radboud univerisy

Master Thesis

Computational-level analysis of insight problem solving

A thesis submitted in fulfillment of the requirements for the degree of Cognitive Neuroscience

Author:
Stefano Gentili [1]

Supervisors:
Iris Van Rooij[2]
Mark Blokpoel[2]
Todd Wareham[3]

[1]Master student at the Donders Institute for Brain, Cognition, and Behaviour, Radboud University, Kapittelweg 29, 6525 EN Nijmegen, The Netherlands
[2]Donders Institute for Brain, Cognition, and Behaviour, Radboud University, Kapittelweg 29, 6525 EN Nijmegen, The Netherlands
[3]Department of Computer Science, Memorial University of Newfoundland, St.John's, Newfoundland, Canada, A1B 3X5

Contents

Abstract

Having an insight is a central aspect of the human ability to do problem solving. When trying to reach a solution for a problem, having an insight is what lets us formulate a problem in a way that we can come up with an answer. However there is no scientific consensus about the cognitive mechanisms of reaching insight. In this thesis we will briefly present the current literature about insight problem solving and show some of the shortcomings that have been affecting it. In particular, we will ask if existing models of insight problem solving are able to generalize and be applicable to all insight problems or only to a selected few they are specifically designed for. We will focus on an existing model of insight problem solving, and give a computationally-backed argument about how the existing model could theoretically be used to encode any type of problem. We will also give an example of an encoding of an insight problem in the existing model and from there we will point out that the model seems to construct its input so that the solution is easily found, almost built in to the model. We will argue about the risks of this in-building and then we will consider what could be a minimum to be built-in. This will lead to a reformulation of a more useful model capable of insight. Finally, with another result we will show that an important aspect of problem solving is often overlooked, namely the amount of time (or steps) necessary before finding the solution. Indeed we will show that pre-specifying this amount is actually necessary for a cognitive model of insight.

These results will give important theoretical constraints to future theories of insight problem solving. Furthermore, the thesis will suggest specific future research approaches for advancing our knowledge in this field.

1 Introduction

1.1 Motivation

Historians narrate how Charlemagne, in around 780 CE, recruited scholars from all around Europe to teach at his court. One of them, Alcuinus of York, regarded at the time as 'The most learned man anywhere to be found' [Einhard, 1960], allegedly wrote a book titled 'Propositiones ad acuendos juvenes' (Problems to Sharpen the Young) to assist his teaching. This is one of the first known collections of recreational mathematical and logical problems, many of which are still well-known today. For example, Problem 18 asks you how to ferry a wolf, a goat, and a cabbage across a river, without the wolf eating the goat or the goat eating the cabbage if left unsupervised, and given that you can transport only one of them on each crossing. Another famous one is Problem 42: in a 100 steps staircase, with 1 pigeon sitting in the first step, 2 sitting in the second, and so on to 100 pigeons sitting in the last one, how many pigeons are there in total? Alcuinus' solution was, instead of naively counting them all, to notice that you could sum up the pigeons in step 1 and 99 to 100, and similarly for pigeons in steps 2 and 98, and so on till steps 49 and 51: this would give us 49 pairs of 100 pigeons plus the pigeons in step 100 and 50, so the solution would be 49*100+100+50=5050. Allegedly, close to a thousand years later, Friedrich Gauss as a pupil solved the same puzzle in a more elegant way by noticing you could sum all the pigeons in 50 pairs if you modify the belief that the sum had to be 100: in fact one could sum step 1 and 100 to 101, step 2 and 99 to 101, etc. giving the solution as 101*50=5050.

Examples like these, and countless more from our experience, show us how many different forms problem solving might have. The importance of problem solving is also clear: human history itself, from our latest scientific discoveries to the indigenous tribe's survival skills, is a reminder of how problem solving is one of the central characteristics that define human beings. If the aim of cognitive science is to understand all aspects of cognition, then problem solving research is of fundamental value to scientific progress.

Yet, research in problem solving has encountered a stalemate: since the first works of Newell and Simon [Newell and Simon, 1972] it doesn't seem that a complete theoretical understanding has been achieved [Batchelder and Alexander, 2011] [Ohlsson, 2011]. Although Newell and Simon's line of research had great influence in the early history of cognitive science, and there have been many implementations of their original architecture [Laird et al., 1987], their framework is also beginning to show some limitations.

In their framework, problem solving is treated as a program exploring a space of possible solutions guided by heuristics. Yet to successfully find a solution the program is highly dependent on a correct encoding of the initial state of the problem and of the search rules that make it possible to reach a solution. This encoding permits the exploration of the various combinations of the search rules over the initial state, until the goal state is found. Yet this encoding of search rules and input states into the program depends more on the programmer's ability to choose a initial state and search rules capable of reaching a solution, rather than the program's actual ability to solve the problem.

More importantly, this lack of an ability to change a problem's representation if necessary leads to the impossibility for the program, in case of being incapable of finding a solution, to achieve one of the most important aspects of problem solving: to have an insight, to re-encode the problem in a way that could lead to a solution.

For example, in the "wolf, goat, cabbage" problem defined above, we could encode the initial problem state in a program as 3 objects (the wolf, goat, and cabbage) on one side of the river and the search rule can be represented as moving just one object at a time to the other side (Figure 1). The goal state is to have all three objects on the other side of the river, without ever having two not permitted objects on the same side. A program could now solve the problem by checking all combinations of moves (a method also known as brute force) until one is found that leads to the desired solution. Yet, an important point to note is that no solution can ever be found if the transition rules don't let some object to be ferried back once they have reached the other shore. So this insight has to be encoded in the starting rules and is impossible for the program to achieve.

When solving a problem by exploring the search space of possible solutions, insight is necessary to select the starting states and the moves applicable to the states to reach the solution. This is a task that humans seem capable of solving, yet its understanding and empirical investigation is still lacking despite the interest shown in it during the history of cognitive science: from the works of Köhler in Gestalt psychology [Köhler, 2013] or the interest of Wallas in social psychology [Wallas,

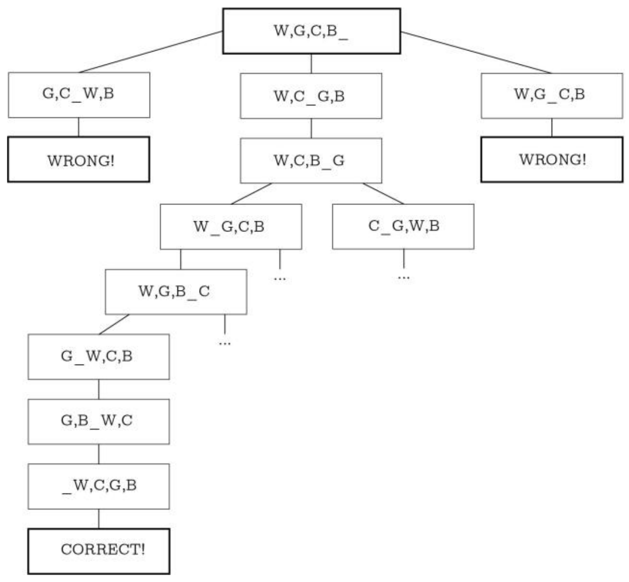

Figure 1: Example by the author of an encoding of the "Wolf, Goat, Cabbage" problem using a classical problem solving architecture. There are three objects encoded with "W, G, C", the boat with "B", and the river with the underscore. From this initial configuration we can create the problem space of all possible combinations of moves by setting a rule that transforms an object "..., X, B_ ..." into a "..._ ..., X, B" object. Some of them are not allowed (e.g. the first one produces a wrong state with G and C alone on one side of the river) and some will lead to the solution. Note that to be able to reach the solution, one must also encode the "insight" of being able to return back one object from one side of the river (that is the rule: "..._ ..., X, B" into "..., X, B_ ...".

1926] to more recent contributions in scientific books [Chu, 2009] [Sternberg and Davidson, 1995] [Vallée-Tourangeau, 2018].

The limitations previously delineated and the state of the empirical investigation in the field has led to some consensus in the literature [Ohlsson, 2011][Batchelder and Alexander, 2011]around the fact that no computational model has yet described successfully the process of generating the encoding in insight problem solving. In this thesis we aim to advance the state of the art in insight problem solving by considering informal proposals about the nature of insight and analyze how they fare when cast in a formal computational model.

