M_1 reaches the Stop state while M_2 reaches a non-stopping state, or M_2 reaches a state with a higher threshold than that of M_1 . In any case, the time until making a choice is larger in M_2 . ■

While the time for making choices increases as procedural costs decrease, the resulting choices are not always u -better. Indeed, reduced costs imply additional aspiration elements and possibly higher thresholds. Higher thresholds may result in ignoring elements that were history-dependent satisfactory before and turned non-satisfactory.

Enlarging the outcome space. Suppose an element w with utility between u_{\min} and u_{\max} is added to the set X . Fix the cost per-state at c . Let M_1 be the automaton that solves MinReg over X and M_2 the automaton that solves MinReg over $X \cup \{w\}$. Then, $t_{M_1}(L) \geq t_{M_2}(L)$ for every list L that does not include w . Intuitively, w makes the outcome space more “dense” and thus the benefit from each additional state weakly decreases. This implies that M_1 either nests M_2 or M_2 has one additional state corresponding to the element w . Thus, processing time in M_2 is shorter than in M_1 except maybe for lists that include w .

6 Concluding remarks

This paper investigated one procedural aspect of decision making. I considered automata implementing choice rules, and measured the procedural complexity of a given choice rule by the minimal number of states required for implementing it. The number of states captures the amount of information processing required for implementation.

In rational choice, information processing depends on the identity of the best element considered so far, so the complexity of rational choice nearly equals the number of feasible alternatives. Hence, any situational cue that makes some of the alternatives “irrelevant” such as a default alternative, simplifies rational choice.

When the outcome space is large, rational choice is complicated, and hence a decision maker may approximate rational choice in order to economize on procedural costs. Theorem 3 establishes that choice rules that emerge as an optimal tradeoff between maximizing utility and minimizing procedural complexity exhibit certain procedural effects. In particular, primacy and recency effects and a default tendency emerge when a “rational” decision maker economizes on procedural costs. Exploring whether additional behavioral phenomena, usually referred to as biases, emerge as natural solutions to decision problems that take procedural costs into account is a task for future research.

7 Appendix

7.1 Proof of Claim 2

Lemma 1 $V_K \leq \min\{u_{K-1} - u_{\min}, u_{\max} - u_1\}$.

Proof. I show that $V_K \leq u_{K-1} - u_{\min}$. Proving that $V_K \leq u_{\max} - u_1$ follows from similar arguments and is left to the reader.

To prove that $V_K \leq u_{K-1} - u_{\min}$, one has to show that (1) $u_z - u_{\min} \leq u_{K-1} - u_{\min}$ and that (2) $u_{\max} - u_y \leq u_{K-1} - u_{\min}$. Because $u_z < u_{K-1}$, part (1) follows. To prove part (2), note that $u_{\max} - u_y \geq u_y - u_{\min}$ because $u_y \geq t_0$. Thus, if $u_{K-1} \geq t_0$, then because $u_y > u_{K-1}$ we obtain that $u_{\max} - u_y < u_{\max} - u_{K-1} \leq u_{K-1} - u_{\min}$ as required. If $u_{K-1} < t_0$, then assume to the contrary that $u_{\max} - u_y > u_{K-1} - u_{\min}$. The last inequality implies that y is u -“closer” than u_{K-1} to the threshold t_0 contradicting the definition of a_{K-1} . Hence, $u_{\max} - u_y \leq u_{K-1} - u_{\min}$ as required. ■

Proof of Claim 2. The history-dependent satisficing procedure of Theorem 3 obtains V_K .

I now show that the regret associated with any automaton M that solves $\text{MinReg}(K)$ is at least V_K . Let M be a solution and C the corresponding choice function. Consider the $K+1$ one-element lists $(a_1), \dots, (a_{K-1}), (y), (z)$. There are several cases to consider:

Case 1. None of these lists is decided. Then, because the informational index is K , two of these lists, (x) and (y) are not separable. Then $C(x, x_{\min}) = C(w, x_{\min}) = x_{\min}$, implying that the regret associated with M is at least $u_{K-1} - u_{\min} \geq V_K$, where the inequality follows from Lemma 1.

