On Evaluating Human Problem Solving of Computationally Hard Problems

Sarah Carruthers[1] *and Ulrike Stege*[1]

Abstract

This article is concerned with how computer science, and more exactly computational complexity theory, can inform cognitive science. In particular, we suggest factors to be taken into account when investigating how people deal with computational hardness. This discussion will address the two upper levels of Marr's Level Theory: the computational level and the algorithmic level. Our reasons for believing that humans indeed deal with hard cognitive functions are threefold: (1) Several computationally hard functions are suggested in the literature, e.g., in the areas of visual search, visual perception and analogical reasoning, linguistic processing, and decision making. (2) People appear to be attracted to computationally hard recreational puzzles and games. Examples of hard puzzles include Sudoku, Minesweeper, and the 15-Puzzle. (3) A number of research articles in the area of human problem solving suggest that humans are capable of solving hard computational problems, like the Euclidean Traveling Salesperson Problem, quickly and near-optimally.

This article gives a brief introduction to some theories and foundations of complexity theory and motivates the use of computationally hard problems in human problem solving with a short survey of known results of human performance, a review of some computationally hard games and puzzles, and the connection between complexity theory and models of cognitive functions. We aim to illuminate the role that computer science, in particular complexity theory, can play in the study of human problem solving. Theoretical computer science can provide a wealth of interesting problems for human study, but it can also help to provide deep insight into these problems. In particular, we discuss the role that computer science can play when choosing computational problems for study and designing experiments to investigate human performance. Finally, we enumerate issues and pitfalls that can arise when choosing computationally hard problems as the subject of study, in turn motivating some interesting potential future lines of study. The pitfalls addressed include: choice of presentation and representation of problem instances, evaluation of problem comprehension, and the role of cognitive support in experiments. Our goal is not to exhaustively list all the ways in which these choices may impact experimental studies, but rather to provide a few simple examples in order to highlight possible pitfalls.

Keywords

problem solving, computational complexity, intractability, algorithmic level theories, cognitive functions

[1]University of Victoria. Please direct correspondence to scarruth@uvic.ca.

http://dx.doi.org/10.7771/1932-6246.1152

Introduction

A common assumption when attempting to gain insight into the power of the cognitive system is that of Marr's Level Theory (Marr, 1982; see, e.g., Horgan & Tienson, 1996; Millgram, 1982; Pylyshyn, 1984; Rumelhart, MacClelland, & PDP Research Group, 1986; van Rooij, 2003, 2008). The cognitive system is thought of performing cognitive tasks that we assume are computational tasks (Massaro & Cowan, 1993) and are described by cognitive functions. Cognitive processes, in turn, are the mechanisms that realize the cognitive functions in the cognitive system. Marr's theory distinguishes three levels: the computational level that contains the descriptions of the cognitive functions, the algorithmic or representational level that describes the algorithm[1] (or set of algorithmic steps) that solves[2] the functions, and the implementation or physical level that is concerned with the realization of these algorithms. These three levels are often considered independently, as it is possible to know the cognitive functions of the cognitive system without necessarily knowing which algorithm or algorithms solve the functions—as often many solutions for a computational problem[3] exist. Similarly, the algorithms may have different realizations on the implementation level. Marr (1982) stresses the importance of the computational level, because understanding the nature of the computational task will more likely lead to an understanding of an algorithm that will solve it. According to the Church-Turing Thesis (Turing, 1936) we postulate that all cognitive functions are computable[4]. Additionally, we require cognitive functions to satisfy the Tractable Cognition Thesis, that is to also be tractable by the human mind (Frixione, 2001; van Rooij, 2003; 2008).

This article is concerned with how computer science, and more exactly computational complexity theory, can inform cognitive science: (1) when studying the power of the cognitive system, that is what properties cognitive functions—besides computability—may and may not possess and (2) when studying how people go about solving computable functions. In particular, we discuss how to investigate how, if at all, people deal with computational hardness. Thus, our discussion will concentrate on the two upper levels of Marr's Level Theory.

Our reasons for believing that humans indeed deal with hard cognitive functions are threefold. First, several computationally hard functions are suggested in the literature, for example, in the areas of visual search (Tsotsos, 1988; 1989; 1990; 1991), visual perception and analogical reasoning (Thagard, 2000), linguistic processing (Wareham, 1996; 1998) and decision making (van Rooij, 2003; van Rooij, Stege, & Kadlec, 2005). A second reason is the prevalence of recreational puzzles and games that have been shown to be computationally hard in their general cases. Examples of such puzzles include Sudoku, Minesweeper and the 15-Puzzle. Third, a number of research articles in the area of human problem solving suggest that humans are capable of solving hard

computational problems, like the Euclidean Traveling Salesperson Problem, quickly and near-optimally (Dry, Lee, Vickers, & Hughes, 2006; Dry, Preiss, & Wagemans, 2012; Graham, Joshi, & Pizlo, 2000).

In this article we aim to illuminate the role that computer science, in particular complexity theory, can play in the study of human problem solving. Theoretical computer science can provide a wealth of interesting problems for human study, but it can also help to provide deep insight into these problems. There is much potential for collaboration between computer scientists and cognitive psychologists, and we hope to contribute to this collaboration. In particular, we discuss the role that computer science can play when choosing computational problems for study and designing experiments to investigate human performance.

The remainder of this article is outlined as follows. We begin with an introduction to some theories and foundations of complexity theory. We then motivate the use of computationally hard problems in human problem solving with a short survey of known results of human performance, a review of some computationally hard games and puzzles, and the connection between complexity theory and models of cognitive functions. Finally, we enumerate issues and pitfalls that can arise when choosing computationally hard problems as the subject of study, in turn motivating some interesting potential future lines of study.

Computer Science Terminology and Foundations

In the following section we begin by defining a number of terms and concepts from computer science, to serve as a foundation for the remains of this paper.

Definition of a Computational Problem

Computational problems are typically defined by, and for, computer scientists. However, the following definition of a computational problem equally applies if the problem solver is a person, a group, an algorithm, a combination of person and computational device, or a cognitive function (van Rooij, 2003).

A computational problem is an input-output mapping, and is specified as follows:

Problem Name
Input: Describes a given instance of the problem, from a general class of problems.
Output: Describes the required output for any given instance that satisfies the input specification.

Many computational problems can be classified into one of three classes: search, decision, or optimization problems[5]. Problems of this type are the focus of this article. These three different classes have slightly different input-output descriptions. In a search problem, the problem solver is asked to output a solution to the problem if one exists, a decision version asks whether or not a solution exists, and an optimization problem,

like the search problem, asks for a solution to the problem. Here, however, the solution is optimized (that is, either minimized or maximized).

As an example, we consider the above mentioned different problem descriptions for the computational problem Vertex Cover. We first define the optimization version of the problem, often called Minimum Vertex Cover (Garey & Johnson, 1979):

Minimum Vertex Cover / Vertex Cover (optimization version)
Input: A graph $G = (E, V)$, with vertex set V, and edge set E.
Output: A smallest *vertex cover* for G, that is, a smallest subset U of V, such that every edge in E is incident to at least one vertex in U.

The search version of the problem is as follows:

Vertex Cover (search version)
Input: A graph $G = (E, V)$, with vertex set V, and edge set E. An integer value k.
Output: A vertex cover for G of size at most k, that is a subset U of V of size at most k, such that every edge in E is incident to at least one vertex in U, if such subset exists. Otherwise output "no solution exists."

Finally, we define the decision version of the problem:

Vertex Cover (decision version)
Input: A graph $G = (E, V)$, with vertex set V, and edge set E. An integer k.
Output: The *YES*/*NO*-answer to the question: Does there exist a vertex cover of size at most k? In other words, does there exist a subset U of V of size at most k, such that every edge in E is incident to at least one vertex in U?

Hard Computational Problems

In computer science, it is desirable—for a specific computational problem—to develop an algorithm that solves the computational problem efficiently. That is, one would like to obtain, from the algorithm or computer program, a correct output for any given instance in a short amount of time. Generally, computational problems that can be solved by an algorithm that runs in polynomial time in terms of the input size, are viewed—by computer scientists—as efficiently solvable (Arora & Barak, 2007). The complexity class of computational problems that are solvable in polynomial time is called *P*. Computational problems that cannot, or for which it assumed that they cannot, be solved in polynomial time in terms of their input sizes, are considered—in the light of classical complexity theory—computationally hard[6]. In this article, we will consistently refer to problems that are diagnosed as such by a classical complexity analysis hard problems.

There is a large number of computational problems for which it is unknown whether or not they belong to *P*. The most famous kind of computationally hard problems are those that are *NP*-complete (Garey & Johnson, 1979). Such problems belong to the class *NP* and are also hard for *NP*. The class *NP* is the class of computational decision problems

for which certificates of YES-instances are verifiable in polynomial time. It is certainly true that all decision problems that are members of *P* are also members of *NP*, and therefore $P \subseteq NP$, but it is unknown whether $P = NP$ or $P \neq NP$. Finding an algorithm that solves one *NP*-complete problem in polynomial time would prove the existence of polynomial time algorithms for each problem in *NP*, and would therefore prove that $P = NP$. Most computer scientists and mathematicians conjecture that $P \neq NP$[7].