We will follow the paradigm of problem solving as a search in a problem space. However, in our framework we will move beyond that paradigm. A key component of problem solving will be the ability to shift from an initial problem space to a different one by changing the representation of the problem once a situation of impasse is reached (what we will refer to as "insight"). Specifically, in this thesis we will consider insight problem solving (IPS) defined as as

> IPS: the human ability of searching an initial problem representation with some rules (as in the Wolf, Goat, Cabbage example) looking for a solution, if no solution is found, some attempts at restructuring the initial problem representation might happen that make the solution reachable.

From the definition above we notice that we consider as IPS also those instances of problem solving where one fails to reach the solution. This might seem excessive given that a consequence would then be to consider as IPS a case where a "solver" just reads a problem and then immediately says that no solution is found, refusing to even think about it. This is technically true, yet just an extreme case of our characterization of IPS. Indeed in our definition the search in an initial problem representation has to be bounded by some factors that will indicate that a restructuring of the initial problem representation has to happen. If these restructurings make the solution reachable within the bounds, then a solution is found, otherwise if also the amount of restructurings becomes excessive, then the search terminates without a solution. With this characterization we can now describe many different examples of human problem solving. Some problem solvers will be able to explore the problem space at great length, while indeed there will also be cases of poorer performances. In an extreme case, like the refuser above, the bounds over the search in the problem space could be so restrictive that the search of the solution might not even start.

Finally, we define IPS problems as all the problems that can be solved through the use of this ability.

1.2 Organization of thesis

As noted before, in the past problem solving was often treated under a computational prospective. Given that there are doubts surrounding computational descriptions of insight problem solving [Batchelder and Alexander, 2011], in our approach we will use analytical tools from computational-level analysis. In Section 2, we will introduce and explain Marr's levels of analysis [Marr, 1981], which comprise the computational-level analysis, a useful paradigm for defining the cognitive processes one aims to explain. Furthermore, we will describe a state-of-the-art IPS computational model and finally define the research questions.

Then we will try to argue how this IPS model could be used to express any problem (this will be explored in Section 3). We will then give an example of a use of a the state-of-the art computational model to give a better intuition for the next arguments (Section 4). Yet we will also express some problems that the current model has, and try to modify the model to be a better definition of IPS (this will be argued for in Section 5). Then we will go back to the original model and note that one aspect of it, namely the amount of steps necessary to reach the solution, must be pre-specified as a bound in the computational model (this will be discussed in Section 6).

Then, in Section 7, we will examine further the implications of each argument presented in this thesis. And finally in the Appendixes the interested reader will be able to find more information about the proofs used in the thesis.

2 Background

2.1 Computational-level analysis: introduction and motivation

Marr's levels of analysis [Marr, 1981], is one of the most influential ideas in computational cognitive science [Peebles and Cooper, 2015]. The levels of analysis treat the brain as an information processing system, and explain its function by distinguishing three different stages of analysis at which one can understand human cognition.

First there is the "computational" level explanation. This level of analysis tries to give an abstract characterization of the problem that the cognitive function is solving. Its task is to precisely define the inputs and outputs that should be taken into account when creating a mechanism capable of simulating a particular cognitive function. Then the construction of an algorithm that could implement this computation and find the wanted solutions is called an "algorithmic" level explanation. Finally, the biological description about how this algorithm is implemented at the neural level is called an "implementational" level explanation.

Table 1: Marr's three levels of analysis

Computational level	Algorithmic level	Implementational level
Description of the computational problem faced	Description of an algorithm solving the computational problem described	Description of the mechanism (usually neurons) carrying out the algorithm described

For a simple example of this tripartition, let's consider the task of sorting some given numbers in input, from the smallest to the biggest. The computational level description of this problem would then define what exactly the input to this problem is and what the output should be. One such computational description for sorting could be

Sorting
Input: A list of n numbers, $a_1, a_2, a_3, \ldots, a_n$
Output: A list of n indexes, $i_1, i_2, i_3, \ldots, i_n$, all different from each other, such that $a_{i_1} \leq a_{i_2} \leq a_{i_3}, \ldots \leq a_{i_n}$

When a computational description is given, then one could attempt to create an algorithm capable of generating the correct output for a given input. For example, given the input from the computational description above, one could devise an algorithm that outputs a random string of numbers $i_1, i_2, i_3, \ldots, i_n$, all different from each other, and then checks whether actually $a_{i_1} \leq a_{i_2} \leq a_{i_3}, \ldots \leq a_{i_n}$. If so, then the algorithm terminates; if not, then it repeats its process by generating another random string.

Of course, this algorithm wouldn't be terribly efficient, and many "smarter" alternatives could be designed. Indeed, an infinite number of algorithmic descriptions could stem from a single computational level description. Yet it is important to notice that all of these possible algorithms will still be constrained by the original problem definition given by the computational description.

Next, when an algorithm is chosen, then its physical implementation is investigated. This depends by the medium we postulate is carrying out the computation. It could be the silicon circuits of the computer we know is sorting the numbers, or it could be a circuit made of neurons, if we postulate that a specific cognitive function dedicated to sorting exists in the human brain.

What is important to realize is that the algorithmic-level and implementation-level descriptions are determined by the nature of the computational problem first described at the computational level. Thus also computational level descriptions of a cognitive function are appropriate aims of scientific investigations, given their importance in giving a clear definition of the problem to be solved by algorithmic theories and their later physical implementation.

Furthermore, computational-level descriptions present a mapping from the inputs of the computational description to specific outputs, thus they are ideal to explore the implications of a theory. For instance, in case of problem solving, one could construct a computational description that is able to work very well for a particular class of problems, but that fails for some others, and thus does not generalize well. Conversely, computational-level analysis could also help individuating when there is an over-generalization of the computational level, for example by the use of too many parameters or by too much human intervention needed when building the algorithm capable of enacting the computational description.

2.2 *eRCT*, a computational-level description of insight problem solving

Getting back to problem solving, although there were some psychological theories attempting to describe aspects of insight, there has been little work done on computational descriptions of insight [Ohlsson, 2011]. The work of the thesis will start by taking into account a recent computational description [Wareham, 2017] that tries to represent the eRCT (extended representational change theory) model [Öllinger et al., 2014], an informal description of insight problem solving. From now on we will use the capitalized "eRCT" to define the psychological theory and informal model of insight problem solving [Öllinger et al., 2014], which we will describe in the Sections below, while we will use the italics "*eRCT*" to talk about the computational description [Wareham, 2017], based on problem solving under eRCT.

To understand the eRCT, one must first understand the theory at its base, the Representational Change Theory (RCT). The Representational Change Theory [Knoblich et al., 1999] describes how an incorrect problem representation can be switched with another one to solve certain problems requiring insight.

2.2.1 Description of RCT and eRCT

The RCT architecture consists of the following elements

- An initial problem representation, an encoding of the problem similar to the one presented in the introduction
- A set of search operators that transform the problem state again similarly to the problem in introduction
- A set of constraints that encode restrictions on the search procedure and the features of the problem states that can be considered solutions
- One or more representation restructuring operators, that shift the initial encoding of the problem and the possible use of the search operators.

To understand how the restructuring operators act, we must also understand how RCT groups the elements of the initial encoding. The problem state elements are grouped into chunks (for an example, consider that the single elements of the Wolf, Goat, Cabbage example could also be chunked, e.g. by having in one chunk the wolf and the cabbage, and in another chunk the boat and the goat). Intuitively, chunks represent patterns that were found useful in previous experiences of problem solving. All the possible subsets of the problem state elements might become chunks, even nested cases (e.g. the chunk "wolf and cabbage" and the chunk "wolf") or intersections, but at any given time just one set of non-nested chunks is "active". That is, search operators are restricted to manipulating those active chunks as wholes. Based on this structure, RCT has two representation restructuring operations. First, the removal of a constraint on the search process or on the form of the solution state, called constraint relaxation. Second, the substitution of one active chunk by the chunks nested under it, called chunk decomposition.

