Running head: Mental simulations

Kinematic mental simulations in abduction and deduction

by

Sangeet S. Khemlani[a], Robert Mackiewicz[b], Monica Bucciarelli[c], and P.N. Johnson-Laird[d]

August 25th 2013

Classification BIOLOGICAL SCIENCES - Psychological and Cognitive Sciences

a. Corresponding author:

Navy Center for Applied Research in Artificial Intelligence, Naval Research Lab, Washington, DC 20375, Tel: +1 571 275 8107 Email: skhemlani@gmail.com

b. Department of Psychology, University of Social Sciences and Humanities, Chodakowska 19/31, 03-815 Warsaw, Poland. Email: robert.mackiewicz@swps.edu.pl

c. Centro di Scienza Cognitiva and Dipartimento di Psicologia, Università di Torino, Turin 10123, Italy. Email: monica.bucciarelli@unito.it

d. Department of Psychology, New York University, 6 Washington Place, New York, NY 10003. Email: phil@princeton.edu

Abstract

We present a theory, and its computer implementation, of how mental simulations underlie the abductions of informal algorithms and deductions from these algorithms. Three experiments tested the theory’s predictions, using a novel environment of a single railway track and a siding. This environment is akin to a universal Turing machine, but it is simple enough for non-programmers to use. They solved problems calling for them to use the siding to rearrange the order of cars in a train (Experiment 1). They abduced and described in their own words algorithms that solved such problems for trains of any length; and, as the use of simulation predicts, they favored while-loops over for-loops in their descriptions (Experiment 2). Given descriptions of loops of procedures, they deduced the consequences for given trains of six cars, doing so without access to the railway environment (Experiment 3). As the theory predicts, difficulty in rearranging trains depends on the numbers of moves and cars to be moved, whereas in formulating an algorithm and deducing its consequences it depends on the Kolmogorov complexity of the algorithm. Overall, the results corroborated the use of a kinematic mental model in creating and testing informal algorithms, and showed that individuals differ reliably in the ability to carry out these tasks.

Keywords: Abduction | Deduction | Informal programming | Mental simulation | Problem solving

Significance statement

We developed a theory of how mental simulations underlie the abductions of informal algorithms and deductions from these algorithms. Experiments tested the theory's predictions using a novel task environment. Participants solved problems; abduced and described in their own words algorithms that solved such problems; and deduced the consequences of algorithms. Difficulty in formulating the algorithm and deducing its consequences depended on the algorithm's Kolmogorov complexity. Results corroborated the use of a kinematic mental model in creating and testing informal algorithms, and showed that individuals differ reliably in the ability to carry out these tasks.

\body

At the root of much human thinking is the ability to make mental simulations, that is, to imagine a process step by step so that it unfolds in the mind in the same temporal order as the events in the actual process. This hypothesis is central to the theory of mental models (1-4). The theory explains how individuals reason, but in tasks such as syllogistic or conditional reasoning, rival theories offer alternative accounts (5, 6), and it is not easy to decide amongst them empirically (7). The aim of the present paper is accordingly to show that human reasoners use kinematic mental models in order to simulate events. This concept of mental models in simulations depends on three assumptions, which derive from the model theory (8).

1. The mental models in simulations are iconic, i.e., their structures correspond to the structures of what they represent (9). Hence, a model of a spatial layout is itself spatial, and so the relations between objects in the world are mirrored in the spatial relations between them in the model (10).

2. A kinematic model unfolds in time, and the sequence of situations that it represents corresponds to the temporal order of events in the world, real or imaginary (2, 11).

3. Mental models can be schematic and more parsimonious than visual images, which they underlie (1), because models need not represent the world from a particular point of view or represent all of its visual features (12). They represent what is common to many possibilities differing in details, and they yield faster inferences than images (13).

Some cognitive scientists are skeptical about the existence of any mental representations (14, 15); some emphasize the role of the environment in constraining, affording, or situating intelligent behavior (16, 17); some allow representations only in the form of syntactically-structured strings of symbols in a mental language (18); and some to the contrary allow representations only in sensory modalities (19). Our experiments were designed to illuminate these various ideas about representations.

The model theory postulates that the formulation of algorithms and computer programs depends on mental simulations. Computer programming calls for knowledge of programming languages, and so our studies focused on how naive individuals – those who knew nothing about programming – formulated algorithms in everyday language. Programs often depend on a loop of operations, e.g., *For each of the n elements in an input list, put that element at the head of the output.* This "for-loop" reverses the order of a list, such as (A B C). The first step places A at the head of an otherwise empty output, the second step puts B at the head of the output, and the third step puts C at the head of the output. The result is (C B A). The same reversal can be carried out with a "while-loop", e.g., *While the input list contains at least one item, put the item at the head of the input list to the head of the output.* While-loops are more powerful than for-loops, because only they can compute certain functions (20).

There have been investigations of deductions that call for a repeated loop of mental operations (21, 22) and of novice programmers' grasp of loops (23, 24). Studies of algorithmic thinking in non-programmers are rare, but they suggest that non-programmers tend not to make spontaneous use of loops (25-27).

In order to investigate the mental simulation of loops, we needed a task suitable for individuals with no knowledge of programming. We devised a simple computer environment of a toy train, which mimics a Turing machine (20), but which can be immediately grasped by naive participants including children. Unlike classical problems, such as the Tower of Hanoi (28) or missionaries and cannibals (29), the railway environment can be used to examine problems that differ in computational complexity (30) as we describe below. Fig. 1 presents the environment as it is shown on a computer screen. It consists of a railway track with a siding and labeled cars. Only three sorts of moves are possible: a move from left track to right track; from left track to siding, and from siding to left track.

We used the train environment to examine naive individuals' performance of three distinct sorts of task. *Problem solving* calls for individuals to rearrange a train, initially on left track, so that it is in a specified order on right track. *Abductive reasoning* yields explanations (31), and we enlarge the term to cover reasoning that yields algorithms. Our task calls for individuals to abduce algorithms that solve whole classes of rearrangements, such as an algorithm that reverses the order of a train of any length. *Deductive reasoning* calls for individuals to infer the consequences of an algorithm for a given train. In what follows, we describe the model theory of these three tasks, its computer implementation, and the results of three experiments that corroborate its predictions about the three tasks. Finally, we draw some general conclusions about mental representations and simulation.

