DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU

HARVARD LIBRARY
Office for Scholarly Communication

Evidence for Recursive Operations in Human Cognition

Citation

Bursley, James K. 2020. Evidence for Recursive Operations in Human Cognition. Doctoral dissertation, Harvard University, Graduate School of Arts & Sciences.

Permanent link

https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37365938

Terms of Use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story .

Accessibility

Evidence for Recursive Operations in Human Cognition

A dissertation presented
by
James K. Bursley
to
The Department of Psychology

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Psychology

Harvard University
Cambridge, Massachusetts
April 2020

©2020 – James K. Bursley
This work is licensed under the Creative Commons Attribution 4.0 International License.

This thesis was typeset using LaTeX and a template for doctoral theses created by Jordan Suchow, released under the MIT (X11) license.

Dissertation Advisor: Joshua D. Greene James K. Bursley

Evidence for Recursive Operations in Human Cognition

Abstract

Recursion has held a prominent place in the cognitive sciences since Noam Chomsky imported the notion from mathematical logic in the 1950s. Despite receiving widespread attention during the ensuing decades, the status of recursive operations—that is, operations that effect a self-call—in cognition remains obscure, in part owing to confusion about the relationship between recursive operations and recursive structures. Though often taken to play a large and far-reaching role in human cognition, such ubiquity cannot simply be assumed and must instead be argued for or demonstrated empirically. Most argumentation and demonstration to this end has followed a common line, which will be described and evaluated in this thesis. Following this, I will propose a novel signature of recursive operations and use behavioral paradigms to detect its presence: First, changes in working memory load during performance of a puzzle task will be shown to follow a broad pattern that would be expected if recursive operations underlie task performance. Then, a minimally-complex recursive computational model will be demonstrated to predict changes in human memory on a fine-grained, trial-by-trial basis using the same task. Finally, a non-recursive model featuring a push-down stack architecture will be shown to be inferior to the recursive model at predicting human memory dynamics. These results signify the first evidence for the existence of recursive operations in human cognition.

Contents

0 Introduction 1
- 0.1 Origins in the Formal Sciences . . . 1
- 0.2 Signatures of Recursive Operations . . . 7

1 Pyramidal Working Memory Hypothesis 11
- 1.1 Study 1 . . . 12
- 1.2 Study 2 . . . 19
- 1.3 Discussion . . . 21

2 Recursive Solver Model 22
- 2.1 Study 3 . . . 24
- 2.2 Study 4 . . . 31
- 2.3 Discussion . . . 32

3 Iterative Solver Model 33
- 3.1 Study 5 . . . 34
- 3.2 Study 6 . . . 36
- 3.3 Discussion . . . 37

4 Conclusion 38

Appendix A Pre-Move Piece Order 45
- A.1 Study 7 . . . 46
- A.2 Study 8 . . . 48
- A.3 Discussion . . . 49

Appendix B Supplementary Figures 50

References 59

This thesis is dedicated to my parents, Kathleen and Barnett.

Acknowledgments

THANK YOU to Josh Greene, who took a risk on a new project and a new student, providing hugely valuable guidance, wisdom, and energy throughout, and always advising with empathy and humanity.

Thank you to my committee members, Susan Carey, Steve Pinker, and Irene Pepperberg, for truly boundless encouragement, enthusiasm, and insight throughout the evolution of this work, from conception to realization.

Thank you to Randy Buckner for helping me to grow and mature as a scientist and thinker, and for providing support, perspective, and advice as I made the transition from one field to another.

Thank you to David Creswell and Mike Tarr, who helped spark and foster some of the earliest parts of my journey into psychology and cognitive science.

Thank you to Dan Palka and Paul Haward for superb companionship, conversation, and camaraderie across time zones, continents, and stages of life.

Thank you to Caterina Magri for unceasing support, exploration, fun, learning, and laughter.

Thank you to Erin Thomas for wonderful and enduring emotional and intellectual engagement.

Thanks to Seth Chizeck and Nial Quinlan for years of tremendously gratifying, colorful, and meaningful friendship.

Thanks to Edward Patzelt, Jason Nemirow, Garth Coombs, Matt Gin, and my Ph.D. cohort for making graduate school immensely more rewarding and enjoyable.

Thank you to Tommy Thompson and Suzanne DiMack for their knowledge and guidance in healing and recovery from physical pain.

Thanks to the Greene and Buckner labs for valuable ideas, suggestions, and feedback over the years.

And thank you to my parents, who have been as supportive, encouraging, and caring as could ever be imagined, and to my Westies, Boo-Boo and Mowgli, for years of creative and stimulating discussion.

0

Introduction

0.1 Origins in the Formal Sciences

In the 1950s, Noam Chomsky (1956; 1957; 1975, written 1955–1956) took seriously the puzzle raised by Wilhelm von Humbolt that the language system appears to make "infinite use of finite means"—that is, that humans have the capacity to understand and produce an unbounded number of

sentences, notably with finite cognitive and neural resources. In order to explain what had long been considered a paradox, Chomsky argued that a derivational account of linguistic knowledge is necessary: Rather than a list of constructions or structural constraints that apply to sentences, a generative system must be specified instead.

Chomsky thus looked to mathematical logic, and specifically to recursive function theory, which was developed in the 1920s, 1930s, and 1940s by Kurt Gödel, Alonzo Church, Stephen Kleene, and others, partly in order to formalize the notion of a computation (Soare, 1996, 2009). Recursive function theory provided a means of specifying a finite system capable of "enumerating an infinite class of [linguistic] expressions," where each expression is "a structured complex of properties of sound and meaning" (Chomsky, 2000).

0.1.1 Recursive Operations

In the decades since the notion of a recursive function was formalized, recursion has played a large role in both mathematical logic and theoretical computer science, and its definition in those fields has been strikingly stable, undergoing no major changes since the late 1930s.[1] For the purposes at hand, it is important to recognize that the different classes and formalizations of recursive functions and operations specified in mathematical logic are all considered recursive for precisely the same reason, and indeed there has been a single understanding in the formal sciences of what a recursive operation is at least since Giuseppe Peano developed his natural number axioms at the end of the 19th century. Namely, *an operation is recursive if and only if it includes a self-call* (Lobina, 2017; Roberts, 2006). To

[1] Kleene updated Church's formalization of the general recursive class of functions in 1938, broadening it to be coextensive with the class of all computable functions and terming it the partial recursive class, while Emil Post introduced his canonical (or "production") systems in 1943. Production systems were the specific recursive formalization Chomsky later imported into linguistics.

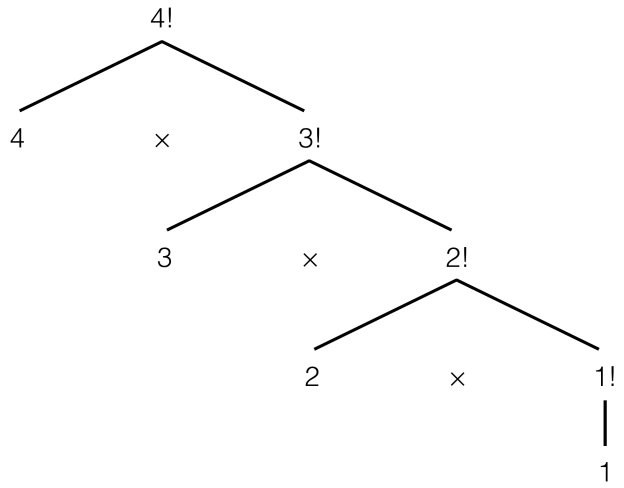


Figure 1: The hierarchy of operations that arises in the recursive computation of the factorial of four. As shown, $4!$ gives rise to $4 \times 3!$, the latter term of which gives rise to $3 \times 2!$, whose latter term gives rise to $2 \times 1!$. From the definition, $1!$ is the base case, which simply returns 1.

illustrate what this means, consider the recursive definition of the factorial class:

(base case)	if $n = 1$, then $n! = 1$
(recursive step)	if $n > 1$, then $n! = n \times (n-1)!$

Note that the definition contains a base case, with instructions for when the input to the formula is 1, and a recursive step, which applies when the input is greater than 1. Critically, the recursive step includes a self-call: In order to calculate the factorial of n, one must first calculate the factorial of $n-1$. If $n - 1$ is itself greater than 1, it will give rise to an additional self-call, with the process continuing until it "bottoms out" at the base case, after exactly $n - 1$ self-calls have been effected. This procedure gives rise to a hierarchy of operations, as can be seen in the calculation of the factorial of four (Fig. 1).

There is no limit to the number of base cases and self-calls that may be included in a recursive

operation; for instance, the recursive definition of the Fibonacci numbers has two of each:

(base case 1)	if $n = 0$, then $fib(n) = 0$
(base case 2)	if $n = 1$, then $fib(n) = 1$
(recursive step)	if $n > 1$, then $fib(n) = fib(n-2) + fib(n-1)$

Note here the presence of two base cases, for $n = 0$ and $n = 1$, as well as two self-calls, $fib(n-2)$ and $fib(n-1)$, in the recursive step. As with the recursive definition of the factorial class, the self-calls in this definition of the Fibonacci numbers give rise to a hierarchy of operations until the base cases are reached.

0.1.2 Recursive Structures

Critically, within the formal sciences there is a substantive distinction between recursive operations and recursive structures. *A structure or object is recursive if and only if it contains a smaller or simpler instance of the same type of structure*, such as the fractal patterns in Fig. 2 (Lobina, 2017; Roberts, 2006; Rodgers & Black, 2004). The ability of humans to construct and manipulate such recursive objects is commonly taken to demonstrate the employment of recursive operations in human cognition. For example, Corballis (2014) argues that recursive operations were necessary for early humans to successfully construct certain kinds of knots and stone tools because of these artifacts' recursive structural features. Similarly, Martins (2012, Martins et al., 2014) argues that recursive operations are necessary for successful performance on a task in which subjects are presented with the first three steps in building a fractal figure and must correctly choose the next step (Fig. 2). Extrapolating from findings like these, some researchers have gone as far as asserting that "[h]umans use recursive operations ... in virtually all aspects of mental life" and that "[a]ll creatures are endowed with recursive ... machinery as part of their standard operating equipment" (Hauser, 2009, p. 49). A number of similar

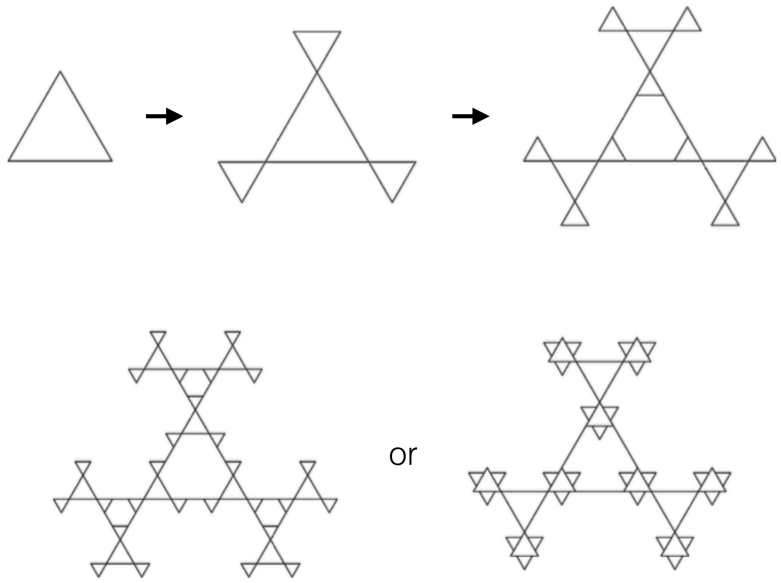


Figure 2: Martins (2012) presented subjects with the first three stages in the generation of a fractal pattern (top row), after which subjects had to choose the correct next stage (bottom row; the correct choice is the figure on the left). Figures adapted from Martins (2012).

claims are described and evaluated by Lobina (2014a, 2017).

In order to evaluate this line of argumentation, the relationship between recursive operations and recursive structures must be explicated. In opposition to the tacit assumption behind such claims, recursive structures are only termed "recursive" *by analogy* to recursive operations, and there is no substantive formal connection between the two (Kleene, 1952; Liu & Stoller, 1999; Lobina, 2014b; Moschovakis, 1998; Rice, 1965; Roberts, 2006). It is a well-known result in mathematical logic that Turing machines, which do not effect self-calls, are extensionally equivalent—that is, they may generate the same input–output mappings (Church, 1951)—to a class of recursive functions defined by Kleene (the partial recursive functions), and both of these identify the class of all computable functions—that is, anything that can be computed.[2] As Kleene (1952) concludes in what is now

[2] This result is variously referred to as the Church–Turing thesis, Church's thesis, Turing's thesis, and by other names, contributing to confusion in the mathematical logic literature about the historical role of recursive functions in formalizing the notion of computability (Soare, 1996, 2009) and in the linguistics and cognitive science literatures about what the thesis actually asserts (Lobina, 2017).

a classic text in mathematical logic, "The following classes of partial functions are coextensive, i.e. have the same members: (a) the partial recursive functions, (b) the computable functions." Thus, any input–output relation computed by a recursive operation can also be computed by an iterative operation (that is, an operation that proceeds without self-calls), and from this it follows that any object that can be generated (or represented, manipulated, etc.) recursively can also be generated iteratively, and vice versa.[3] As a simple example, consider again the calculation of factorials. As an alternative to the recursive approach shown above, factorials may also be calculated iteratively:

```
target = n

counter = 1

product = 1

while counter <= target

    product = product * counter

    counter = counter + 1

return product
```

Note that this solution uses a while loop and three variables to compute the factorial of n, and initiates no self-calls. Despite these differences, for any input in its domain (the natural numbers) it will return the same output as the recursive approach. To reiterate, the existence of an iterative solution for calculating factorials that is extensionally equivalent to the recursive solution is not an anomaly or a coincidence, since "all recursive relations can be reduced to recurrence or iterative relations" (Rice, 1965).

