Considerations in Representation Selection for Problem Solving: a Review

Aaron Stockdill[1][0000-0003-3312-5267], Daniel Raggi[1][0000-0002-9207-6621], Mateja Jamnik[1][0000-0003-2772-2532], Grecia Garcia Garcia[2][0000-0002-7327-7225], and Peter C.-H. Cheng[2][0000-0002-0355-5955]

[1] University of Cambridge, UK
{aaron.stockdill, daniel.raggi, mateja.jamnik}@cl.cam.ac.uk
[2] University of Sussex, UK
{g.garcia-garcia, p.c.h.cheng}@sussex.ac.uk

Abstract. Choosing how to represent knowledge effectively is a long-standing open problem. Cognitive science has shed light on the taxonomisation of representational systems from the perspective of cognitive processes, but a similar analysis is absent from the perspective of *problem solving*, where the representations are employed. In this paper we review how representation choices are made for solving problems in the context of theorem proving from three perspectives: cognition, heterogeneity, and computational demands. We contrast the different factors that are most important for each perspective in the context of problem solving to produce a list of considerations for developers of problem solving tools regarding representations that are appropriate for particular users and effective for specific problem domains.

Keywords: Representations · Problem solving · Theorem proving.

1 Introduction

Problem solving is a fundamental activity for intelligent agents – people included. The ability to solve problems is dependent on *expertise*, but what is expertise? Can we leverage expertise with one type of problem to solve others? And how can we support problem solvers in choosing the right representation to make their task easier? This paper reviews the current research on representations for problem solving, both from a human-centred perspective, and from a software-centred perspective. Our goal is to understand how we can support problem solvers, and use this knowledge to inform the design of problem solving software – specifically, theorem provers – by compiling a list of considerations for software developers.

We begin this paper in Section 2 by exploring the cognitive aspects of the human reasoning system: how people understand and solve problems, and how expertise affects the solving process. In Section 3 we consider how representations interplay with human reasoning: the different modalities, their effectiveness, and how human reasoners choose representations for their problems. Diagrammatic representations in particular exhibit many of the aspects identified. From the software angle, we consider how problems are solved by automated and interactive theorem provers; Section 4 explores different types of theorem provers, and how some incorporate multiple representations to varying degrees. In Section 5 we analyse the cognitive aspects of these systems, and conclude with a list of considerations for software designers to better align their software with the cognitive needs of the user.

2 Cognitive factors in problem solving

Whilst the importance of choosing the right representation for a problem and a person attempting to solve the problem has long been established [28], computationally choosing the right representation remains an open problem. Much taxonomisation of knowledge representations has been done before, but little in the context of problem solving. Some people are able to solve problems more effectively than others, exhibiting expertise in particular domains. But changing representations, adapting and updating problem solving strategies remains hard for people – it seems these are the skills of experts. We focus on *problem solving* because of its general nature: there is an initial state, some way of identify goal states, and actions that can be taken that modify the state. A wide array of tasks can be modelled as problem solving, so we wish to understand how human experts model problems in order to solve them, and how expertise is related.

2.1 Problem solving

Solving a problem is conjectured to be a tight loop of understanding, planning, executing, and evaluating progress until a condition is met [28]. Pólya's influential work on problem solving, *How to Solve It: A New Aspect of Mathematical Method*, lays out these four steps clearly, presents many varied examples of each step, and exemplifies the loop in its entirety. A more formal treatment of problem solving comes from Simon and Newell, where they introduce the *problem space* [33]. The problem space is modelled as a (possibly infinite) graph, where nodes are the problem state and the edges are the actions that allow movement between them; a walk in this graph originating from the initial state and ending on a goal state is a solution to the problem. The nature of the problem, and the representation of the problem, determine the problem space. The person solving the problem must traverse the problem space.

In this paper, we shall work with Simon and Newell's model of problem solving as our grounding. We choose this model because it maps cleanly to common models of theorem proving, making our discussion more direct. Other ways to frame problem solving (such as Zhang's distributed cognition [47], or Johnson-Laird's mental models [18]) may also function as a suitable model for this discussion, but are beyond the scope of this paper.

When a person is traversing the problem space, some factors are fixed: the fundamentally serial information processing, small-capacity[3] but rapid-recall

[3] Famously, seven plus or minus two *chunks* [22].

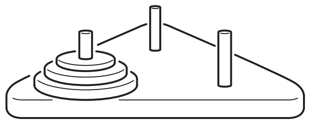


Fig. 1: A typical Tower of Hanoi puzzle. The goal is to move all three discs from the starting peg to another peg, without placing a larger disc atop a smaller disc.