The model theory of algorithms

In order to create an algorithm that solves any problem in a class of problems, the first step is to solve representative instances in the class. The second step is to use a simulation of the process of their solution in order to abduce an algorithm that solves any problem in the class. And, in order to test the algorithm's correctness, the third step is to use the algorithm itself, or to simulate it, to deduce its consequences for some new problems in the class. Each of these steps is a component of the model theory, and we have implemented each component in a computer program, *mAbducer* (for "model-based abducer", available at http://mentalmodels.princeton.edu/models/). We describe the theory of its three components in turn.

Problem solving. Although there are only three possible sorts of move in rearrangement problems (R – move one or more cars to right track; S – move one or more cars to siding; and L – move one or more cars to left track), trial and error soon leads to an explosion of possibilities. A problem such as the Tower of Hanoi can be solved using means-ends analysis in which one works backwards from the desired goal, invoking operations to reduce the difference between it and the current state (32). A Sudoku puzzle, however, cannot be solved using means-ends analysis, because by design it lacks a complete description of the goal (33). Rearrangement problems can be solved in a relatively unusual way, using a *partial* means-ends analysis, in which individuals decompose the goal, starting with the right-most car on the right track, and solve the problem of getting one or more adjacent cars into their required position in a piecemeal way.

The input to mAbducer is the starting state of the track and the required goal. It maintains a model of the current state of the track and of the goal, and it solves the problem in a psychologically plausible way. The kinematic model that it uses to represent the railway is highly schematic. For example, this model from a kinematic sequence

A[BA]BCC

represents the car, A, on left track, the cars BA on the siding as denoted by the square brackets, and the cars BCC on right track. The goal is represented as a single sequence of cars, which need to be on right track, with no cars on the siding or left track, e.g., [ ]AABBCC. The program, which implements a partial mean-ends analysis, matches cars on left track and the siding with those required to be on right track, updating the goal whenever at least one car is moved to right track until it solves the problem. Its output is a trace of the successful sequence of moves.

The sequences of moves in the program's solutions are intended to be psychological plausible. Hence, the relative difficulty of a problem should depend on the number of moves in the program's solution, and the mean number of operands per move. In a reversal problem, as the trace above shows, each move after the first one has an operand of a single car. We can contrast this case with the solution of a *palindrome* problem, such as the rearrangement from ABCCBA[ ] to [ ]AABBCC. We refer to the problem as a "palindrome", because when the input is a palindrome, as in this case, it is sorted into the order illustrated above. The program's solution calls for 6 moves and the total number of operands (moved cars) is 10, which is greater than the 7 operands for the reversal problem. Even though the two problems have the same number of moves, the theory therefore predicts that the palindrome should be more difficult to solve than the reversal. Number of operands has a family resemblance to "relational complexity", which concerns the number of arguments in a relation, and which affects the difficulty of solving problems (34). However, the number of operands concerns, not the number of arguments of an operator, but whether the value of a single argument is one or more cars. The two have in common that they increase the processing load on working memory. A corollary is that individuals should be likely to make unnecessary moves in their solutions, i.e., they should often fail to solve problems parsimoniously, because they move just one car instead of two or more.

An alternative theoretical approach is that solution depends instead, say, on a proof procedure, or on an algebraic manipulation (35). The difficulty of a problem is then likely to depend on the Levenshtein "edit" distance (36), i.e., the number of additions, deletions, or substitutions to get from the starting string of cars to the goal string of cars. This metric predicts the difficulty of certain deductive tasks (37).

Abductions of algorithms. Consider the task of formulating an algorithm for reversing a train of any length, i.e., given an input of a train of some arbitrary length, ABC…XYZ, the algorithm should yield: ZYX…CBA. A train with a small number of cars can be reversed with a small number of moves with no loops. But, the example calls for reversing trains of any length, and so a correct solution is bound to call for a loop of operations. The model theory postulates that individuals can nevertheless carry out the task. The process is abductive because it depends on creating an explanation of how to get from the input to the output (31). A putative solution can be tested using deduction, but it is not discovered by deduction alone – no more than is the discovery of a mathematical proof. According to the model theory, the creation of an algorithm depends on three steps, which are each modelled in the *mAbducer* program.

The program's first step is to simulate the solutions to two instances of the problem in order to avoid ambiguity. It makes the simulations using the process described above. Because each move concerns a set of one or more cars, which move together, the process parallels the piece-meal simulation of the workings of complex mechanisms (4).

Its second step is to recover the loop of moves, and any moves that have to be performed before or after the loop. The program finds the repeated sequences of at least two moves. But, what determines the number of iterations of the loop? Since the loop can be either a for-loop or else a while-loop, there are two ways to proceed. One way is to solve a pair of simultaneous linear equations to obtain the values of a and b in n = a*length + b, where n is the number of iterations of a for-loop, and length is the number of cars in the train. So, the two reversals above yield the values, 3 = 4a + b, and 4 = 5a + b, and the solution is that a = 1 and that b = -1. Hence, for a train of length 6, a for-loop can be constructed in which the number of iterations of the loop for a reversal, n, equals 1*6 -1 = 5. Another way to ensure that a loop is carried out for the required iterations is to determine the conditions under which a while-loop halts. A simulation shows that for a reversal the while-loop halts as soon as the siding is empty. Other sorts of problem have different halting conditions. They can be used in the description of a while-loop.

Next, *mAbducer* determines any moves that precede or follow the loop. In the present example, the loop is preceded by a move, S3 or S4, where the number of operands again depends on the length of the train, or in the simulation when there is only one car remaining on the left track. After the end of the loop of moves, a final R1 occurs. The loop in the present example is *static* in that the number of operands for the moves in the loop remains constant from one iteration to the next. In other rearrangement problems, including those that use two stacks for their solution, loops are *dynamic*, i.e., the number of cars in a move within a loop varies depending on the length of the train and on whether the loop is in its first iteration, its second iteration, and so on (see the faro shuffle in SI 1).