Case 2. Two or more of these lists are decided. Then the regret associated with M is at least $u_{\max} - u_1 \geq V_K$ where the inequality follows from Lemma 1.

Case 3. One of these listed is decided. Then, by case 2, it must be the one-element list (y) . Thus, the regret associated with M is at least $u_{\max} - u_y$. The remaining K lists are undecided. Because the informational index is K , either two of them are not separable or one of them is not separable from the empty list. In any case, $C(x, x_{\min}) = x_{\min}$ for at least one of these lists, implying that regret is at least $u_z - u_{\min}$. Thus, regret is at least V_K as required. ■

7.2 Proof of Theorem 3

As shown in Claim 1 in the main text, the unique solution to $\text{MinReg}(1)$ is a history-dependent satisficing procedure with one phase. It is straightforward to verify u - and *regret*-Efficiency.

Consider the problem $\text{MinReg}(K)$ for $K \geq 2$. Assume that

- (1) For every element x and for every i , $u(x) \neq t_i$, and
- (2) there is a unique solution to $\arg \min_{x \in X \setminus \{a_1, \dots, a_{K-2}\}} \{|t_0 - u(x)|\}$.

Footnote 10 discusses the additional solutions when any of these properties is violated.

By (2), Lemma 1 can now be restated as follows.

Lemma 1 $V_K < u_{K-1} - u_{\min}$ and $V_K < u_{\max} - u_1$.

I begin by outlining properties of any automaton that solves $\text{MinReg}(\mathbb{K})$.

Claim 4 Let M be a solution to $\text{MinReg}(\mathbb{K})$, and let $q_i = g(q_0, a_i)$ for $1 \leq i \leq K - 1$. Then,

(1) M remembers a_i for $i \geq 1$: $g(q, a_i) \neq \{q_0, \text{Stop}\}$, if $g(q, x) = q_i$ then either $q = q_i$ or $x = a_i$, $g(q_i, x) \neq q_0$ for $i \geq 1$, and $f(q_i, x) \in \{a_i, x\}$.

(2) M satisfices in the initial state: for any element $x \neq a_i$, $g(q_0, x) = \text{Stop}$ if $u_x \geq u_y$, or $g(q_0, x) = q_0$ otherwise.

Proof. Let L_i be the one-element list (a_i) for $i \geq 1$, and L_0 be the empty list. Consider the transition $g(q_k, a_i)$ in M for $0 \leq k \leq K - 1$:

(i) If $g(q_k, a_i) = q_0$ then M reaches q_0 after reading the list (L_k, a_i) . Because $f(q_0, x) = x$, M then chooses x_{\min} from the list (L_k, a_i, x_{\min}) . Thus the regret of M is at least $u_i - x_{\min} > V_K$ in contradiction to M being a solution.

(ii) If $g(q_k, a_i) = \text{Stop}$ then M moves to the Stop state after seeing (L_k, a_i) and thus it chooses either a_i or a_k if $k \neq 0$. But then $\text{regret}_u(M) \geq u_{\max} - u_1 > V_K$ because (x_{\max}) is a possible continuation.

Thus, $g(q_k, a_i) \notin \{\text{Stop}, q_0\}$. Similarly to (i), if $g(q_i, x) = q_0$ and $i \geq 1$ then $M(a_i, x, x_{\min}) = x_{\min}$ and M cannot be a solution. Thus, $g(q_i, x) \neq q_0$ for $i \geq 1$. Clearly, $f(q_i, x) \in \{x, a_i\}$ or else M fails to choose from the list $L = (a_i, x)$ an element that appears in L .

Assume $g(q_k, x) = q_i$ but $q_k \neq q_i$ and $x \neq a_i$. Then M reaches state q_i after reading either (L_k, x) or (a_i) . But then, since $a_i \neq x$ and $a_i \neq a_k$ for $k \geq 1$, we must have $M(a_i, x_{\min}) = M(L_k, x, x_{\min}) = x_{\min}$ and M cannot be a solution.

In particular, $g(q_0, x) \in \{\text{Stop}, q_0\}$ for any $x \neq a_i$. To conclude the proof, note that if $u_x \geq u_y$ but $g(q_0, x) = q_0$ then the regret associated with M is at least $u_x - u_{\min} > u_{k-1} - u_{\min} > V_K$. Similarly, $g(q_0, x) = q_0$ when $u_x \leq u_z$. ■

I now identify an automaton that solves $\text{MinReg}(\mathbb{K})$ subject to u - and *regret*-Efficiency.