One type of problems where RCT has been empirically tested is match-stick arithmetic problems [Knoblich et al., 1999]. In these problems, some matchsticks are arranged in the form of Roman numerals to form an incorrect mathematical equation (e.g. "V + II = V"). The solver then has to move one or more matchsticks to form a correct example (e.g. in the previous example, by shifting a matchstick from the "2" to the initial "5" we obtain "IV + I = V", a correct equation). In RCT terms, the problem representation consists of a structure that represents the arrangement of matchsticks. Furthermore, there are constraints defining what a valid formula is. We can see the usefulness of the constraints in finding an insight by looking at one matchstick example. Consider the solving of the matchstick arrangement "III + III = III". In this tricky case, the solver must notice that there is no one-matchstick-move starting from the matchsticks that form the numbers that can solve the problem. The insight that the problem solver must reach is the realization that one matchstick from the "+" has to be moved to form an "=" sign so that the solution "III = III = III". In RCT, the search operators can move a matchstick-chunk from one point in the problem state to another, so to solve the previous problem, first a chunk decomposition must happen to let the matchsticks forming the "=" sign be active. Then also a restructuring operator must be applied to relax the constraints of what a possible solution formula is, so that 3 equalities are considered a solution.

RCT provides a useful informal model and a possible theoretical explanation of insight. Yet it seems to be restricted to matchstick arithmetic problems [Batchelder and Alexander, 2011, Page 79]:

> "While the RCT offers a plausible theoretical account of some subclasses of insight problems, it has a major disadvantage as a general theory of insight problem solving. The disadvantage is that for many insight problems [...] it is not at all clear how a theorist can determine possible activity distributions from problem statements. Instead, it appears that for most insight problems, the RCT might be a useful post hoc tool to discuss problem solving behavior rather than a theory that makes formal predictions across a wide class of insight problems"

More recently an extended version of RCT has been developed, called eRCT [Öllinger et al., 2014]. The new aspect added to the model is the inclusion of a recursive step to the process of problem solving: that is, the ability of nested repetitions of search, impasse, and representational change [Danek et al., 2016]. More specifically, under eRCT, and similarly to RCT, insight is assumed to be caused by a representational change. Yet eRCT doesn't specify any particular class of problems that necessarily requires insight: it assumes that each problem can be solved without re-representation if an appropriate initial problem representation and search operators are used. This is an important point, as it means that eRCT (but also our definition of IPS) is technically also a computational description, with the restructuring happening only when an insight is needed.

eRCT describes insight problem solving as a search procedure involving different stages. First, an initial problem representation is created, based on the interplay between the perception of the problem and some prior knowledge. Then this problem space is explored with the help of heuristics, similarly to the previous classical models [Newell and Simon, 1972]: if this search is successful then a solution is found, otherwise a situation of impasse is reached that must be overcome by a representational change. Similarly to the previous RCT model [Ohlsson, 1992], this is achieved with relaxation of prior constraints and decomposition of elements in their subparts. When this is accomplished, a new problem representation is established, with a new problem space to be searched.

Despite these changes, it's still not clear if this new theory can be applied just to toy problems (like matchstick problems, or the 9 dot problem). This raises some doubts about its ability to generalize to new Insight problems, and this will indeed be one of the thesis' starting points.

2.2.2 Description of *eRCT*

eRCT was translated into a computational-level description [Wareham, 2017] as follows.

- The problem representation is encoded as a predicate structure: a formalization consisting of objects and predicates describing a relationship among those objects. This problem representation is then decomposed into chunks: partitions of the problem-state of usually contained size, that are used by the search operators to come to a solution.
- The search operators operate on the active chunks of the predicate structure by replacing them with other predicate structures.
- A set of constraints is used to specify the restrictions in the applications of the search operators and a constraint is used to define the accepted problem solutions.
- The restructuring operators then may be either relaxations represented with the removal of one of the constraints previously specified, or restructurings of the problem space with the "de-chunking" of one chunk into the chunks nested inside of it.

An example of the *eRCT* computational-level description applied to a matchstick problem may be seen in Figure 2. In matchstick problems the task is to rearrange some matchsticks forming an incorrect arithmetic operation into a configuration that is correct (for example, in Figure 2a , to shift one matchstick from the "+" symbol in the "1 = 2 + 2" equation to form a "3" in the correct equation "1 = 3 - 2"). The matchstick representation is encoded as a predicate structure (seen above the matchsticks) where each object represents a matchstick, plus some predicates describing the orientation of those matchsticks and the relationship they have between them (e.g. "v" for "vertical" and "tl" for "to the left"). Furthermore this problem representation is partitioned (by dotted lines) into chunks. Informally speaking, these chunks represent the units of the problem taken into consideration when solving the problem. In Figure 2b we see an example of chunk decomposition (formalizing insight ability), where a chunk restructuring operator relaxed the chunks around the "2" and the "+" symbols, so that we can apply a search operator capable of moving a matchstick from the "+" symbol to form a "3". In Figure 2c we see the result of this, with the creation of new chunks around the "3" and the "-" symbol.

A more formal description of the elements outlined above follows (see [Wareham, 2017] for more details):

- The problem representation is described as a predicate structure. Predicate structures are a type of representation used in cognitive science, and are composed of objects (for example the matchsticks are represented as dots in Figure 2), and predicates relating those objects (for example defining the position of a matchstick as vertical or horizontal, or relating a matchstick to the left or above of another). Predicate structures can be mathematically represented as vertex-labelled directed acyclic graphs. In this type of graph, objects can be represented as leaves, predicates are internal vertices, and objects can be linked with directed edges to their predicates. From now on, we will call this predicate structure p.

  The chunks can thus be modelled as sub-predicate structures: a subset of the objects in a predicate structure with all the predicates in that structure that are based on the objects in this subset. Furthermore, these chunks can overlap, or be nested. Then we can define the active chunks as a chunk-structure, a collection of any non-nested chunks that covers all objects in a predicate-structure. Since there are more than one of the same type of chunk in a problem representation (for example, the individual matchsticks chunks that can be repeated) then we distinguish between the set of chunk types available for making chunk structures and the individual chunk-instances making up a chunk structure. From now on, we will label the chunk-type set T and the chunk-structure D.

- The search operators can be modelled as substructure replacement rules of the form X → Y that operate on predicate structures. When we apply a search operator to a predicate structure, it replaces one occurrence of a predicate substructure X with a predicate substructure

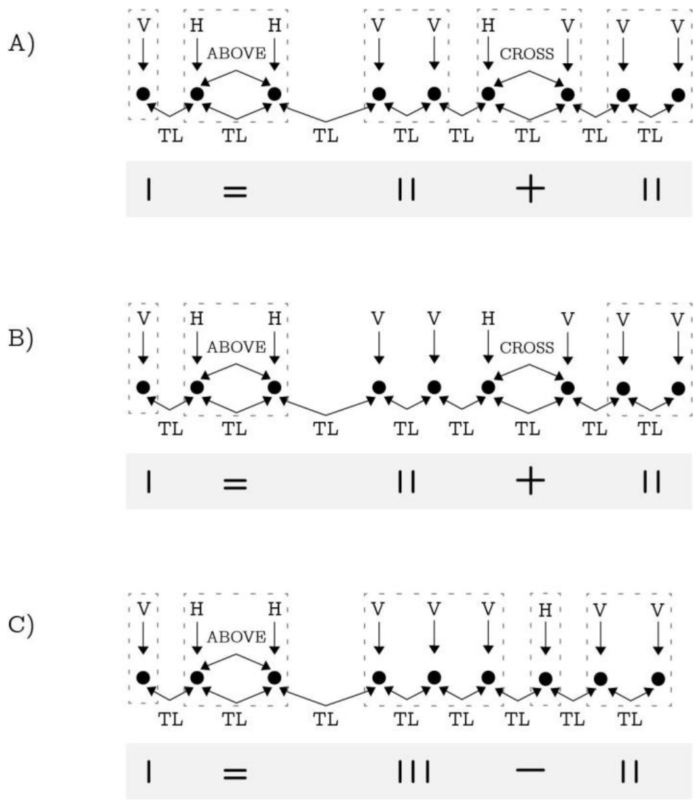

Figure 2: An example (adapted from Figure 4 in [Wareham, 2017]) of a matchstick problem described with *eRCT* . In A) we can see an example of an encoding of a matchstick problem in a predicate structure. The dots intuitively represent the individual matchsticks, all of them have specified their orientation ("V" for vertical, "H" for horizontal) and the relationship between them ("TL" being "to the left"). Furthermore we can see the chunking represented as subsets of the predicate structure delineated by dotted lines. B) Same predicate structure as the above, but with a chunk decomposition around the right side of the matchstick equation. C) result of a search operators "moving" the vertical matchstick of the "+" sign to the left to form a "3".