The program’s third step is to convert the structure of the solution, including the loop, into a verbal description of the algorithm. It translates both for-loops and while-loops into explicit descriptions in the programming language LISP (see SI 1 for the translations). It also translates while-loops into informal English.

The theory predicts that naive individuals use simulations to abduce algorithms, and so it should be easier for them to detect the halting conditions needed for while-loops than to solve the simultaneous equations needed for for-loops. They should therefore be biased to use while-loops. The prime difficulty in *solving* a problem is the number of moves and operands. But, the prime difficulty in *abducing* an algorithm should be the complexity of the algorithm itself. We used

Kolmogorov complexity as the relevant metric (38, 39), and we applied it to *mAbducer*'s while-loops, because of their psychological plausibility. We used the numbers of characters in its algorithms in Common Lisp (see SI 1), multiplied by the number of bits in a character (i.e., 7 for ASCII). The first three problems in Table 1 call for static loops, but the faro shuffle, which is the converse of the parity sort, calls for a dynamic loop. The faro shuffle of cards (also known as a 'riffle') has interesting mathematical properties relating to parallel computation and to the Fast Fourier transform (40). The four algorithms, which we used in our experiments, increase in complexity and in computational power – two stacks are needed to solve faro shuffles. But, Kolmogorov complexity is a simple general metric that captures this increase, which is otherwise hard to quantify.

Deductions from descriptions of algorithms. The final task that we investigated is to deduce the consequences of an algorithm. *mAbducer* carries out this procedure to check the algorithm that it has abduced. For a train of a new length, it simulates the consequences of the algorithm. An obvious sign of an erroneous algorithm is that it halts prior to solving the problem. This sort of error has not occurred with *mAbducer,* and so it is capable of automatic programming (for other methods, see 41-43). Suppose that naive individuals familiar with the railway environment have to deduce the consequences of the reversal algorithm for the train, ABCDEF. They should carry out this task by mentally simulating a sequence of operations. Of course, the task of imagining this sequence could be too difficult for most individuals without access to pencil and paper, and so one aim of our empirical research was to determine whether they could cope with it. The primary factor that should cause difficulty in such simulations, given that they are of comparable numbers of moves and operands, is the Kolmogorov complexity of the algorithms.

We have outlined the model theory, and its computer implementation, of how individuals solve rearrangement problems, how they use simulations to abduce algorithms to solve them, and how they use simulations of the algorithms to deduce their consequences. We now turn to empirical tests of the theory's predictions that number of moves and operands should determine the difficulty of solving problems, whereas Kolmogorov complexity should determine the difficulty of the abductions and deductions.

Experiment 1 Problem solving

The experiment examined the ability of 20 students to solve rearrangement problems – a prerequisite for the subsequent studies, because if individuals cannot solve these problems with reasonable efficiency, they can hardly devise algorithms for their solution. But, the experiment was also a test of the first component of *mAbducer* – its procedure for solving rearrangement problems. It uses a single algorithm to carry out a partial means-ends analysis in order to decide what move to make next, which may have one or more operands. The experiment allowed the participants to manipulate the trains (on a computer screen), and so they did not have to simulate the process of solution, but could carry out it directly. The aim was to determine whether naïve individuals could carry out the task, whether its difficulty depended on *mAbducer*'s numbers of moves and operands, and whether they tended to err in overlooking parsimonious moves. The problems were presented using a graphical interface on a computer, and consisted of all 24 possible rearrangements of trains containing four cars.

The important result was that naive individuals were able to solve these problems with ease. They produced very few incorrect solutions. We dropped the two extreme problems from the statistical analysis so that they would not bias the results, i.e., the problem that required only one move to solution, and the problem that had a total of 12 operands. The participants' mean number of moves to solve a problem increased with the *mAbducer*'s number of moves (Page's trend test, L = 1809.5, z = 8.47, p < .0001) and the mean number of moves also increased with *mAbducer*'s number of operands (Page's trend test, L = 276, z = 5.69, p < .0001; see SI 2 for means and additional analyses). In other words, as the number of operands increased so did the mean number of moves, independently of the number of moves in a *mAbducer*'s solution. The latency results likewise corroborated both of these effects. There was a reliable tendency for the participants to make redundant moves. Every participant made at least one redundant move, and we replicated this tendency in a follow-up experiment designed to elicit such errors. The main reason for redundant moves was perseveration. That is, when the participants moved a single car from siding to left track, they often overlooked the possibility of moving two cars together from left track to right track. The participants differed reliably in their ability to find parsimonious solutions (Friedman test, χ^2 = 45.05, p < .001), and the best participant made a mean of 5.63 moves over all the problems, and the worst participant made a mean of 7.54 moves over all the problems. After the end of the experiment proper, the participants had to think aloud as they tackled two further problems, and their protocols corroborated the use of a partial means-ends analysis in which they focused on the successive parts of the goal rather than the goal as a whole.