Claim 5 The history-dependent satisficing procedure of Theorem 3 solves $\text{MinReg}(\mathbb{K})$ subject to u - and *regret*-Efficiency.

Proof. Let M be an automaton implementing the history-dependent satisficing procedure of Theorem 3. Clearly, it solves $\text{MinReg}(\mathbb{K})$. Let q_0 be the initial state of M and denote $q_j = g_M(q_0, a_j)$.

To establish u -Efficiency, assume there exists an automaton M' , $|M'| = |M|$, that is u -superior to M . Then M' solves $\text{MinReg}(\mathbb{K})$ and thus satisfies the conditions of Claim 4. To be u -superior to

M , the automaton M' must have at least one transition (not from the initial state) that is different than M . There are three possible cases:

(1) $g_M(q_k, x) \neq \text{Stop}$ but $g_{M'}(q_k, x) = \text{Stop}$. Then $x \neq x_{\max}$ and M outputs a u -larger element from the list (a_k, x, x_{\max}) .

(2) $g_M(q_k, x) = \text{Stop}$ but $g_{M'}(q_k, x) \neq \text{Stop}$. Then, $x \neq a_i$ and $u(x) > (u_k + u_{\max})/2$. Thus M outputs a u -larger element from (a_k, x, x_{\min}) .

(3) $g_M(q_k, x) = q_i$. Then, by (1) $g_{M'}(q_k, x) \neq \text{Stop}$. Thus, $g_{M'}(q_k, x) = q_j$ (where I abuse notation and denote $q_j = g_{M'}(q_0, a_j)$). Because M remembers the u -best element a_t it sees, it must be that $u(a_i) > u(a_j)$. But then $M(a_k, x, x_{\min}) = a_i$ while $M(a_k, x, x_{\min})$ is either a_j or x_{\min} .

To establish *regret*-Efficiency, let the regret of state q_j in M be defined by:

$$p_j = \max\{u_{\max} - u_{y_j}, u_{z_j} - u_j\}$$

where y_j is the u -minimal element for which $g(q_j, y_j) = \text{Stop}$ and z_j the u -maximal element for which $g(q_j, z_j) = q_j$. Then,

Lemma 2 $p_1 \leq p_2 \leq p_{k-1} \leq p_0$.

Proof. I show that $p_{i-1} \leq p_i$ (a similar proof holds for p_{k-1} and p_0). By the definition of M , $u_{y_i} \leq u_{y_{i-1}}$ and thus $u_{\max} - u_{y_i} \geq u_{\max} - u_{y_{i-1}}$. In addition $u_{z_i} \leq u_{z_{i-1}}$. If $u_{z_i} = u_{z_{i-1}}$, then $u_{z_{i-1}} - u_{i-1} < u_{z_i} - u_i$. If $u_{z_{i-1}} > u_{z_i}$ then $u_{z_{i-1}} \geq u_{y_i}$ (because y_i is just above z_i in terms of utility), and thus $u_{\max} - u_{y_i} \geq u_{\max} - u_{z_{i-1}} > u_{z_{i-1}} - u_{i-1}$, where the right inequality follows from $u(z_{i-1}) < u_{i-1}$. Thus $p_{i-1} \leq p_i$

Assume *regret*-Efficiency is violated and let M' be *regret*-superior to M . Then M' solves MinReg(K) (take L to be the empty list in the definition of *regret*-Efficiency), and thus satisfies the conditions of Claim 4. Let L be a M -undecided list such that $\text{regret}_u(M' | L) < \text{regret}_u(M | L)$.

Assume M reaches state q_i after processing L . Then $\text{regret}_u(M | L) \leq p_i$ because M never moves to state q_j , $j > i$, and because $p_i \geq p_m$ for $m < i$. Since M' solves MinReg(K), it reaches a state q' , which “remembers” some element a_m . Note that $u(a_m) \leq u_i$ because M remembers the u -highest element from L . Because M' must either ignore or output y_m and z_m , $\text{regret}_u(M' | L) \geq p_m$. Since $p_m \geq p_i$, $\text{regret}_u(M' | L) \geq \text{regret}_u(M | L)$, which is a contradiction. ■

I now show uniqueness, and thus conclude the proof of Theorem 3.