Y. Furthermore, we also specify that search operators can only be applied to active chunks in the chunk-structure of a predicate-structure. We will refer to the set of search operators as O.

- The constraints are modelled as first order logic formulas over the objects, predicates, and chunks in a problem representation. Furthermore we also have to distinguish two sets of constraints to be satisfied. One defines the goal states and the other specifies restrictions in the application of search operators. These constraints will be labelled as C, which will be formed by the goal constraints C_G and the restrictions on the search operators constraints C_O
- Finally the constraint relaxation is defined as a deletion of one of the constraints defined above in the accepted goal states or in the restrictions of the search operators, and the restructuring is defined as a chunk decomposition of a chunk in a chunk structure with the chunks nested inside of it.

With this we can finally summarize the above description in the following computational definition of *eRCT* [Wareham, 2017]:

eRCT
Input: A problem representation p with chunk-structure D; a chunk-type set T, a search-operator set O, a constraint-set $C = (C_G, C_O)$, and integers k_C, k_D, and k_S .
Output: A sequence s consisting of the application of $\leq k_C$ constraint relaxation and $\leq$ k_D chunk decomposition operators and $\leq$ k_s search operators from O in any order

that transforms p into a goal state consistent with C_G, if such an s exists, and special symbol $\perp$ otherwise.

We furthermore notice that the special symbol$\perp$in the above computational description is the computational equivalent of finding no solution to the problem that is being solved.

This computational-level description, named *eRCT* , will be the starting point of the thesis investigation.

2.3 Research Questions

The literature concurs [Batchelder and Alexander, 2011] [Ohlsson, 2011] that an open issue with computational models of problem solving is the difficulty to create a model that is not reliant on the initial problem encoding to reach a solution. Often a model is created for a particular class of problems, and this creates some difficulties when it comes to the generalization of the model to other types of problems.

Given the impasse in research in insight problem solving identified by the literature, and given the proposed *eRCT* computational description for IPS, we will first try to assess if *eRCT* is generalizable to all IPS problems, and thus if it is a good computational description of IPS. The presence of some difficulties in the generalization might also point out that insight problem solving, as formulated, is actually unexplainable in scientific terms. Thus the first aim of the thesis will be to address the following Research Question.

- **Is *eRCT* generalizable to all insight problems?** We will start by analyzing the generalizability of a recent computational description of insight problem solving [Wareham, 2017] built upon eRCT [Öllinger et al., 2014], and we will try to understand if indeed this model is applicable to all IPS problems.

What we will find is that eRCT is actually theoretically applicable to all insight problems, although with an important caveat. The success of the model depends not just upon the initial encoding of the problem, but also upon the encoding of the restructuring abilities of the model. That means the model already receives in input the information about the insight to be reached. To better explain this, following the example from the Section above, the model receives in input (in the chunk restructuring operators) the knowledge that it can restructure its chunking of the "2" and "+" matchsticks. This means that it receives a great help in its search of the correct insight, and what the computational description actually depicts is the steps of the restructuring. Yet the computational description gives no information about the origin of the chunk restructuring operators. From this follows that maybe another computational definition might be able to better describe IPS, and this leads us to a different question:

- **Is it possible to redeftne a new computational description of IPS?** Preserving the generalizability of eRCT, but furthermore having a description capable of explaining the origin of insight would be of great help for the future of the field. As already described, various problem solving architectures seem to be difficult to generalize, and furthermore their description already encodes part of the problem that is going to be solved. Finding a way to get rid or reduce that in-building of the initial encoding would be an important advancement, as having a well formalized computational-level description would be necessary for the construction of an algorithmic theory of IPS, which would in turn be of great empirical and practical relevance.

Furthermore, another result stemming from the Research Question 1 is the importance of the bound k_s in the input of the *eRCT* description. Indeed while arguing for the Research Question 1 we will find out that the bound k_s is critical for finding an answer to the generalizability of *eRCT* . From this we conclude that:

- eRCT is an informal theory of IPS while *eRCT* is a formalized computational description that added the k bounds to the computational definition to allow for the complexity analysis of the model[Wareham, 2017]. So we ask ourselves: **Are these bounds necessary for a proper deftnition of *eRCT* ? If so, how can they inform us about some characteristics that are missing from eRCT?**

In the following Sections we will present some arguments, backed by a computational-level analysis of *eRCT*, that will elucidate the previous Research Questions.

3 Generalization Argument: Can *eRCT* describe all IPS problems?

3.1 Introduction

Our aim is to discover whether *eRCT* is applicable to all possible IPS problems. To do this we could argue in favour of *eRCT* 's generality by trying to show that many IPS problems could be encoded and solved with the *eRCT* architecture. In this way, we could show that some insight problems, that may look very different from one another, could be described by *eRCT* . Yet this would be considered just tangential evidence for the generality of *eRCT*, and of course this approach would still leave many IPS problems (supposedly infinite in nature) as not yet shown to be encodable. Given that we cannot 'prove' a computational-level description's generality by showing all cases, we can try to provide an argument backed by a mathematical proof about eRCT's generality.

We will explain the argument properly in the next section, and for now we will introduce it just briefly. The intuition underlying the argument is that human insight problem solving, being a cognitive function, could be described as a "procedure of information manipulation". If *eRCT* is generic enough to be able to formalize this "procedure of information manipulation", then the *eRCT* description could certainly be used to encode all IPS problems. This "procedure of information manipulation" refers to the fact that cognitive processes can be described as computations, and this assumption we will discuss at the start of the next section.

3.2 Argument

One important assumption of cognitive science (or at least of the more computationally oriented branches of cognitive science) is that cognitive capacities can be represented as computations [Anderson, 1990], [Massaro and Cowan, 1993]. Historically, this has been a dominating view in the field of cognitive science and is often called the computational theory of mind. This theory considers cognitive functions as computable, that is, it considers all cerebral activities that lead us to attention, language, memory, perception, problem solving, and thinking in general, as possible algorithms that could in principle be written on a computer [Pinker, 1999]. Even if there have been different oppositions to the description of cognitive function as being computable ([Fodor, 2001], [Penrose, 1994], [Van Gelder, 1995]), to this day the dominant position in cognitive science is that cognitive functions can be explained as computations. So in the following, we will likewise assume that cognitive functions are computable.

Another important assumption that we will use for the argument is that IPS is a cognitive function. Arguably, we consider insight problems solving as a mental phenomenon that happens whenever a solver is faced with a problem. If the solution is reached, then it was obviously a cognitive process that lead to the exploration and restructuring of the search space until a solution was found. Otherwise, if a solution is not reached, then this was due to a failure of re-representation or to an unsuccessful search of the problem space. For example, not finding the solution might be due to an excessive size of the search space that leads to difficulties in exploring it properly. Similarly, not reaching a re-representation that could easily lead to a solution might be due to some strong constraints that the problem solver has difficulties in relaxing. So, whether the solution is reached or not, IPS still seems to define a human mental phenomenon during problem solving, and so we assume that IPS could be a cognitive function.

As a conclusion of these two assumptions (cognitive functions being computable, and IPS being a cognitive function), we expect that there exists some computable functions capable of computing IPS.

The next step, as shown in Appendix A, is to prove that an algorithm solving the *eRCT* description is "strong" enough to simulate any computation. An algorithm solving *eRCT* takes in input a problem representation p with chunk-structure D; a chunk-type set T, a search-operator set O, a constraint-set $C = (C_G, C_O)$, and integers k_C, k_D, and k_S. Then it searches a sequence of constraint relaxations, chunk decomposition operators, and search operators from O that transforms p into a goal state consistent with C_G. Indeed a class of algorithms that solves a version of *eRCT* with just one application of a chunk decomposition and constraint relaxation operation already exists, and are called ideal-observer models [Wareham, 2017]. Starting from these algorithms, we can create an even simpler algorithm, that takes in input just the predicate structure, some search operators and a constraint set, (the other inputs of *eRCT* can be set as null), and then manipulates the predicate structure with those search operators until it is capable to output the search operators that, applied to a particular predicate structure, satisfy the constraint set. We will show that this algorithm, that we dub N&S-PS system (since it's a version of the Simon and Newell problem solving model discussed in the introduction), is capable to simulate any computation.