Experiment 2 Abduction of algorithms

The experiment examined the model theory of how naïve individuals abduce informal algorithms that solve rearrangement problems. They should rely on mental simulations of solutions of the problems. The experiment accordingly tested three empirical predictions. First, algorithms to solve rearrangements of trains of 8 cars should be easier to create than those for trains of any length. The former do not require loops of operations and so they should be simpler to deal with than the latter. Second, the difficulty in formulating algorithms should depend on their Kolmogorov complexity, not on metrics such as edit distance or number of moves (see Table 1 above). Third, if participants use mental simulation, then they should be biased in favor of while-loops rather than for-loops, because they can observe the condition on the track when a while-loop ends, whereas the abduction of a for-loop calls for mental arithmetic to solve simultaneous equations. The experiment examined the three sorts of problem with static loops, namely, reversals, palindromes, and parity sorts, which call for loops with a constant number of operands in their instructions (see SI 3). The 20 participants, who were not programmers, first solved five practice problems (different from those in the experiment) using the railway environment. The environment was then switched off, and they had to create algorithms for solving the three sorts of problem either for trains of 8 cars or for trains of any length. The problems of these two sorts were presented in separate blocks in two counterbalanced orders to make a total of 6 trials. The participants wrote their algorithms in everyday language, and here is a typical example of a participant's correct algorithm for a reversal of trains of any length: *Move all cars to the right of A to the side. Then move A to the right. Shift B to left, then right. Shift C to left, then right...repeat until pattern is reached.* It is based on a while-loop (for other examples of informal algorithms, see SI 3). Because solutions were near ceiling for the 8 car trains (92% correct), Fig. 2 presents the percentages of correct algorithms and the times the participants took to produce them (whether correct or not) only for trains of any length. The results corroborated the three predictions of the model theory. First, it was easier to formulate algorithms for trains of 8 cars (92% correct) than for trains of any length (52% correct, Wilcoxon test, $z = 3.29$, $p < .001$). Second, the three sorts of rearrangement yielded the predicted trend in accuracy, i.e., reversals (90% correct), palindromes (70% correct), and parity-sorts (63% correct, Page’s trend test, L = 256.5, z = 2.60, p < .005). Participants created accurate algorithms more often when they tackled 8 car trains in the first block than when they tackled trains of any length in the first block (82% vs. 65%, Mann-Whitney test, z = 1.70, p < .05). However, there was a three-way interaction (Mann-Whitney test, z = 1.94, p < .05) in that 8 car problems were close to ceiling regardless of block or sort of problem, whereas algorithms for trains of any length were affected by both variables. Once again, the latencies showed the same pattern of results (see SI 4). Third, analyses of the algorithms revealed that the participants used reliably more while-loops than for-loops. For trains of 8 cars, 61% of correct algorithms embodied loops (38% while-loops and 23% for-loops). For trains of any length, correct solutions were bound to use loops (82% while-loops and 18% for-loops). These data are based on the 18 participants who formulated at least one correct algorithm for trains of any length; 12 of them used more while-loops than for-loops and there were 3 ties (Binomial test, p < .02). The bias towards while-loops was greater for trains of any length (Wilcoxon test, z = 2.4, p < .01). The use of while-loops had a reliable correlation with accuracy (r = .43, p < .005), whereas the use of for-loops tended towards a negative correlation with accuracy (r = -.26, p = .09). Finally, the participants, who knew nothing about programming, differed overall in their ability to formulate correct algorithms (Friedman non-parametric analysis of variance, χ^2 = 35.96, p = .01). The best participant was correct on every problem, whereas the worst participant was correct for less than 20% of the problems.

Experiment 3 Deduction from algorithms

The model theory postulates that when naïve individuals deduce the consequences of carrying out an algorithm on a particular train, they rely on simulating the sequence of the algorithm's operations. Hence, according to the theory, the difficulty of the task should depend, not on the number of moves to be carried out, but on the Kolmogorov complexity of the algorithm. The experiment tested this prediction using while-loops for all four sorts of problem in Table 1, i.e., reversals, palindromes, parity sorts, and faro shuffles. Each of them, however, was described in exactly the same number of words. The participants, who were not programmers, first watched a movie that explained and illustrated the railway environment. They then had no access to this environment for the deduction task, and they were not allowed to write anything down. After two simple practice problems, they had to deduce the consequences of the descriptions of algorithms on a given train of 6 cars. They did the task twice for each of the four sorts of algorithm, once with trains labelled with letters and once with trains labelled with numbers. The descriptions of the algorithms were in Polish, the native language of the participants, and they were not the minimal descriptions in Table 1, but were rewritten to be as clear as possible and to contain the same number of words (see SI 5).

The percentages of correct deductions for the 43 participants who produced at least one complete answer corroborated the model theory's predictions. The participants were correct for 41% of reversals, 35% of palindromes, 32% of parity sorts, and 23% of faro shuffles (Page's L test, $z = 1.94$, $p < .03$). The latencies of correct deductions also supported this trend for those participants who were correct on at least one deduction of each sort, i.e., 77 s for reversals, 130 s for palindromes, 106 s for parity sorts, and 151s for faro problems. The means are slightly misleading because the stochastic increase in latencies for individual participants corroborated the predicted trend in a highly reliable way (Page's L, $z = 3.55$, $p < .0005$). The number of moves in the simulations, the number of operands, or the edit distance (see Table 1) cannot explain the trends in accuracy and latency. The participants differed overall in their ability to make correct deductions (Friedman non-parametric analysis of variance, $\chi^2 = 17.29$, $p < .001$). The best participants got all eight problems correct; the worst got none of them correct.

General Discussion

In reasoning, the mind is fallible about both logical and probabilistic conclusions (44-46), but it has a striking ability to make mental simulations. They can be static mental models or kinematic sequences of them in which the sequences represent temporal orders (11). The model theory that we outlined in this article, and its computer implementation in *mAbducer*, show how such simulations can underlie the abduction of algorithms and the deduction of their consequences – at least in the case of a seemingly simple environment of toy trains. The environment is easy to understand and, skeptics may say, too easy because its problems are as narrow and trivial as toy trains. In fact, unlike, say, syllogistic inferences (7), the number of rearrangement problems is unbounded, and some of them call for considerable computational power. Faro shuffles (43), as illustrated in Table 1, call for the use of two stacks so that a car shifted from the siding to left track has to be shifted back to the siding again. The computational power needed here – two stacks – exceeds the power embodied in a well-known conjecture about the syntax of natural languages (47).

Individuals readily solve problems in the railway domain when they manipulate the cars on the track. The difficulty of solving these problems, as Experiment 1 showed, depends on *mAbducer*'s number of moves in a solution, but also independently on the number of cars in these moves. Participants often overlooked parsimonious moves of more than one car at a time. In the experiment, they did not have to simulate the moves, because they could use the track itself.