Thus, evidence that humans can generate, represent, and manipulate recursive structures such as fractal patterns does not reveal the nature of the underlying operations; at present, these remain an unexplored empirical matter. Stated differently, examining the output of an operation will not reveal

[3] In fact, automated methods exist for transforming any recursive operation into an iterative one, and these methods are sufficiently general to handle any number of self-calls and base cases (Liu & Stoller, 1999).

whether the operation proceeds recursive or iteratively. In essence, this is a distinction between *what an operation does* (say, generates fractal figures) and *how that operation proceeds* (that is, recursively or iteratively; Lobina, 2017).

0.2 Signatures of Recursive Operations

This independence of procedure and input–output relation being the case, what signatures might one search for as signatures or evidence of recursive operations? The definition of recursive operations provides a clue. Because such operations definitionally include a self-call, they must store values *and* deferred operations at each step, while iterative operations only must store values (Lobina, 2014a). This can be seen in the steps of calculating factorials recursively and iteratively. Consider first the recursive implementation, used in this case to calculate the factorial of four (Table 1). At each step, values (the integer multiplicands and the inputs to each call of *fact*) and deferred operations (the unresolved multiplication steps themselves) both must be stored in memory while the self-calls (the additional instances of *fact*) are being executed. The deferred multiplication steps are only resolved in the second half of the process, when no additional self-calls are forthcoming, and must be stored in memory until then. By contrast, consider the steps in the iterative implementation. As can be seen, only values must be stored at each step (the *target*, *counter*, and *product* from the iterative approach specified earlier), and no deferred operations must be stored in memory because no self-calls occur.

Of course, this holds as we extend the process and compute larger factorials (Table 2), in which case the accumulation of deferred operations in the recursive approach becomes even more salient. This accumulation results in a marked increase followed by a decrease in memory requirements for a computer following the recursive implementation, in contrast to a constancy of memory requirements for the iterative implementation.[4] Put in cognitive terms, there is a conspicuous curve in working

[4] I am here ignoring the potential difference in memory required for storing larger integers in the later steps of the iterative implementation. Such a difference is particular to arithmetical examples of the sort I have been

Recursive Implementation	Iterative Implementation		
	target	*counter*	*product*
$fact(4)$	4	1	1
$4 \times (fact(3))$	4	2	1
$4 \times (3 \times (fact(2)))$	4	3	2
$4 \times (3 \times (2 \times (fact(1))))$	4	4	6
$4 \times (3 \times (2 \times 1))$	4	5	24
$4 \times (3 \times 2)$			
4×6			
24			

Table 1: The steps in calculating the factorial of four using the recursive and iterative implementations specified earlier. Note the presence of both values and deferred operations in the steps of the recursive implementation, in contrast to the presence of only values in the steps of the iterative implementation.

memory required to carry out the recursive implementation, but not the iterative implementation. Given that this curve is an obligate consequence of the self-calls that are included by definition in any recursive implementation, *predictable changes in working memory load should be evidence of recursive operations in cognition*.

Intuitively, it does not appear highly likely that humans calculate factorials via recursive operations. Given this, where should one look for potential recursive operations in human cognition? Despite early expectations, work in psycholinguistics has not uncovered evidence for their involvement in the production or comprehension of sentences. While conflation between characteristics of operations and the structures they produce is as rife within linguistics as in other subfields of cognitive science (Lobina, 2014b), investigation along the lines I have been outlining here has not uncovered evidence of recursive operations. Lobina (2017, chapter 5) employed a tone-detection paradigm to detect changes in working memory load at boundaries in auditorally presented sentences as evidence of recursive self-calls, but found none. Note that the likely non-recursiveness of the parser does not

employing up to now and will not be relevant in the experiments to be described. Similarly, the fact that multiplication is commutative and the deferred multiplication operations could in principle be resolved at any step in the recursive process is idiosyncratic to the present illustration.

Recursive Implementation	Iterative Implementation		
	target	*counter*	*product*
$fact(10)$	10	1	1
$10 \times (fact(9))$	10	2	1
$10 \times (9 \times (fact(8)))$	10	3	2
$10 \times (9 \times (8 \times (fact(7))))$	10	4	6
$10 \times (9 \times (8 \times (7 \times (fact(6)))))$	10	5	24
$10 \times (9 \times (8 \times (7 \times (6 \times (fact(5))))))$	10	6	120
$10 \times (9 \times (8 \times (7 \times (6 \times (5 \times (fact(4)))))))$	10	7	720
$10 \times (9 \times (8 \times (7 \times (6 \times (5 \times (4 \times (fact(3))))))))$	10	8	5040
$10 \times (9 \times (8 \times (7 \times (6 \times (5 \times (4 \times (3 \times (fact(2)))))))))$	10	9	40320
$10 \times (9 \times (8 \times (7 \times (6 \times (5 \times (4 \times (3 \times (2 \times (fact(1))))))))))$	10	10	362880
$10 \times (9 \times (8 \times (7 \times (6 \times (5 \times (4 \times (3 \times (2 \times 1)))))))))$	10	11	3628800
$10 \times (9 \times (8 \times (7 \times (6 \times (5 \times (4 \times (3 \times 2)))))))$			
$10 \times (9 \times (8 \times (7 \times (6 \times (5 \times (4 \times 6))))))$			
$10 \times (9 \times (8 \times (7 \times (6 \times (5 \times 24)))))$			
$10 \times (9 \times (8 \times (7 \times (6 \times 120))))$			
$10 \times (9 \times (8 \times (7 \times 720)))$			
$10 \times (9 \times (8 \times 5040))$			
$10 \times (9 \times 40320)$			
10×362880			
3628800			

Table 2: The steps in calculating the factorial of 10 using the recursive and iterative implementations specified earlier.

entail that the internal computational procedure situated by generative grammarians at the heart of the language faculty be similarly non-recursive, as Stabler (2014) shows in detail. As will be discussed in the Conclusion, a recursive generative procedure may be effected in real time by non-recursive mechanisms. Thus, to date, real-time recursive operations have not been discovered in cognition, and their existence remains an open question.

Usefully for our purposes, computer science provides criteria for when recursive solutions tend to be most efficient (Roberts, 2006), and these serve to constrain the kinds of tasks and cognitive domains for which the involvement of recursive operations is most plausible. The criteria are fairly intuitive: First, the problem to be solved must be decomposable into simpler instances of the same problem. Second, the sub-problems must eventually be so simple that they may be solved without further subdivision. Third, it must be possible to combine the solutions of the sub-problems into a solution to the original problem.

To be clear, these criteria do not preclude the involvement of recursive operations in particular tasks *a priori*: The extensional equivalence of recursive and iterative operations described above ensures that operations of either sort may accomplish any computable task. Rather, the criteria are merely guidelines for the kind of problems for which recursive solutions are usually computationally more efficient than iterative solutions. If we assume, plausibly, that computational efficiency is one determinant of the kind of operations the mind employs, identifying tasks that fit the above criteria is a sensible methodological strategy for establishing the possible existence of recursive operations. This is the strategy that will be pursued in the present dissertation, and a human-solvable task will be employed that meets these criteria and thus is a plausible candidate for being underlain by recursive operations. The ensuing results, comprising data on working memory load and computational models of recursive and non-recursive task strategies, will for the first time provide evidence for the existence of recursive operations in human cognition.

1

Pyramidal Working Memory Hypothesis

In order to maximize the likelihood of detecting evidence for recursive operations in human performance, a task was chosen that fits Roberts's (2006) criteria for when recursive operations are most efficient for solving a problem. While this certainly does not guarantee such operations will be used in task performance, it provides a reasonable place to search in the hope of identifying their existence. The chosen task consists of rectangular blocks that are moved around a game board in order

to free a target block to exit the board. This structure allows the task to be broken down into subtasks whose solutions may be recombined into an overall solution. In this respect, the task is similar to several classic tasks used by cognitive scientists to study planning and the use of heuristics, most prominently the Tower of Hanoi (Hofstadter, 1985), but it differs in several ways. Most importantly for the aims of the present investigations, the selected task has a large number of possible starting configurations featuring different numbers and sizes of movable pieces, so individual puzzles (that is, starting configurations presented to subjects) may require dramatically different solutions and, it is hoped, patterns of changes in working memory, as will be relevant in Studies 3–6. In addition, the selected task has the experimentally desirable features of being easy to pick up and relatively difficult to master: That is, most subjects intuitively understand the constraints and goal of the task, and quickly being solving individual puzzles with minimal instruction. On the other hand, there are no simple and general heuristics, as there are in the Tower of Hanoi task (Simon, 1975), that may be straightforwardly applied to a wide variety of starting configurations. This is advantageous as dependent variables like working memory load (and, later, piece order) could become more difficult to measure with high resolution and precision as task performance speeds up and becomes automatic. The task is thus well-suited for reasons both theoretical and methodological to detecting the kind of broad pyramidal pattern of change in working memory load described in the Introduction. As discussed, such a pattern would be expected as a result of accumulating and resolving self-calls if recursive operations underlie task performance.

1.1 Study 1: Pyramidal Working Memory Hypothesis

1.1.1 Participants

Undergraduate and graduate students (N = 75) aged 18 through 35 were recruited for a half-hour-long lab session and compensated with either course credit or cash payment of $10.00.

1.1.2 Procedure

Subjects performed spatial block-puzzle tasks inspired by the children's puzzle game Rush Hour,[1] which featured a computer screen with an eight-by-eight game board containing rectangular pieces two, three, or four squares in length. Subjects used a trackpad to click on and move the pieces around the board in order to free the red piece and move it to the board's exit on the right-hand side. Pieces could only be moved lengthwise: That is, horizontally-oriented pieces could be moved left and right, and vertically-oriented pieces could be moved up and down. There were 27 puzzles total, each including one red piece and between four and 12 gray pieces. Of these 27, 18 were target puzzles to be analyzed later (Fig. 1.1). This set of target puzzles was designed to include a wide variety of configurations and solution strategies. Subjects completed as many puzzles as they could during the 26-minute experimental session (some of which was spent on instructions and practice puzzles). Metrics including the timestamp, sequence, and location of each move were recorded, as well as click locations and the duration of each puzzle.

In order to probe working memory load during performance of the task, subjects were interrupted at random intervals (with a mean of 15 seconds and minimum of five seconds) and asked to report how much information it felt like they were holding in mind by making a key press ranging from 1 to 9 (Fig. 1.2).

The timestamps of subjects' memory reports were scaled on a per-trial basis so that each timestamp ranged from 0 to 1, reflecting the proportional time during a trial at which a subject was interrupted and asked to report his or her memory. Memory reports were also scaled to range from 0 to 1 and were plotted by their scaled times to visualize changes in memory load during the task.

Individual trials (that is, instances of a single subject attempting a single puzzle) were excluded from analysis if (1) the subject failed to solve that trial (i.e., clicked a "skip" button on the screen), (2)

[1] Developed by Nobuyuki Yoshigahara and sold by ThinkFun.

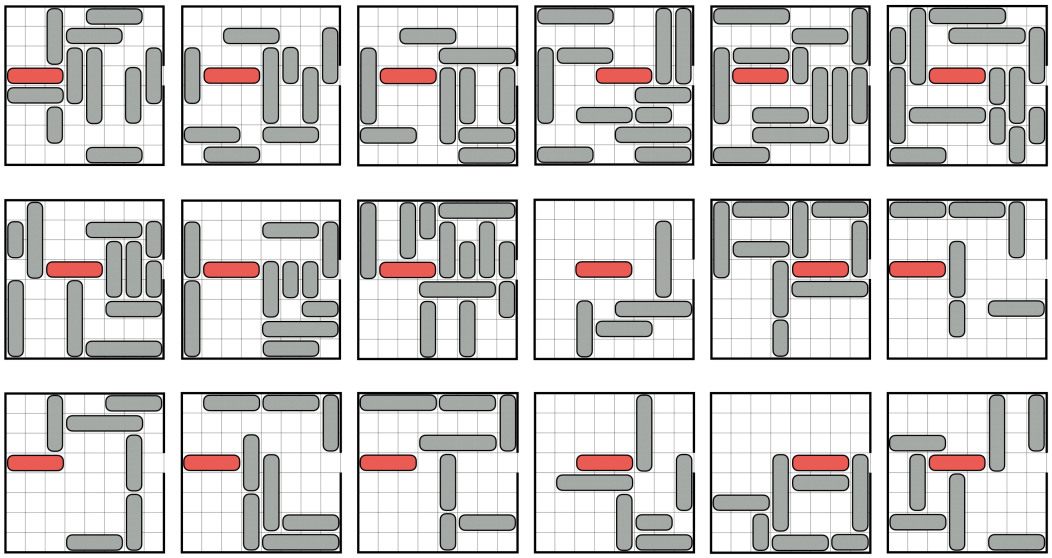


Figure 1.1: The 18 target puzzles used in Studies 1–6.

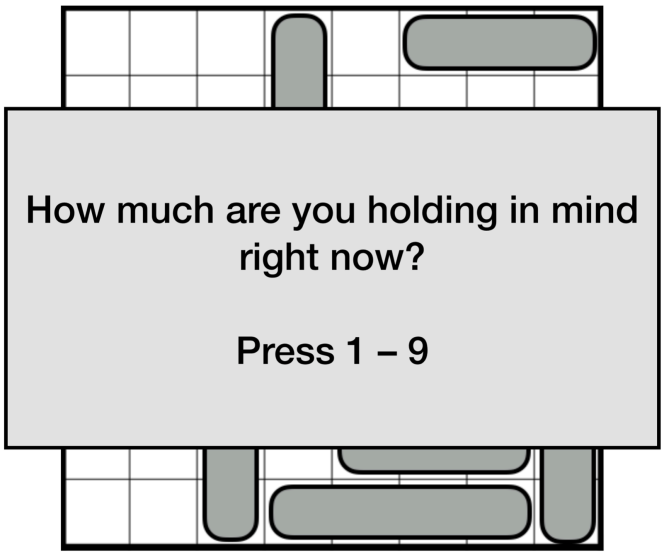


Figure 1.2: The memory probe, which appeared at random time intervals during trials of the puzzle.

waited more than 20 seconds before making the first move, (3) waited more than 15 seconds between moves, or (4) made a total number of moves more than two standard deviations above or 1.5 times the minimum number of moves all subjects made on that puzzle.