To show *NP*-hardness for a decision problem *D*, it is sufficient to find a polynomial-time reduction[8] from a known *NP*-complete problem *A*, to *D*. While *P* and *NP* make up the most widely known complexity classes, there are many other complexity classes that, should a decision problem be proven hardness for, imply intractability for the problem (Arora & Barak, 2007). Examples include: co-*NP*[9] and *PSPACE* (Arora & Barak, 2007).

Note that, for an intractable computational problem, many instances can still be efficiently solvable, despite the property that there is no algorithm that can solve *every* instance efficiently (in fact, there is an infinite number of such instances for each algorithm).

Problem Instances

A computational problem, as defined above, can be thought of as a set of problem instances (short: instance), and a set of corresponding solutions (that is, at least one solution for each instance, if the problem is solvable). A specific problem instance is given as input of the problem, as defined above. It is critical to differentiate between an instance of a problem and the problem itself, in particular because an algorithm that correctly solves a particular instance of a problem (or even a set of instances) will not necessarily yield a correct solution to all instances of the same problem. In contrast, an algorithm that correctly solves the computational problem will always yield a correct solution to any instance of the problem.

Plausible Algorithmic Level Explanations for Computationally Hard Cognitive Functions

When a cognitive function or a computational problem is diagnosed as computationally hard, then we typically expect that there does not exist an algorithm that solves the function in polynomial time. This naturally constrains the number of instances that can be optimally solved efficiently, as we require cognitive functions to be tractable. A number of cognitive scientists suggest that computationally hard cognitive functions invoke heuristics[10] as algorithmic level explanation (e.g., Millgram, 2000; Thagard & Verbeurgt, 1998). Their suggestion coincides with the call by Garey and Johnson (1979), namely, that when dealing with *NP*-hard computational problems to focus on other approaches.

In their recent *Synthese* article, van Rooij, Wright, and Wareham (2012) argue, however, that a heuristic for a specific computational problem cannot serve as an algorithmic level

explanation for the corresponding cognitive function, since the heuristic does not compute the given function, but a different one. Another suggestion to deal with computational hardness is that of using approximation algorithms[11] as a problem solving strategy. Also here, van Rooij et al. (2012) argue that, if this was the case, the computational function solved would be a different one: namely one that allows for approximate solutions.

Some researchers reject computational problems as cognitive functions if they are computationally hard: For example, Frixione (2001) postulates the *P*-Cognition Thesis that requires every cognitive function to be computable in polynomial time. van Rooij (2003; 2008) suggests, instead of rejecting computationally hard functions in their entirety, a refined analysis of the computational problem. Her interpretation of the Tractable Cognition Thesis is the *FPT*-Cognition Thesis: cognitive functions must have a parameterization[12] that is a member of the parameterized complexity class *FPT*[13].

Motivation

Computationally hard problems are a subject of interest in cognitive science and human problem solving for several reasons: (1) In the literature, computationally hard problems have been proposed as models of cognitive functions. (2) People seem to have a genuine interest in puzzles and games that are shown to be computationally hard. (3) A number of computationally hard problems have been used to investigate human performance; people appear to be good at solving those.

Computational Problems as Models of Cognitive Functions

In the literature, several computationally hard problems are suggested as models for cognitive functions (van Rooij, 2003). Examples, among others, are the computational problems Coherence, and Subset Choice. According to Thagard (2000), the Coherence problem asks to maximize logical consistence for a given set of propositions connected by positive and negative constraints by partitioning the propositions into accepted and rejected propositions. The coherence value of this partitioned network or graph is obtained by adding all weights of all satisfied constraints, that is, positive constraints that are both incident to accepted propositions or both incident to rejected propositions, and negative constraints that are incident to each one accepted and one rejected proposition.

The problem Subset Choice, introduced in van Rooij (2003), is defined for hypergraphs, a generalization of a graph in which an edge can connect any number of vertices.

We formally define these two problems.

Coherence (decision version)
Input: A simple undirected graph[14] $G = (P, C)$, where C is partitioned into C^+ and C^-, that is $C = (C^+ \cup C^-)$ and $C^+ \cap C^- = \varnothing$. For each constraint $pq \in C$ there is an associated positive integer weight $w(pq)$; a positive integer c. Here, P denotes the set of vertices or propositions,

and C denotes the set of edges or constraints. In particular, constraints that belong to C^+ are called *positive constraints*; constraints that belong to C^- are called *negative constraints*. *Output:* Can P be partitioned into A and R such that $Coh_G(A, R) \geq c$? Here, $Coh_G(A, R) = \sum_{pq \in S_G(A, R)} w(pq)$. $S_G(A, R)$ denotes the graph G induced by the partition of P with respect to the *satisfied* positive and negative constraints of G, that is $S_G(A, R) = (P, C_{sat})$ with satisfied constraint set $C_{sat} = \{xy \in C^+ | x, y \in A \text{ or } x, y \in R\} \cup \{xy \in C^- | x \in A \text{ and } y \in R\}$.

Subset Choice (decision version)
Input: A weighted hypergraph[15] $H = (V, E)$, $E \subseteq \bigcup_{2 \leq h \leq |V|} V^h$; for every $v \in V$ a weight $w_V(v) \in \mathbb{Z}$; for every $e \in E$ a weight $w_E(e) \in \mathbb{Z} \setminus \{0\}$; a positive integer p
Output: Does there exist a subset $V' \subseteq V$ such that value$(V') = \sum_{x \in V'} w_V(x) + \sum_{e \in (V' \times V') \cap E} w_E(e) \geq p$?

Coherence is suggested as a model for scientific reasoning, legal justification, social judgement, as well as visual perception (see, e.g., Thagard, 2000). Coherence as defined above is known to be *NP*-complete (Thagard & Verbeurgt, 1998). Subset choice, also *NP*-complete, models decision making scenarios, for example, in medical applications as well as management and consumer choice (van Rooij, 2003; van Rooij et al., 2005).

Known Hard Problems We Play for Fun

Many people choose to play—recreationally—games that are in their general form computationally hard. These include many one-player games—often also called puzzles. Before we list some examples, we note that, for these hardness results, the decision question asked is typically of the form: does there exist a solution to the instance given? We set out that this differs from what people typically encounter when playing; puzzles when presented to a human are typically solvable. Their experience will show that a solution typically exists, but the problem they have to solve is to find such a solution. It may still be that for a puzzle of general size—with a promised existing solution—it is computationally hard to find such a solution. Minesweeper is an example of such a problem: It is *NP*-complete to determine whether or not—for a given minesweeper board of general size—a solution exists (Kaye, 2000). The problem of uncovering the (hidden) information of a cell on a given minesweeper board is also computationally hard (in fact, this problem is co-*NP*-complete). In this latter version, we assume the existence of a solution to the minesweeper board (Scott, Stege, & van Rooij, 2011). We will pick up on this topic in the last section of this article.

The decision versions of the following generalized puzzles, asking for the existence of a solution, are shown to be *NP*-complete. We assume the reader's familiarity with the typically played instances of these famous puzzles. For a survey on *NP*-complete puzzles see (Kendall, Parkes, & Spoerer, 2008).

- Instant Insanity (Alt, Bodlaender, van Kreveld, Rote, & Tel, 2007; Robertson & Munro, 1978)

- Sudoku (Yato & Seta, 2003)
- Slitherlink (Yato, 2000; Yato & Seta, 2003)
- Battleship (Sevenster, 2004)
- Mastermind (Stuckman & Zhang, 2006)

Further, it is *NP*-complete to determine whether there exists a solution to the generalized 15-Puzzle, namely *n*-Puzzle in at most *k* moves (Ratner & Warmuth, 1990). The general, optimization version of this problem is defined here.

n-Puzzle (optimization version)
Input: For *n*, an integer, such that $n+1$ has an integer square root, a set of *n* tiles, numbered $1 \ldots n$ arranged in a $(\sqrt{n+1} \times \sqrt{n+1})$-grid, with one square left open.
Output: A sequence of moves that results in the ordered tiles in the $(\sqrt{n+1} \times \sqrt{n+1})$-grid, such that the number of moves is minimized. Legal moves are limited to sliding a tile into the single empty location.

Several versions of Tetris are shown to be *NP*-complete (Demaine, Hohenberger, & Liben-Nowell, 2008), including the decision questions: whether or not there exists a sequence of trajectories that eliminates *k* rows, maximizing the number of 4-row eliminations, maximizing the number of deleted rows, and maximizing the number of pieces placed. A generalized decision version of RushHour that asks whether or not a solution exists has been shown to be *PSPACE*-complete, a higher level of complexity than *NP*-complete. For the following problems, the version of the problem shown to be *NP*-complete is that of reaching the goal state from the start state, generalized over map sizes: Super Mario Bros., Donkey Kong, Legend of Zelda, and Pokemon (Aloupis, Demaine, & Guo, 2012).

The apparent addictive nature of some of these games and puzzles is also interesting to note. At this point it is not clear if the addictiveness is a cause, result, or not at all related to their complexity. In reality, the versions of these games and puzzles that are played are constrained in some manner, often limited to a fixed and finite size. However, human interest in these still raises some interesting open questions.