The ability to solve problems is a prerequisite for abducing algorithms for their solution. The *mAbducer* program depends on simulating solutions using schematic models that it updates kinematically. Given that a loop of operations has to be repeated, it formulates a while-loop from its observations of the halting condition in the simulations. The program can also describe a for-loop and determine the number of times that the loop should iterate from its solution of a pair of simultaneous equations. The task of abducing algorithms is difficult, and at first we doubted whether naive individuals would be able to perform it, because previous studies of informal programming showed that they avoided the use of loops (25-27). But, without access to the railway environment, as Experiment 2 showed, they were able to simulate loops of operations, to figure out what was going on in them, and to describe them in informal algorithms. The participants had the predicted bias towards while-loops rather than for-loops. Likewise, the difficulty of the four sorts of rearrangement depended, not on the numbers of moves or cars to be moved, but on the Kolmogorov complexity of the Lisp algorithms that *mAbducer* creates (see Table 1).

Prudent programmers debug their code by deducing its consequences for specific inputs. This task also provided evidence for the role of simulation. With no access to the railway environment and without being allowed to write anything down, naive individuals in Experiment 3 were able to infer the results of carrying out the four sorts of algorithm on trains containing six cars. As the theory predicts, the difficulty of making the deductions depended, not on numbers of moves or cars to be moved, but on the complexity of the algorithms, which varied from reversing the order of cars to the more complex faro shuffle (see Table 1).

The evidence we have reported supports the theory of the simulation using kinematic mental models. It provides a unified account of the abduction of algorithms and the deductions of their consequences. As far as we know, no other theory of naive reasoning about algorithms exists. Probabilities hardly enter the process and so Bayesian theories of reasoning may be irrelevant (5). But, a theory could be developed from an axiomatization of the railway domain in logic (6). The difficulties for this approach are to frame a complete set of axioms in a way that captures both what changes and what does not change with each move (48), and to ensure that the resulting system makes the correct predictions about human performance.

As we mentioned in the Introduction, psychologists hold almost all possible views about mental representations, from the claim that they are not needed for intelligent behavior (16) to the competing views that they are either abstract strings of symbols (18) or rooted in sensory modalities (19). Our results seem impossible to explain without invoking mental representations, and, most plausibly, kinematic models with an iconic structure that corresponds to the railway environment. These models may be mapped into visual images or they may be as abstract as they are in *mAbducer* (see 4, 12). Individuals can reason from models without forming visual images from them, and evidence suggests that images impede reasoning (13). Of course, it does not follow that all reasoning depends on simulating the world: a person can learn to use formal rules of inference. Likewise, it does not follow that all mental representations are iconic models (49). The model theory itself relies on another sort of representation to capture the meaning of an assertion, which it then uses to construct models (50).

Mathematicians, logicians, and computer programmers reason about the repeated loops of operations in algorithms. Previous studies have examined how novice programmers try to formulate such algorithms in a programming language (e.g., 23-27). But, as computer scientists often complain, no valid test exists to predict the ability of naive individuals as computer programmers (51). The results show that individuals differ reliably in their ability to abduce informal algorithms and to deduce the consequences of these algorithms. It remains to be seen whether such tasks, which depend on mental simulation, are reliable predictors of ability in programming. But, the evidence corroborates the theory that naïve individuals use mental simulations to create informal algorithms, even those containing loops of operations, and to infer their consequences.

Methods

Experiment 1. Twenty undergraduate students at Princeton University served as participants (mean age of 19.7 years), and none had had any prior training in logic or computer science. The participants were tested individually, and carried out the experiment on a PC running LispWorks 4.4. They interacted with the system using the mouse and the keyboard of the computer. They were shown a three-minute instructional video that guided them through the elements of the railway environment, and that presented the instructions. The problems showed the initial state with the cars on the left track, and the required goal state with the cars on the right track. The participants made moves using a mouse to control a graphical interface. The key instruction stated that they should try to solve each problem with as few moves as possible. They acted as their own controls and carried out all 24 problems, which were presented in a different random order to each of them.

Experiment 2. Twenty participants from the same population as before were tested individually. The session began with five practice problems akin to those in Experiment 1, which the participants had to solve by interacting with the railway system. These problems were unrelated to the experimental problems, and each of them used a train of 6 cars with a solution of 8 moves. The experiment proper followed, and the participants' task was to type out a procedure that would solve each problem, but they could not interact with the railway environment or write anything down. They carried out two blocks of trials, one with problems for trains of 8 cars and one with problems for trains of any length, i.e., a total of 6 trials. The blocks were presented in a counterbalanced order to two groups of participants. The order of the three sorts of rearrangement was randomized for each participant within each block. For the problems with trains of any length, the participants were told that a car containing an ellipsis stood in place for any number of cars that had the same pattern. They were free to use their own words in any way that they wanted. Two independent judges – one of the authors and a research assistant – scored the informal algorithms in terms of whether were correct or incorrect, and whether they contained a while-loop or a for-loop. The two judges agreed 93% about the accuracy of the algorithms (111 out of 120 problems, Cohen's κ = .82). The judges agreed 83% about the nature of the loops in the algorithms (99 out of 120 problems, Cohen's κ = .73). A third independent judge resolved the discrepant evaluations in both cases.

Experiment 3. Fifty-four undergraduate psychology students from Warsaw University of Social Science and Humanities took part in the experiment (mean age 21.6 years), and because logic is obligatory in most Polish universities, over half of them had taken at least one course in logic. Twenty-two participants were paid a small sum (equivalent to $2) for participating in the experiment, and the rest took part in exchange for course credit. This difference had no reliable effect on either of the dependent variables, and so we pooled the data from those two conditions. Each participant carried out two versions of the reversal, palindrome, parity, and faro problems. One version had cars labelled with letters, and one version had cars labelled with numbers. Each description of an informal algorithm started and ended with the same phrases, and each description contained 109 words in Polish (see SI 4 for the original descriptions and translations into English). The descriptions were presented in one of eight counterbalanced orders allocated at random to the participants. The experiment was presented on a computer screen and the students typed in their answers. They were instructed not to type their response until they knew the position of all six cars on the right track, and they were not allowed to write anything down.