1.1.3 Results

Fig. 1.3 depicts the expected pattern of working memory load if recursive operations underlie task performance, as well as binned mean memory load data from 75 subjects across the 18 target puzzles. Upon visual inspection, these data are suggestive of the employment of recursive operations. Low memory loads appear at the beginning and end of the trial, and high memory load is seen midway through, as predicted if recursive operations underlie performance. See Figs. S.1 to S.3 in Appendix B for scatter plots.

More precisely, the prediction of the recursive hypothesis is that working memory load should increase while deferred operations are accumulating and then decrease as they are being resolved. In the present task, deferred operations should only be accumulating prior to subjects' execution of their first move on each puzzle, and they should in general be resolving afterward as subjects are making moves on the board. Figs. 1.4 and 1.5 show hypotheses and memory loads from these periods of the task. Once again, the data support involvement of recursive operations. Fig. 1.4 shows a low memory load at the beginning of the trial, increasing as subjects approach their first move (which occurred at a mean of 6.61 seconds; standard deviation = 3.87 seconds). Fig. 1.5 shows high memory load following subjects' first move, dropping by the end of the trial (mean trial duration = 20.82 seconds; s.d. = 6.13 seconds).

Linear regression revealed that memory load indeed increased from the beginning of the trial to the first move ($p < 0.0001$), with subjects providing memory reports that were 3.17 points higher on the nine-point scale by the time of the first move. A separate linear regression showed that subjects' memory loads were 2.64 points lower at the end of the trial than immediately after the first move

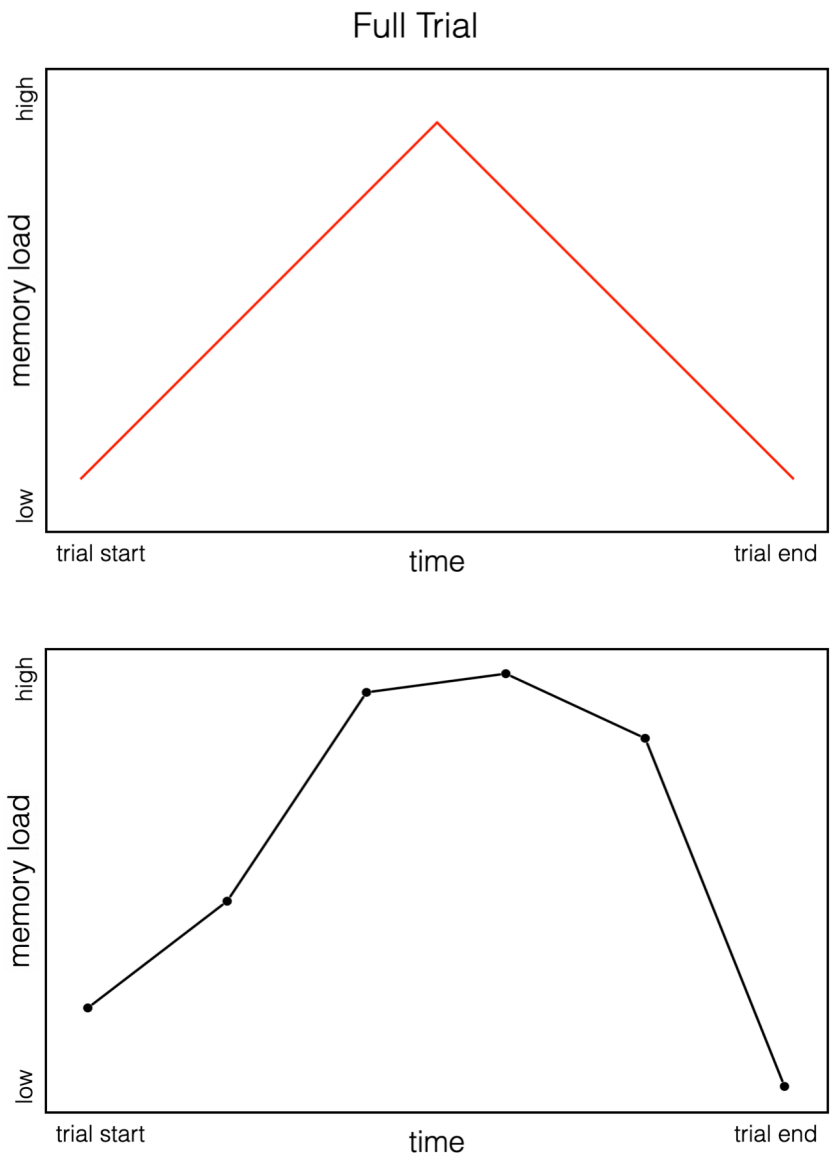


Figure 1.3: (Top) Predicted memory curve if recursive operations underlie performance of the Rush Hour task. (Bottom) Mean memory reports from 75 subjects across 18 different puzzles, scaled for visualization. In both figures, the horizontal axis depicts time from the beginning to the end of a puzzle—that is, from when a particular puzzle first appeared on the screen to when subjects solved the puzzle by sliding the red piece off the board. The vertical axis shows reported memory load from low to high. See Fig. S.1 for scatter plot.

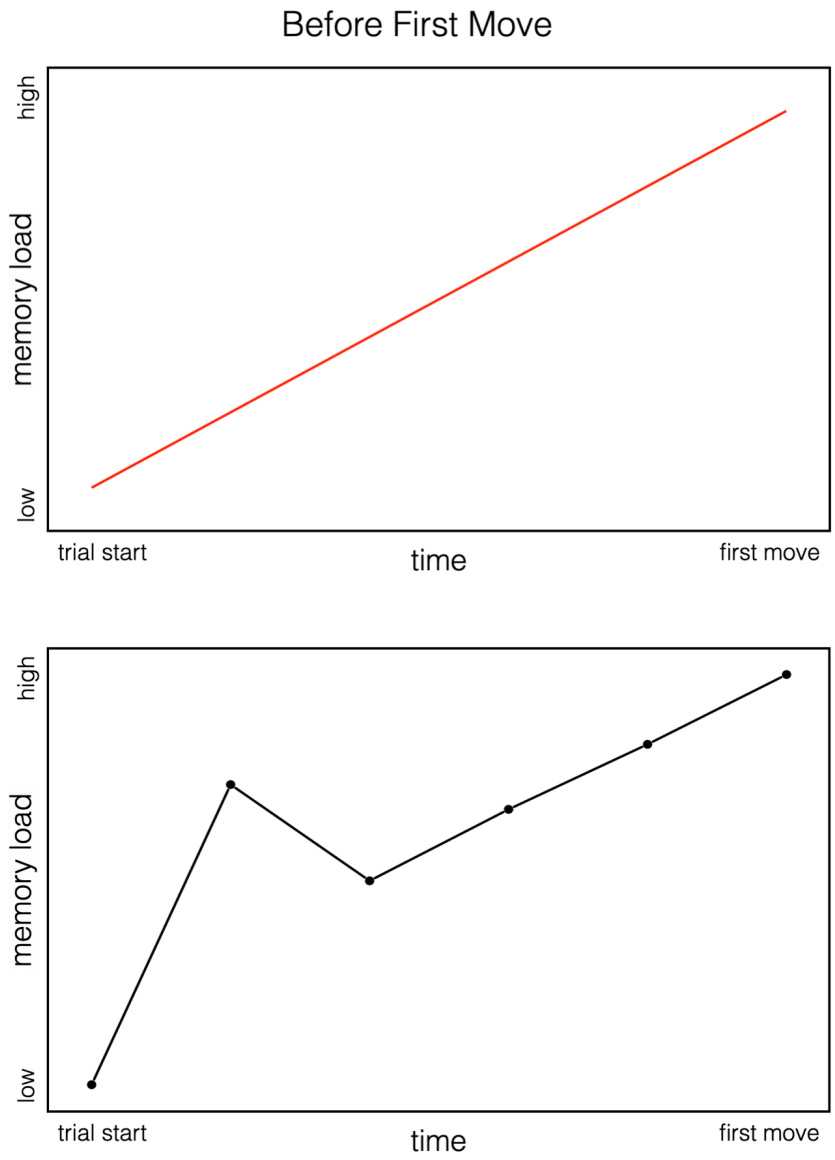


Figure 1.4: (Top) Predicted memory curve for recursive operations prior to the first move on each puzzle. (Bottom) Mean memory reports from 75 subjects across 18 different puzzles, scaled for visualization. In both figures, the horizontal axis depicts time from the beginning of a puzzle to when a subject made his or her first move on that puzzle. The vertical axis shows reported memory load from low to high. See Fig. S.2 for scatter plot.

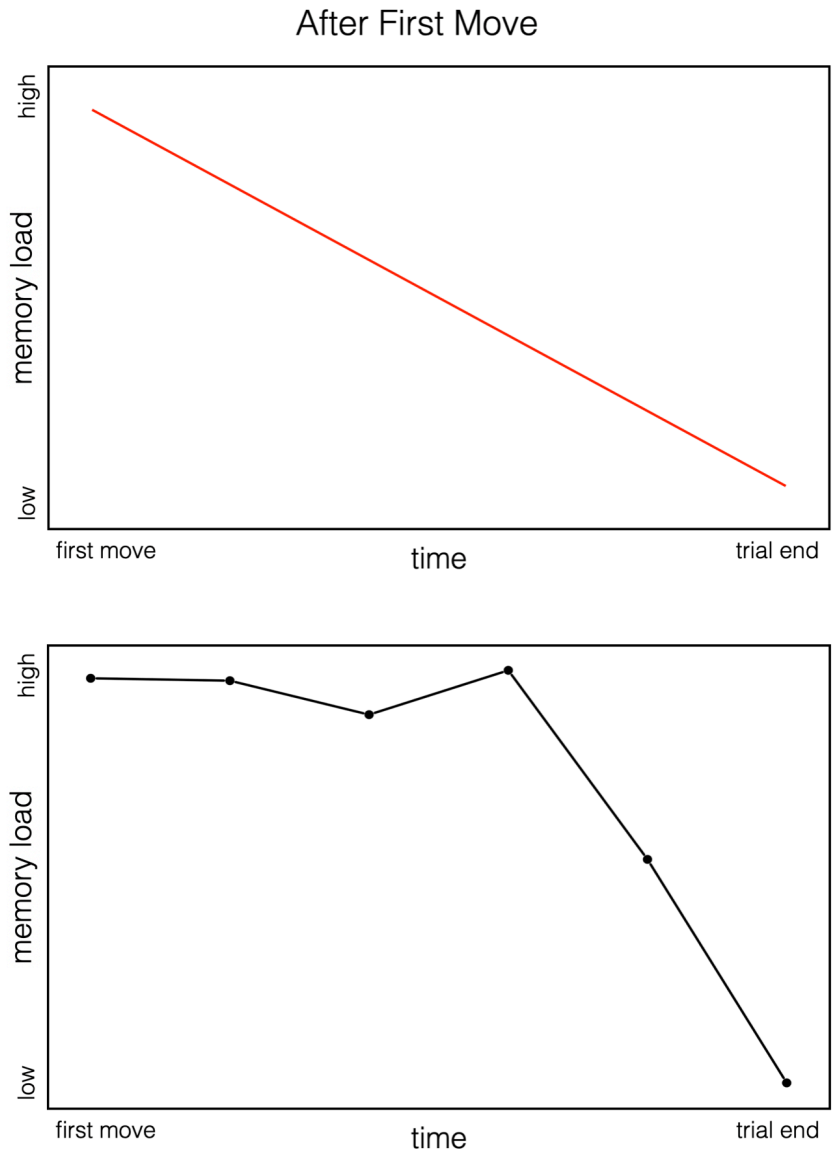


Figure 1.5: (Top) Predicted memory curve for recursive operations after the first move on each puzzle. (Bottom) Mean memory reports from 75 subjects across 18 different puzzles, scaled for visualization. In both figures, the horizontal axis depicts time from the beginning of a puzzle to when a subject made his or her first move on that puzzle. The vertical axis shows reported memory load from low to high. See Fig. S.3 for scatter plot.

($p < 0.0001$).

1.2 Study 2: Pyramidal Working Memory Hypothesis: Replication

In order to further confirm the existence of a pyramidal working memory effect, a replication with a larger sample size was undertaken.

1.2.1 Participants

Undergraduate and graduate students (N = 104) aged 18 through 35 were recruited for a half-hour-long lab session and compensated with either course credit or cash payment of $10.00.

1.2.2 Procedure

A replication of Study 1 that followed identical procedures was preregistered (AsPredicted #27479) and carried out.