To the best of our knowledge, human performance has been investigated on only a few of these games and puzzles, namely: *n*-Puzzle, and Minesweeper. The study of human performance on *n*-Puzzle investigated individuals' ability to judge distance and direction from an intermediate state to goal state, as well as general performance (Pizlo & Li, 2005). Findings indicate that distance is hard to judge, however direction may be easier to judge. Performance in terms of solution-path length and solution time varied greatly both between subjects, and within. Recent work on human performance on Minesweeper puzzles indicates that human performance is not close to optimal. In this pilot study participants had low success (measured in terms of wins) rates, between 10% and 35% on (8x8)-instances of the puzzle with 10 mines (Walker, 2010).

Human Performance Results for Computationally Hard Problems

The study of human performance on computational problems known to be hard has focused, to date, on variations of the Euclidean Traveling Salesperson Problem (E-TSP). However, a handful of other problems have been investigated, including: Generalized Steiner Tree problem (Burns, Lee, & Vickers, 2006), and the Minimum Vertex Cover Problem (Carruthers, Masson, & Stege, 2012).

The E-TSP asks, given a set of points *P* in the Euclidean plane, for a shortest tour (a path starting and ending at the same point) that visits every point of *P* in the plane. Variations that have been studied in the context of human performance include: E-TSP with obstacles (Haxhimusa, Kropatsch, Pizlo, Ion, & Lehrbaum, 2006), TSP using a city-block configuration (Walwyn & Navarro, 2011), and 3D E-TSP (Haxhimusa et al., 2011). The problem has been presented visually on paper (MacGregor & Ormerod, 1996; MacGregor, Ormerod, & Chronicle, 1999; MacGregor, Chronicle, & Ormerod, 2004; van Rooij, Stege, & Schactman, 2003; Tak, Plaisier, & van Rooij, 2008; Walwyn & Navarro, 2011) and on a computer screen (Best & Herbert, 2003; Dry et al., 2006; Dry et al., 2012; Graham et al., 2000; Kong & Schunn, 2007; Ormerod & Chronicle, 1999; Saalweachter & Pizlo, 2008), virtually using a simulator, and in a real-world interactive environment using objects placed on a floor (Haxhimusa et al., 2011). In general, studies claim that human solutions are close to optimal (Graham et al., 2000; Ormerod and Chronicle, 1999; van Rooij, Schactman, Kadlec, & Stege, 2006; Walwyn & Navarro, 2011), taking approximately linear (in terms of instance size) time to complete (Dry et al., 2006; Dry et al., 2012; Graham et al., 2000). A number of models have been proposed to explain human performance, including: a nearest-neighbour heuristic (Best & Herbert, 2003), a convex-hull heuristic (MacGregor & Ormerod, 1996; MacGregor, Ormerod, & Chronicle, 2000), and hierarchical and pyramid models (Graham et al., 2000; Haxhimusa et al., 2006). One possible explanation for the quality and speed of human performance on instances of the E-TSP is that humans are able to leverage perceptual processes when searching for a solution (van Rooij et al., 2006).

Preliminary work has been done on the human performance on the Minimum Vertex Cover Problem (Carruthers et al., 2012), investigating the quality of human performance on a problem where perceptual processes are less likely to be useful in the same way they are on E-TSP. Results of this study focused on properties of instances of Minimum Vertex Cover that impact human performance, and what strategies participants adopt when tackling instances. Participants' solution quality appears to be impacted by the neatness of graphs, and not by the number of crossings. Further, participants appear to be able to identify and apply two optimal strategies. A study of the Generalized Steiner Tree Problem looked at possible correlations between performance on this, and other computational problems, and general intelligence measures. No specific performance results were discussed (Burns et al., 2006).

Designing Human Problem Solving Experiments

When designing human experiments involving computationally hard problems, a number of factors should be carefully considered. There is a large number of computationally hard problems, every one of which could potentially serve as a basis of human study. Each of these computational problems (regardless of its complexity) can be presented and represented in more than one manner, and further can typically be posed as a search, decision or optimization problem. The choice of problem, question, presentation, and representation may impact human performance, and as such should be carefully considered.

The goal, in this paper, is not to exhaustively list all the ways in which these choices may impact performance measures. Rather, we provide a few simple examples that we hope highlight possible pitfalls. Each computational problem will provide unique challenges, and we therefore aim to motivate the need for a close and careful analysis of a computational problem when it is to be used in human performance experiments.

It is also important to ensure that participants are working on the problem intended by the researcher, and that they have available to them the resources necessary to do so. In this section, we illustrate some challenges and pitfalls one may encounter when studying human performance on computational problems.

Presentation

There often exists more than one valid way of presenting a computational problem: e.g., a booklet of instances printed on paper; or a series of digital representations on a computer screen. The choice of presentation impacts human performance on hard problems, in part due to the constraints a particular presentation may impose on the cognitive aids participants have access to. Consider, for example, the difference between paper and pencil presentation, and computer presentation of a hard computational problem. Computer presentation of a problem—if implemented properly—can, for example, make it easier for participants to undo steps, giving the advantage of being able to backtrack if a chosen step is not on the path to the goal state (that is, the optimal solution is not directly reachable without backtracking). Backtracking is a fundamental decision-making mechanism, allowing for testing of theories. In contrast, when working with paper and pencil, participants may be less willing to test theories because the cost of undoing is steeper (the exact order of steps taken may be hard to recall, for instance). This is particularly important when problem solving or tackling hard computational problems, where the size of the problem space may preclude testing options purely in one's mind due to limitations of working memory (Baddeley & Hitch, 1974).

According to Newell and Simon (1972), the problem space for a given instance of a problem is an internal representation generated by the problem solver. One possible representation of the problem space would be all possible paths from the start state to all

goal states, where the transitions from state to state are made using legal operators. While the problem solver does not generate and retain the entire problem space in memory it is useful to think about it in its entirety (Newell & Simon, 1972).

However, even some digital representations of problems can limit, or even prevent, backtracking. Consider the game Minesweeper, which has been shown to be computationally hard in its general case (Scott et al., 2011), and which human performance has been studied, at least in a limited sense (Walker, 2010; also see above). A typical implementation of the Minesweeper game—as played for example on a PC—prevents backtracking. In the case where a player or participant reaches a juncture in the gameplay where no more cells can be eliminated exactly they are forced to have to guess between some number of valid cells. If they guess incorrectly (that is, they detonate a mine), the game is over, and no further attempts can be made. This, in effect, prevents players from exploring moves that may lead down incorrect paths through the problem space. Indeed, even the problem to decide whether the content of another cell on a given minesweeper board can be inferred is computationally hard (Scott et al., 2011). Even if the correct answer is that an inference is possible, backtracking may be a useful aid for humans playing minesweeper. Most players take it as a given that a wrong choice will end the game instantly, but when approaching this problem from the perspective of searching for a path to the goal state, the inability to backtrack prevents some branches of search from being explored.

Representation

Instances of computational problems can be represented in different ways. Let us consider graph problems, for example. Graphs are typically represented by vertices and edges (e.g., lines that connect the vertices). An example of an undirected graph is given in Figure 1. Another way of conveying the same information, namely the relations between pairs of nodes, is in the form of a matrix (Figure 2). In this representation, a *1* in cell (*i*, *j*) indicates

Figure 1. An undirected graph.

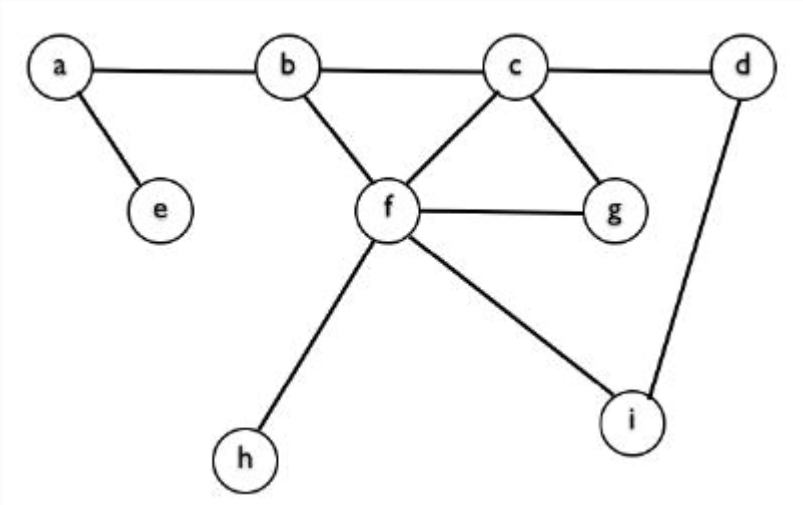


Figure 2. Matrix representation of the graph representation in Figure 1.

	a	b	c	d	e	f	g	h	i
a	0	1	0	0	1	0	0	0	0
b	1	0	1	0	0	1	0	0	0
c	0	1	0	1	0	1	1	0	0
d	0	0	1	0	0	0	0	0	1
e	1	0	0	0	0	0	0	0	0
f	0	1	1	0	0	0	1	1	1
g	0	0	1	0	0	1	0	0	0
h	0	0	0	0	0	1	0	0	0
i	0	0	0	1	0	1	0	0	0

that there is an edge connecting vertices v_i and v_j. In human studies, graph problems like TSP, E-TSP, and Vertex Cover are usually represented as points in the plane (representing vertices) connected by lines (representing edges). In the case of E-TSP, the edges are often omitted, with the implication that there exists an edge (a straight line) between every pair of points. This graph representation seems to be a natural choice. However, instances of all of these problems can be represented in other, computationally equivalent, ways.