short-term memory, and effectively infinite slow-recall long-term memory. But other factors are mutable, such as *how* the space is traversed. Kotovsky et al. presented participants with variations on the 'Towers of Hanoi' problem, recording how they interacted with the problem and made progress towards (and away from) the solution [19]. The Towers of Hanoi puzzle, depicted in Figure 1, involves three discs with holes stacked atop one peg – a small disc on top, then a medium sized disc, then a large disc at the bottom. Alongside, there are two pegs without any discs. The goal is to move all three discs from the first peg to either of the remaining pegs such that they all end up in one single peg, with the condition that at no point in the process should a larger disc be on top of a smaller disc. Kotovsky et al. analysed how people perform when presented with isomorphic variants of the Towers of Hanoi puzzle, such as monsters-and-globes, boxes-and-dots, or acrobats-and-flagpoles, and with different types of action: either *moving* objects (as in moving a disk from one peg to another) or *changing* their size. Notably, representations involving unfamiliar scenarios (e.g., monsters rather than acrobats), and representations involving changing rather than moving, strongly hindered problem-solving performance, in spite of the problem being isomorphic. Moreover, when facing an unfamiliar problem, participants tended to *probe* the problem space: they would perform a short sequence of actions with minimal deviation from the planned sequence before returning to the initial state. After these probes had been completed, and the participants were satisfied with their ability to traverse the space, they applied short leaps of two actions chained together, rapidly converging on the goal state. These leaps achieved sub-goals, unblocking the next action [19].

Kotovsky et al.'s work has two implications: first, there are two distinct methods of traversing the problem space (the probing back-and-forth approach, and the rapid sub-goal chaining approach); and second, changing the representation of the problem without changing its nature made the Towers of Hanoi-like problems easier or harder. These two results are tightly coupled: the representation of the problem impacted how the participants were able to traverse the problem space, and the participants' relative expertise in the problem space affected how difficult they found the task. To better understand this, we must understand expertise.

2.2 Space traversal and expertise

Algorithmically, there are many ways to traverse a graph: breadth first search, depth first search, A* heuristic search, etc. While people are less procedural, Larkin et al. identify two strategies that solvers use to traverse the problem space: *means-ends analysis* and *knowledge development* [20]. The former is similar to the behaviour seen by Kotovsky et al., using probing then sub-goal unblocking; the latter uses heuristics to avoid the probing and sub-goal analysis to immediately start chaining actions. Further, the solvers who use each strategy can be identified: means-ends analysis is indicative of *novices* in the problem domain, while *experts* employ knowledge development [20].

The strategy of *means-ends analysis*, which is employed by novices, is a type of 'working backwards'. The solver must identify what necessary conditions must be met to move towards the goal, and then work towards this new sub-goal [19]. Thus the novice begins to probe the problem space, understanding what effect their actions have, and then can begin to achieve their sub-goals. Maintaining this internal sub-goal chain is cognitively demanding, using working memory that could otherwise be devoted to the problem itself, not the 'traversal state'; even small problem spaces overwhelm human working memory [19].[4] Worse, the high cognitive load required to employ means-ends analysis can inhibit *schema acquisition*, a method of becoming an expert [40].

Expert problem solving is best modelled through *knowledge development*, in which powerful heuristics guide the expert through the problem space [40]. Because experts are familiar with the domain – and thus the problem space – there is little to no 'probing' phase; they have seen and solved similar problems in the past. Instead, experts can immediately begin applying *schemas*, which are patterns that the expert can recognise in the new problem space, and so immediately apply actions [40]. Not only does this approach eliminate the probing and sub-goal creation, this approach induces less cognitive load – the utilisation of working memory – than means-ends analysis [40]; experts will be faster *and* more cognitively efficient.

2.3 Cognitively effective representations

A representation is a view of a problem: the problem is expressed using some representation. The representation itself belongs to some representational *system*: a collection of syntax and semantics that generate some agreed-upon notation and interpretation. This is sometimes called an *external* representation because it exists outside the mind; there is a corresponding *internal* representation that exists within the mind of the problem solver [30]. Cheng links internal and external representations in two directions: an appropriate external representation can induce an effective internal representation, while an effective internal representation encourages external representation generation [7].

With a diverse range of representational systems at our disposal, some with more diagrammatic aspects than others, we must consider: what makes a representation *effective*? In the context of problem solving, there are quantifiable

[4] By analogy to computers, we devote 'registers' that would otherwise be used on the problem to maintaining the 'call stack', but the human brain's 'call stack' capacity is small, and – due to the nature of graph search – easy to overflow.

results we might be interested in: lower cognitive load, shorter times to generate a solution, shorter solution paths. But in this subsection we look at the representations themselves, not the results they generate: in order to achieve these results, what properties do our representations have?

We consider effective external representations in relation to the internal representations they induce. Green and Blackwell created the 'Cognitive Dimensions' framework as a guide on creating representations, but note that it is not intended for a deep analysis of existing representations [13].[5] Instead we here consider the 5 general criteria containing the more specific 19 criteria identified by Cheng [7] for effective representations. These are: direct encoding, low-cost inference, conceptual transparency, generality, and conceptual-syntactic compatibility. In the next section, we will compare them to diagrammatic aspects of representations.