1.2.3 Results

As in Study 1, visual inspection of Fig. 1.6 (top) once again reveals a pyramidal shape, and Fig. 1.6. (Note that binning of the data influences the appearance of the curves; for scatter plots see Figs. S.4 to S.6.) Linear regression showed that memory load increased ($p < 0.0001$) from the beginning of the trial to the first move (which occurred at a mean of 6.57 seconds; s.d. = 3.84 seconds), with memory reports 5.26 points higher by the time of the first move (Fig. 1.6, middle). A separate linear regression ($p = 0.015$) revealed that memory loads were 0.86 points lower at the end of the trial (mean trial duration = 21.19 seconds; s.d. = 6.26 seconds) than just after the first move (Fig. 1.6, bottom; this curve is somewhat less clean than that seen in Study 1).[2]

[2] In addition to Study 1 and Study 2, these effects have also been replicated in several other datasets (N = 20, 39, 61, 33, and 23) using different puzzles from the 18 mentioned here and an alternative probe question as an

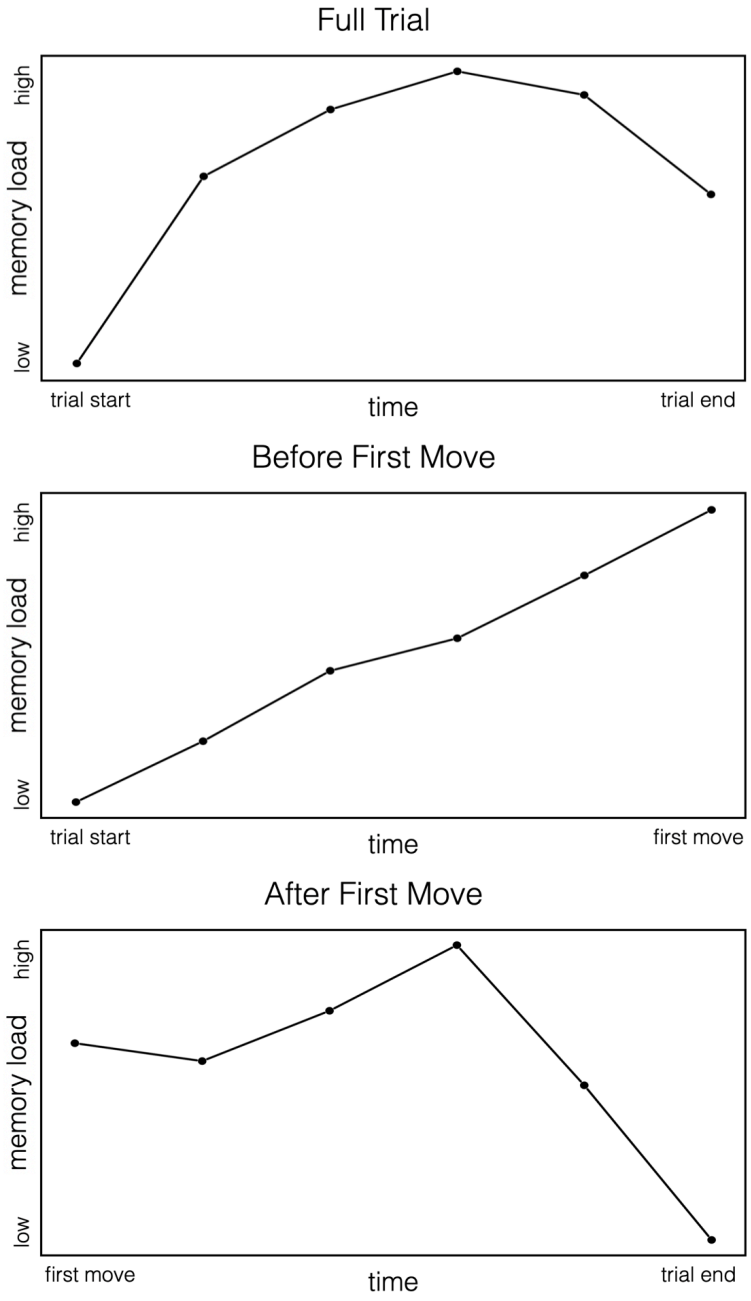


Figure 1.6: Mean memory reports from 104 subjects across 18 different puzzles, scaled for visualization, for the entire trial (top), before the first move of each trial (middle), and after the first move of each trial (bottom). In all three figures, the horizontal axis depicts time from the beginning of a puzzle to when a subject made his or her first move on that puzzle. The vertical axis shows reported memory load from low to high. See Figs. S.4 to S.6 for scatter plots.

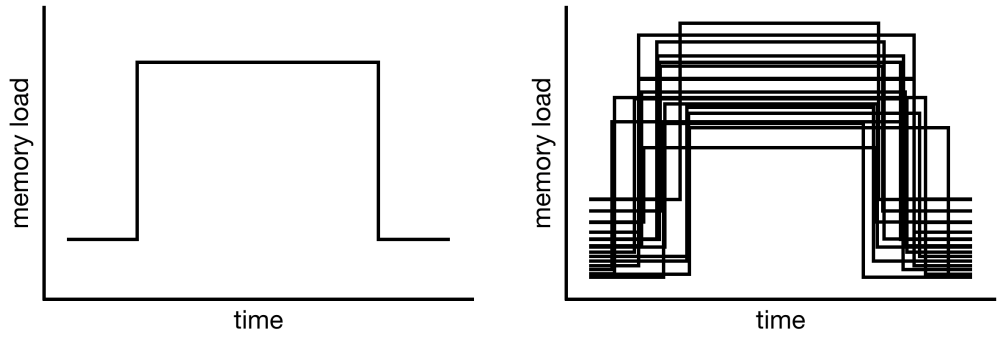


Figure 1.7: (Left) A simple step function that could reflect a subject's working memory load in the absence of recursion operations during the puzzle. (Right) When a number of such step functions are superimposed, random sampling could produce high-density clusters for low memory load at the beginning and end of trials and for high load in the middle of the trial, as observed in Study 1 and Study 2.

1.3 Pyramidal Working Memory Hypothesis: Discussion

As demonstrated in Studies 1 and 2, measuring self-reported memory load during performance of the puzzle task provides evidence for the employment of recursive operations in task performance. The pyramidal pattern of working memory load during task completion would be expected across trials if self-calls are present, causing deferred operations to generally accumulate prior to a subject's first move on a puzzle, and then generally terminate thereafter.

However, the presence of the observed pyramidal pattern does not by itself conclusively demonstrate the involvement of recursive operations, and a particularly plausible alternative hypothesis remains. Namely, the pattern could be the result of a number of superimposed step functions across subjects and puzzles (Fig. 1.7). Such memory load step functions could be produced by various non-recursive procedures, with the stepwise increase in memory load simply reflecting the onset of task engagement, and the later stepwise decrease reflecting disengagement upon solving or nearly solving the puzzle. This alternative account for the pattern of results in Studies 1 and 2 should be ruled out.

---

indicator of working memory load ("How hard are you thinking right now?"), among other methodological differences. In some cases these datasets yielded cleaner increases and decreases in reported memory load than those seen in Study 1 and Study 2.

2

Recursive Solver Model

If recursive operations truly underlie performance of the puzzle, and self-calls result in increases in working memory load at predictable points in time, it should be possible to make finer-grained predictions of memory load dynamics on the basis of the particular sequences of moves required by individual puzzles. One way to generate such predictions is by constructing a recursive computational model that solves puzzles and whose memory usage may be tracked step by step during

puzzle performance. If such a model predicts unique features of human memory load dynamics across individual puzzles, it would not only rule out the simple step function alternative hypothesis discussed above, but would also constitute stronger evidence for the presence of recursive operations, as puzzle-by-puzzle variations in memory dynamics constitute a more complexly structured phenomenon to predict.

Given that human performance of the present task depends on a number of factors, including heuristics, attentional strategies, visuospatial factors, and the like, and given that such factors are ancillary to the present aim of identifying recursive operations, a minimalist modeling strategy is appropriate. The solver model to be described is not intended to capture human behavior with maximum fidelity in each attempt at a puzzle, and instead sacrifices humanlike performance in favor of extreme parsimony. That is, the solver essentially has the ability to (1) move pieces, (2) check whether a move is possible or is blocked by another piece, and (3) initiate self-calls. No strategy or heuristics have been included for making performance more efficient or humanlike; instead, the model employs randomness to select between different available moves, thus generating a large number of solutions to each puzzle. After this, the solution that is closest to a human subject's is chosen for comparison. A key advantage of this minimalist strategy is that if the solver's memory usage is indeed found to predict moment-to-moment changes in human working memory load, there will be no possibility this is due to auxiliary model components like sophisticated heuristics. That is, a positive result will demonstrate that the ability to initiate self-calls alone is sufficient to capture memory load dynamics across individual puzzles.

2.1 Study 3: Recursive Solver Model

2.1.1 Procedure

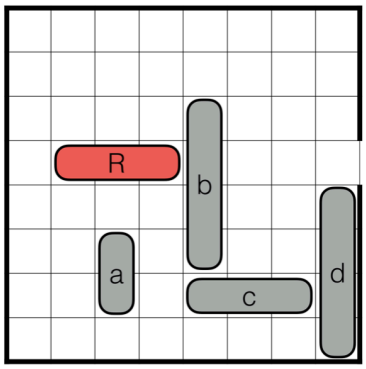
In order to generate more detailed predictions of working memory load during the task, we constructed a computer program that solves individual puzzles recursively, that is, making use of self-calls. Simplified pseudocode for the recursive solver program is as follows:

```
move_piece(target_piece, destination):
    while target_piece to destination is blocked:
        move_piece(blocking_piece, random_dir)
    move target_piece to destination
```

The move_piece function takes as input a *target_piece*—that is, the piece to be moved—and the *destination* it is to be moved to. As long as the path from the *target_piece* to its *destination* is blocked, it initiates a self-call, giving rise to a new instance of the move_piece function to move the *blocking_piece* out of the way in a random direction (i.e., randomly choosing between up and down or left and right, depending on the orientation of the piece). Once the *target_piece* is no longer blocked, it is moved to its *destination*.

Because the move_piece function is nondeterministic and chooses randomly among a set of possible moves, the solver generates a variety of solutions to each individual puzzle: See the steps for two different sequences of moves for the puzzle in Fig. 2.1. Moment-to-moment memory requirements for each sequence may be ascertained, with required memory defined as recursive depth, or the number of instances of the move_piece function simultaneously maintained in memory at each step of the process.[1]

[1] A step is here defined as any move, self-call, or alteration of a data structure. Data structure manipulation is not relevant for the recursive solver, but will be for the stack-based iterative solver described in Study 5 and Study 6.



move_piece(*R*, *exit*)
(move_piece(*b*, *down*)), move *R*
((move_piece(*c*, *left*)), move *b*), move *R*
(((move_piece(*a*, *up*)), move *c*), move *b*), move *R*
(((move *a*), move *c*), move *b*), move *R*
((move *c*), move *b*), move *R*
(move *b*), move *R*
move *R*

move_piece(*R*, *exit*)
(move_piece(*b*, *down*)), move *R*
((move_piece(*c*, *right*)), move *b*), move *R*
(((move_piece(*d*, *up*)), move *c*), move *b*), move *R*
(((move *d*), move *c*), move *b*), move *R*
((move *c*), move *b*), move *R*
(move *b*), move *R*
(move_piece(*d*, *down*)), move *R*
((move_piece(*c*, *left*)), move *d*), move *R*
(((move_piece(*b*, *up*)), move *c*), move *d*), move *R*
(((move *b*), move *c*), move *d*), move *R*
((move *c*), move *d*), move *R*
(move *d*), move *R*
move_piece(*R*, *exit*)
(move_piece(*b*, *down*)), move *R*
((move_piece(*c*, *left*)), move *b*), move *R*
(((move_piece(*a*, *up*)), move *c*), move *b*), move *R*
(((move *a*), move *c*), move *b*), move *R*
((move *c*), move *b*), move *R*
(move *b*), move *R*
move *R*

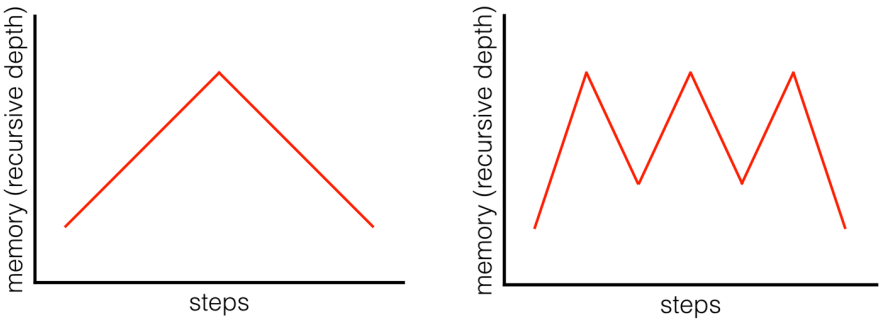


Figure 2.1: (Top and Middle) The steps of operations in two different solutions to a particular puzzle carried out by the recursive solver. Note the self-calls to the move_piece function as well as the deferred moves that accumulate and then are resolved during the process. The solution on the left requires substantially fewer steps than the one on the right, owing to which moves were selected during the process. (Bottom) Curves depicting required memory or recursive depth for the two solutions.

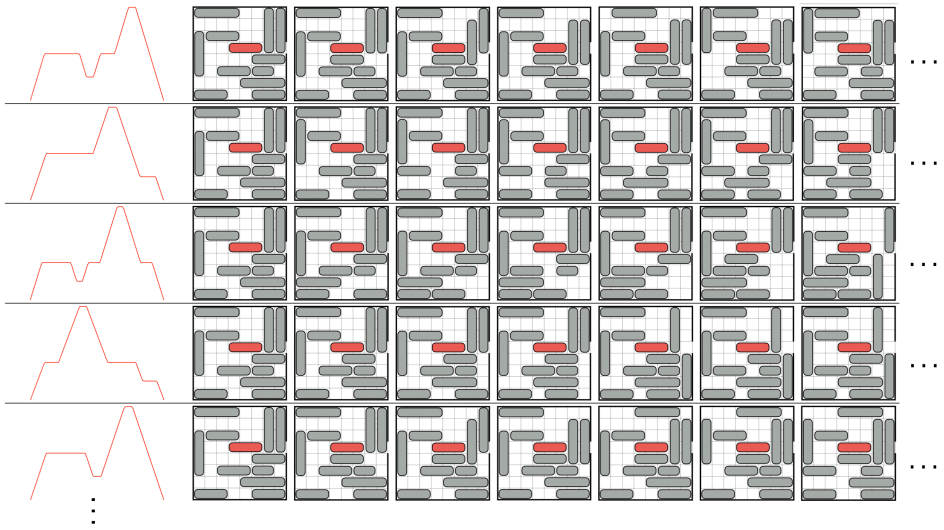


Figure 2.2: The opening moves of five different solutions for solving a puzzle (Right), and their accompanying memory curves (Left).

The move_piece function is here presented in somewhat simplified form for concision and clarity. Among the aspects that are omitted are provisions for identifying which pieces are blocking a piece from its destination and for choosing a particular move from multiple options. This latter functionality is implemented with a degree of randomness: If there are two different blocking pieces that may be moved out of the way, or if there are three possible destination positions for a given blocking_piece, they will be chosen from randomly so that each will be selected some proportion of the time across many runs of the solver. This randomness allows a wide range of move sequences—and thus solutions to a given puzzle—to be reached over many attempts.

Specifically, the solver was applied to each puzzle 100,000 times in order to generate a variety of solutions (dozens or hundreds, depending on a given puzzle's complexity; see Fig. 2.2), and this process was repeated for all puzzles (Fig. 2.3).