The choice of representation can impact what information is available to the problem solver. To illustrate how solving a computational problem on both a graph and a matrix for the same instance might differ, consider the optimization version of the *NP*-complete decision problem Dominating Set.

Minimum Dominating Set/Dominating Set (optimization version)
Input: A graph $G = (E, V)$, with vertex set V, and edge set E.
Output: A smallest dominating set for G, that is a subset U of V, such that every vertex in $V - U$ is joined to at least one vertex in U by an edge in E.

Now consider two different representations of an instance of the problem, using the graph and the matrix representations of a graph in Figures 1 and 2, respectively. Figures 3 and 4 show the state of the problem after a first vertex, vertex *f* is added to the dominating set U to be determined. In Figure 3 we see vertex *f* colored red, indicating that it has been added to U, and the vertices *b*, *c*, *g*, *h*, *i* have been colored blue, indicating they are dominated by *f*. The ultimate goal is to color as few vertices red as necessary, such that all vertices in the graph are colored red or blue. Similarly, in Figure 4, the row- and

Figure 3. Graph representation of the Dominating Set instance from Figures 1 and 2, with vertex *f* (red) selected into the dominating set. The vertices dominated by *f* are colored in blue.

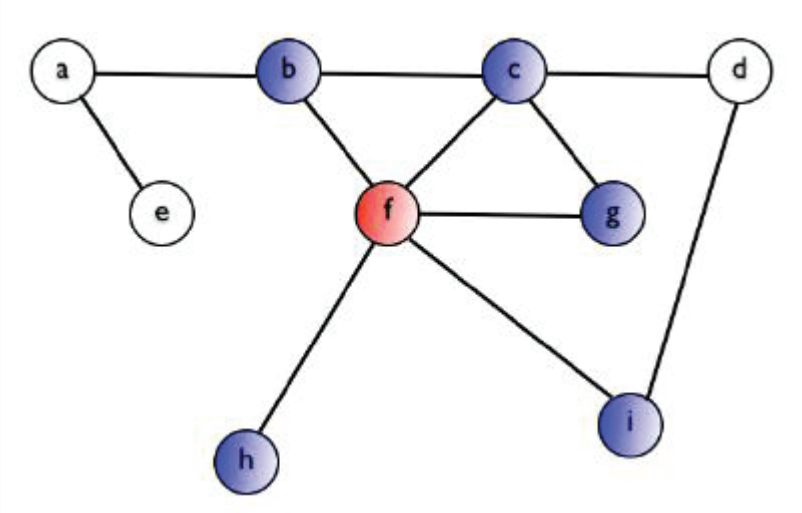


Figure 4. Matrix representation of Figure 3.

	a	b	c	d	e	f	g	h	i
a	0	1	0	0	1	0	0	0	0
b	1	0	1	0	0	1	0	0	0
c	0	1	0	1	0	1	1	0	0
d	0	0	1	0	0	0	0	0	1
e	1	0	0	0	0	0	0	0	0
f	0	1	1	0	0	0	1	1	1
g	0	0	1	0	0	1	0	0	0
h	0	0	0	0	0	1	0	0	0
i	0	0	0	1	0	1	0	0	0

column-labels for vertex *f* are colored red, and the vertices *b*, *c*, *g*, *h*, *i* are colored blue as they are dominated by *f* . The ultimate goal is to color as few row labels red as necessary such that all column labels in the matrix are colored red or blue. The effect these different representations might have on performance has not yet been studied, and it is interesting to wonder how they impact the strategies identified by participants. Manipulation of the representation of a problem could be used to investigate what aspects of the problem/ representation impact human performance.

Problem Type

As discussed earlier, a computational problem can be typically phrased as an optimization, search, or decision problem. To date, human performance studies have focused on the optimization versions (that is, find the shortest route in E-TSP; find the shortest path to solution in *n*-Puzzle), but it is also important to consider how humans might perform on, for example, either search or decision versions of the same computational problem. In the case of Vertex Cover, the decision problem in particular is a natural choice as its natural parameterization, where the decision problem is parameterized by the size of the size of the vertex cover permitted, is a member of the class *FPT*.

Problem Selection

To date, studies of human performance on computationally hard problems have focused on a handful of the large number of such problems that may be suitable for human performance studies. Future studies would benefit from the careful selection of problems based on an in-depth knowledge of their complexity, attributes and properties. Hard problems differ in a number of ways. They may be tractable under certain constraints (e.g., if the instances in question all belong to a class of problem instance that can be solved efficiently) or considerations (such as parameterizations that are in *FPT*), easy or hard to approximate, or even related to other problems. We illustrate these constraints and considerations by providing a number of examples.

E-TSP was likely initially chosen for human study because of its famous intractability results, and performance results from these studies have sparked a further interest in studying other problems. Vertex Cover was selected as a non-Euclidean alternative to E-TSP (Carruthers et al., 2012), but, as mentioned above, also has the desirable quality of being in the class *FPT* when parameterized by its vertex cover size; thus for small enough *k* (in its decision version), Vertex Cover's time complexity grows exponential only in *k*, and is polynomial in terms of its input size (Downey & Fellows, 1999)[17]. A natural question arises: is there a difference in human performance on computational problems that are *FPT* for natural parameterizations versus those that are (likely) not[18]? In other words, are humans able to identify and leverage techniques consistent with reduction rules (polynomial-time decisions that reduce the size of the problem space) that are used to simplify problems in *FPT*? In addition to problems that are in *FPT*, there exist hard problems that lend themselves nicely to approximation algorithms such as E-TSP (Arora, 1996), and others that cannot be approximated within arbitrary accuracy. Again, are there differences in human performance on these two types of problem? Finally, it may be of interest to study performance on two problems for which the solution to one, exactly yields a solutions to the other, which differ either in terms of ease of approximation or fixed parameter tractability.

When selecting problems for human study, all these factors should be considered in attempting to explain or model human performance. Furthermore, for problems that

are known to have approximation algorithms, or that are in *FPT*, researchers may look to the literature for techniques that humans may be able to identify and apply. For example, when solving Vertex Cover instances, people appear able to identify and apply two known reduction rules (Carruthers, et al., 2012), which may help to explain human performance, at least for instances where these reduction rules are available.

Selection of Instances

The selection of instances given to participants in a study can have a significant impact on performance results. Certainly, the size of an instance may impact the difficulty of the task, but other factors must also be carefully considered. A deep understanding of a

Figure 5. An optimal solution for the Minimum Vertex Cover on this instance.

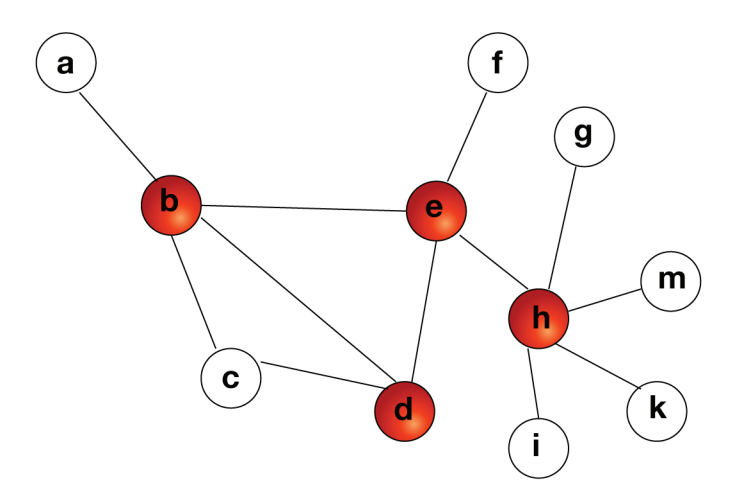


Figure 6. A solution obtained when applying the greedy strategy described above to another instance.