So we will show that for any possible computation, we could create a predicate structure p, search operator set O and constraint set $C = C_G$ that the N&S-PS system takes in input and uses to simulate the given computation. This means that the N&S-PS system could be used to compute anything that a computer could compute.

To do this, we will show that the N&S-PS system could simulate any Turing machine, an important theoretical model of computation [Sipser, 2006]. Intuitively this is true because a predicate structure can encode any language, or any string, and the search operators of a predicate structure can be set to have the same properties of a Turing Machine head (the element of a Turing machine that "computes" by reading and writing symbols). For a proper proof, the reader is redirected to Appendix A (specifically A.2).

From this proof follows that anything that could be computed could also be computed by a N&S-PS system. Thus any cognitive function could be simulated on a N&S-PS system and thus a N&S-PS system could compute an IPS cognitive function. Since the N&S-PS system is an algorithm that solves *eRCT* under some particular caveats (having some inputs as null), this means that there exist some input/output pairings of the *eRCT* that could be used to define that particular cognitive computation that the N&S-PS system is simulating, and among those is IPS problem solving. Hence we can conclude that surely *eRCT* is powerful enough to describe IPS problem solving.

Another way to look at this argument is that the eRCT is almost "too powerful" and it doesn't need even all of its input to represent the cognitive function that it wants to represent (indeed, it could represent any computable function, and so any cognitive function). Yet this is not to say that the *eRCT* description is not useful to define insight, since the restructuring that the *eRCT* tries to describe might still be a good representation of how the IPS cognitive ability is acting. Yet, whatever the insight problem at hand, still the N&S-PS system (a "subset" of *eRCT*) is enough to compute it.
To summarize, the general outline of this argument is as follows:

Assumption 1: cognitive functions are computable.

Assumption 2: IPS is a cognitive function.

Conclusion 1: IPS is computable.

Theorem (appendix A): An N&S-PS system is capable of simulating any computation.

Conclusion 2: An N&S-PS system can simulate an IPS cognitive function.

Definition: An N&S-PS system is an algorithm that solves a subset of the problems defined by *eRCT*

Conclusion 3: *eRCT* could be used to define an IPS cognitive function.

Thus we have argued that the *eRCT* computational-level description could perfectly describe IPS, and thus *eRCT* could be used to model the solution of any IPS problem.

This line of reasoning answers Research Question 1, that is, *eRCT* is capable of encoding any insight problem. Yet some important observations have to be made.

First, we didn't show how to encode any problem, we just theoretically argued that given the assumptions above, an *eRCT* description exists for all possible IPS problems (and even more, for all cognitive functions). Hence this description we know exists might even be very unintuitive and different from the usual encoding of a problem in eRCT.

Second, in the argumentation above there is a switch between the computational level and algorithmic level when we talk about the *eRCT* algorithms that correspond to the N&S-PS system that could simulate any computation. This might cause some doubts concerning whether questions about computational expressibility can be addressed at the algorithmic level. Yet the basic characteristic of computable functions is the necessity of a finite procedure, an algorithm, to tell how to compute the function [Sipser, 2006]. So, since the computational level deals with problems that should be computable, then we conclude that it is sound to inform the computational level analysis with some algorithmic aspects.

Third, an important caveat that we omitted in the above argumentation is that the N&S-PS system is Turing complete just when allowed to have an arbitrarily large bound k_s of search operator applications. This is an interesting aspect also for *eRCT* , as that bound was not present in the informal eRCT model. Indeed we will show, in the Unbounded Argument, that the presence of that bound is actually indispensable for the computational definition.

4 Intermezzo: Example encoding of an IPS problem in *eRCT*

4.1 Introduction

Even with this formal proof at our disposal, one might still ask how to encode a new IPS problem in the aforementioned eRCT architecture. In fact, the previous argument just pointed out that an encoding using eRCT is possible, but didn't show us how to do it. So we will now present an encoding of the previously cited Problem 42 by Alcuinus.

This will be useful not just to give some insight to the reader into the process of deriving the eRCT computational-level description from a problem, but also to illustrate that not every aspect of eRCT seems to be necessary for it to be a good description of IPS. This observation will be backed by some formal analysis about the definition of *eRCT* , and will illustrate an important point about how specific the encoding of the information into the computational-level description has to be. This will be an important point in Section 5 (the Minimality Argument), where we will argue that it seems that it is the actual human-encoding that is necessary to come to a solution, and not the *eRCT* description per se.

4.2 Example

Recall the following problem from the introduction.

> PROBLEM 42:
> in a 100 steps staircase, with 1 pigeon sitting in the first step, 2 sitting in the second, and so on to 100 pigeons sitting in the last one, how many pigeons are there in total?

We will now give one example of how to encode the problem in the *eRCT* description
We will encode the 100 steps of the "staircase" in a predicate structure (Figure 3). Each stair step will have some number of sub-predicates equal to the number of "pigeons" in that specific step. Then the role of a "counting" search operator could be that of adding two adjacent objects into one object with the sum total of the previous ones. This would result in a long calculation of 99 operators before reaching the conclusion (we can describe the goal state as the constraint of having just one predicate with the total number of pigeons). This process would clearly reach a conclusion, but without any insight. Yet recall from the introduction that more insightful ways of solving this problem exist, by noticing that a rearrangement of the steps permits much easier calculations. So we will now try to describe this problem with a restructuring that permits to reach the sum in a more clever way.

In the eRCT model, if the solution is too difficult to reach given the present operators and problem representation, then a restructuring occurs. In this case, it might be that the rather lengthy calculation needed to solve the first problem might indeed create an impasse, so a restructuring operator is needed to model insight. In the picture, centre column, a first restructuring operator (called "Alcuinus Insight") is introduced. This operator can be modelled as an operator belonging to the set O, yet with any constraint in C_O associated to it that doesn't permit its application to the initial predicate structure. Intuitively this operator shifts the predicates so that the first step is close to the 99th, the second close to the 98th and so on until the 49th step is close to the 51st step, and then adds the remaining two steps, the 50th and the 100th. So when no solution is found the restructuring that will happen will be due to a constraint relaxation of the constraint in C_O that didn't let "Alcuinus insight" search operator to be active. Then this operator will modify the structure in a way that will permit us to use other operators, previously encoded in O but unable to act due to the predicate structure being in a form that didn't let their application (In Figure 3, the operators that sum "1" to "99", "2" to "98", etc.). In this way we can use fewer search operators, and all structurally similar to each other, to sum to 100 the first 49 couples. Then we need the introduction of a multiplication operator (operator 50), that is able to "collect" the 49 predicates that are close to one another and that have the same attribute (100) into one predicate that corresponds to the multiplication 49x100. At this point we can use the first operators we introduced to sum the remaining numbers into an accepted solution. In this way we achieved a