Resultant solver working memory curves were compared to human subjects' self-reported memory load data from Study 1 ($N = 75$). To do this, of the variety of curves generated for each puzzle, a single memory curve was selected for comparison with a given subject's data points on a given puzzle. To

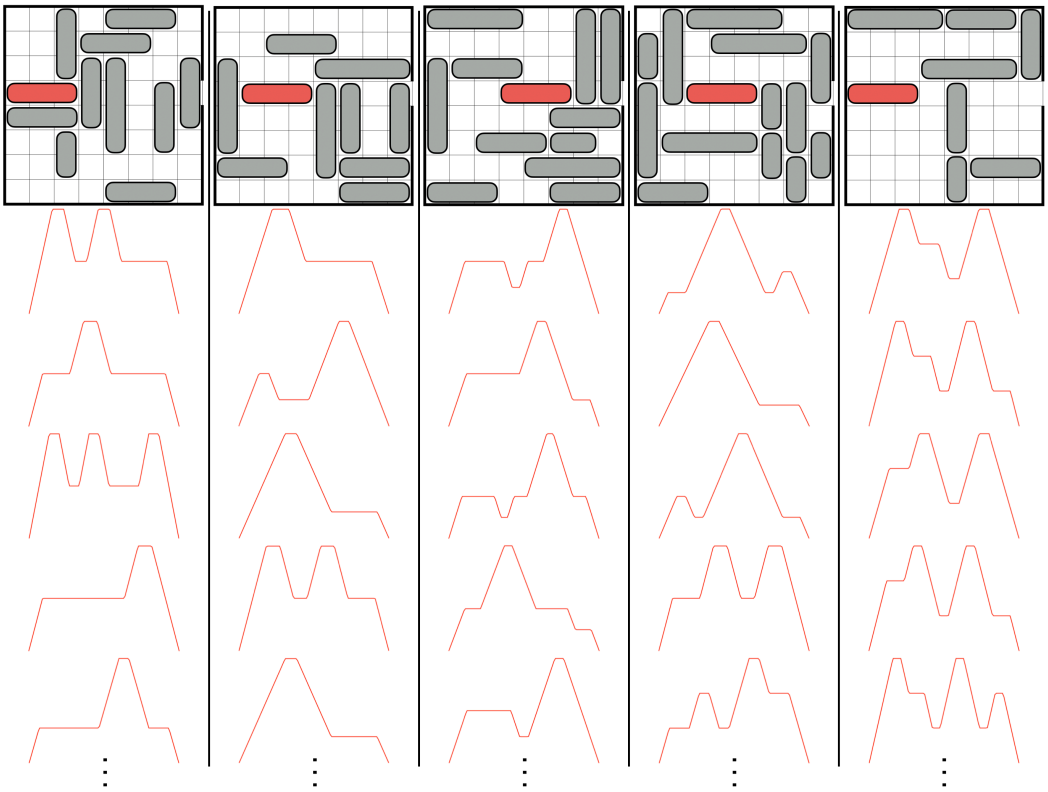


Figure 2.3: Five different memory curves for each of five different puzzles. Some of the memory curves differ slightly from one another, while others differ dramatically.

make this selection, dynamic time warping was used to identify the sequence of moves most similar to the sequence the human subject used to solve a given puzzle. Dynamic time warping is a technique to bring two time series into alignment with one another by generating a distance measure that may be used to compare similarity across a group of time series. The solver's and the human subject's move sequences for each puzzle were turned into n-dimensional vectors, with n being the number of pieces on the board. Each unique solver vector was then warped to the human subject's vector, and the solver vector with the lowest total warping distance was selected for analysis—that is, the solver solution whose moves were most similar to the human subject's solution. The memory curve accompanying that sequence of moves could then be compared to the subject's self-reported memory load for that puzzle. Critically, this method for selecting the prototype solver solution for a puzzle is independent of the memory curve that will be compared with human memory load—that is, selection is only on the basis of the moves made on the board and does not use information about the memory curve itself or the subject's memory reports.

After the solver was run 100,000 times on each of the 18 puzzles (see Fig. 1.1) and a single solver solution was selected for each human solution of each puzzle, the solver solutions' accompanying memory curves could be compared with human memory reports. To do this, solver memory load and human memory reports were scaled to a range of 0 to 1, as were real-world time (for humans) and computational steps (for the solver) (see Fig. 2.4). As a measure of fit between the solver and human data, the sum of squared residuals was calculated between the solver curve and human memory reports for each puzzle. To determine whether the solver predicts unique features of changes in human working memory load across puzzles, a permutation test was carried out, with the 18 solver curves and the 18 sets of human reports (that is, one for each puzzle) scrambled with respect to one another 25,000 times and the squared residuals calculated for each permutation. The true sum of squared residuals was then compared to the distribution of scrambled residuals to obtain a p-value.

To assess the possibility the the solver may simply be capturing working memory differences be-

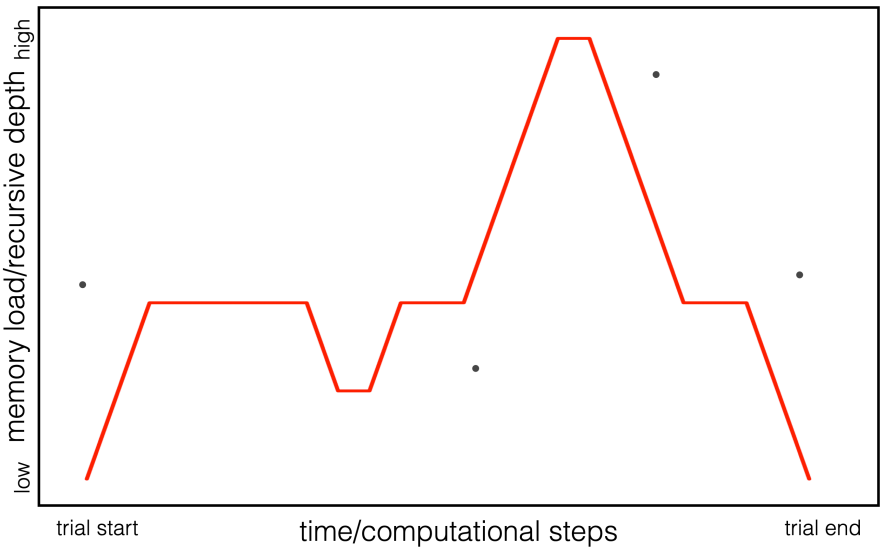


Figure 2.4: Prototype solver curve (red) and a human subject's memory reports (points) for a single trial of the puzzle.

tween puzzles of different difficulties or complexities—say, generating one kind of curve for simple puzzles and another for more difficult puzzles and thereby capturing broad difficulty-dependent differences in human memory load—the number of moves the solver made was used as a proxy for difficulty. Specifically, on each of the 25,000 permutations described above, the difference between the number of moves the solver made to generate the "true" curve was subtracted from the number of moves the solver made to generate the new, scrambled curve, and this difference was squared. The Pearson correlation between these 25,000 differences and the 25,000 permuted sums of squared residuals was calculated. A positive correlation would suggest that the fit between the solver curve and the human memory data is better when a solver curve from a solution of similar difficulty or complexity is swapped in, and worse when a solver curve from a solution of very different difficulty is used instead. This would raise the possibility that the solver is doing little more than fitting human memory data on the basis of broad changes in difficulty across puzzles, analogous to merely working harder on harder puzzles.

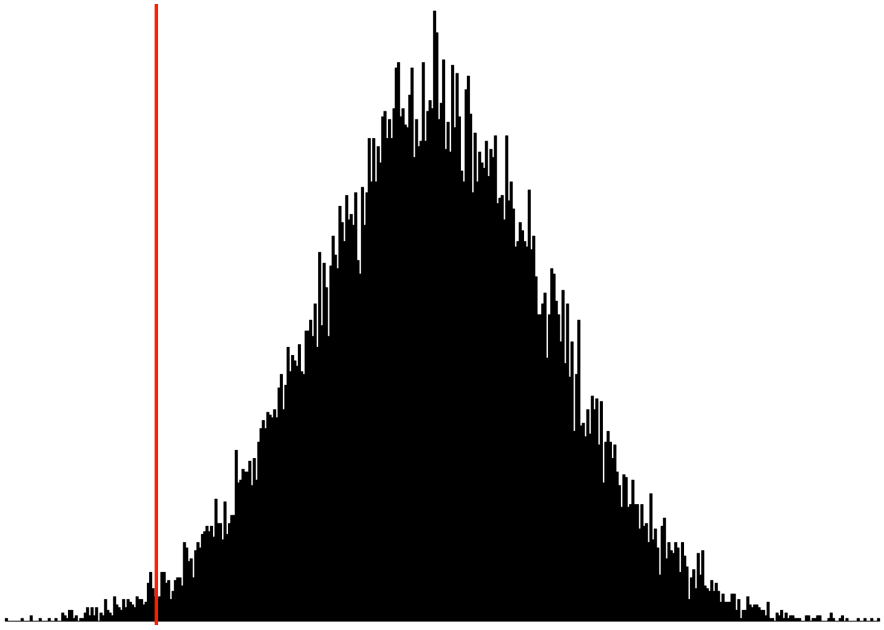


Figure 2.5: True sum of squared residuals (red line) reflecting the fit between recursive solver memory usage and human working memory load reports in Study 3, and permuted sums of squared residuals (black distribution) resulting from scrambling solver curves with human data.

2.1.2 Results

The recursive solver indeed predicted unique features of changes in human working memory load across individual puzzles ($p = 0.009$; Fig. 2.5), suggesting the employment of recursive operations in human task performance. The correlation between permuted sums of squared residuals and differences in numbers of moves was -0.007 ($p = 0.293$), indicating that the solver was not merely capturing differences in puzzle difficulty.

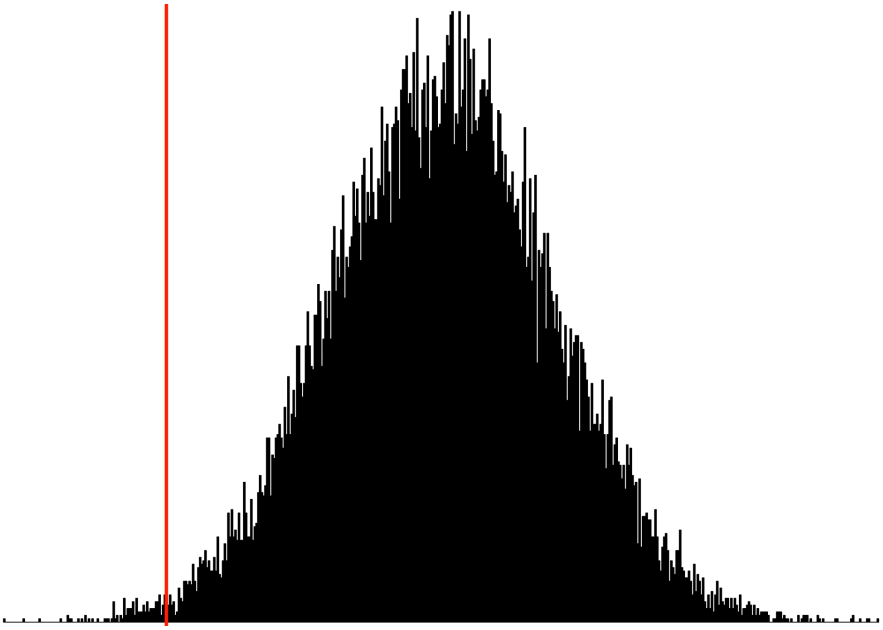


Figure 2.6: True sum of squared residuals (red line) reflecting the fit between recursive solver memory usage and human working memory load reports in Study 4, and permuted sums of squared residuals (black distribution) resulting from scrambling solver curves with human data.

2.2 Study 4: Recursive Solver Model: Replication

2.2.1 Procedure

The human data from Study 2 (N = 104) was analyzed with the solver model from Study 3, using identical analysis procedures. This replication was preregistered (AsPredicted #27479).

2.2.2 Results

As in Study 3, the solver was again found to predict unique features of human working memory load dynamics across individual puzzles, as shown by a permutation test ($p = 0.006$; Fig. 2.6). The correlation between permuted sums of squared residuals and differences in numbers of moves was 0.004 ($p = 0.572$), again showing the solver was not simply capturing differences in puzzle difficulty.

2.3 Recursive Solver Model: Discussion

Studies 3 and 4 demonstrate that the recursive solver exhibits memory dynamics that correspond to those of humans with a high degree of specificity; that is, the solver's memory usage is predictive of human data at the level of individual puzzles. This finding is particularly notable given the simplicity of the recursive solver, which was designed in a minimalist fashion to consist of little more than a capacity to execute self-calls and the ability to move pieces. Additionally, Study 7 and Study 8 in Appendix A demonstrate that the recursive solver also predicts the order in which subjects attend to pieces prior to making a move in the puzzle. This is not to say that no non-recursive model could do the same, though; indeed, non-recursive models that capture human memory dynamics in a similar manner to the recursive solver certainly exist, as will be discussed in the next section. In order to address the central question of whether recursive operations are present in human performance, the way forward will be to identify a non-recursive model that is similarly plausible and parsimonious to the recursive solver and compare their ability to predict human data.

3

Iterative Solver Model

The minimalist recursive algorithm described above and implemented in the recursive solver is certainly not the only possible approach to solving the Rush Hour task, and there in principle exist non-recursive procedures that closely replicate the working memory dynamics of the recursive

one.[1] (In the extreme case, there should exist iterative procedures that simply simulate the working memory requirements of the recursive procedure at each step and adjust their own memory usage to match, say by storing random data in memory at times when the recursive solver's memory load would increase and discarding that random data when the recursive solver's memory load would decrease.) As is the state of affairs in science generally, one should not expect all such alternatives to ever be ruled out, so alternatives should be considered and tested when they meet a minimum threshold of plausibility. One highly plausible iterative procedure whose memory dynamics makes use of a pushdown stack containing not-yet-moved pieces and their destinations. By constructing a solver model of this kind of algorithm and comparing its predictiveness of human working memory load with that of the recursive solver model, this alternative could potentially be ruled out.