Acknowledgments

We thank Sam Glucksberg, Adele Goldberg, Geoffrey Goodwin, Louis Lee, David Lobina, Max Lotstein, Paula Rubio, and Carlos Santamaría, for their advice. Our research was supported by a National Science Foundation Graduate Research Fellowship to the first author, by the Polish Ministry of Science and Higher Education Grant No. 2836/01/E/560/S/2012 to the second author, by the Italian MIUR Grant No. 2010RP5RNM to the third author to study problem solving and decision making, and by National Science Foundation Grant No. SES 0844851 to the fourth author to study deductive and probabilistic reasoning.

References

Shepard RN, Metzler J (1971) Mental rotation of three-dimensional objects. *Science* 171: 701-703.

Johnson-Laird PN (1983) *Mental Models* (Cambridge University Press, Cambridge).

Bower GH, Morrow DG (1990) Mental models in narrative comprehension. *Science* 247: 44-48.

Hegarty M (2004) Mechanical reasoning as mental simulation. *Trends Cogn Sci* 8: 280- 285.

Oaksford M, Chater N (2007) *Bayesian Rationality: The Probabilistic Approach to Human Reasoning* (Oxford University Press, New York).

Rips LJ (1994) *The Psychology of Proof* (MIT, Cambridge, MA).

Khemlani S, Johnson-Laird PN (2012) Theories of the syllogism: A meta-analysis. *Psychol Bulletin* 138: 427-457.

Johnson-Laird PN (2010) Mental models and human reasoning. *Proc Natl Acad Sci USA* 107: 18243–18250.

Peirce CS (1931-1958) *Collected Papers of Charles Sanders Peirce*. Vol 4, eds Hartshorne C, Weiss P, Burks A (Harvard University Press, Cambridge, MA).

Johnson-Laird PN, Byrne RMJ (1991) *Deduction* (Erlbaum, Hillsdale NJ).

Schaeken WS, Johnson-Laird PN, d'Ydewalle G (1996) Mental models and temporal reasoning. *Cogn* 60: 205-234.

Hegarty M, Stieff M, Dixon BL (2013) Cognitive change in mental models with experience in the domain of organic chemistry. *J Cogn Psychol* 25: 220-228.

Knauff M, Fangmeier T, Ruff CC, Johnson-Laird PN (2003) Reasoning, models, and images: Behavioral measures and cortical activity. *J Cogn Neurosci* 4: 559-573.

Margolis E, Laurence S (2007) The ontology of concepts—abstract objects or mental representations? *Noûs* 41: 561-593.

Ramsey WM (2007) *Representation Reconsidered*. (MIT, Cambridge, MA).

Brooks R (1991) Intelligence without representation. *Artif Intell* 47: 139-160.

Thelen E, Smith LB (1994) *A Dynamic Systems Approach to the Development of Cognition and Action*. (MIT, Cambridge, MA).

Pylyshyn Z (2003) Return of the mental image: Are there really pictures in the brain? *Trends Cognit Sci* 7: 113–118.

Barsalou, LW (2008) In *Embodied Grounding: Social, Cognitive, Affective, and Neuroscientific Approaches*, eds. Semin GR, Smith ER (Cambridge University Press, New York), pp. 9–42.

Rogers H (1967) *Theory of Recursive Functions and Effective Computability*. (McGraw-Hill, New York).

Cherubini P, Johnson-Laird PN (2004) Does everyone love everyone? The psychology of iterative reasoning. *Thinking & Reasoning* 10: 31-53.

Mazzocco K, Cherubini AM, Cherubini P (2013) On the short horizon of spontaneous iterative reasoning in logical puzzles and games. *Organ Behav Hum Dec Process* in press.

Kurland DM, Pea, RD (1985) Children's mental models of recursive LOGO programs. *J Educat Comp Res* 1: 235-244.

Anderson, J R, Pirolli P, Farrell R (1988) In *The Nature of Expertise,* eds. Chi M, Glaser R, Farr M (Erlbaum, Hillsdale NJ), pp 153-183.

Miller L (1974) Programming by non-programmers. *Int J Man-Mach Stud* 6: 237-260.

Miller L (1981) Natural language programming: Styles, strategies, and contrasts. *IBM Sys J* 20: 184-215.

Pane JF, Ratanamahatana CA, Myers BA (2001) Studying the language and structure in non-programmers' solutions to programming problems. *Int J Human-Comp Stud* 54: 237-264.

Simon HA (1975) The functional equivalence of problem-solving skills. *Cognit Psychol* 7: 268-288.

Simon HA, Reed SK (1976) Modeling strategy shifts in a problem-solving task. *Cognit Psychol* 8: 86-97.

Hopcroft JE, Ullman JD (1979) *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA).

Peirce CS (1955) *Philosophical Writings of Peirce*, ed Buchler J (Dover, New York).

Newell A (1990) *Unified Theories of Cognition* (Harvard University Press, Cambridge, MA).

Lee NYL, Goodwin GP, Johnson-Laird PN (2008) The psychological problem of Sudoku. *Thinking & Reasoning* 14: 342-364.

Halford GS, Wilson WH, Phillips S (1998) Processing capacity defined by relational complexity: Implications for comparative, developmental, and cognitive psychology. *Behav Brain Sci* 21: 803-865.

Anderson JR, Betts S, Ferris JL, Fincham JM (2011) Cognitive and metacognitive activity in mathematical problem solving: prefrontal and parietal patterns. *Cogn Affective Behav Neurosci* 11: 52-67.

Levenshtein V (1966) Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10: 707–710. (Originally published in Russian in: *Doklady Akademii Nauk SSSR* 163: 845–848, 1965.)

Ragni M, Khemlani S, Johnson-Laird PN (2013) The evaluation of the consistency of quantified assertions. *Mem & Cog*: in press.

Li M, Vitányi P (1997) *An Introduction to Kolmogorov Complexity and Its Applications,* 2nd ed (Springer-Verlag, New York).