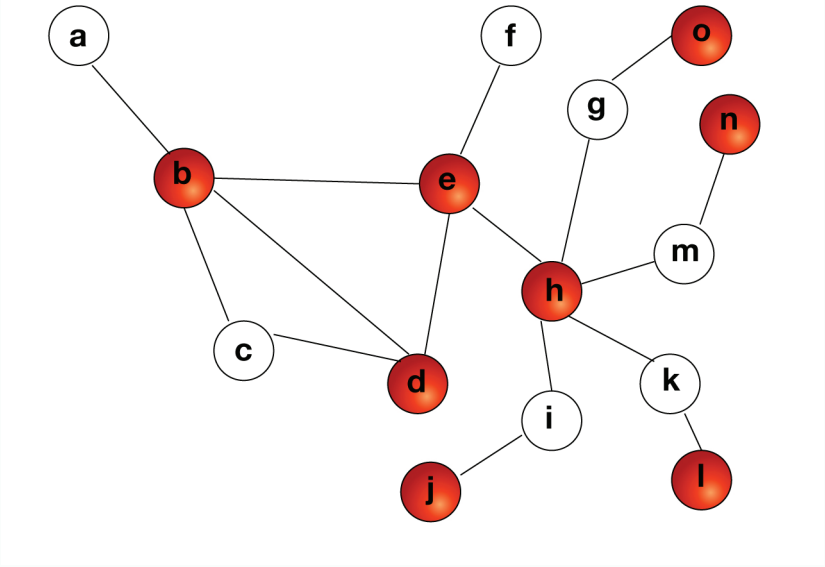
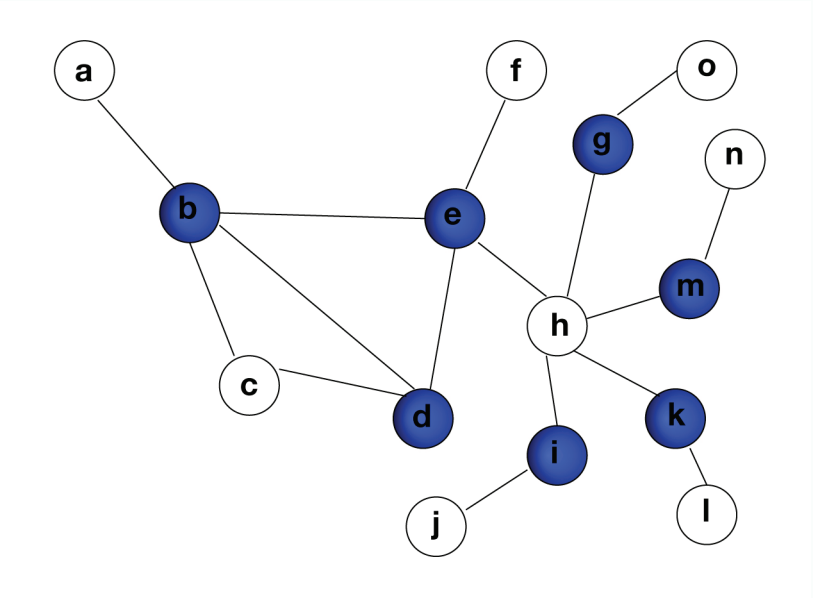


Figure 7. An optimal solution for the Minimum Vertex Cover instance in Figure 6.



computational problem can facilitate the selection of instances that somehow reflect the difficulty of the problem in general. For instance, for all computational problems there exist (possibly large) subsets of instances for which a particular algorithm will generate correct output. Should an instance set contain only, or mostly, instances of this type, performance results could be skewed. We motivate this with an example.

Recall that an algorithm that (exactly or optimally) solves a particular instance of a problem may not necessarily (exactly or optimally) solve the problem (that is, solve all instances of the problem). Here we give an example from Vertex Cover.

Consider its optimization version, Minimum Vertex Cover, as defined above. Figure 5 shows an optimal solution to a particular instance of this problem. A strategy that yields this optimal solution for the particular instance can be described as follows.

- As long as there exists at least one uncovered edge in the graph
  - find a vertex that is incident to the most uncovered edges
  - include this vertex in the vertex cover

However, this strategy (called a *greedy* strategy) does not give an optimum vertex cover for the instance in Figure 6; in Figure 7 a better result is achieved.

A common practice in studies of human performance on E-TSP is to use randomly generated instances. On the surface, this may seem like a wise choice. However, this implicitly assumes somehow that the difficulty of problem instances is normally distributed, and that random generation will yield a set of problems that are representative of the difficulty of the problem as a whole. This caution speaks to a more general issue, that more purposeful means of generating instances may yield more variation in performance results, and therefore more insight into human problem solving.

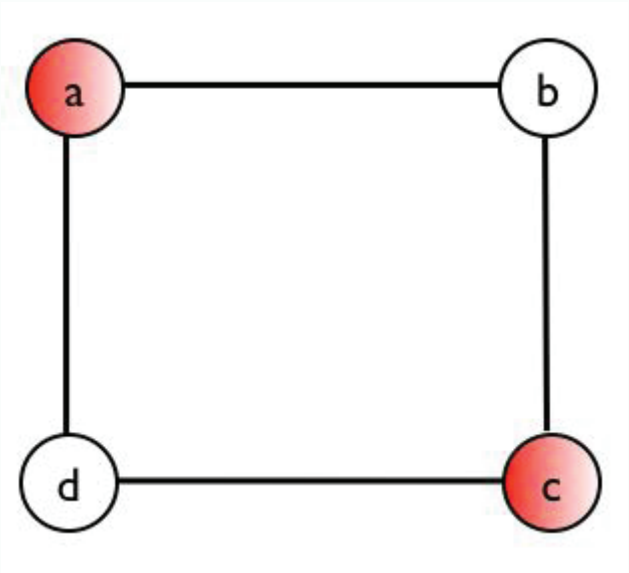
Participant Comprehension

As in other human performance studies, in order to make meaningful inferences from data collected on hard computational problems, it is important to determine that participants fully understand the problem given. This is typically accomplished through training and trials. In some cases, participants have also been tested on identifying the correctness of solutions. However, it is possible that these strategies may fail to identify some kinds of problems, for instance where participant output may also be correct output for other problems. There exist problem instances for which a correct solution may be correct for more than one computational problem given the same instance. Take the three problems Vertex Cover, Independent Set, and Dominating Set, where the optimization version of the *NP*-hard decision problem Independent Set is defined as follows:

Maximum Independent Set/Independent Set (optimization version)
Input: A graph $G = (E, V)$, with vertex set V, and edge set E.
Output: A largest independent set for G, that is a subset U of V, such that no vertex in U is joined to any other vertex in U by an edge in E.

Vertex Cover and Independent Set are particularly interesting since a correct solution to the optimization version of one directly yields a correct solution of the optimization of the other (namely, the complement of the vertices selected as solution for the one problem is the solution for the other one). Given the instance in Figure 8 as input, a participant's (correct) output when tasked with the Vertex Cover could be the one indicated in red. However, this same output is also correct for both Independent Set and Dominating Set. This error would not be identified by some of the protocols used to determine compre-

Figure 8. Optimal solution for Minimum Vertex Cover, Maximum Independent Set, and Minimum Dominating Set.



hension. This instance, while admittedly trivial, serves to highlight this issue as a caution. When designing experiments and testing participant comprehension, careful design of protocols may be needed. A verbal protocol could be designed to demonstrate that an output reflects an understanding of the problem given, rather than a (still correct) output to a closely related problem. In addition, careful selection of instances for trials can make these kinds of errors easier to discern.

In computationally hard Euclidean problems, is it possible to ensure that it is within participants' perceptual abilities to discern optimal from sub-optimal solutions? We identify two possible issues that arise when attempting to answer this question. First, combinatorially speaking, participants may be required to compare the length of a tour to (potentially many) alternatives in order to ascertain that none are shorter. For large instances, the number of tours required for comparison can be very large, however there may be perceptual mechanisms that allow participants to eliminate some number of tours which are far from optimal. Second, does the human visual perception system provide enough accuracy in measurement of distances to identify optimal tours in E-TSP? With no means other than visual assessment, there are differences in length that cannot be determined. Studies of human performance on visually presented E-TSP have largely neglected these issues. In one study, participants were asked to decide if a solution was optimal or not (Ormerod & Chronicle, 1999). The solutions were either (about) 0%, 15%, 25%, 35%, 45% above optimal. The best scores were for the 45% above optimal sets, but still with about 12% error rate. Worst scores were in the 25–35% range with approximately 50% error rate. These results do not support the premise that participants are able to identify optimal tours.

One final note of caution comes to mind when studying human performance on popular games or puzzles. In these cases, habitual rules of play may undesirably undermine the researchers' intentions of what problem participants are indeed tackling. Habitual game-play may impede participants' ability to correctly follow instructions. Take the *n*-Puzzle for example, studied by Pizlo and Li (2005). The authors clearly instructed participants to solve the optimization version of the problem, that is, to find the shortest path to the goal. The 15-puzzle, a special case of the *n*-Puzzle, is relatively popular. The question posed by the researcher, that of optimizing the length of the path to the solution, differs from how it is typically played, and as such players may not consistently work towards the fewest moves possible. Can we be certain that participants were able to overcome their habit of trying to find a solution in order to follow the researcher's instructions to find the shortest path?

Cognitive Support

As described above, the choice of presentation and representation can impact participants' performance. When problem solving, a number of factors may limit people's ability to efficiently solve a problem, including limitations of working memory, ability to backtrack,

and the ability to determine the path or distance to the goal state. Digital representations of problem instances can be designed to provide additional support, thus reducing these challenges and allowing participants to focus more fully on the problem.

According to a means end analysis approach to problem solving, problem solvers choose their next step(s) in part by identifying the step that provides the most gain from the current problem state to the goal state (Newell & Simon, 1972). This calculation can be challenging in computationally hard problems, as the problem space (which can be thought of as a search tree rooted at the initial state of the problem, with branching paths, some of which lead to the goal state), can be very large. Consider an instance for Vertex Cover with n vertices. One (naive) way to build the search tree is as follows. At the i-th step, we consider whether or not to add vertex v_i to the cover, giving us two options: yes or no, we then consider whether or not to add vertex v_{i+1}, and so forth. This leads to a binary search tree of depth n, and of size 2^n. The distance to the goal state (all edges are covered, using the fewest vertices) is not necessarily easily measured based on the current location in the search tree and may require backtracking, and traversing down a different branch. This difficulty, of ascertaining the distance from a current state to the goal state, is not limited to this problem, and has been observed in human behaviour. In their study, Pizlo and Li (2005) found that participants, given a randomly selected intermediate state from one of their solutions, could not reliably judge the distance to the goal state. A participant's ability to measure the distance to the goal state is challenging. In the Vertex Cover problem, for instance, it may be helpful to give participants feedback upon completion of an instance about the quality of their vertex cover. This could allow them to identify patterns of decisions (such as making greedy choices) that lead to sub-optimal solutions. While this does not directly inform the individual about the distance to the goal state, it allows them to eliminate decisions which lead them away from short paths to the goal. Each computational problem will present different challenges when considering how to best support participants ability to judge the distance to the goal state.