3.1 Study 5: Iterative Solver Model

3.1.1 Procedure

A solver model was constructed that makes use of a pushdown stack, that is, a data structure where pieces and destinations may be stored ("pushed") and then returned ("popped") in the reverse order. This model does not effect any self-calls, and thus is iterative in nature, as the following simplified pseudocode illustrates:

```
move_piece:
    move_stack.push([red_piece, board_exit])
    while length(move_stack) > 0:
        if move_stack.pop(piece) to move_stack.pop(dir) is blocked:
```

[1] I am here using the terms "algorithm" and "procedure" mostly interchangeably, though strictly speaking the former may be taken to denote an abstract mathematical object, while the latter is the implementation of such an object in a computer language. As such, considerations like memory usage arise primarily in procedural implementations.

```
            move_stack.push([blocking_piece, random_dir])
        else:
            move move_stack.pop(piece) to move_stack.pop(dir)
```

This iterative move_piece function makes use of a pushdown stack called *move_stack*, which stores piece–destination pairs that are added to and removed from it, or "pushed" and "popped." The function begins by adding the *red_piece* and its destination—the *board_exit*—to the *move_stack*. A while loop is then entered that only terminates when the *move_stack* has been emptied of pieces and destinations. During each cycle of the while loop, if the path from *move_stack*'s most recently-added piece to its destination is blocked, the *blocking_piece* and a destination (that is, a randomly-selected direction) are pushed onto the *move_stack* (analogous to a blocking piece and a destination being passed as input to the recursive solver's self-call). Otherwise, if the path of *move_stack*'s most recently-added piece is *not* blocked, it is simply moved to its destination. This process repeats within the while loop until the *move_stack* is empty, having just moved the *red_piece* to the *board_exit* and finished the puzzle, at which point the function terminates.[2]

Like the recursive solver described in Study 3, the iterative solver is nondeterministic, choosing randomly among a set of possible pieces and destinations to add to *move_stack*. Accordingly, as with the recursive solver, the iterative solver was run 100,000 times on each puzzle, yielding a variety of different solutions. Iterative solver memory curves were then compared to human subjects' self-reported memory load data from Study 1 (N = 75) using the analytic methods described in Study 3. These methods were identical, including dynamic time warping of move sequences to select which iterative solver solution to compare with each subject's memory data, scaling of solver memory load and human memory reports to a range of 0 to 1, and permuting solver curves and human memory reports to

[2]Several similar iterative solver models were also implemented, with different conditions for while loop termination and differences in which commands fell within and outside of the while loop. In some cases more than one while loop was used, or one loop was embedded within another. None of these variations yielded substantially different results to the ones presented here.

determine whether the iterative solver predicts unique features of human memory load dynamics across individual puzzles.

To directly compare iterative solver and recursive solver predictiveness, a separate analysis was carried out in which the two solvers were run 25,000 times on each puzzle, from which a single solution was selected to compare with each human trial, as in Study 3. The fit of the recursive and iterative solvers to the trial-by-trial human memory data from Study 1 (N = 75) was then compared using sum of squared residuals. This process was repeated 100 times in order to determine what proportion of the time the recursive solver predicted human memory data better than the iterative solver.

3.1.2 Results

The iterative solver did not predict the human memory load data from Study 1 better than chance ($p = 0.187$). Direct comparison of the two solvers revealed that the recursive solver predicts human memory data more accurately than the iterative solver: All 100 runs of the recursive solver resulted in a lower sum of squared residuals (that is, better fit to the human memory data; mean sum of squared residuals = 88.8; s.d. = 0.55; units are arbitrary) with the human data than all 100 runs of the iterative solver (mean s.s.r. = 106.7; s.d. = 0.57).

3.2 Study 6: Iterative Solver Model: Replication

3.2.1 Procedure

The human data from Study 2 (N = 104) was analyzed with the recursive and iterative solver models using identical analysis procedures to Study 5.

3.2.2 Results

The iterative solver predicted human memory load data from Study 2 better than chance ($p = 0.029$). Direct comparison of the recursive and iterative solvers again showed that the recursive solver predicts human memory data more accurately: All 100 runs of the recursive solver led to a lower sum of squared residuals (mean s.s.r. $= 122.6$; s.d. $= 0.76$) than all 100 runs of the iterative solver (mean s.s.r. $= 155.2$; s.d. $= 0.33$).

3.3 Iterative Solver Model: Discussion

While the iterative solver, consisting of a pushdown stack that stores pieces and destinations, may weakly predict human working memory load dynamics during the puzzle task, it does so less well than the recursive solver. The recursive solver, consisting of the ability to initiate self-calls and provisions for moving pieces, thus predicts changes in human working memory load better than a plausible non-recursive alternative. Keeping in mind that the recursive solver was designed in a minimalist fashion such that the memory curves it generates are the product of little more than the solver's self-calls, its predictiveness of human dynamics constitutes strong evidence for the involvement of recursive operations in performance of the present puzzle task.

4

Conclusion

The results described in this thesis constitute the first empirical evidence that recursive operations exist in human cognition. A basic, definitional property of recursive operations—that they effect self-calls—was leveraged in the present work and enabled the prediction of human working memory dynamics at both a broad level (Study 1 and Study 2) and on a fine-grained, puzzle-by-puzzle basis (Study 3 and Study 4) when formalized into a simple recursive computational model. This model,

which essentially consists of the abilities to make self-calls and move pieces, predicted human memory dynamics significantly better than an iterative model featuring a pushdown memory stack for storing pieces and their destinations (Study 5 and Study 6). The success of the recursive model is the strongest evidence obtained here that recursive operations underlie performance of the puzzle task and thus exist in human cognition.

Given the historical conflation of recursive operations and structures and the kind of evidence that has traditionally been advanced on the matter as a result, the present work signifies the first empirical evidence for recursive operations in human cognition. While the possibility remains open that non-recursive operations more complex than a pushdown stack model may actually underlie the phenomena under examination, the balance of evidence is decidedly against this alternative. The recursive model not only predicts human memory dynamics across individual puzzles via an extremely simple procedure; it also does so better than a comparably simple iterative model that may in some ways be thought of as its non-recursive counterpart, making use as it does of a pushdown stack for future moves rather than self-calls. Thus, if future investigations identify iterative algorithms that predict human memory dynamics as well as the recursive algorithm, these should either be *simpler* than the extremely simple recursive algorithm described here, or they will need to have independent support for their veracity—a tall order considering the dearth of evidence one way or the other over previous decades. At a bare minimum, we may suggest that the present work narrows the range of possible operations to just those members of an equivalence class with recursive operations on the basis of memory dynamics. That is, the true operations the mind makes use of in the puzzle task are ones that, when applied, result in memory demands mirroring those seen during the execution of self-calls and the accumulation of deferred operations, whether or not these are truly the mechanisms at play.

On a slightly different note, it goes almost without saying that any task whose performance involves recursive operations may also involve non-recursive ones. The possibility (or perhaps likelihood) of

such a "hybrid" strategy in task performance does not undermine the present approach: The aim of this research program has been to establish the *existence* of recursive operations in cognition, but certainly not to establish their exclusivity. Thus, positive evidence for recursive operations, as has been obtained here, is no less compelling—and their existence would be of no less import—due to the possibility that non-recursive operations may also contribute to task performance.

Having obtained evidence for the employment of recursive operations in at least one domain of human cognition, it is appropriate to consider potential reasons for their involvement. In general, recursive operations tend to be more memory-intensive and often take more time to execute than their iterative counterparts, which is why, at least in computer systems, "orbiting conditions—such as memory limits, architectural complexity, efficiency—more often than not bring about iterative implementations" (Lobina, 2017, p. 34). Given that iterative operations can in principle compute any input–output relation recursive operations can, why would the latter ever be used? When programmers employ recursive approaches, they often do so to minimize the complexity and length of the procedure that must be written in code—that is, recursive solutions are often more compact, as well as easier to read and understand, with fewer individual cases that must be hard-coded, than iterative solutions to the same problem. Thus, an iterative procedure for computing a given input–output relation oftentimes requires a lengthier and more complex specification than an extensionally equivalent recursive procedure. This was in fact the case for the two implementations of the factorial function described in the Introduction: The iterative implementation used three variables instead of one and required roughly twice as many lines of code to express (depending on how they are counted). That means that any system for which an operative architectural constraint is algorithmic simplicity may in some cases employ a recursive procedure over its more complex iterative counterpart, even at the cost of speed and memory usage. Or, looked at differently, when matched for their complexity, a recursive procedure may be more powerful or more flexible than an iterative one. This is not a point about recursive or iterative operations *as a class*; rather, it is about *particular* recursive and

iterative operations, and this is the grain of analysis that is called for as future investigations aim to more precisely characterize the operations evinced in the present work.

One may wonder whether the recursiveness or non-recursiveness of operations involved in a task could be established by simply asking participants to describe how they perform that task, either retrospectively or with a think-aloud paradigm. When considering this suggestion, it is important to keep in mind the issue of levels of analysis. Descriptions of the plans, strategies, and goals a subject uses to perform a task are situated at Marr's "computational" level—roughly analogous to Chomsky's "competence" (1984; 2017)—which concerns "what is being computed and why" (Marr, 1977, p. 37) and the mapping that applies between a system's input and its output—say, between a particular board state and a sequence of moves that will lead to a goal state. By contrast, the kind of real-time recursive or iterative operations under investigation here fall squarely within Marr's "algorithmic" level—analogous to Chomsky's "performance"—which deals with how the mappings at the computational level are effected or computed *in real time*, a matter that is constrained not only by "what is being computed," but also by the cognitive mechanisms available, such as memory capacity and attentional resources (Lobina, 2017). Thus, in generative linguistics, a theory of the internal computational procedure (often referred to as the grammar) that maps lexical items to structured sound–meaning complexes is situated at the level of competence, while questions about language comprehension and production—involving real-time operations dependent on working memory, among much else—are questions of performance and are far from wholly determined by the competence-level computational procedure.

In the case of individuals' competence-level descriptions while working on puzzles, these typically include the plans they are carrying out, comprising information like which move they are currently executing, which sub-problems they are solving and why, the way the present task state relates to the goal state, and so forth (Miller et al., 1960; Newell & Simon, 1972). The described plans are more akin to the recursive structures discussed in the Introduction than they are to real-time operations

effecting self-calls: While they may be recursive in their structure (Khemlani & Johnson-Laird, 2007), the recursiveness of these envisioned plans does not require that the operations that carry them out in real time contain self-calls. (The analogue in linguistics, again, is that the recursiveness of the internal generative procedure does not mandate that recursive operations be present in production; Stabler, 2014.) Real-time, algorithmic-level operations are explicitly the focus of the present work, and because of the aforementioned independence of how an operation proceeds (recursively or iteratively) and what that operation does (say, carries out a particular plan), facts about the latter do not strictly determine the former, and insight into the nature of a plan does not guarantee what sort of operations come into play when it is carried out. That is to say, non-recursive operations can execute the steps of a recursively-structured plan, and vice-versa. Put simply, there is nothing a subject could reasonably report about his or her planning and execution that would unambiguously attest the occurrence of a self-call at the algorithmic level. Thus, an individual could provide a description of a plan that is recursive in structure (indeed, subjects sometimes offer such descriptions on the Tower of Hanoi task after hints or training (Simon, 1975; Simon & Reed, 1976), and in other cases are taught to use recursively-structured plans from the outset (Xu & Corkin, 2001)), but this does not reveal whether these plans are being accomplished via algorithmic-level operations that effect self-calls. Because of this, a think-aloud paradigm will not reveal the presence or absence of self-calls at the algorithmic level and so could not take the place of experiments that are designed to detect signatures of these self-calls, like those described here.

Moving to a higher level, the overall aim of the present dissertation has been to establish the existence of recursive operations. The studies described here are thus not optimized for individuating specific properties of the recursive operations the mind may employ, and a future shift in focus from detection to characterization will require different methodologies. As a first pass at discovering properties of recursive operations, determining boundary conditions for their employment would be extremely useful, including the domains and tasks that evoke them as well as task characteristics

that selectively promote or inhibit their use across domains, such as decomposability (Roberts, 2006), recursiveness of objects to be processed or manipulated, recursiveness of task structure, simultaneity with linguistic tasks, and peak memory burden or number of items to be processed. Such investigations would be a step toward cataloging the parts of the mind in which recursive operations take place. As described in the Introduction, only one other search for these operations has been carried out to date that does not fall victim to familiar conflations (Lobina, 2017); it is hoped that more will soon follow. Future work should attempt to identify the specific cognitive components that enable recursive operations in the visuospatial problem-solving domain probed here, as well investigate where else in the mind recursive and iterative operations take place.

As conditions for recursive operations are elucidated, some may aid in determining signatures of these operations in the human mind *per se*, rather than of recursive operations generally. That is, a memory load signature of predictable accumulations and resolutions of deferred operations is a signature of recursive operations irrespective of hardware and ought to be present for any kind of computational system executing such operations. The particular recursive operations that may be employed by the human mind certainly have their own properties above and beyond this general sort, some of which may serve as additional signatures, apart from what a formal understanding of recursion would dictate.

Given that the present results point to the existence of recursive operations in at least one system of the mind (underlying visuospatial cognition) shared with nonhuman animals, the question of whether such operations are found elsewhere in nature may be pursued. While a fair amount of work (most prominently, Gentner et al., 2006) has already attempted to address this question, it generally falls victim to the same structure–operation conflation discussed in the Introduction, as has been reviewed elsewhere (Lobina, 2017). It remains an exciting and unexplored avenue of investigation to apply methods similar to those used here—attempting to detect signatures in task performance—to nonhuman species, including avians and nonhuman primates. Such methods, if successful, would be

a first step toward establishing a phylogenetic history of algorithmic architectures in cognition.

Relatedly, the locus of the recursive operations pursued in the present work being outside of the language system, an important opportunity for the exploration of such operations in children is presented. Studies tracing the emergence of recursive operations during development and their timing with respect to other capacities' emergence could shed light on a number of issues, including the possibility that the recursive operations studied here may underpin performance in a variety of domains—a possibility often raised but, as discussed, still unresolved.