Chater N, Vitányi P (2003) Simplicity: a unifying principle in cognitive science? *Trends in Cognit Sci* 7: 19-22.

Diaconis P, Graham RL, Kantor WM (1983) The mathematics of perfect shuffles. *Adv Appl Math* 4: 175–196.

Koza JR (1994) *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT, Cambridge, MA).

Flener P, Yilmaz S (1999) Inductive synthesis of recursive logic programs: Achievements and prospects. *J Logic Program* 41: 141-195.

Gulwani, S. (2010). Dimensions in program synthesis. In *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming* (pp. 13-24). Hagenberg, Austria.

Johnson-Laird PN (2006) *How We Reason* (Oxford University Press, New York)

Nickerson RS (2008). *Aspects of rationality* (Psychology Press, New York).

Khemlani S, Lotstein M, Johnson-Laird PN (2012) The probability of unique events. *PLOS-ONE* 7: 1-9.

Gazdar G (1981) On syntactic categories. *Phil Trans Royal Soc London*, B 295: 267-283.

McCarthy J (1986) Applications of circumscription to formalizing common-sense knowledge. *Artif Intellig* 28: 89-116.

Khemlani S, Orenes I, Johnson-Laird PN (2012) Negation. *J Cogn Psychol* 24: 541-559.

Khemlani S, Johnson-Laird PN (2012). The processes of inference. *Argument and Computation*, 1–17, iFirst.

Bornat, R, Dehnadi S, Simon (2008) Mental models, consistency and programming aptitude. *Proc Tenth Austral Comp Educ Conf* 10: 53–61.

Table 1. Examples of four sorts of rearrangements, the informal algorithms for trains of any length that *mAbducer* discovered from simulating solutions, the total number of moves for each example of 6 cars, their mean number of operands, their edit distance, and the Kolmogorov complexities of the Lisp functions for re-arranging trains of any length.

Rearrangements of ABCDEF	mAbducer's informal algorithms	No. of moves	Mean no. operands	Edit distance	Kolmogorov complexity
Reversal yields: FEDCBA	Move one less than cars to siding. While there are more than zero cars on siding move one car to right track move one car to left track. Move one car to right track.	12	1.3	6	1288
Palindrome yields: AFBECD	Move one less than half the cars to siding. While there are more than two cars on left track move two cars to right track move one car to left track. Move two cars to right track.	6	1.6	4	1295
Parity sort yields: ACEBDF	While there are more than two cars on left track move one car to right track move one car to siding. Move one car to right track. Move one less than half the cars to left track. Move half the cars to right track.	7	1.4	4	1519
Faro shuffle yields: ADBECF	Set the number of operands to be moved, n-of-s, to one less than half the cars. Set the decrement to one. While n-of-s is more than zero, move one car to right track, move n-of-s cars to siding, move one car to right track, move n-of-s cars to left track, take decrement from n-of-s. Move two cars to right track.	9	1.3	4	1771

Fig 1.

The railway environment with an example of an initial configuration in which a set of cars is on the left side (A) of the track, the siding (B) can hold one or more cars while other cars are moved to the right side of the track (C). The program allows individuals to select a car, e.g., the highlighted "E" car, and to move it and all the cars in front of it to the siding or right track.

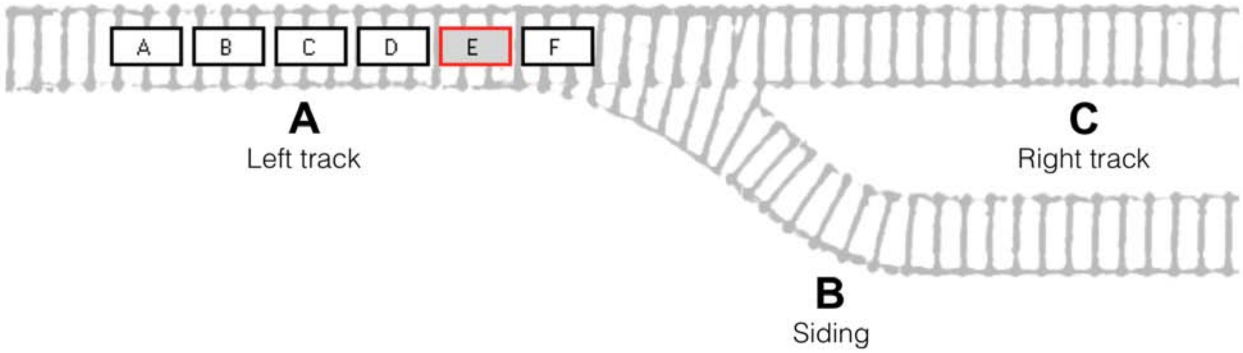


Fig 2. The proportions of correct algorithms in Experiment 2 for trains of any length depending on the sort of rearrangement and whether the participants carried out problems of trains of any length in the first block (A) or the second one (B).