Direct encoding Cheng's first criterion for effective representations is that it *directly encodes* the types, structures, and relations of the problem [7]. This is the same benefit that diagrammatic representational systems provide. But *why* is this necessary for a representation to be effective? Consider, for example, Duncker's 'candle problem': given a box of tacks, some matches, and a candle, attach the candle to the wall [45]. Participants will attempt to tack the candle to the wall, or melt some wax to use as glue, neither being effective; rarely do they consider they could pin the tack box to the wall and sit the candle in the box [45]. Condell et al. call this inability to re-contextualise the tack box *functional fixedness*: the 'type' of the box is wrong, since in the 'representation' people have, the box is a container for tacks, not a container for candles as required for the solution [8]. Tversky highlights a similar point in regards to structure: people have a mental hierarchy to categorise their environment, and benefit when the representation follows the same hierarchy [41].[6] Thus a representation that more directly encodes a problem is likely more effective than those that encode the problem indirectly.

Low-cost inference The cost of inference in representations is a combination of factors: while the inferential actions themselves should be low-cost to perform, they must also be low-cost to identify [7]. In diagrammatic representational systems, this is a mixture of geometric and spatial aspects, syntactic constraints, and syntactic plasticity. One notable variety of low-cost inference is the *free ride* – an inference that can be made without specifically taking steps to make that inference [31]. Stapleton et al. generalise this to *observational advantages*: some representations allow information to be observed 'for free' that would require purposeful inference in other representations [35]. 'Free' is certainly low-cost; representations exhibiting observational advantages are likely to be more effective than their disadvantaged counterparts.

[5] Although work that builds upon these dimensions (e.g., [4]) often includes concepts very close to those we are about to discuss.

[6] In this case, the hierarchy is that the box is restricted to tacks, and there is no hierarchical relationship to the candle; the *necessary* hierarchy has box restricted to *objects*, which includes tacks and candles.

Conceptual transparency More difficult to define, conceptual transparency is the ability to 'see through' the representation to its underlying meaning. Cheng decomposes conceptual transparency into five aspects: coherence and unambiguity; small conceptual gulf; integration of conceptual perspectives; integration of granularity scales; and the comparing and contrasting of typical, special, and extreme cases [6]. These are themselves difficult to resolve, and are beyond the scope of this review.

Generality It is desirable that a representational system allows us to represent a variety of situations (generality) and, equally important, that there are available a variety of syntactic operations that allow us to traverse the space (generativity) towards desirable goals. This relates to the syntactic power of the systemdiscussed next.

Conceptual-syntactic compatibility Cheng's final criterion for effective representations is conceptual-syntactic compatibility. In diagrammatic representational systems, this is related to the idea of syntactic constraints: a close relationship between 'expressible' and 'valid' results in a more effective system [7]. By analogy, in computer science we discuss making illegal states unrepresentable [23] for the same effect: if you *cannot* say something incorrect, then you have reduced the ways in which you can make a mistake. Other mechanisms through which representations have conceptual-syntactic compatibility is by having a construction process which mirrors the problem solving process, and by allowing for distinct phases in encoding, interpreting, and making inferences [7].

In the problem solving context, we can consider an effective representation to be one that provides a problem space in which the solver is sufficiently expert: they already have access to low-cost inferences and powerful schemas. All the above criteria contribute to making the space easier to traverse for the solver. However, it is worth noting that they are not independent. For example, while a general representational system might be more likely to guarantee the existence of a solution, the problem space will often be more 'branchy', making correct inferences more costly. This compromise between generality and low-cost inference is captured by Cheng's concept of *syntactic plasticity* [7]. This sort of compromise indicates that choosing an effective representation is a complex optimisation problem where many competing factors have to be weighted. Furthermore, often they cannot be weighted equally for solvers with differing levels of expertise.

Novices and experts alike solve problems by traversing a problem space, applying actions to change state within the space such that they eventually reach a goal state. But their traversal methods are very different: novices have a costly, means-ends analysis approach to searching the problem space; experts apply powerful heuristics called schemas to efficiently work from the start to the goal. Clearly, being an expert is advantageous: can we somehow transfer these advantages to a novice? Or perhaps, can we change the problem space so that our novice is already expert?

3 Heterogeneity of representations

As Kotovsky et al. discovered, the way a problem is represented can significantly impact how difficult the problem is to solve [19]. But why is this, and what exactly is involved in the representation of a problem? In this section we consider different modalities of representations, and what it means for a problem to be represented effectively.

3.1 Diagrammatic aspects of representations

Restricting ourselves to external representations, we can attempt to classify representations: a common distinction is between 'sentential' – a sequence of characters composed only through concatenation [37] – and 'diagrammatic' representations.[7] Despite their apparent value of '10 000 words' [21], diagrammatic representations are often second-class in mathematics, even in highly visual domains such as graph theory. Only in the last 25 years have diagrammatic representational systems begun to be treated formally to address this gap. Informally, diagrammatic representations are widely used by mathematicians; formally, diagrams are often stripped from the discussion, because mathematicians consider them unsuitable for proof [16]. Even educational materials such as textbooks often present only sentential solutions to problems, obscuring any intuition that a diagram can provide [46]. Perhaps it is because diagrammatic systems are difficult to define: what makes a diagrammatic representation *diagrammatic*?