Backtracking is a necessary technique when solving hard problems, and can be supported by allowing sequential undoing, that is, allowing participants to undo choices in the reverse order in which they were done. Without sufficient cognitive support, undoing may be a burden on working memory. Consider the case where a choice is made, followed by a (possibly large) number of subsequent choices. If this number of choices is too large, participants may not be able to correctly recall how to return to the state where the initial choice was made. Sequential undoing has been used in a number of studies of E-TSP. In addition, in their study of performance on n-Puzzle, researchers noted that very little backtracking was done (Pizlo & Li, 2005). It may, however, be that short backtracking (1–2 moves only) was all that participants were safely able to perform without risk of error. It would be interesting to determine if, had they had access to a reliable means of backtracking, participants would more readily use backtracking to shorten their solution path.

Figure 9. Instance of Vertex Cover.

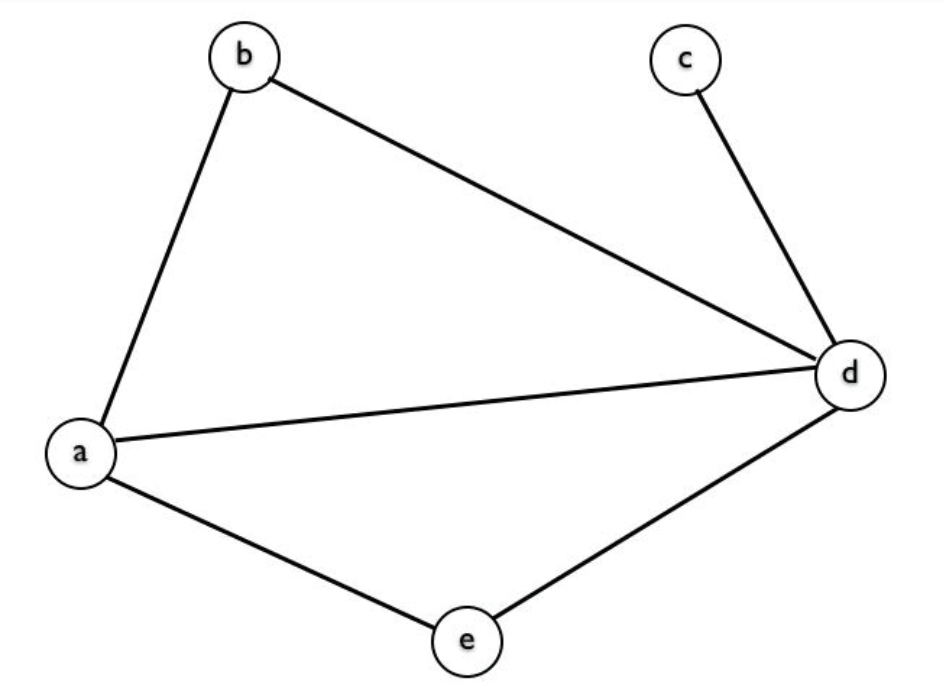
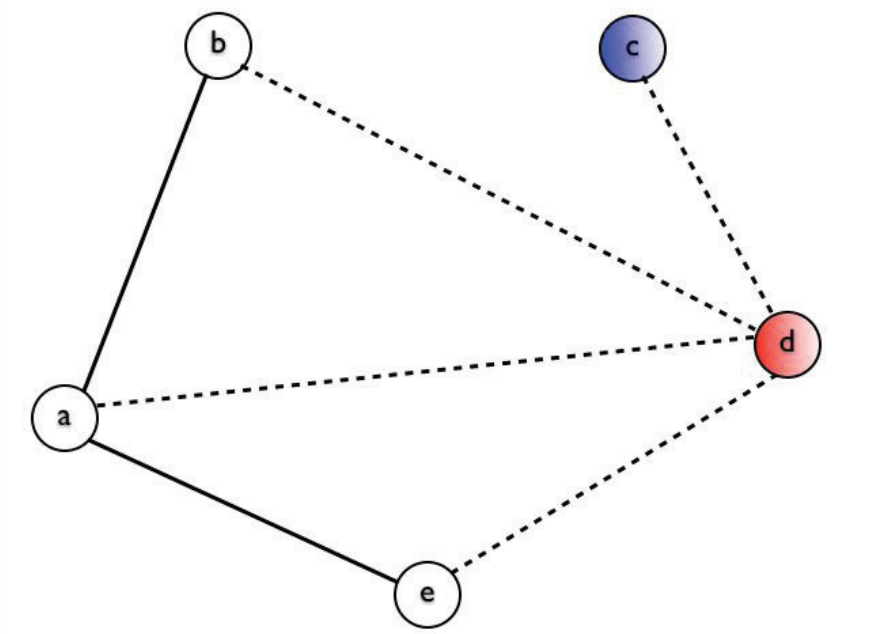


Figure 10. Instance of Vertex Cover with vertex *d* added.



When investigating human performance on computationally hard problems, it is desirable to allow participants to focus on the hard problem, and not be burdened by menial tasks and bookkeeping. For example in Vertex Cover, once participants demonstrate their understanding of the problem, in particular that the addition of a vertex to the vertex cover implies that all adjacent edges are covered, marking the edges covered

can be tedious, and lead to unnecessary errors. A digital interface can assist in this area, by cleaning up the state of the instance to reflect which edges are already covered. Consider Figure 9. When vertex *d* is added to the cover, the vertex *c* need no longer be considered, leaving only *a*, *b*, *e*, leading to the state in Figure 10. In the E-TSP, the constraints of the human visual system can make it difficult to determine, with great enough accuracy, the difference of similar length paths. One could assist problem solvers by giving them a tool which more accurately measures the distance between points than is possible by eye alone.

Conclusion

With all this in mind (problem selection, problem presentation, problem representation, instance selection, problem type selection, participant comprehension, cognitive support), how can researchers be sure to carefully present their studies of human performance on hard computational problems? A first step is to address how all these issues are dealt with in the design of the experiment. It is not sufficient to justify the selection of a problem with knowledge that it is hard; a more in-depth analysis is called for. Since problem question, presentation, and representation can all impact performance, these choices should be carefully considered and explained. When describing the complexity results of a problem, researchers should ensure that they: 1) match the problem given to participants, and 2) match the problem being solved by participants. The hard computational problems used in human studies have been shown to be hard in general, that is, for any input size. However, in human studies we are limited to giving participants a finite number of instances of relatively small size. Further, participants are rarely tasked with the computational problem known to be computationally hard: that is, to find a general algorithm which always yields the correct result. Instead, participants are tasked with finding (close to) optimal solutions to the instances given. As we have illustrated, the selection of instances can have a great impact on the difficulty of the task, as well as what strategies can succeed. Therefore, authors should be explicit in describing how and why specific instances were selected, and how this selection might impact the hardness of the task. Since different problems can lead to identical correct output, it may also be important to use protocols other than trials and training to ensure that participants indeed understand the problem intended. And finally, in order to best support participants in focusing on the hard problem, researchers may wish to provide adequate cognitive support to minimize the amount of bookkeeping necessary.

Future Research: Research Questions

We have presented examples of how—when using known puzzles or other known computational problems—there is a potential disconnect between the computational problem assumed to be studied, and the problem as interpreted by participants. This situation

presents the opportunity to study how it may impact performance. Holding presentation, representation, support, question and complexity constant, is there a performance difference on problems that are habitually played in a manner that differs from the problem posed?

Further, it can happen that the problem as posed in the study differs from the problem as defined by the complexity theorists. For a number of popular puzzles, the computational problem for which hardness results are cited in the literature may differ from the computational problem normally played. That is, there is a disconnect between the question posed, and the version that is played recreationally. Examples are Sudoku, Minesweeper and Slitherlink. For all three puzzles, the generalized decision version that asks for a given problem instance whether or not a solution exists is the one that is shown *NP*-complete (Kendall et al., 2008).