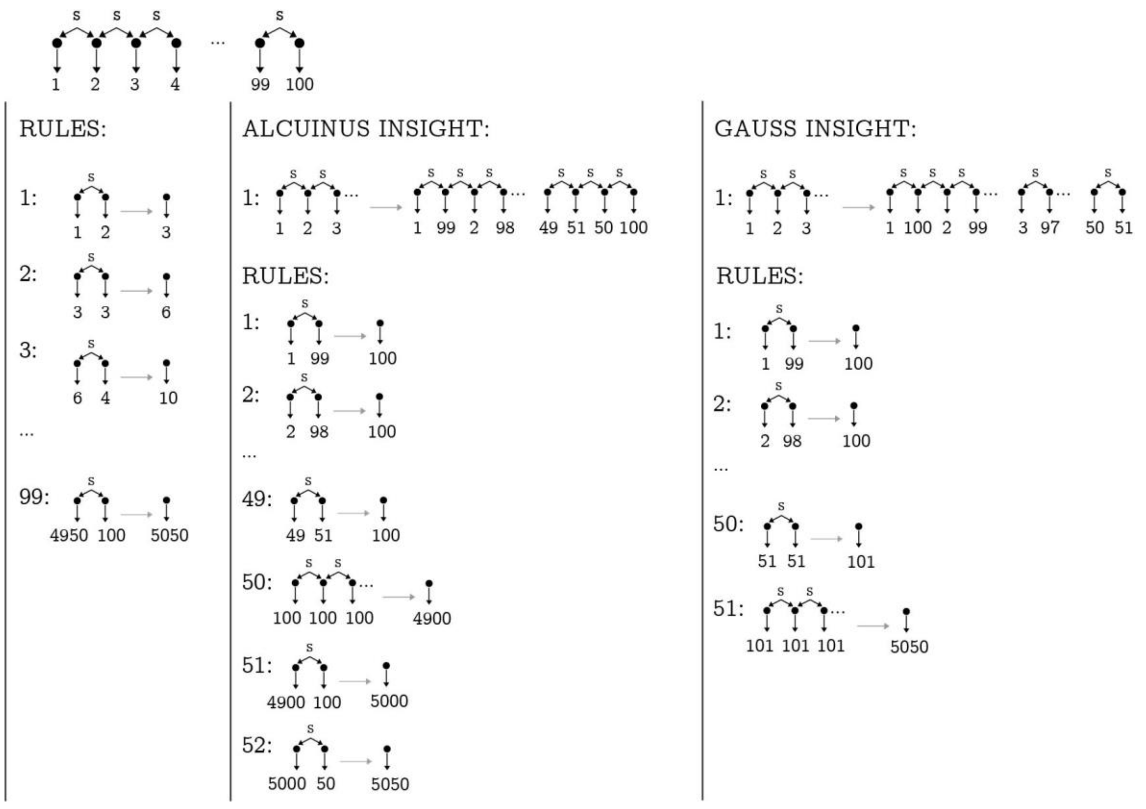

Figure 3: An example of how to encode the staircase problem in a predicate structure with search operators. The leftmost columns represent the base case of an encoding that doesn't lead to insight, and needs 99 steps of search operator application to reach the solution. While the centre and rightmost columns have a first restructuring rule, representing insight, and then need half the steps to reach the solution. Note that we use no chunking for this example, although probably some ways to model the problem with chunking might exist.

solution that is quicker (since it uses way less search steps), and undoubtedly more "insightful" (given that the restructuring permits a quicker solution) than the first solution proposed.

Furthermore, we could encode another restructuring operator, to come to the solution as Gauss supposedly did. This operator (the rightmost column in the picture) would move the initial predicates in a way that the first is close to 100th, the second to the 99th, and so on to the 50th close to the 51st. At this point a "sum to 101" operators would get the predicate structure ready to be solved directly by a a single multiplication operator.

Now that we have this sample problem formalized, we can see that the information about the restructuring was given by a search operator with a constraint. And if we recall the *eRCT* definition,

> *eRCT*
> **Input:** A problem representation p (a predicate structure) with chunk-structure D (a grouping of predicates forms a chunk); a chunk-type set T (possible chunks that can make a chunk-structure), a search-operator set O, a constraint-set $C = (C_G, C_O)$ (constraints in using the operators and in what the solution looks like), and integers k_C, k_D, and k_S.
> **Output:** A sequence s consisting of the application of $\leq k_C$ constraint relaxation and $\leq k_D$ chunk decomposition operators and $\leq k_S$ search operators from O in any order that transforms p into a goal state consistent with C_G, if such an s exists, and special symbol $\perp$ otherwise.

We can see that search operators and constraints are given in input to the computational description. Furthermore, we also notice that much of the *eRCT* input (related to the chunking) doesn't need to be specified for this problem. This is testament to our previous results (Appendix A.2) that an N&S-PS system (comprised of predicate structure, search operators, and constraint in the solution) is well-capable of handling IPS.

To summarize, the conclusion from this observations is as follows.

> The insight information is given to *eRCT* with a series of restructuring operators that are very similar to the search operators from previous architectures. Furthermore the restructuring (insight) operators are defined in the input.

This example and analysis is important because it sheds light on the inner workings of *eRCT* and makes some clarity regarding what exactly characterizes the restructuring (and consequently the insight) in *eRCT* . This conclusion doesn't answer any research question per se, but will be used to make an important point for the Minimality Argument.

5 Minimality Argument: In-building the solution

5.1 Introduction

In this Section we will argue how the information given in input to *eRCT* seems to strip it of the ability to achieve insight (we will refer to this as *in-building*). With the Generalization Argument we have seen how *eRCT* could be used to encode any insight problem, yet we will now argue that the information given in input may pose some problems if one aims to describe IPS. This will be an important point to investigate if one aims to answer Research Question 2 (about the possibility of redefining a more general IPS computational description).

More in particular, we will observe how the information given in input in the computational description needs an explanation of its origin. And this may cause a problem of infinite regress of this explanation. This is because this built-in information might be as well solved by another problem solving cognitive function, which we know nothing about. This problem will make us ask for a computational description of IPS that specifies a minimum set of information in input so that this infinite regress problem is solved. We will observe how this issue of in-building is similar to other debates in some rather famous topics in cognitive science (e.g. Fodor's puzzle of concept learning [Fodor, 1975]). Finally we will introduce some results from computability theory that may help to clarify what could be an appropriate set to be given in input to an IPS computational description so that any type of insight problem might be encoded with it. This will be useful to give some insights about an answer to Research Question 2, about the possibility of explaining insight problem solving with a computational description explaining the origin of insight.

5.2 Argument

An important point stemming from the example problem above is that it seems that some of the information that we are giving in input to the *eRCT* is by itself solving the insight problem, with little need for *eRCT* to do anything "clever". In our example, and from the analysis of *eRCT* , it's important to notice that the information about the possibility to restructure is given *in input*, so that an algorithm solving this computation has nothing much to do, except to apply the restructuring rules already given. Indeed this aspect was already discussed in Section 1, and was an issue already well established in the literature. To give in input to a problem solving architecture such a specification could be seen as in-building the solution itself. This is because then the computational definition of problem solving becomes a description of the process through which the correct operators, pre-written in the model, can be combined to reach a wanted state.

Of course, one could counter-argue that if *eRCT* is actually designed to describe this process of operator combination, and defines this process as insight, then indeed *eRCT* achieves exactly this. Yet, as we will argue, this still spells some problems for the present IPS models.

Consider the following informal computational description of a Problem Solving architecture under Simon and Newell.

> **Problem Solving**
> **Input:** a starting problem representation, a set of search operators, and a final solution state.
> **Output:** A sequence of search operators that, starting from the problem representation, reach the solution state.

Having those search operators and representations in input was not considered an issue of in-building for their Problem Solving model. That is because one could explain the origin of that input by bringing into play other human cognitive functions [Ohlsson, 2011], while the focus of their model was in showing the capabilities that combined search operators had in finding a solution. As an example of a problem solved with one such problem solving architecture, we can go back to the example of the "Wolf, Goat, Cabbage" problem from the introduction, and we can see that the search operator that was transporting objects back from the other side of the river was encoded in the problem definition from the beginning. Yet this input was not really seen as a problematic built-in information in input, as the origin of this input was supposedly coming from other cognitive functions, e.g. memory of past problems or, indeed, insight abilities.

Similarly, also the present *eRCT* might make a similar case: having the insight restructuring operators in input is not in-building as long as another lower level problem perception cognitive ability exists, that can observe the previously unnoticed features of an unsolved problem [Ohlsson, 1992, Page 13]. This is a sound stance, but not free from issues. Indeed, if we look back at the *eRCT* definition, we see that it uses the restructuring search operators in input to explain how to reach an appropriate restructuring. Very similarly, the procedure used by previous PS models to find a problem solution was also taking in input a correctly restructured problem representation that could be used to reach a solution.

This similarity creates some problems for the computational description of IPS, since it makes it become just a "double" application of a PS architecture. In input we have chunks and restructuring operators, but they could very well be seen as instances of problem representations and search operators. This gives the idea that the computational description is explaining away the crux of the matter, from "insight" to "lower level problem perception abilities". With the first models of PS, the problem was figuring out how could the correct problem representation and search operators be created, and now, with IPS models, the problem becomes how "lower lever problem perception abilities" are created. Considering the pattern used so far, will also "lower level problem perception abilities" be explained by some search operator combination, which will be given in input? And will that input too have to undergo the same procedure, of being explained by some search operators given in input, and so on recursively and ad infinitum?