Claim 6 *The unique solution to MinReg(K) subject to u - and regret-Efficiency is the history-dependent satisficing procedure of Theorem 3.*

Proof. Let M' be an automaton that solves $\text{MinReg}(K)$ subject to u - and regret -Efficiency. Let M implement the history-dependent satisficing procedure of Theorem 3. Denote $q_i = g_{M'}(q_0, a_i) = g_M(q_0, a_i)$. By u -Efficiency, $f_{M'}(q_i, x)$ must be the u -maximal element among a_i and x , and thus $f_{M'}$ coincides with f_M .

Thus, M' must differ from M in some transition $g(q_i, x)$ where $q_i \neq q_0$. To see that this is impossible, I will show that any different transition implies that M is regret -superior to M' . Note that by Claim 5, $\text{regret}_u(M \mid L) \geq \text{regret}_u(M' \mid L)$ for every list which is M - and M' -undecided, and thus it is enough to show a strict improvement for just one list. Define y_i, z_i and p_i as in Claim 5, and consider the following cases.

Case 1. $x \notin \{a_1, \dots, a_{k-1}\}$, $u_x > \frac{u_{\max} + u_i}{2}$ but $g_{M'}(q_i, x) = q_i$. Then, $\text{regret}_u(M' \mid (a_i)) \geq u_x - u_i$ (e.g. consider the list (a_i, x, x_{\min})). But $u_x - u_i > u_{\max} - u_x \geq u_{\max} - u_{y_i}$ by the definition of x and y_i , and $u_x - u_i > u_{z_i} - u_i$ by the definition of z_i . Thus, $u_x - u_i > \max\{u_{\max} - u_{y_i}, u_{z_i} - u_i\} = p_i = \text{regret}_u(M \mid (a_i))$. Thus, $\text{regret}_u(M' \mid (a_i)) > \text{regret}_u(M \mid (a_i))$, contradicting regret -Efficiency.

Case 2. $x \notin \{a_1, \dots, a_{k-1}\}$, $u_x < \frac{u_{\max} + u_i}{2}$ but $g_{M'}(q_i, x) = \text{Stop}$. Then, similarly to case 1,

$$\text{regret}_u(M' \mid (a_i)) \geq u_{\max} - u_x > \max\{u_{\max} - u_{y_i}, u_{z_i} - u_i\} = p_i = \text{regret}_u(M \mid (a_i)).$$

By cases 1 and 2, the threshold in M' for outputting an element x in state q_i is t_i , just like in M . For the remaining cases assume $x = a_j$ for $j \geq 1$.

Case 3. $g_{M'}(q_i, a_j) \neq g_M(q_i, a_j)$ and $p_i \neq p_j$. Again, this implies regret -Efficiency is violated. For example, if $g_{M'}(q_i, a_j) = q_j$ then $u_j < u_i$, and thus $\text{regret}_u(M' \mid (a_i, a_j)) \geq p_j > p_i = \text{regret}_u(M \mid (a_i, a_j))$.

By case 3, M' differs from M only in transitions of the form $g_{M'}(q_i, a_j)$ for which $p_i = p_j$, and at least one such transition exists. This implies that M' violates u -Efficiency.

Indeed, if $g_{M'}(q_i, a_j) \neq g_M(q_i, a_j)$ then M outputs a u -larger element from the list (a_i, a_j, x_{\min}) . In addition, consider any other list L . When reading L , the transitions of M' and M are identical until the first time in which an element a_t appears such that $q_l = g_{M'}(q_r, a_t) \neq g_M(q_r, a_t) = q_h$ where $l, h \in \{r, t\}$ and $u_l < u_h$. The two automata then move to different states in which they make the same decisions for elements $x \in X \setminus \{a_1, \dots, a_{K-1}\}$. For an element a_m , there are several possibilities:

(1) $g_{M'}(q_l, a_m) \neq g_M(q_l, a_m)$ where q_l is the current state of M' . Then $p_m = p_l = p_z$ and the two automata continue to make the same decisions regarding elements in $x \in X \setminus \{a_1, \dots, a_{K-1}\}$.