All three puzzles, however, when played, typically are presented as *promise problems*, that is: the puzzle is assumed to have a solution, but the challenge is to *find it* (You can always place those numbers correctly in your newspaper Sudoku, it is just a matter of your skills how fast you will succeed!). The complexity of these different problems can be different: for example, in the case of Minesweeper, the Minesweeper Consistency problem that asks whether or not a given Minesweeper board does have a solution, is shown to be *NP*-complete (Kaye, 2000), whereas Minesweeper Inference, the promise problem that assumes as input a consistent Minesweeper board that can be solved asks, whether or not with the information available the content of a hidden cell can be inferred, is co-*NP*-complete (Scott et al., 2011). It is theoretically possible that the promise problem of its intractable non-promise version is a tractable one. This presents an opportunity to examine the complexity of the problems being used. While only a few of these problems have been formally studied in terms of human performance (*n*-Puzzle, Minesweeper), it would still be interesting to know complexity results for the way the game is typically played, for example, the promise problems of the puzzles Sudoku, Slitherlink, Mastermind and Instant Insanity. In this article we have focused on the hard computational problems, but there are equally many opportunities to mine complexity theory for other kinds of problems as well. While a number of problems that are known to be tractable have been studied in terms of human performance, it may be of interest to specifically compare performance on tractable vs. intractable problems. Problems could easily be identified which share common presentation, representation, question and/or support, thereby allowing for the focus of the study to be on particular strategies or problem solving techniques identified by participants.

Another issue arises due to the necessary use of a small number of instance of problems in human studies. It appears that for a number of problems—including E-TSP—the question posed is not to solve the problem in general (such as "Find an algorithm that solves E-TSP for any given instance") but instead to solve the problem for one or several specific instances. This leaves open the question of human performance measures when tasked with identifying a general algorithm for a hard computational problem. This is of

particular interest to computer scientists, because it is precisely the problem they face when studying a problem.

We introduced computational problems that have parameterizations that are in *FPT*, as well as problems that cannot be closely approximated. Both of these complexity distinctions raise interesting open questions. Is there a difference in human performance on problems that are *FPT* versus those that are (likely) not? Are humans able to identify and leverage techniques consistent with reduction rules that are used to simplify problems in *FPT*? Are there differences in human performance on problems which can be nicely approximated and those that cannot? For problems where the parameterized complexity or approximability are not known, an analysis towards an understanding is encouraged as it may shed light on the human performance of those computational problems.

Finally, in order to best support participants tackling hard computational problems, we believe it is important to come up with the best kinds of cognitive support to allow participants to focus on the hard (interesting) problem and not on menial tasks and bookkeeping. As such, it may be interesting to investigate how best to do this, both for problems that are currently being studied, but also in general.

Acknowledgments

We would like to thank Iris van Rooij, Michael Masson, and Laurie Heyer for fruitful discussions, inspiration and constructive criticism. This work was funded by NSERC.

Notes

1. An algorithm is a finite set of well-defined steps that transforms an input into an output (Cormen, Leiserson, Rivest, & Stein, 2003).
2. An algorithm *solves* a function if, on any input, it comes to a stop with the correct output (Cormen et al., 2003).
3. A *computational problem* is typically a formally described question or task that is to be solved for a given input of a specified format.
4. A function is called *computable* if there exists an algorithm that solves the function (Turing, 1936).
5. Other types of computational problems include *counting problems* and *promise problems* (Goldreich, 2008).
6. *Parameterized Complexity* (Downey & Fellows, 1999)—that studies the complexity of computational problems in a multivariate view—considers many problems that are diagnosed as intractable by a classical complexity analysis as tractable, namely, the parameterized computational decision problems that are in the class of *Fixed Parameter Tractability* (FPT) problems. See also footnotes 13 and 14.

7. The Clay Institute offers 1 million US dollars for solutions to any of its Millennium Prize Problems, including the question of $P \neq NP$. http://www.claymath.org/millennium/.
8. A *polynomial-time reduction* from a decision problem *A* to a decision problem *D* is a polynomial-time algorithm that takes as input any instance I_A for problem A and transforms it into an instance I_D for problem D such that I_A is a YES-instance if and only if I_D is a YES-instance (Garey & Johnson, 1979).
9. The class co-*NP* contains all those computational problems which have the property that the problem's *complement* is in *NP*. Note that co-*NP* and *NP* have a non-empty intersection, that is, there are problems which are both in *NP* and in co-*NP*. The class co-*NP*-complete contains all those problems that are in co-*NP*, and which are hard for co-*NP* (Arora & Barak, 2007).
10. A *heuristic* for a computational problem is an algorithm that has no guarantee of correctness or quality of bounds for a given computational problem (Cormen et al., 2003).
11. An *approximation algorithm* for a computational problem is a well-defined set of steps that transforms an input into an output that guarantees the optimality of the solution to be within specific bounds of the correct solution (Vazirani, 2004). Typically, the goal when designing an approximation algorithm is that the approximation achieved by the algorithm is optimal up to a (small) constant factor (for example, $\leq 2 \times$ optimal). This still holds when there exists more than one optimal solution, as the bound is in terms of the optimality of the solution.
12. A parameterization or parameterized decision problem of a computational decision problem is a problem where a (fixed) parameter is specified separately from the input.
13. The parameterized complexity class *FPT* is the class of all parameterized decision problems $< I, k >$—where *I* constitutes the (non-parameterized) input and *k* denotes the specified parameter—that are solvable by an algorithm in a time $f(k)|I|^{\alpha}$ where α is a positive constant and f is a computable function. Note that *f* does not have to be polynomial.
14. For a book on basic graph theory see, for example, the book by Voloshin (2009).
15. Note that it is indeed possible that a minesweeper board does not contain enough information to infer the content of at least one further cell on the board.
16. The running time for Vertex Cover parameterized by its vertex cover size is $O(1.2738^k + k|V|)$ (Chen, Kanj, & Xia, 2010).
17. There exist parameterizations of computationally hard decision problems that are likely not in *FPT*. Examples are Dominating Set parameterized by the size of the dominating set permitted, and Independent Set parameterized by the size of the independent set needed (Downey & Fellows, 1999).
18. For example, there exists a constant $\rho < 1$ such that: if there exists a polynomial time ρ−approximation for the problem MAX3-SAT, then it is the case that P = NP (Arora & Barak, 2007).

References

Aloupis, G., Demaine, E. D., & Guo, A. (2012, Mar 8). Classic Nintendo games are (NP-)hard. Retrieved from arXiv:1203.1895

Alt, H., Bodlaender, H. L., van Kreveld, M., Rote, G., & Tel, G. (2007). Wooden geometric puzzles: Design and hardness proofs. In P. Crescenzi, G. Prencipe, & G. Pucci, (Eds.), *Fun with Algorithms: 4th International Conference, Castiglioncello, Italy* (pp. 16–29). http://dx.doi.org/10.1007/978-3-540-72914-3_4

Arora, S. (1996). Polynomial time approximation schemes for Euclidean TSP and other geometric problems. *Proceedings, 37th Annual Symposium on Foundations of Computer Science, Princeton, NJ* (pp. 2–11). http://dx.doi.org/10.1109/SFCS.1996.548458

Arora S., & Barak, B. (2007). *Computational complexity*: *A modern approach*. New York: Cambridge University Press.

Baddeley, A. D., & Hitch, G. (1974). Working memory. *Psychology of Learning and Motivation*, *8*, 47–89. http://dx.doi.org/10.1016/S0079-7421(08)60452-1

Best, B., & Herbert, S. (2003). Simulating human performance on the traveling salesman problem. *Proceedings of the Fifth International Conference on Cognitive Modeling*, Bamberg, Germany (pp. 1–6).

Burns, N. R., Lee, M. D., & Vickers, D. (2006). Are individual differences in performance on perceptual and cognitive optimization problems determined by general intelligence? *Journal of Problem Solving*, *1*(1), 5–19.

Carruthers, S., Masson, M. E. J., & Stege U. (2012) Human performance on hard non-Euclidean Graph problems: Vertex cover. *Journal of Problem Solving*, *5*(1), 34–55.

Chen, J., Kanj, I. A., & Xia, G. (2010). Improved upper bounds for vertex cover. *Theoretical Computer Science*, *411*(40–42), 3736–56.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2003). *Introduction to algorithms (2nd ed.)*. Cambridge, MA: MIT Press.

Demaine, E. D., Hohenberger, S., & Liben-Nowell, D. (2003). Tetris is hard, even to approximate. In T. Warnow & B. Zhu (Eds.), *Computing and Combinatorics: Proceedings of the 9th Annual International Conference*, *Big Sky, MT* (pp. 351–63).

Downey, R. G., & Fellows, M. R. (1999). *Parameterized complexity*. New York: Springer Verlag. http://dx.doi.org/10.1007/978-1-4612-0515-9

Dry, M., Lee, M. D., Vickers, D., & Hughes, P. (2006). Human performance on visually presented traveling salesperson problems with varying numbers of nodes. *Journal of Problem Solving*, *1*(1), 20–32.

Dry, M., Preiss, K., & Wagemans, J. (2012). Clustering, randomness, and regularity: Spatial distributions and human performance on the traveling salesperson problem and minimum spanning tree problem. *Journal of Problem Solving*, *4*(1), 1–17.

Frixione, M. (2001). Tractable competence. *Minds and Machines*, *11*, 379–97. http://dx.doi.org/10.1023/A:1017503201702

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco, CA: W. H. Freeman.

Goldreich, O. (2008). Computational complexity: A conceptual perspective. *ACM SIGACT News*, *39*(3), 35–39. http://dx.doi.org/10.1145/1412700.1412710

Graham, S. M., Joshi, A., & Pizlo, Z. (2000). The traveling salesman problem: A hierarchical model. *Memory & Cognition*, *28*(7), 1191–204. http://dx.doi.org/10.3758/BF03211820

Haxhimusa, Y., Carpenter, E., Catrambone, J., Foldes, D., Stefanov, E., Arns, L., & Pizlo, Z. (2011). 2D and 3D traveling salesman problem. *Journal of Problem Solving*, *3*(2), 167–93.