We can summarize this "infinite regress" problem as follows: To do problem solving, humans start with some information (the problem representation in input) that can help them solve the problem. But where does this information come from? That is a problem by itself, and to do problem solving, humans start with some information (the problem restructuring operators in input) that can help them solve the problem. But where does this information come from? That is a problem by itself, and to do problem solving... (and so on ad infinitum as above)

This type of circular explanation necessarily follows if the IPS computational description we have chosen is built in such a way that some unexplained in-building is required to reach the solution.

Yet we believe that this "infinite regress" problem might have an answer. Before presenting it, we will give an intuition for it by looking at an example of a famous similar case of in-building that happened in the history of cognitive science.

Let's look back to Fodor's claim of necessity of nativism for concept learning [Fodor, 1975]. The claim goes as follows. Imagine that to do concept learning you need some old concepts to be associated to the new concept being learned. But where do these old concepts come from? Well, if they are learned concepts, to do concept learning you need some old concepts to be associated to the new concept being learned. But where do these old concepts come from? Well, if they are learned concepts, to do concept learning you need ......and so on ad infinitum". So one must either believe that the human brain is somehow capable of managing this infinite regress, or one must simply postulate the existence of a nativism of some sort: there must exist primitive concepts already encoded in the brain.

We can similarly propose a small modification to *eRCT* [1] that should make it so that the

[1]To be fair to the eRCT capabilities, also the search operators, if built in a specific manner, might show a great deal of generative effects. This is because search operators might always be engineered to work and produce an insight just for a particular combination for them. In the examples always really simple operators are shown, but we could imagine we could split them, so that a search operator of the from $X \to Y$ (where X and Y are predicate structures) could be split into many search operators $X \to X_1$, $X_1 \to X_2$, $X_2 \to X_3$, ..., $X_n \to Y$, so that the solution would be reached after a lengthy search. This would make the operators appear more generative, and less built-in. Yet the computational definition does not specify how to design this type of more generative operators, while further down in the argument we will try to explain exactly that. So we will hold true that eRCT still has

restructuring operators (which represent the insight gained) given in input, are both simple and powerful enough to create any possible insight. So that, given their simplicity, the issue of in-building will not be that relevant. And given their power, the infinite regress in the search of a correct input won't happen, given that by themselves they will be able to create all possible search operators. In this way we will theoretically achieve a computational definition that should be able to describe the totality of human insight capacities, and thus be an answer to Research Question 2.

Let's suppose that there might exist some core, primitive, problem solving information, from which one could build up all the rest of problem solving skills. This might be an appropriate solution to avoid this infinite regress (and indeed similar to the idea of concept learning primitives). This minimum quantity of innateness will be our focus for the rest of the argument. We will give a look at a mathematical theory that guarantees us that a minimum subset of simple computable functions (something akin to our search operators) can be combined together to form all possible computable functions. This, we argue, is an important step in the construction of a computational-level definition for IPS, since this will provide us with absolute generality. We want our definition to be applicable to all IPS problems (whatever they might be), so we want our innate built-in operators to be capable of creating all possible operators, so that our IPS model can be used to solve any IPS problem (that may well need some operators never seen before). To achieve this, in the following text we are going to argue that we can swap search operators and predicate structures with computable functions per se. In this way we can use already existing mathematical theories of computation to look at the minimal origin of these computable functions. This can make for some interesting observations about the minimality of innateness.

First, we notice that predicate structures and search operators of *eRCT* could be replaced by an input and some computable functions, so that the computable functions act on the input in a similar way that search operators act on the predicate structure. This is trivially true since search operators and predicate structures are graphs and operations on graphs that can be easily encoded in computer algorithms, thus substituting them with computable functions doesn't create a loss of generalization. Yet this new description would also be equally powerful as search operators and predicate structures, since they can be proved to be Turing complete (Appendix A.1). Thus substituting them with computable functions and an input doesn't cause any gain in generality either.

Now let's recall that in *eRCT* the restructuring is due to a chunk decomposition rule relative to the input D and by a constraint relaxation rule relative to C_O. These are the built-in aspects of eRCT that cause the infinite regress problem (as their origin is given by the not specified "lower lever problem perception abilities"). We can ask ourselves how to replace them in order to have a core set that could achieve generality and solve the infinite regress problem. To answer this, we have to ask ourselves what could the building blocks of computable functions be.

This aspect would depend on which way to represent computable functions we are investigating, as there are different Turing equivalent models of computation (Turing machines, lambda calculus, mu-recursive functions,...and probably the human brain). Yet one of these models, namely mu-recursive functions, is interesting because of the small initial class of basic computable functions that can be used to achieve all the other possible computable functions [Kleene, 1968]. The initial set of just 3 classes of functions (constant function, successor function, projection function) is combined together by 3 rules (composition operator, primitive recursion operator, minimization operator). The interested reader is referred to Appendix C for more information and an example about mu-recursive functions. Starting from those basic functions and combinations, one can achieve all possible computable functions. A hypothetical computational description that would have these innate characteristic would be powerful enough to create all computable functions, and thus achieve all insights ever needed, without any in-building.

To formalize this new computational description:

IP S
Input: A problem representation input p (an input), an initial mu-recursive function set O (the initial search operators), an insight mu-recursive function set I (comprised of the basic set of functions that can span all possible computable functions), constraint-set $C = C_G$ (constraints in what the solution looks like) .
Output: A sequence s consisting in the application of computable functions from O

some unexplained in-building.

and I in any order that transforms p into a goal state consistent with C_G, if such an s exists, and special symbol $\perp$ otherwise.

So if we imagine the mu-recursive function set I being innate, then we can avoid the infinite regress trap, as this set could span all the computable functions/search operators we would ever need, and explain the totality of possible insights.

As an interim summarization we can solve to the infinite regress problem as follows: to do problem solving, humans start with some information (the problem representation in input) that can help them solve the problem. But where does this information come from? No infinite regress now: there is some innate information that is capable of creating the re-representation of the problem representation in input.

This line of thought can bring us to the following conclusions, more relevant for cognitive science. First, some of these basic functions and operators of mu-recursive functions can be translated to cognitive claims. E.g. we could hypothesize that the innate abilities of the brain to do recursion or composition or minimization are the ones generating all possible computable functions (or search operators), and thus letting us do IPS. This prospect, based upon this computational-level analysis of IPS, would be theoretically solid and rich in possible empirical predictions.

Alternatively, the mu-recursive functions example is also a good illustration that points out that very little innateness can achieve a lot (in fact, totality) in terms of computational power. Mu-recursive functions won't probably be the end explanation for IPS, and probably better descriptions exists for cognitive search operators then them. Yet they are an example that lets us see how theoretically innateness could generate totality of computational power and thus generalization.

Additionally, computability theory could be of use, in case, to prove that another innate model of search operators, maybe more cognitively plausible than mu-recursive functions, is indeed strong enough to generate all the operators necessary to solve any problem. And not fall into the infinite regress trap.
To summarize, the general outline of this argument is as follows:

Assumption 1: the innateness of IPS computational problem operators and the infinite regress problem warrant to investigate what is the minimum set of operators that can be combined to achieve the function of any operator.

Assumption 2: IPS computational problem operators can be represented as computable functions.

Assumption 3: computability theory methods can prove what minimum computable functions set is needed to achieve all possible computable functions.

Conclusion: computability theory methods can be used to identify a minimum set of operators capable of creating all the rest of them, and in this way they can be used to define a good hypothesis for insight operators.

The argument presented would try to answer Research Question 2 (about the possibility of a computational-level definition describing insight)

The answer hinted by this argument is that the built-in aspects of a IPS computational description wont have to be necessarily relative to the problem at hand: computability theory might guarantee for a minimum set of operators capable of achieving all operators that will be ever necessary. Thus, achieving a single computational level problem describing insight might be possible.

As a last note, for completeness, we point out also some potential problems that this framework might have.