(2) $g_{M'}(q_l, a_m) = g_M(q_l, a_m)$. Then there are two possibilities:

(2.1) $u_m \geq u_h$. Then M and M' move to the same next state q_m .

(2.2) $u_m < u_h$. Then M' moves to state q_s ($s \in \{l, m\}$) while M stays in state q_h . Since this transition in M' is identical to the corresponding transition in M it must be that $p_s \leq p_l = p_h$. But since $u_s < u_h$ (or else M would move to state q_s as well) it must also be that $p_s \geq \bar{p}$. Thus

$p_s = p_h$, which means M and M' continue to make the same decisions (except maybe when the list ends in which case M outputs a weakly u -higher element).

Thus, M is u -superior to M' as required. ■

7.3 Procedural complexity and learnability

The procedural complexity of a choice function corresponds to the difficulty of learning the function. To establish this point, I use the model of Probably Approximately Correct (PAC) Learning. The PAC-learning model is studied in depth in Kearns and Vazirani [11] and Vidyasagar [22]. Kalai [10] and Salant [19] contain applications to economics.

Consider an econometrician Bob who wishes to learn the choice function of a consumer. Bob knows how sophisticated the consumer is, and observes examples of how the consumer chooses, where an example is a list and the chosen element from the list. Examples are drawn randomly and independently according to some fixed probability measure over the collection of all lists. How many examples are needed in order to formulate a hypothesis that will predict most of the choices of the consumer with high probability (with respect to the same measure that draws the examples)?

Formally, let C_K be the family of all choice functions with complexity K . Bob wishes to learn a choice function $c \in C_K$. Let \mathcal{P} be a probability measure over the collection of all lists. The measure \mathcal{P} provides a natural measure of error between any function $h \in C_K$ and c defined by $error_c(h) = Pr[L : c(L) \neq h(L)]$. Let $0 < \epsilon, \delta < 1/2$.

PAC-Learning. The family C_K is PAC-learnable from t examples with confidence $1 - \delta$ and accuracy $1 - \epsilon$ with respect to \mathcal{P} if:

For every $c \in C_K$, if L_1, \dots, L_t are drawn at random and independently according to \mathcal{P} , then with probability at least $1 - \delta$:

$$\text{if } h \in C_K \text{ satisfies } h(L_i) = c(L_i) \text{ for } i = 1, \dots, t, \text{ then } error_c(h) \leq \epsilon.$$

If this holds for every measure \mathcal{P} , then we say that C_K is PAC-learnable from t examples with confidence $1 - \delta$ and accuracy $1 - \epsilon$.

Figure 7 provides graphical intuition. On the right side, the large oval represents a family of functions C_K . A particular function $c \in C_K$ is represented by a point. The probability measure \mathcal{P} induces a distance function (or an error measure) on C_K . The small grey oval includes all the functions whose distance from c is $\leq \epsilon$. The probability measure \mathcal{P} also induces a probability measure over samples of t examples. On the left side, these samples are classified into “good” and “bad” samples. A sample is good if any $h \in C_K$ that agrees with the sample lies in the grey oval around c . The family C_K is PAC-learnable from t examples if for every probability measure \mathcal{P} and every $c \in C_K$, the proportion (with respect to \mathcal{P}) of good samples is at least $1 - \delta$.

Thus, if the family C_K is PAC-learnable from t examples, then with high probability, after seeing a random sample of t examples of some function $c \in C_K$, any function $h \in C_K$ that “agrees”

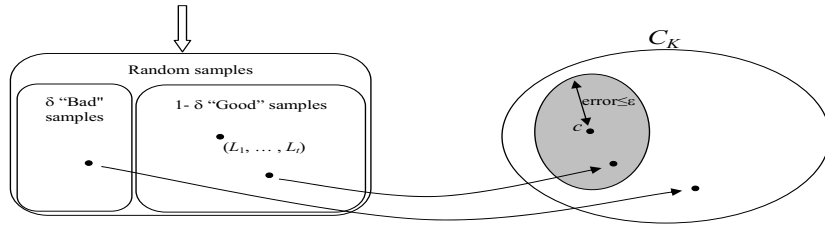


Figure 7: PAC-learning

with the examples will predict a large proportion of the values of c ; Hence the name Probably Approximately Correct Learning.