To the extent that such efforts in humans and nonhuman animals are successful—and, of course, to the extent that recursive operations truly exist in the mind, as the present work suggests—robust characterization of their properties, origins, and role in cognition will be a valuable step in integrating recursion into a broader computational–representational picture of the mind and its capacities.

A

Pre-Move Piece Order

Studies 1–6 used working memory load dynamics as evidence of the involvement of recursive operations during performance of the puzzle task. A complementary approach to gain insight into whether something like the recursive algorithm described above underlies task performance is to ascertain the order in which subjects attend to or think about pieces on the board prior to making their first move. That is, if human performance of the puzzle involves initiating self-calls and accumulating

deferred operations in a manner similar to the recursive solver, there should be observable similarities between the human and solver not just in memory load throughout the puzzle but also in mental content, specifically which piece a subject is thinking about at a given time. By probing human subjects to discover the order in which they think about pieces prior to making their first move, the degree of correspondence between solver and human pre-move preparation may be determined as an additional signature of recursive operations' involvement.

A.1 Study 7: Pre-Move Piece Order

A.1.1 Participants

Undergraduate and graduate students (N = 31) aged 18 through 35 were recruited for a half-hour-long lab session and compensated with either course credit or cash payment of $10.00.

A.1.2 Procedure

Experimental procedure was nearly identical to that described in Study 1, with the following differences: Subjects performed a set of 23 different puzzles (Fig. A.1), all of which were used for analyses and contained between six and ten gray pieces. In order to probe which piece subjects were thinking about, subjects were interrupted at random intervals (prior to first move: mean = 3 seconds; after first move: mean = 15 seconds) and asked to report "Which piece [they were] thinking about right now," which they indicated via trackpad click. Subjects were instructed that it did not matter how much or how little time they had been thinking about a piece they might select, that the piece did not need to be one they were planning on moving later on, and that the piece did not need to be the one they were currently looking at. The recursive solver generated 20,000 solutions to each puzzle, and, as in Study 3 and Study 4, the solution whose move sequence was most similar to a subject's sequence on a puzzle was chosen for analysis.

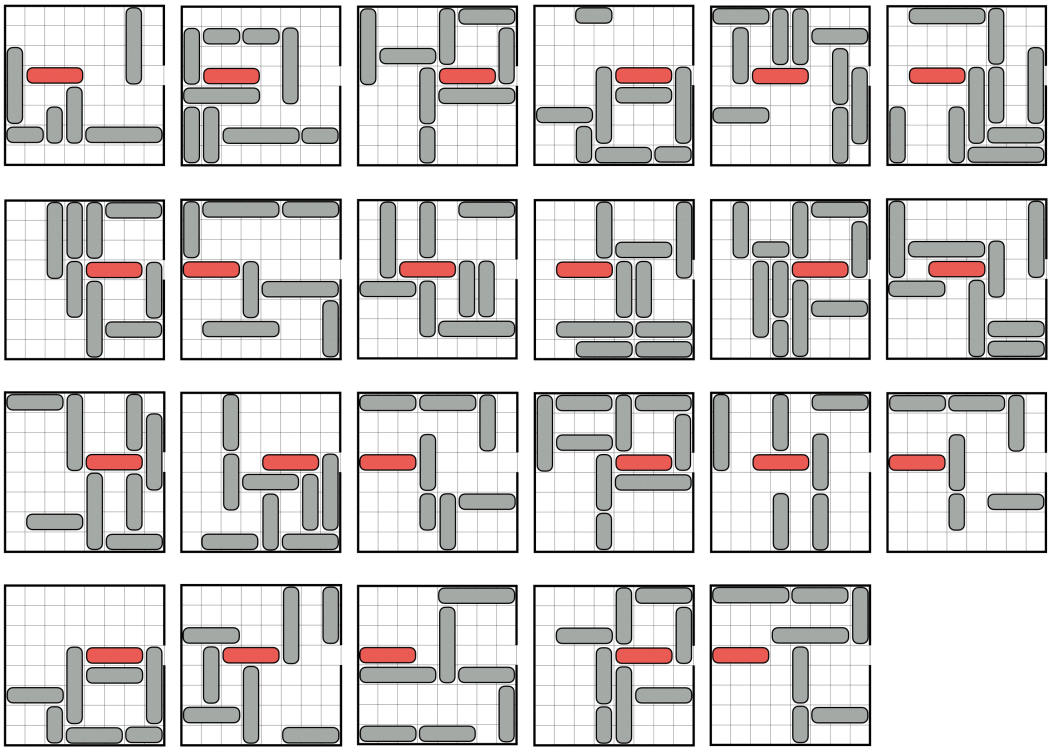


Figure A.1: The 23 puzzles used in Study 7 and Study 8.

For analysis, two data vectors were created. For the first vector, the time points of subjects' probe responses were transformed to reflect the proportion of time elapsed prior to the first move (probe responses after the first move had taken place were not analyzed). That is, if a subject was probed four seconds into the trial and she first moved a piece at ten seconds, the corresponding value in the first vector was 0.4. To create the second vector, a subject's response was assigned a value in proportion to how far along the sequence of solver-identified puzzle pieces it was. That is, if the solver model stored pieces in memory in the order piece c, piece d, piece b, piece a, and the subject indicated she was thinking about piece d, this corresponded to a value of 0.33, i.e., (target piece ordinal -1) / (number of pieces in sequence -1): $(2-1)/(4-1)$. Only pieces the solver assigned to variables prior to its first move were included in the analysis, and the piece that would eventually be moved first was also excluded to avoid biasing the analysis in favor of a positive result, as the solver would store this piece in memory just prior to moving it, and humans would almost certainly think of it immediately prior to moving it, as well. Values of the second vector were linearly regressed onto those of the first vector, with a significant positive slope taken as support for humans' use of recursive operations in the puzzle.

A.1.3 Results

A linear regression revealed a positive relationship ($p = 0.011$; adjusted $R^2 = 0.035$) between the order in which human subjects attended to pieces and the recursive solver identified and stored pieces in memory prior to the first move.

A.2 Study 8: Pre-Move Piece Order: Replication

A.2.1 Participants

Undergraduate and graduate students (N = 46) aged 18 through 35 were recruited for a half-hour-long lab session and compensated with either course credit or cash payment of $10.00.

A.2.2 Procedure

A replication of Study 7 that followed identical procedures was preregistered (AsPredicted #35521) and carried out.

A.2.3 Results

As in Study 7, a linear regression again revealed a positive relationship ($p = 0.0002$; adjusted $R^2 = 0.041$) between the order in which humans attended to pieces and the order the recursive solver stored them in memory prior to the first move.

A.3 Pre-Move Piece Order: Discussion

The correspondence between the order in which human subjects think about pieces and the recursive solver selects them prior to the first move of the puzzle serves as additional confirmation that the solver—consisting of little more than the ability to make moves and initiate self-calls—approximates well the way humans perform the puzzle, both in terms of their working memory usage and the pieces they attend to prior to making any moves.

B

Supplementary Figures

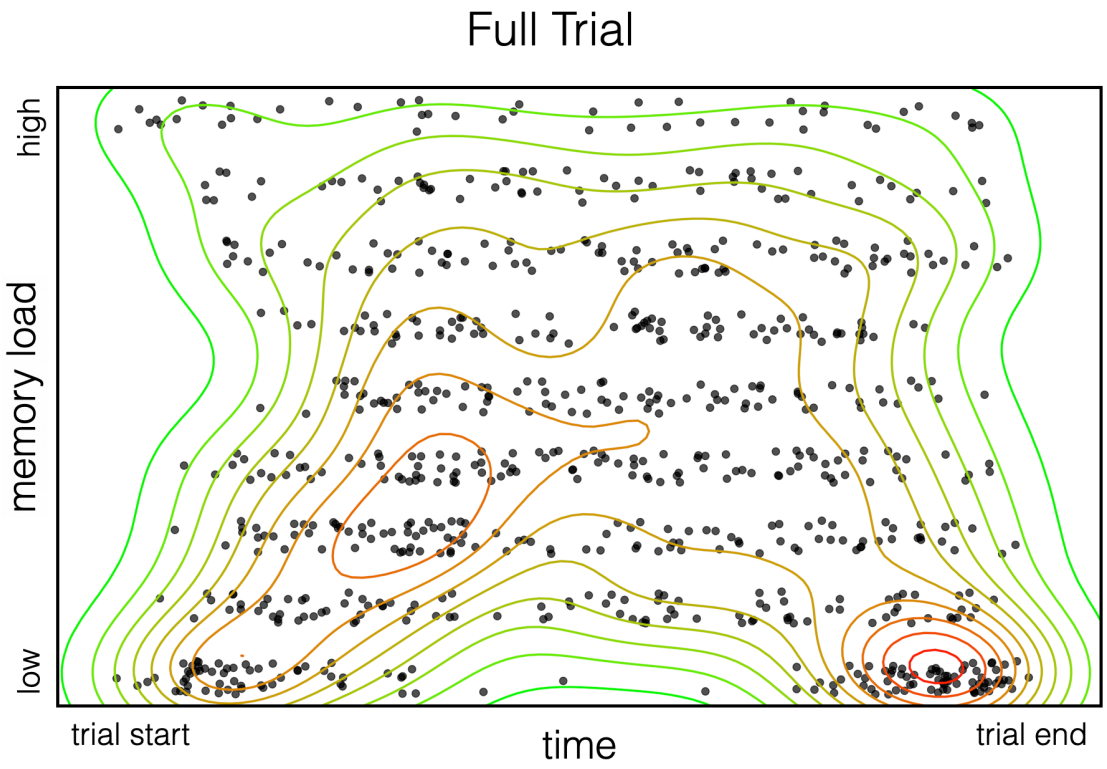


Figure S.1: From Study 1, 822 memory reports from 75 subjects across 18 different puzzles, with vertical jitter and colored density overlays to aid visual inspection. This data covers the entire trial period, from the beginning of a puzzle until its completion. Higher-density clusters of points are circumscribed by orange and red lines. The horizontal axis depicts time from the beginning to the end of a puzzle—that is, from when a particular puzzle first appeared on the screen to when subjects solved the puzzle by sliding the red piece off the board. The vertical axis depicts subjects' subjective memory reports, from low to high load.

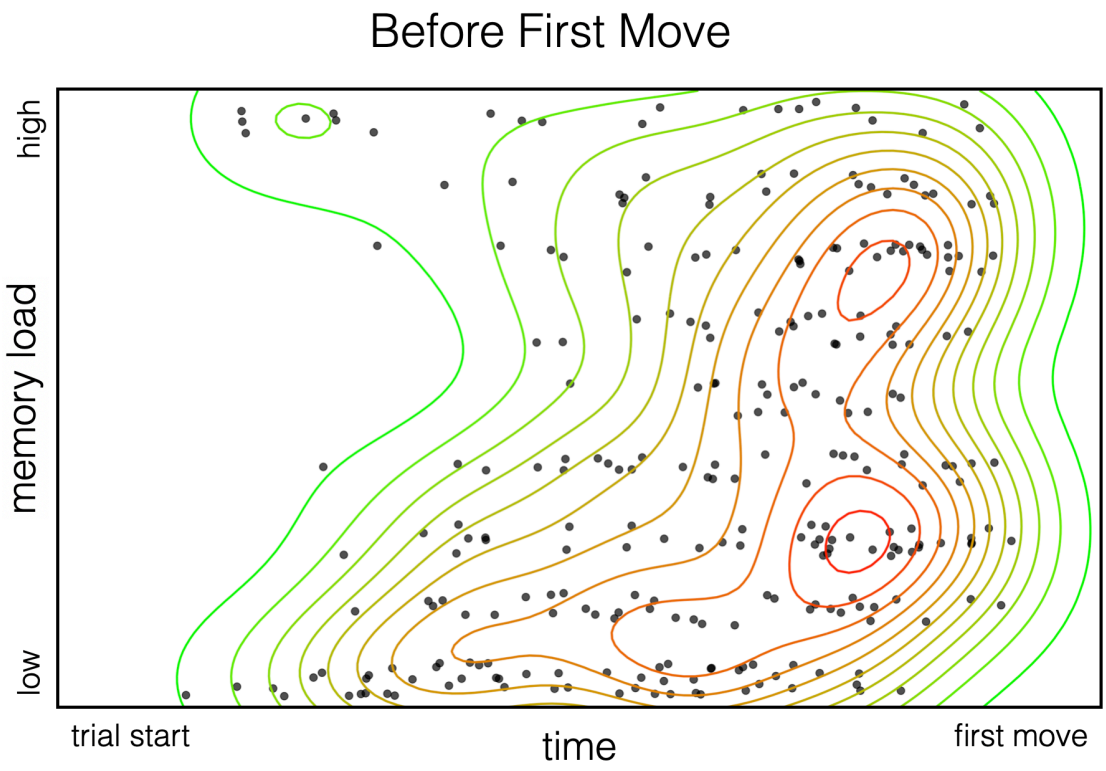


Figure S.2: From Study 1, 275 memory reports from 75 subjects across 18 different puzzles, with vertical jitter and colored density overlays to aid visual inspection. This data covers the beginning of the trial until just before subjects made their first move. Higher-density clusters of points are circumscribed by orange and red lines. The horizontal axis depicts time from the beginning to the end of a puzzle—that is, from when a particular puzzle first appeared on the screen to when subjects solved the puzzle by sliding the red piece off the board. The vertical axis depicts subjects' subjective memory reports, from low to high load.

Figure S.3: From Study 1, 547 memory reports from 75 subjects across 18 different puzzles, with vertical jitter and colored density overlays to aid visual inspection. This data covers the time from subjects' first move to the end of the trial. Higher-density clusters of points are circumscribed by orange and red lines. The horizontal axis depicts time from the beginning to the end of a puzzle—that is, from when a particular puzzle first appeared on the screen to when subjects solved the puzzle by sliding the red piece off the board. The vertical axis shows reported memory load from low to high.