Taken in the extremes, there is obvious consensus around which representations are 'sentential' and which are 'diagrammatic': in mathematics, standard propositional logic notation is sentential, while Euler diagrams are diagrammatic. But as we drift away from these extremes, the boundary becomes indistinct: positioning limits on a summation is not concatenative, and hints towards some vertical-positioning relationship; a table filled with words uses space and positioning to encode information, but uses strings extensively. The distinction is difficult because, as Giardino observes, there is no sharp distinction to be made [10]. Representations exist on a continuum, some with more diagrammatic aspects than others; when we discuss diagrammatic representations we are referring to representations exhibiting four diagrammatic aspects: direct encoding, syntactic constraints, syntactic plasticity, and heavy use of geometric and spatial attributes and relations. Let us consider each of these in more detail.

Direct encoding Diagrammatic representations directly encode the types, structures, and relations of the problem, rather than using some indirect association as in sentential representations [37]. Consider a relation 'to the right of': we can easily state an instance of this sententially:

a is to the right of b

while observing that a is visibly *left* of b. By comparison,

[7] We consider only *visual* representations; representations that are audial or tactile, for example, are beyond the scope of this review.

$$b \qquad a$$

is a more *direct* encoding: a is literally to the right of b. This extends to all levels: rather than using the word 'square', diagrams can include squares; rather than explaining how nodes and edges form a graph, we can draw the graph. But this can also enforce specificity: we can sententially state that 'a zebra has some stripes', without making any claim to how many stripes, but any particular drawing of a zebra has a *fixed number* of stripes. This makes the representation easier to process at the cost of reducing generality [38].

Syntactic constraints Shimojima observed that rules of a representational system come in two broad classes: intrinsic, and extrinsic[8] [32]. An *intrinsic* (or *syntactic*) constraint is imposed by the syntax of the representational system: the geometry, topology, or physics of the representation enforces the rules. An *extrinsic* constraint is imposed by the problem solver: the representational system allows for statements that the solver wishes to avoid. Going back to our 'to the right of' example, let us assume a system where we can write $a >_r b$, meaning a is to the right of b.[9] Then we can state the following three facts:

$$a >_r b,\ b >_r c, \text{ and } c >_r a$$

Now, if we try to represent this in our 'positional' notation from earlier, we hit an *intrinsic* constraint: we cannot arrange the letters on the page such that this is true! The representational system has prevented us from representing some state. On a plane, the sentential notation is too permissive: we failed to apply the *extrinsic* constraints necessary to identify a nonsense statement. But on a sphere, the positional representation is overly restrictive: the intrinsic constraints are preventing us from encoding a valid state.

Geometry and space Finally, diagrammatic representations make use of *geometry* and *space* [37]. The benefit of this is that it exploits the human visuo-spatial reasoning system – the Towers of Hanoi variants presented by Kotovsky et al. to participants consistently demonstrated that participants more efficiently solved the 'physically plausible' variants [19]. Humans evolved in a physical world that obeys particular rules: we are well-adapted to manage systems that follow these rules. But geometry and space are limiting; just as we identified in direct encoding and syntactic constraints, we forfeit *abstraction* and *generality* by following the physical rules.

Characterising diagrammatic representations by these properties exhibits why diagrams fit various criteria from Section 2.3 for effective representations.

3.2 Recommending a representation

We have seen that representations can affect how difficult a problem is to solve and argued that diagrammatic representations often exhibit favourable aspects for problem solving. Undeniably, changing to an effective representation

[8] [32] identified variations on this divide, but all are sufficiently similar for our discussion.
[9] Note again that, visually, a is to the *left* of b.

is useful [1,6,12] – the problem is that students do not necessarily *change* to a more effective representation [39]. Ideally, we want to support students to change representation to one that is appropriate for the problem and for their expertise. But how should they be guided to change towards effective representations?

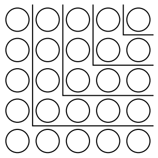
In the restricted domain of extracting information from a database, Grawemeyer's External Representation Selection Tutor (ERST) was able to recommend an information visualisation to users to answer queries [12]. The visualisations were scatter plots, sector graphs, pie charts, bar charts, tables, and Euler diagrams; when supported by ERST in choosing an effective representation, participants were more effective at answering the queries [12]. But to consider tasks beyond information extraction, the literature on representation recommendation becomes scarce. To *solve* a problem, we explore the representation *design* literature: what factors are important when designing representational systems, which we may consider for representation recommendation?

Representation design recommendation is a product of three factors: what is the problem, who is approaching it, and why are they working on it? This combination of factors determines the *cognitive fit* of a representation [24,44]. Vessey introduces cognitive fit as the combination of the specific problem under consideration, and the overarching task and context in which the problem is encountered, which together influence the internal representation a person constructs [44]. But implicit in Vessey's discussion is that the *person* influences the internal representation; as we saw earlier, an expert and a novice will be operating with different internal representations: the novice's internal representation is tuned for search, and the expert's for heuristics [20]. Moody makes this explicit: cognitive fit is the interaction between the problem, the person, and the task [24].

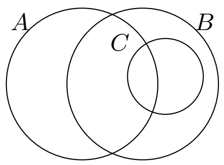
A representation is a complex thing: it is an encoding of information into the real world, which induces an *internal* representation in people. A range of factors determine representational efficacy, and diagrammatic aspects of representations align to allow for effective representations. By considering cognitive fit, we can begin to understand how to recommend a representation based on the problem being solved, the person solving the problem, and the task and context in which the problem was encountered.