Note that learning in the PAC model is susceptible to two kinds of failure. The confidence parameter δ is necessary since a random sample may be “unrepresentative” of the underlying function one wants to learn. For example, the sample might include repeated draws of the same example despite the fact that \mathcal{P} is a uniform measure. The accuracy parameter ϵ is necessary since a small random sample may not distinguish between two functions that differ on only a few examples that have high enough probability.

The number of examples needed to PAC-learn a class of functions is asymptotically determined by the P -dimension.

P -dimension. The P -dimension of a class of functions C_K is the maximal number of lists L_1, \dots, L_t and corresponding elements x_1, \dots, x_t such that for every subset $A \subseteq \{1, \dots, t\}$ there is a function $f_A \in C_K$ that attains A , that is, $f_A(L_i) = x_i$ for $i \in A$ and $f_A(L_i) \neq x_i$ for $i \notin A$.

The following is a fundamental result from statistical learning theory (see Vidyasagar [22]).

Theorem 4 For fixed values of ϵ and δ , the number of examples t needed to PAC learn a class of functions with confidence $1 - \delta$ and accuracy $1 - \epsilon$ is bounded below and above by linear functions of the P -dimension.

I now show that the amount of information needed to PAC-learn the family of functions with procedural complexity K is intimately related to K and the size of the outcome space X .

Theorem 5 The number of examples needed to PAC learn the family of choice functions with complexity K is bounded below and above by linear functions of NK and $NK \log(NK)$ respectively, where $N = |X|$.

Proof. To prove this result, it suffices to show that the P -dimension of the class of choice functions with informational index K is bounded below and above by linear functions of NK and $NK \log NK$.

I first provide an upper bound on the P -dimension. By definition the P -dimension is bounded above by $\log |C_K|$ since at least 2^t distinct functions are needed in order to attain all the subsets of t examples, where an example is a pair (L, x) such that L is a list and $x \in X$.

The number of functions with informational index K is bounded above by the number of automata with K states. The number of automata with K states is bounded above by $(K+1)^{NK} N^{NK}$ which is the number of possible transition and output functions that use a hardware of K states. Taking log we get that the P -dimension of the class of functions with informational index K is bounded above by $NK \log N(K+1)$.

Showing that t is a lower bound on the P -dimension requires finding a collection of lists L_1, \dots, L_t and elements x_1, \dots, x_t , such that for every set $A \subseteq \{1, \dots, t\}$ there is a function $f_A \in C_K$ that attains A .

Let a be an element of X and denote the remaining elements of X by $\{b_1, \dots, b_{N-1}\}$. Consider the following collection of $K(N-1)$ examples denoted by $ji = (L_{ji}, b_i)$, where $0 \leq j \leq K-1$ and $1 \leq i \leq N-1$:

- (1) For $j = 0$, $L_{ji} = (b_i, a)$, and
- (2) for $1 \leq j \leq K-1$, $L_{ji} = (\underbrace{a, \dots, a}_{j \text{ times}}, b_i)$.

Thus, the list L_{ji} contains j appearances of the element a followed by b_i and possibly a again.

Fix a subset of examples $A \subseteq \{01, 02, 0(N-1), 11, \dots, (K-1)(N-1)\}$. I construct a function in C_K that attains A by describing the corresponding automaton.

Denote the states of the automaton by q_0, \dots, q_{K-1} . Consider the examples $\{0i\}_{i=0}^{N-1}$. If $0i \in A$ define $g(q_0, b_i) = \text{Stop}$, otherwise $g(q_0, b_i) = q_0$. In addition, define $g(q_0, a) = q_1$ (if $K = 1$, define $g(q_0, a) = \text{Stop}$) and $f(q_0, x) = x$. Thus, the chosen element from (b_i, a) is b_i if and only if $0i \in A$.