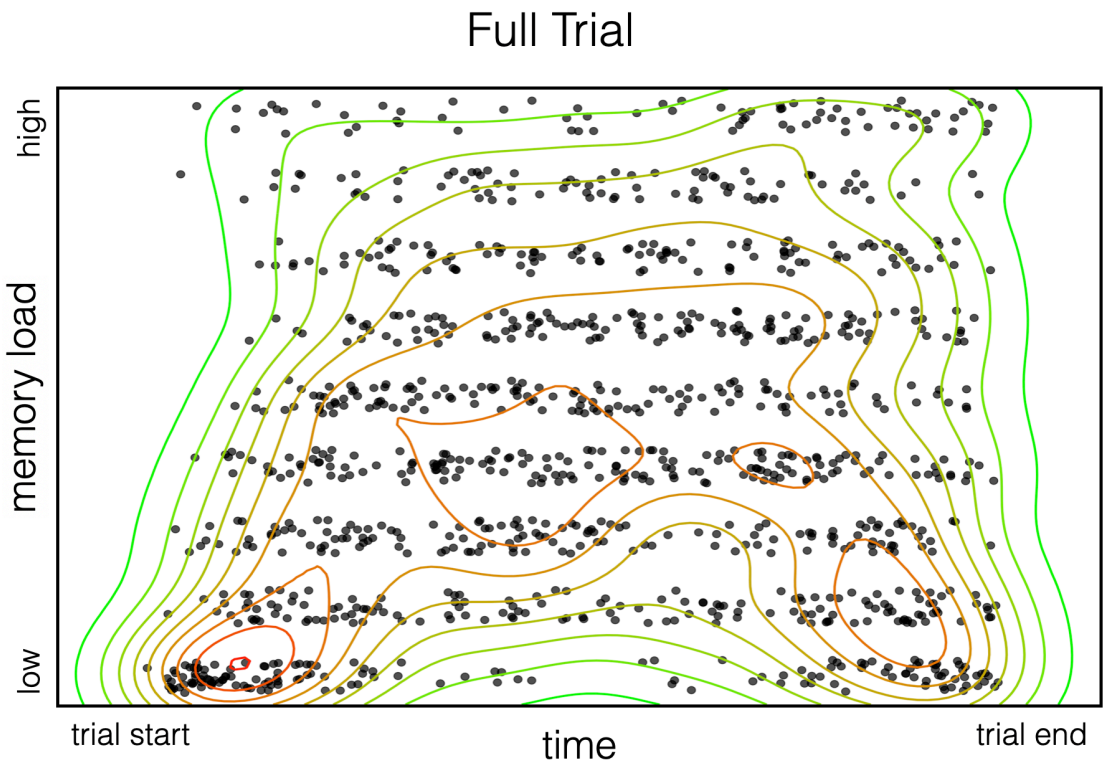


Figure S.4: From Study 2, 1,143 memory reports from 104 subjects across 18 different puzzles, with vertical jitter and colored density overlays to aid visual inspection. This data covers the entire trial period, from the beginning of a puzzle until its completion. Higher-density clusters of points are circumscribed by orange and red lines. The horizontal axis depicts time from the beginning to the end of a puzzle—that is, from when a particular puzzle first appeared on the screen to when subjects solved the puzzle by sliding the red piece off the board. The vertical axis shows reported memory load from low to high.

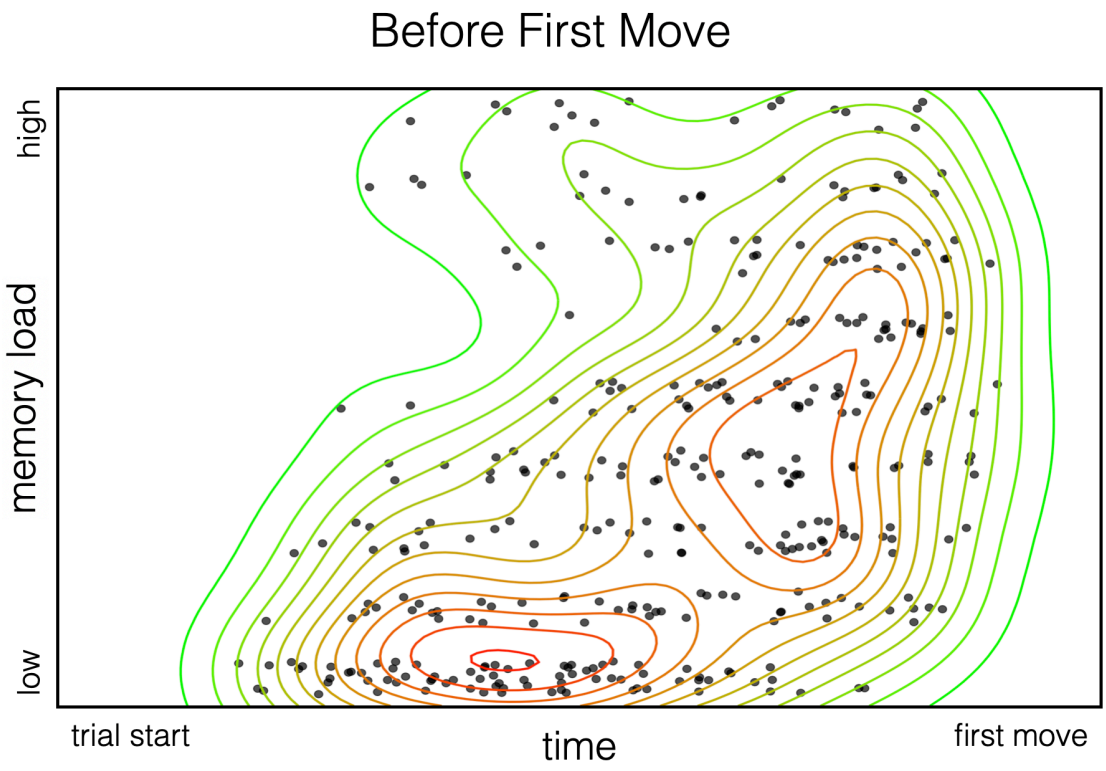


Figure S.5: From Study 2, 353 memory reports from 104 subjects across 18 different puzzles, with vertical jitter and colored density overlays to aid visual inspection. This data covers the beginning of the trial until just before subjects made their first move. Higher-density clusters of points are circumscribed by orange and red lines. The horizontal axis depicts time from the beginning to the end of a puzzle—that is, from when a particular puzzle first appeared on the screen to when subjects solved the puzzle by sliding the red piece off the board. The vertical axis shows reported memory load from low to high.

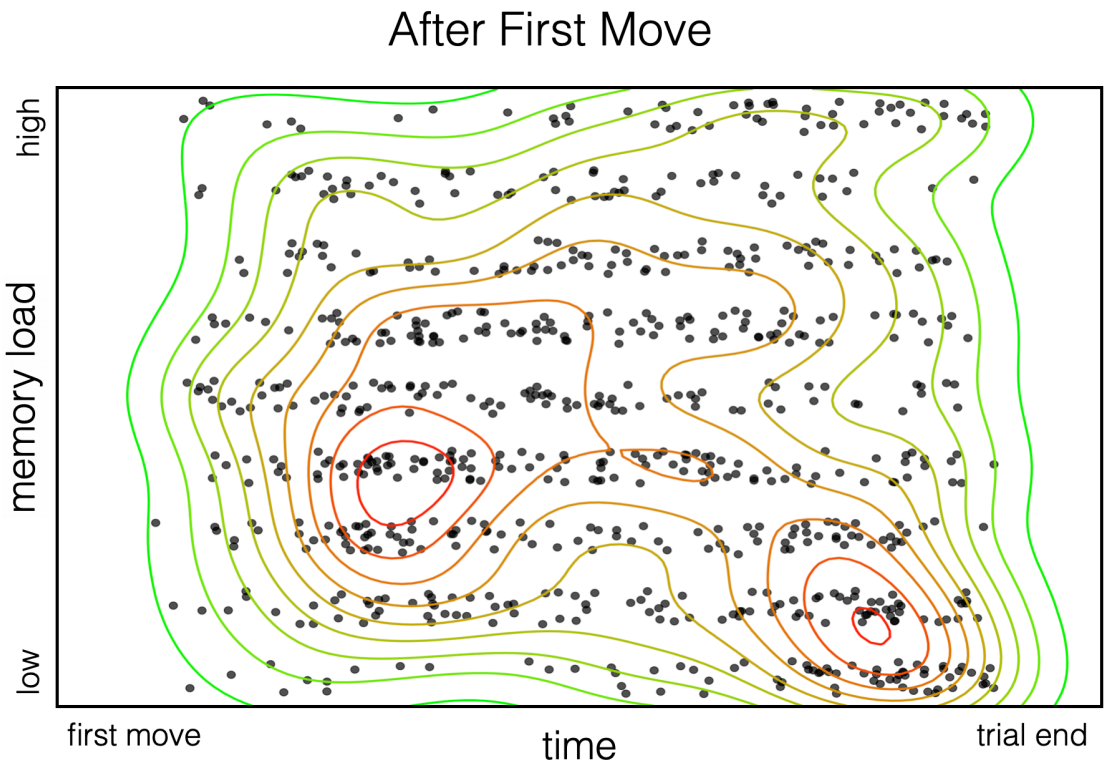


Figure S.6: From Study 2, 790 memory reports from 104 subjects across 18 different puzzles, with vertical jitter and colored density overlays to aid visual inspection. This data covers the time from subjects' first move to the end of the trial. Higher-density clusters of points are circumscribed by orange and red lines. The horizontal axis depicts time from the beginning to the end of a puzzle—that is, from when a particular puzzle first appeared on the screen to when subjects solved the puzzle by sliding the red piece off the board. The vertical axis shows reported memory load from low to high.

Bibliography

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 3, 113–124.

Chomsky, N. (1957). *Syntactic structures*. The Hague, Netherlands: Mouton Publishers.

Chomsky, N. (1975). *The logical structure of linguistic theory*. New York City, NY: Plenum Press.

Chomsky, N. (1984). *Modular approaches to the study of the mind*. San Diego, CA: San Diego State University Press.

Chomsky, N. (2000). Linguistics and brain science. In A. Marantz, Y. Miyashita, & W. O'Neil (Eds.), *Image, language, brain* (pp. 13–28). Cambridge, MA: The MIT Press.

Chomsky, N. (2017). Two notions of modularity. In R. G. D. Almeida & L. R. Gleitman (Eds.), *On concepts, modules, and language: Cognitive science at its core* (pp. 25–40). New York City, NY: Oxford University Press.

Church, A. (1951). A formulation of the logic of sense and denotation. In P. Henle, H. Kallen, & S. Langer (Eds.), *Structure, method, and meaning: Essays in honor of Henry M. Sheffer* (pp. 3–24). New York City, NY: The Liberal Arts Press.

Corballis, M. C. (2014). *The recursive mind: The origins of human, language, thought, and civilization*. Princeton, NJ: Princeton University Press.

Gentner, T. Q., Fenn, K. M., Margoliash, D., & Nusbaum, H. C. (2006). Recursive syntactic pattern learning by songbirds. *Nature*, 440, 1204–1207.

Hauser, M. (2009). Origin of the mind. *Scientific American*, 301, 44–51.

Hofstadter, D. R. (1985). *Metamagical themas: Questing for the essence of mind and pattern*. New York, NY: Basic Books.

Khemlani, S. & Johnson-Laird, P. N. (2007). Thinking about algorithms. In D. S. McNamara & J. G. Trafton (Eds.), *Proceedings of the 29th Annual Meeting of the Cognitive Science Society* (pp. 1786). Austin, TX: Cognitive Science Society.

Kleene, S. C. (1952). *Introduction to metamathematics*. Amsterdam, Netherlands: North Holland Publishing Co.

Liu, Y. A. & Stoller, S. D. (1999). From recursion to iteration: what are the optimizations? *ACM SIGPLAN Notices*, 34, 37–82.

Lobina, D. J. (2014a). Probing recursion. *Cognitive Processing*, 15, 435–450.

Lobina, D. J. (2014b). What linguists are talking about when talking about.... *Language Sciences*, 45, 56–70.

Lobina, D. J. (2017). *Recursion: A computational investigation into the representation and processing of language*. Oxford, United Kingdom: Oxford University Press.

Marr, D. (1977). Artificial intelligence: A personal view. *Artificial Intelligence*, 9, 37–48.

Martins, M. D. (2012). Distinctive signatures of recursion. *Philosophical Transactions of the Royal Society: B*, 367, 2055–2064.

Martins, M. J., Fischmeister, F. P., Puig-Waldmüller, E., Geißler, A., Robinson, S., Fitch, W. T., & Beinsteiner, R. (2014). Fractal image perception provides novel insights into hierarchical cognition. *NeuroImage*, 96, 300–308.

Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the structure of behavior*. New York City, NY: Henry Holt and Co.

Moschovakis, Y. N. (1998). On founding the theory of algorithms. In H. G. Dales & G. Oliveri (Eds.), *Truth in mathematics* (pp. 71–104). New York City, NY: Oxford University Press.

Newell, A. & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice–Hall.

Rice, H. G. (1965). Recursion and iteration. *Communications of the ACM*, 8, 114–115.

Roberts, E. (2006). *Thinking recursively with Java*. Hoboken, NJ: Wiley.

Rodgers, P. & Black, P. E. (2004). Recursive data structure. In V. Pieterse & P. E. Black (Eds.), *Dictionary of algorithms and data structures*. National Institute of Standards and Technology. xlinux.nist.gov/dads, accessed March 28th, 2020.

Simon, H. A. (1975). The functional equivalence of problem solving skills. *Cognitive Psychology*, 7, 268–288.

Simon, H. A. & Reed, S. K. (1976). Modeling strategy shifts in a problem-solving task. *Cognitive Psychology*, 8, 86–97.

Soare, R. (1996). Computability and recursion. *Bulletin of Symbolic Logic*, 2, 284–321.

Soare, R. (2009). Turing oracle machines, online computing, and three displacements in computability theory. *Annals of Pure and Applied Logic*, 160, 368–399.

Stabler, E. P. (2014). Recursion in grammar and performance. In T. Roeper & M. Speas (Eds.), *Recursion: Complexity in cognition* (pp. 159–177). London, United Kingdom: Springer International Publishing.

Xu, Y. & Corkin, S. (2001). H. M. revisits the Tower of Hanoi puzzle. *Neuropsychology*, 15, 69–79.