4 Computational considerations of representation

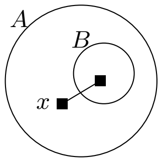
In the previous two sections we considered why and how representations are evaluated and recommended. In this section, we explore the use of representations in computational systems. While artificial intelligence researchers have attempted to build general problem solvers for a long time – consider the aptly named 'General Problem-Solving Program' [25] – most success has been had in solvers specialised to particular domains. We focus on interactive and automated theorem provers, as this class of software is forced to consider concerns similar to ours: solving problems, representing them effectively, and considering their users.



(a) $\sum_{i=1}^{5}(2i-1) = 5^2$



(b) $A \cap B \neq \emptyset, C \subset B$



(c) $B \subset A, \exists x \in A$

Fig. 2: (a) Dot diagrams, (b) Euler diagrams, and (c) Spider diagrams.

4.1 Homogeneous systems

Theorem provers are used by people to solve a very specific type of problem: given some assumptions, derive a specific conclusion. To make progress, the set of assumptions is updated using already-proved theorems (or axioms) through inference mechanisms. This maps directly to the problems space we discussed: the current state is the current set of assumptions, a goal state is any set of statements which contains the desired conclusion, and the actions to move between states are the inference mechanisms. So the difference between the theorem provers is the state space they model – and so the representations they exploit.

Most theorem provers are *homogeneous*: that is, they use a single representational system. This single representational system is usually sentential, but the details vary. One family of theorem provers are those based on type theory: two notable members are Coq [15], and Nuprl [9]. These systems use Martin-Löf type theory as their foundational system, and proofs are constructions of a value that has the type which is an encoding of the theorem to prove. A second notable family of theorem provers are those with HOL/LCF[10] ancestry [11]: HOL4 [34], HOL Light [14], and Isabelle/HOL [26].[11] These systems use a small core of actions that is intended to be easy to verify, and all other actions must be built on top of this core. Both families use a syntax that is programming-language-like, and purely sentential.

Equally homogeneous, but no longer sentential, are the family of *diagrammatic* theorem provers. DIAMOND focuses on diagrammatic proofs of arithmetic using grids of dots (Figure 2a), and ways of partitioning the grid [17]. The high-level approach of DIAMOND is different to that of the sentential provers mentioned earlier: it works with instances of a proof and generates a generalised version automatically, rather than expecting the person proving the theorem to work in the most general case at all times. Edith, and its successor Speedith, focus on Euler diagrams (Figure 2b) and Spider diagrams (Figure 2c), respectively [36,43].

[10] HOL stands for higher-order logic, an extension of predicate/first-order logic. LCF stands for the logic for computable functions, a theorem prover based on the logic *of* computable functions.

[11] Isabelle (without ‘HOL’) is a *meta*-logic system: a developer tailors Isabelle to work in their particular system. For example, Isabelle/ZF allows people to use Zermelo-Fraenkel (ZF) set theory rather than higher-order logic. This is an interesting step towards heterogeneity, but the different logics are inaccessible to each other.

Their proof structure more closely resembles that of the sentential systems: from some diagrams you can construct a new diagram; analogously, from some assumptions you derive a new conclusion. These systems show that software is capable of supporting diagrammatic representational systems, yet they do not yet push the bounds to *heterogeneous* reasoning: exploiting multiple representations.

4.2 Semi-heterogeneity

Homogeneous theorem provers have continued to grow in sophistication and power, but their generality comes at the cost of speed: certain problems are best left to dedicated tools that have a better representation for that problem. In the HOL/LCF tradition, these tools are integrated as *hammers* [5]. For example, Isabelle/HOL uses 'Sledgehammer' to transform a higher-order logic problem into a first-order logic problem before passing the transformed problem (along with a set of relevant lemmas) to automated first-order logic provers; the proof is returned to Isabelle/HOL, and validated in the verified core like any other proof [27]. While not obviously heterogeneous – every representational system involved is sentential – Isabelle/HOL exploits a system with a more effective problem space by transforming the problem.

Isabelle has a second means of semi-heterogeneous reasoning: the *Transfer* package. The Transfer package was designed as a general tool for code generation, and for the development of quotient types and subtypes. Raggi et al. used Transfer as the basis for a tool for heterogeneous reasoning [29], so that statements are transferred across a network of theories that formalise natural numbers in various ways. With this tool, theorems about natural numbers can be proved under their representation as either successors of zero, multisets of primes, classes of finite sets, and others. The one heuristic used to select between potential representations is the size of sentences. Otherwise, the task of selection is left to the user.

We consider this approach semi-heterogeneous, as it enables the transformation of sentences across different mathematical theories, while remaining in purely sentential representational systems.