Consider the j 'th group of examples and the corresponding state q_j , which captures the fact that the element a appeared j times so far. Define $g(q_j, b_i) = \text{Stop}$ and $f(q_j, b_i) = b_i$ if $ji \in A$ or $f(q_j, b_i) = a$ if $ji \notin A$. In addition, define $g(q_j, a) = q_{j+1}$ (or $g(q_j, a) = \text{Stop}$ if $j = K-1$) and $f(q_j, a) = a$. Thus, the chosen element from $(\underbrace{a, \dots, a}_{j \text{ times}}, b_i)$ is b_i if and only if $ji \in A$. Continuing in this fashion, the automaton attains A . Because a similar construction applies for any $A \subseteq \{01, 02, 0(N-1), 11, \dots, (K-1)(N-1)\}$, the P -dimension is at least $K(N-1)$ as required. ■

References

- [1] Dilip Abreu and Ariel Rubinstein, *The structure of nash equilibrium in repeated games with finite automata*, *Econometrica* **56** (1988), no. 6, 1259–1281.

- [2] Taradas Bandyopadhyay, *Revealed preference theory, ordering and the axiom of sequential path independence*, Review of Economic Studies **55** (1988), no. 2, 343–351.
- [3] Donald E. Campbell, *Realization of choice functions*, Econometrica **46** (1972), no. 1, 171–180.
- [4] James Dow, *Search decisions with limited memory*, The Review of Economic Studies **58** (1991), no. 1, 1–14.
- [5] Kfir Eliaz, *Nash equilibrium when players account for the complexity of their forecasts*, Games and Economic Behavior **44** (2003), no. 2, 286–310.
- [6] John E. Hopcroft and Jeffrey D. Ullman, *Introduction to automata theory: Languages and computation*, Addison Wesley, Cambridge, Massachusetts, 1979.
- [7] Daniel Kahneman and Amos Tversky (eds.), *Choices, values and frames*, Cambridge University Press, New York, 2000.
- [8] Daniel Kahneman, Jack L. Knetsch, and Richard H. Thaler, *Anomalies: The endowment effect, loss aversion, and status quo bias*, Journal of Economic Perspectives **5** (1991), no. 1, 193–206.
- [9] Ehud Kalai and William Stanford, *Finite rationality and interpersonal complexity in repeated games*, Econometrica **56** (1988), no. 2, 397–410.
- [10] Gil Kalai, *Learnability and rationality of choice*, Journal of Economic Theory **113** (2003), no. 1, 104–117.
- [11] Michael J. Kearns and Umesh V. Vazirani, *An introduction to computational learning theory*, The MIT Press, Cambridge, Massachusetts, 1994.
- [12] Barton L. Lipman and Sanjay Srivastava, *Informational requirements and strategic complexity in repeated games*, Games and Economic Behavior **2** (1990), no. 3, 273–290.
- [13] Abraham Neyman, *Bounded complexity justifies cooperation in the finitely repeated prisoner’s dilemma*, Economic Letters **19** (1985), 227–229.
- [14] Michael Rabin, *Psychology and economics*, Journal of Economic Literature **36** (1998), 11–46.
- [15] Ariel Rubinstein, *Finite automata play the repeated prisoners’ dilemma*, Journal of Economic Theory **39** (1986), 83–96.
- [16] ———, *Why are certain properties of binary relations relatively more common in natural language*, Econometrica **64** (1996), 343–356.
- [17] ———, *Modeling bounded rationality*, Zeuthen Lecture Book Series, The MIT Press, Cambridge, Massachusetts, 1998.
- [18] Ariel Rubinstein and Yuval Salant, *A model of choice from lists*, Theoretical Economics **1** (2006), no. 1, 3–17.

- [19] Yuval Salant, *On the learnability of majority rule*, Journal of Economic Theory **135** (2007), no. 1, 196–213.
- [20] William Samuelson and Richard Zeckhauser, *Status quo bias in decision making*, Journal of Risk and Uncertainty **1** (1988), 7–59.
- [21] Herbert A. Simon, *A behavioral model of rational choice*, Quarterly Journal of Economics **69** (1955), 99–118.
- [22] Mathukumalli Vidyasagar, *A theory of learning and generalization*, Communications and control engineering series, Springer, London, 1997.
- [23] Andrea Wilson, *Bounded memory and biases in information processing*, NAJ Economics **5:3** (2002).