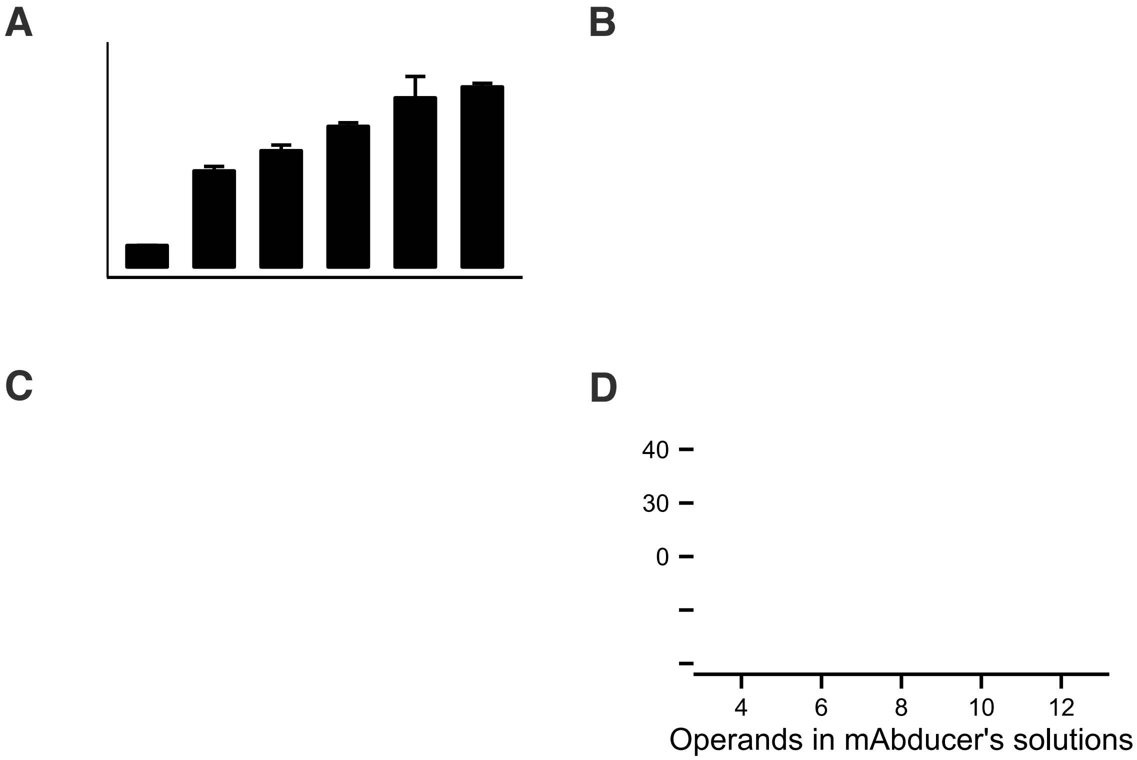
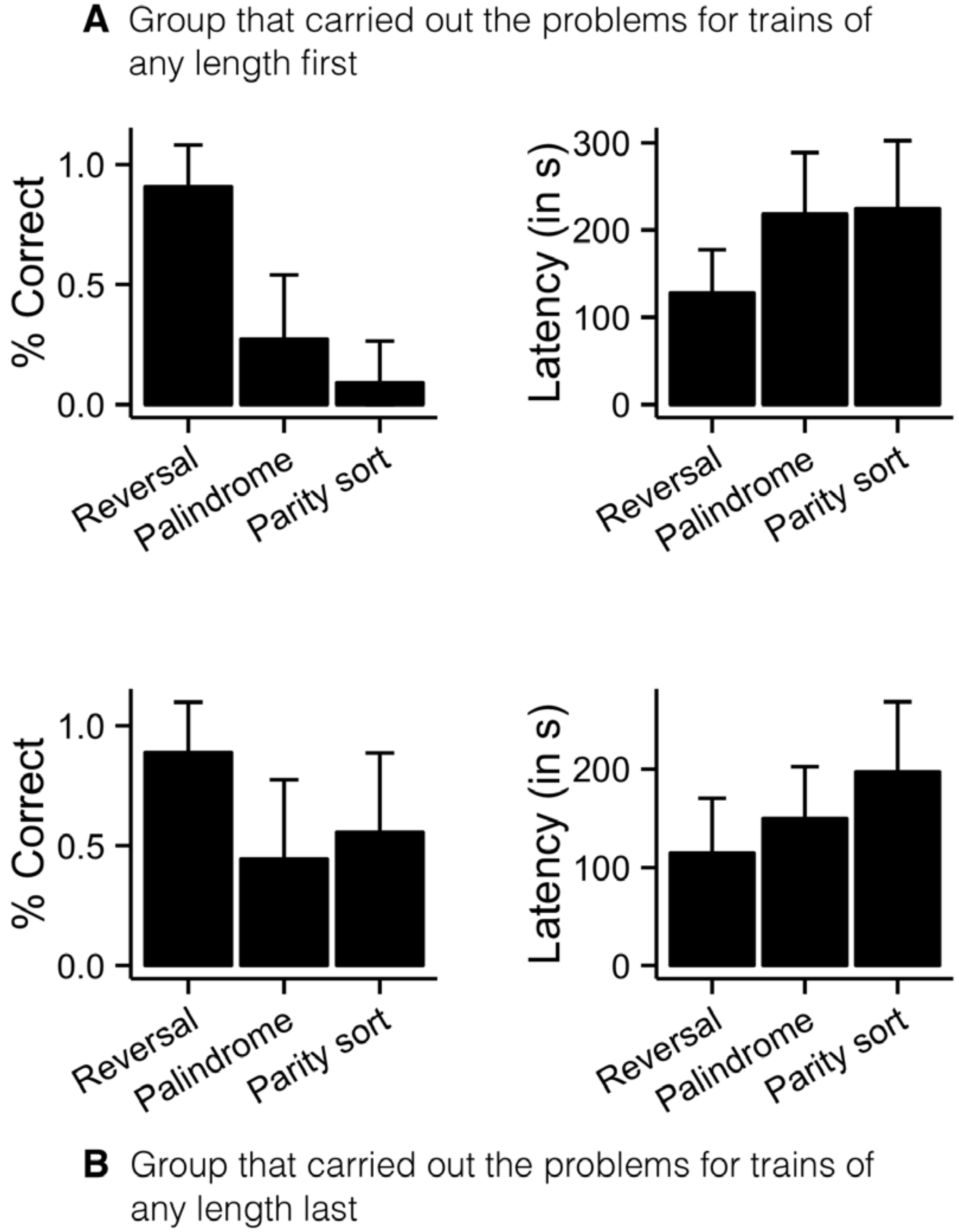


Fig. 3. The percentages of correct algorithms for trains of any length and their mean latencies (in s) depending on the sort of rearrangement, and the order of the two blocks.


A Group that carried out the problems for trains of any length first
% Correct
Reversal
Palindrome
Parity sort
Latency (in s)
Reversal
Palindrome
Parity sort
% Correct
Reversal
Palindrome
Parity sort
Latency (in s)
Reversal
Palindrome
Parity sort
B Group that carried out the problems for trains of any length last