4.3 Fully heterogeneous theorem provers

We move now from homogeneous or purely sentential systems to heterogeneous reasoning systems. An early and notable heterogeneous system, Hyperproof, was an educational tool for first-order logic that used a three-dimensional chessboard environment alongside a more typical sentential representational system [3]. The actions available in the two representational systems were different, as would be expected; proofs in the sentential first-order logic system are often more verbose than their chessboard counterparts [3]. Barker-Plummer et al. generalised Hyperproof to Openproof, a framework allowing heterogeneous reasoning with many different representational systems [2]. The framework avoids an *inter-lingua* (common language) but maintains a common proof state; this avoids some 'lowest-common-denominator' expressiveness concerns while maintaining a valid proof. But as a result, there is a tight coupling between the representational systems in Openproof: there is a one-to-one correspondence between the objects and relations in each representation, and formal translations between them.

MixR is a heterogeneous theorem proving framework that grew out of a desire to integrate Speedith, the spider diagram reasoner, with Isabelle [42]. The MixR framework consisted of two parts: one theorem prover that 'owned' the proof state, and many 'working' theorem provers that could modify the proof state. MixR aimed to reuse existing theorem provers, rather than develop specialist heterogeneous theorem provers. Moreover, MixR allowed for unsound transformations between the representational systems used by each of the 'working' theorem provers. MixR also introduced heterogeneous statements – using multiple representational systems simultaneously – through *placeholders*.[12] MixR, like all the heterogeneous systems we have discussed, provides the *option* for heterogeneous reasoning. But it does not *encourage* or *guide* heterogeneous reasoning: representation selection is a human-driven process.

This section explored how current software, designed to work towards solving problems alongside a person, manages the issue of representation. For the most part, software systems maintain a single representational system, whether sentential or diagrammatic. Some software is heterogeneous, notably MixR: it allows the user to combine multiple representational systems together to solve a single problem, in particular, allowing for informal transformations between representations. But the decision on which representation system to use at any given point is driven by the user – while multiple representations may be available, the user is *not* helped to use them.

5 Cognitive analysis of computational systems

Each of these computational systems comes with compromises: their representational systems exist at different points along each of the cognitive factors we have looked at, with varying degrees of heterogeneity. This in turn interacts with their generality. We would encourage the developers of existing and future problem solving software to be cognisant of the compromises they are making for the cognitive benefit of their users.

Many of the systems we explored are sentential, meaning they can be incredibly general tools at the cost of abstraction. By being able to encode effectively all of mathematics, the encoding becomes less *direct*, with less *conceptual transparency*, and potentially more effort is required by the user to map the notation to the problem. Exceptions to this are tools like DIAMOND, Edith, and Speedith, which trade *generality* for directness: DIAMOND encodes natural numbers as arrangements of dots, Edith encodes sets as Euler diagrams, and Speedith directly encodes sets with existential elements as spider diagrams. They restrict their domain to afford a more direct representation of said domain.

Similarly, we see these diagrammatic tools lower the *inference cost* for the user in their domains, again trading generality for lowered cognitive cost. The user can

[12] For example, $(x = \square) \rightarrow (\text{shape}(x) = \texttt{square})$ places the square in the statement.

often exploit the observational advantages of the diagrammatic representations over more general sentential representations. In contrast, sentential tools like Isabelle or Coq typically exhibit fewer observational advantages, meaning even simple inferences still require effort.

This impacts the accessibility of these systems. On the one hand we have domain-specific diagrammatic tools which exhibit all the cognitive benefits of diagrams and, on the other hand, we have very general tools that require high skill from the user. Systems such as Hyperproof were designed with the goal of education from the very beginning: the creators understood the value of re-representation and diagrammatic aspects as a powerful tool for novices to reach for. Conversely, Isabelle and Coq are designed for research-level mathematics: learning to use these tools is *hard*, but their generality and power reward the effort. Neither is 'better' – these are different classes of tools with different audiences.

The heterogeneous systems we examined (MixR and Hyperproof) have seen most of their success limited to research, with only limited deployment or applications in the real world (Openproof). As we have argued in this paper, cognitive considerations are of utmost importance for the selection of effective representations for problem solving. Thus, we contend that the limited success of these systems is due to their lack of flexibility in regards to the variety of users with different levels of expertise and familiarity. This flexibility could be harnessed through intelligent recommendation by estimating the cognitive fit of representation using the criteria presented in this paper. As we argued, this is necessarily a complex assessment that involves the calculation of trade-offs between competing criteria.

In summary, we recommend software designers consider the following:

Consideration	**Explanation**
Direct encoding	Information is usually easier to extract, but the representation is typically less general.
Low-cost inference	Make each step simple, and the next step easy to identify. These are relative to your target users. Exploit free rides.
Generality	Be general enough to be useful, but not so general as to be confusing. More expert users can typically handle more general representations.
Conceptual-syntactic compatibility	Make it 'easy' to do the right thing, but 'hard' to do the wrong thing. Easy and hard depend on the target users.
Syntactic constraints	The representation imposes rules, rather than forcing users to remember them. More constraints typically limit generality.
Geometry and space	Favour physically plausible interactions. This typically limits the ability to generalise.

No one representation will be perfect for every user and every problem, so we encourage heterogeneous (but coherent) systems: use direct representations with strong syntactic constraints when available, but more general representations when flexibility is needed. We hope that software designers will be more open to different modalities and develop new, more effective tools to solve problems.