Haxhimusa, Y., Kropatsch, W. G., Pizlo, Z., Ion, A., & Lehrbaum, A. (2006). Approximating TSP solution by MST based graph pyramid. In F. Escolano & M. Vento (Eds.), *Graph-Based Representations in Pattern Recognition: 6th IAPR-TC-15 International Workshop, Alacante, Spain* (pp. 1–11). New York: Springer Verlag.

Horgan, T., & Tienson, J. (1996). *Connectionism and the philosophy of psychology*. Cambridge, MA: MIT Press.

Kaye, R. (2000). Minesweeper is NP-complete. *Mathematical Intelligencer*, *22*(3), 9–15. http://dx.doi.org/10.1007/BF03025367

Kendall, G., Parkes, A., & Spoerer, K. (2008). A survey of NP-complete puzzles. *ICGA Journal*, *31*(1), 13–34.

Kong, X., & Schunn, C. D. (2007). Global vs. local information processing in visual/spatial problem solving: The case of traveling salesman problem. *Cognitive Systems Research*, *8*(3), 192–207. http://dx.doi.org/10.1016/j.cogsys.2007.06.002

MacGregor, J. N., Chronicle, E. P., & Ormerod, T. C. (2004). Convex hull or crossing avoidance? Solution heuristics in the traveling salesperson problem. *Memory & Cognition*, *32*(2), 260–70. http://dx.doi.org/10.3758/BF03196857

MacGregor, J. N., & Ormerod, T. C. (1996). Human performance on the traveling salesman problem. *Perception & Psychophysics*, *58*(4), 527–39. http://dx.doi.org/10.3758/BF03213088

MacGregor, J. N., Ormerod, T. C., & Chronicle, E. P. (1999). Spatial and contextual factors in human performance on the travelling salesperson problem. *Perception*, *28*(11), 1417–1427. http://dx.doi.org/10.1068/p2863

MacGregor, J. N., Ormerod, T. C., & Chronicle, E. P. (2000). A model of human performance on the traveling salesperson problem. *Memory & Cognition*, *28*(7), 1183–90. http://dx.doi.org/10.3758/BF03211819

Marr, D. (1982). *Vision: A computational approach*. San Francisco, CA: W. H. Freeman.

Massaro, D. W., & Cowan, N. (1993). Information processing models: Microscopes of the mind. *Annual Review of Psychology*, *44*, 383–425. http://dx.doi.org/10.1146/annurev.ps.44.020193.002123

Millgram, E. (1982). The knowledge level. *Artificial Intelligence*, *18*, 87–127. http://dx.doi.org/10.1016/0004-3702(82)90012-1

Millgram, E. (2000). Coherence: The price of the ticket. *Journal of Philosophy*, *97*(2), 82–93. http://dx.doi.org/10.2307/2678447

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

Ormerod, T. C., & Chronicle, E. P. (1999). Global perceptual processing in problem solving: The case of the traveling salesperson. *Perception & Psychophysics*, *61*(6), 1227–38. http://dx.doi.org/10.3758/BF03207625

Pizlo, Z., & Li, Z. (2005). Solving combinatorial problems: The 15-puzzle. *Memory & Cognition*, *33*(6), 1069–84. http://dx.doi.org/10.3758/BF03193214

Pylyshyn, Z. W. (1984). *Computation and cognition: Towards a foundation for cognitive science*. Cambridge, MA, MIT Press.

Ratner, D., & Warmuth, M. (1990). Finding a shortest solution for the ($N \times N$)-extension of the 15-puzzle is intractable. *Journal of Symbolic Computation*, *10,* 111–37. http://dx.doi.org/10.1016/S0747-7171(08)80001-6

Robertson, E., & Munro, I. (1978). NP-completeness, puzzles and games. *Utilas Mathematica*, *13*, 99–116.

Rumelhart, D. E., MacClelland, J. K., & PDP Research Group. (1986). *Parallel distributed processing. Explorations in the microstructure of cognition. Volume 1: Foundations*. Cambridge, MA, MIT Press.

Saalweachter, J. & Pizlo, Z. (2008). Non-Euclidean traveling salesman problem. In T. Kugler, J. C. Smith, T. Connolly, & Y.-J. Sun (Eds.), *Decision modeling and behavior in complex and uncertain environments* (pp. 339–58). New York: Springer. http://dx.doi.org/10.1007/978-0-387-77131-1_14

Scott, A., Stege, U., & van Rooij, I. (2011). Minesweeper may not be NP-complete but is hard nonetheless. *The Mathematical Intelligencer*, *33*(4), 5–17. http://dx.doi.org/10.1007/s00283-011-9256-x

Sevenster, M. (2004). Battleships as decision problem. *ICGA Journal*, *27*(3), 142–49.

Stuckman, J. & Zhang, G.-Q. (2006). Mastermind is NP-Complete. *INFOCOMP Journal of Computer Science*, *5*, 25–28.

Tak, S., Plaisier, M., & van Rooij, I. (2008). Some Tours are more equal than others: The convex-hull model revisited with lessons for testing models of the traveling salesperson problem. *Journal of Problem Solving*, *2*(1), 4–28.

Thagard, P. (2000). *Coherence in thought and action*. Cambridge, MA, MIT Press.

Thagard, P. & Verbeurgt, K. (1998). Coherence as constraint satisfaction. *Cognitive Science*, *22*(1), 1–24. http://dx.doi.org/10.1207/s15516709cog2201_1

Tsotsos, J. K. (1988). A 'complexity level' analysis of immediate vision. *International Journal of Computer Vision*, *1*(4), 303–20.

Tsotsos, J. K. (1989). The complexity of perceptual search tasks. *Proceedings of the Eleventh Annual International Joint Conference on Artificial Intelligence, Detroit, MI: Vol. 2*

(pp. 1571–77). Retrieved from http://ijcai.org/Past%20Proceedings/IJCAI-89-VOL-2/PDF/114.pdf

Tsotsos, J. K. (1990). Analyzing vision at the complexity level. *Behavioral and Brain Sciences*, *13*(3), 423–69. http://dx.doi.org/10.1017/S0140525X00079577

Tsotsos, J. K. (1991). Is complexity theory appropriate for analyzing biological systems? *Behavioral and Brain Sciences*, *14*(4), 770–73. http://dx.doi.org/10.1017/S0140525X00072484

Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, *42*, 230–65. http://dx.doi.org/10.1112/plms/s2-42.1.230

van Rooij, I. (2003). *Tractable cognition: Complexity theory in cognitive psychology* (Unpublished doctoral dissertation). University of Victoria, Victoria, Canada.

van Rooij, I. (2008). The tractable cognition thesis. *Cognitive Science*, *32*, 939–84. http://dx.doi.org/10.1080/03640210801897856

van Rooij, I., Schactman, A., Kadlec, H., & Stege, U. (2006). Perceptual or analytical processing? Evidence from children's and adult's performance on the Euclidean traveling salesperson problem. *Journal of Problem Solving*, *1*(1), 44–73.

van Rooij, I., Stege, U., & Kadlec, H. (2005). Sources of complexity in subset choice. *Journal of Mathematical Psychology*, *49*(2), 160–87. http://dx.doi.org/10.1016/j.jmp.2005.01.002

van Rooij, I., Stege, U., & Schactman, A. (2003). Convex hull and tour crossings in the Euclidean traveling salesperson problem: Implications for human performance studies. *Memory & Cognition*, *31*(2), 215–20. http://dx.doi.org/10.3758/BF03194380

van Rooij, I., Wright, C., & Wareham, H. T. (2012). Intractability and the use of heuristics in psychological explanations. *Synthese*, *187*, 471–87. http://dx.doi.org/10.1007/s11229-010-9847-7

Vazirani, V. V. (2004). *Approximation algorithms*. San Francisco, CA: Springer.

Voloshin, V. I. (2009). *Introduction to graph and hypergraph theory*. Hauppauge, NY: Nova Science Publishers.

Walker, J. J. (2010). *Minesweeper & Hypothetical Thinking Action Research & Pilot Study*. Retrieved from http://www.minesweeper.info/articles/MinesweeperHypothetical-ThinkingActionResearch.pdf

Walwyn, A. L., & Navarro, D. J. (2011). Minimal paths in the city block: Human performance on Euclidean and non-Euclidean traveling salesperson problems. *Journal of Problem Solving*, *3*(1), 93–105.

Wareham, H. T. (1996). The role of parameterized computational complexity theory in cognitive modeling. In C. X. Ling & R. Sun (Eds.), *Working notes of AAAI-96 Workshop on Computational Cognitive Modeling: Source of the Power.*

Wareham, H. T. (1998). *Systematic parameterized complexity analysis in computational phonology* (Unpublished doctoral dissertation). University of Victoria, Victoria, BC, Canada.

Yato, T. (2000). On the NP-completeness of the slither link puzzle. *IPSJ SiG Notes*, *74*, 25–32.

Yato, T., & Seta, T. (2003). Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions: Fundamentals of Electronics, Communications and Computer Sciences*, 86(5), 1052–1060.