One final recommendation is for the software to *guide* heterogeneous reasoning by making the right representation available at the right time. Consider:

theorem provers that suggest illuminating diagrams; spreadsheets that encourage explanatory charts; video editing software that intelligently switches between timelines and nodes. We wish to encourage software that dynamically switches between – not just allows – varied representations. Research in this direction could open up new opportunities in human-computer collaboration.

6 Conclusion

We examined in this paper how human problem solving can be modelled as traversing a problem space: the solver is attempting to reach goal states by applying actions to the current state. The expertise of the solver impacts their ability to navigate the problem space, but by selecting an effective representation we can induce a problem space in which the solver is already expert. The nature of the representation determines its effectiveness, and specific aspects – each with trade-offs – are generally agreed to be better; conveniently, these align with diagrammatic aspects of representational systems. We also discussed how representations are used in software, specifically theorem proving software: few support heterogeneous reasoning, and those that do, fail to support the user in selecting an appropriate representational system. We encourage developers of these problem solving tools to be aware of how the design decisions they are making impact the cognitive aspects of their tools, and what effect this will have upon their users. This review identifies and illuminates some of the factors impacting those decisions.

Acknowledgements Aaron Stockdill is supported by the Hamilton Cambridge International Scholarship. This work was supported by the EPSRC grants EP/R030650/1, EP/T019603/1, EP/R030642/1, and EP/T019034/1.

References

1. Ainsworth, S.: The Educational Value of Multiple-representations when Learning Complex Scientific Concepts, chap. 9, pp. 191–208. Springer (2008)
2. Barker-Plummer, D., Etchemendy, J., Liu, A., Murray, M., Swoboda, N.: Openproof - a flexible framework for heterogeneous reasoning. In: Stapleton, G., Howse, J., Lee, J. (eds.) Diagrammatic Representation and Inference, Diagrams 2008. Lecture Notes in Computer Science, vol. 5223, pp. 347–349. Springer (2008)
3. Barwise, J., Etchemendy, J.: Visual information and valid reasoning. In: Allwein, G., Barwise, J. (eds.) Logical Reasoning with Diagrams, chap. 1, pp. 3–25. Oxford University Press, Inc., New York, NY, USA (1996)
4. Blackwell, A.F., Whitley, K.N., Good, J., Petre, M.: Cognitive factors in programming with diagrams. Artificial Intelligence Review **15**(1), 95–114 (2001)
5. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards qed. Journal of Formalized Reasoning **9**, 101–148 (2016)
6. Cheng, P.C.H.: Electrifying diagrams for learning: principles for complex representational systems. Cognitive Science **26**(6), 685 – 736 (2002)

7. Cheng, P.C.H.: What constitutes an effective representation? In: Jamnik, M., Uesaka, Y., Elzer Schwartz, S. (eds.) Diagrammatic Representation and Inference, Diagrams 2016. Lecture Notes in Computer Science, vol. 9781, pp. 17–31. Springer (2016)
8. Condell, J., Wade, J., Galway, L., McBride, M., Gormley, P., Brennan, J., Somasundram, T.: Problem solving techniques in cognitive science. Artificial Intelligence Review **34**(3), 221–234 (2010)
9. Constable, R.L., Allen, S.F., Bromley, H.M., Cleaveland, W.R., Cremer, J.F., Harper, R.W., Howe, D.J., Knoblock, T.B., Mendler, N.P., Panangaden, P., Sasaki, J.T., Smith, S.F.: Implementing Mathematics with the Nuprl Proof Development System. Prentice-Hall, Inc., USA (1986)
10. Giardino, V.: Towards a diagrammatic classification. The Knowledge Engineering Review **28**(3), 237–248 (2013)
11. Gordon, M.J., Milner, A.J., Wadsworth, C.P.: Edinburgh LCF: A Mechanised Logic of Computation, Lecture Notes in Computer Science, vol. 78. Springer, 1 edn. (1979)
12. Grawemeyer, B.: Evaluation of ERST – an external representation selection tutor. In: Barker-Plummer, D., Cox, R., Swoboda, N. (eds.) Diagrammatic Representation and Inference, Diagrams 2006. pp. 154–167. Lecture Notes in Computer Science, Springer (2006)
13. Green, T., Blackwell, A.: Cognitive dimensions of information artefacts: a tutorial. In: BCS HCI Conference. vol. 98 (1998)
14. Harrison, J.: HOL Light: An overview. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2009. Lecture Notes in Computer Science, vol. 5674, pp. 60–66. Springer, Munich, Germany (2009)
15. Huet, G., Kahn, G., Paulin-Mohring, C.: The Coq proof assistant: A tutorial. The Coq Project (2004)
16. Inglis, M., Mejía-Ramos, J.P.: On the persuasiveness of visual arguments in mathematics. Foundations of Science **14**(1), 97–110 (2009)
17. Jamnik, M., Bundy, A., Green, I.: On automating diagrammatic proofs of arithmetic arguments. Journal of Logic, Language and Information **8**(3), 297–321 (1999)
18. Johnson-Laird, P.N.: Models and heterogeneous reasoning. Journal of Experimental & Theoretical Artificial Intelligence **18**(2), 121–148 (2006)
19. Kotovsky, K., Hayes, J., Simon, H.: Why are some problems hard? evidence from tower of hanoi. Cognitive Psychology **17**(2), 248–294 (1985)
20. Larkin, J.H., McDermott, J., Simon, D.P., Simon, H.A.: Models of competence in solving physics problems. Cognitive Science **4**(4), 317–345 (1980)
21. Larkin, J.H., Simon, H.A.: Why a diagram is (sometimes) worth ten thousand words. Cognitive Science **11**(1), 65–100 (1987)
22. Miller, G.A.: The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychological Review **63**(2), 81–97 (1956)
23. Minsky, Y.: Effective ML revisited. Blog Post (accessed 28 Nov. 2020). `https://blog.janestreet.com/effective-ml-revisited/` (2011)
24. Moody, D.: The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Transactions on Software Engineering **35**(6), 756–779 (2009)
25. Newell, A., Shaw, J., Simon, H.A.: Report on a general problem-solving program. In: Proceedings of the International Conference on Information Processing, pp. 256–264 (1959)
26. Paulson, L.C.: The foundation of a generic theorem prover. Journal of Automated Reasoning **5**(3), 363–397 (1989)

27. Paulson, L.C., Blanchette, C.: Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers (2010)
28. Pólya, G.: How to Solve It: A New Aspect of Mathematical Method. Princeton Science Library, Princeton University Press (1957)
29. Raggi, D., Bundy, A., Grov, G., Pease, A.: Automating change of representation for proofs in discrete mathematics (extended version). Mathematics in Computer Science **10**(4), 429–457 (2016)
30. Scaife, M., Rogers, Y.: External cognition: how do graphical representations work? International Journal of Human-Computer Studies **45**(2), 185–213 (1996)
31. Shimojima, A.: Operational constraints in diagrammatic reasoning. In: Allwein, G., Barwise, J. (eds.) Logical Reasoning with Diagrams, chap. 2, pp. 27–48. Oxford University Press, New York, NY, USA (1996)
32. Shimojima, A.: The graphic-linguistic distinction. In: Blackwell, A.F. (ed.) Thinking with Diagrams, pp. 5–27. Springer (2001)
33. Simon, H.A., Newell, A.: Human problem solving: The state of the theory in 1970. American Psychologist **26**(2), 145–159 (1971)
34. Slind, K., Norrish, M.: A brief overview of HOL4. In: Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics, TPHOLs '08. pp. 28–32. TPHOLs '08, Springer (2008)
35. Stapleton, G., Jamnik, M., Shimojima, A.: What makes an effective representation of information: A formal account of observational advantages. Journal of Logic, Language and Information **26**(2), 143–177 (2017)
36. Stapleton, G., Masthoff, J., Flower, J., Fish, A., Southern, J.: Automated theorem proving in euler diagram systems. J Autom Reasoning **39**(4), 431–470 (2007)
37. Stenning, K., Lemon, O.: Aligning logical and psychological perspectives on diagrammatic reasoning. Artificial Intelligence Review **15**(1), 29–62 (2001)
38. Stenning, K., Oberlander, J.: A cognitive theory of graphical and linguistic reasoning: Logic and implementation. Cognitive Science **19**(1), 97–140 (1995)
39. Superfine, A.C., Canty, R.S., Marshall, A.M.: Translation between external representation systems in mathematics: All-or-none or skill conglomerate? The Journal of Mathematical Behavior **28**(4), 217 – 236 (2009)
40. Sweller, J.: Cognitive load during problem solving: Effects on learning. Cognitive Science **12**(2), 257–285 (1988)
41. Tversky, B.: Visualizing thought. topiCS **3**(3), 499–535 (2011)
42. Urbas, M., Jamnik, M.: A framework for heterogeneous reasoning in formal and informal domains. In: Dwyer, T., Purchase, H., Delaney, A. (eds.) Diagrammatic Representation and Inference, Diagrams 2014. Lecture Notes in Computer Science, vol. 8578, pp. 277–292. Springer (2014)
43. Urbas, M., Jamnik, M., Stapleton, G., Flower, J.: Speedith: A diagrammatic reasoner for spider diagrams. In: Cox, P., Plimmer, B., Rodgers, P. (eds.) Diagrammatic Representation and Inference, Diagrams 2012. pp. 163–177. Lecture Notes in Computer Science, Springer (2012)
44. Vessey, I.: Cognitive fit: A theory-based analysis of the graphs versus tables literature. Decision Sciences **22**(2), 219–240 (1991)
45. Weisberg, R., Suls, J.M.: An information-processing model of Duncker's candle problem. Cognitive Psychology **4**(2), 255 – 276 (1973)
46. Zazkis, D., Weber, K., Mejía-Ramos, J.P.: Bridging the gap between graphical arguments and verbal-symbolic proofs in a real analysis context. Educational Studies in Mathematics **93**(2), 155–173 (2016)
47. Zhang, J.: The nature of external representations in problem solving. Cognitive Science **21**(2), 179–217 (1997)