- Building up the search operators from such a small set of initial ones would make the computation really intractable, since one would need the combination of many simple operators to reach any useful operator.

This is a fair point, but the computation described by *eRCT* was also intractable to begin with. As for the excessive number of possible combinations of innate operators, which seem to make even the simplest insight impossible, one might further theorize that the set I is updated after every IPS problem solved, so that I doesn't simply contain the minimum set of search operators, but also other search operators found useful from previous experiences and learning.

- Technically speaking, the aforementioned set of basic operators from mu-recursive functions, although simply definable, is still composed of an infinite number of basic functions (for example, the set of constant functions contains a function for each possible constant output). This would create some problems for a search tree algorithm (the algorithms that usually solve such computational descriptions) trying to find the solution to the computational-level description.

This algorithmic technical detail might be solved by using a different dovetailing search algorithm in the infinite tree in width and length that is being spanned. Alternatively, one could also use other means of computation. For example, SKI calculus is comprised by just 2 initial functions, but is still Turing complete [Van Heijenoort, 1967].

- The present IPS computational description would be able to output any computable function as mu-recursive functions are designed to be able to span all computable functions. This might be seen as a problem as a computational description that has in its output the possibility of forming any computable function might be seen as too powerful to correctly define the human problem solving cognitive function.

This is a correct observation, yet this might be actually necessary for a correct definition of human insight problem solving abilities, given the many everyday examples we have of humans solving many different kinds of problems. The following though experiment might be able to give an intuition for this. Imagine receiving as a problem a computational description (in Marr's sense), and the given task is to output an algorithm capable of solving that description. Now this computational description problem that needs an algorithm as a solution is undoubtedly an instance of a problem that human problem solving capacities can try to solve. Yet the output asked to solve this problem is potentially any algorithm (indeed, this is somewhat the job that a programmer does). So one must conclude that human problem solving abilities could technically span all computable functions. As a further note to the previous objection, in the next argument and in Section 7 we will point out that the previous IPS computational description, to be computable by an algorithm, actually needs a finite bound in the number of applications of the search operators. This bound will make the computational description described above incapable of reaching all computable functions as outputs, and so the issue of excessive power should furthermore be resolved.

6 Unbounded Argument: Necessity for a bound in the steps used to reach the solution

6.1 Introduction

Looking closer at the *eRCT* computational definition, it's noticeable that one piece of information that is in input to the description is the integer k_s, representing the maximum number of applications of the search operators ("steps") necessary to reach the solution. Informally speaking, this number tells us how many search operators we can combine together before having to stop and start anew with a new combination of operators. When all combinations are exhausted the only alternative would be to restructure the problem representation and start again.

The fact that this information is given as part of the input of the problem is important because it gives a bound to the computation, after which a possible PS algorithm solving the problem has to stop, and if no solution is found, the necessity for an insight arises.

In this argument we will analyze the role of this k_s bound, and discover its critical role in the computational description. This finding warrants further empirical research about the bound k_s introduced, and will make us notice that a possible algorithmic theory of IPS would then be also critically influenced by the presence of this bound. This argument will give us an answer to Research Question 3, about how the differences from *eRCT* and eRCT inform us about characteristics missing from eRCT.

6.2 Argument

If we recall the *eRCT* definition, we can notice the presence of three bounds k_S,k_c,k_d, given in input, that are used to know the maximum sequence of operators allowed before restructuring. The reason given for the introduction of these integers, as stated in the original article, is that they permit the complexity analysis of the computational description. Similarly, in the eRCT informal model it is not specified that this bound must exists [Danek et al., 2016] [Öllinger et al., 2014]. So it would be interesting to have a closer look and analyze a version of *eRCT* without this bound, given that its introduction was due to technical reasons and not backed by the theory. In the following argument, we will analyze such version of *eRCT*, called *unbounded− eRCT*, that is deprived of the bound k_s. By doing so, we will discover that *unbounded− eRCT* is actually uncomputable, that is, there is no algorithm capable of finding a solution to such computational problem definition. Put more formally:

> *unbounded− eRCT*
> **Input:** A problem representation p (a predicate structure) with chunk-structure D (a grouping of predicates forms a chunk); a chunk-type set T (possible chunks that can make a chunk-structure), a search-operator set O, a constraint-set $C = (C_G, C_O)$ (constraints in using the operators and in what the solution looks like), and integers k_C, k_D.
> **Output:** A sequence s consisting of the application of $\leq k_C$ constraint relaxation and $\leq k_D$ chunk decomposition operators and an unspecified number of search operators from O in any order that transforms p into a goal state consistent with C_G, if such an s exists, and special symbol $\perp$ otherwise.

This formalism describes an *IPS* description based on the eRCT architecture, without the previously cited k_S. We wonder (and give a negative proof of it in Appendix B) if we could ever find an algorithm capable of computing this problem.

The intuition behind the impossibility of such algorithm, is that, as we can recall from the Generalization Argument, search operators and predicate structures are Turing complete, that is capable of representing any possible Turing machine. This makes them very powerful tools, but this also comes with some limitations.

One famous result from computability theory, called the Halting problem, tells us that is impossible to devise an algorithm that, when given the description of a Turing machine and an input, could tell us whether the machine would halt or not while running on that input. This fact has an important implication for our computational description. By not giving a limit k_s to the number of operators, it is as if we were asking if a possible combination of operators ever "halts", that is, if there is a combination of the operators that ever satisfies the goal constraints C_G. And considering the correspondence between search operators and Turing machines, we can conclude that if we had an algorithm to solve *unbounded−eRCT*, we could also solve the Halting problem. But since this is impossible, an algorithm solving *unbounded−eRCT* does not exist.

For another way to look at this issue, we can argue for the necessity of this bound because otherwise we would have an a priori way to know if a problem has a solution or not. This suggests that this bound k_s is necessary for creating a computational problem that is computable to begin with. And, since we believe that cognitive functions are computable to begin with, this prerequisite would be necessary for any computational level explanation.

The impossibility of knowing a priori if some search operators combination leads to solution means that the constraint k_S is actually necessary, and not just a complexity analysis bound. Without it the problem would actually be uncomputable. It may be that the solution searched is just a few operators application away, as it may be that it could never be reached with those operators. This is an important fact about PS or IPS models, they will never be perfect solvers for problems, as the solution for the problem to be solved might lie a few steps away from k_s.

Furthermore, the necessity of this bound warrants further speculations about its origins. For example, as we will better see in the discussion, we could speculate about motivational or working memory aspects being predictors for it. Additionally some thought should also be given to how this bound might change over different problems, and whether it is fixed in nature or mutable for different problem representations. This bound might not be a simple correction to make the problem computable, its criticality might suggest its use as a criterion to know when restructuring is necessary in an insight problem.

It is interesting that this aspect seems novel to the literature, considering also that this impossibility result stems from well-known results of computability theory. One hypothesis about why this issue arose is that the initial PS models by Simon and Newell were based on search trees of propositional logic proofs [Newell et al., 1959]. Propositional logic is decidable, that is, it is possible to know a priori if a formula is a theorem that follows from some logical rules (it is indeed

less "powerful" that the Turing complete models we are dealing with). So the bound on the steps was not logically necessary on that kind of model. This might have caused the omission of the bound in the number of steps in many following models of PS.
The line of reasoning of this argument can be summarized analytically as follows:

> Assumption 1: It is not clear whether $unbounded - eRCT$ could be a good description for an IPS cognitive function.
>
> Theorem 1: $unbounded - eRCT$ isn't computable.
>
> Assumption 2: cognitive functions are a subset of computable functions.
>
> Conclusion: $unbounded - eRCT$ is not a good description for an IPS cognitive function.

This line of reasoning would be an example answer to Research Question 3. When developing an algorithmic theory trying to predict human data, starting from an $unbounded - eRCT$ computational description would lead to failure, as no algorithm solving $unbounded - eRCT$ exists. So the computational analysis hints that a bound k_S in input is necessary. This is an interesting example about how a well-thought-out computational level analysis is important in the process of understanding scientifically cognitive functions.

7 Discussion

In the following Subsections we will discuss how the results shown in this thesis might influence our empirical understanding of problem solving

7.1 Discussion of the Generalization Argument

With the Generalization Argument we showed that the *eRCT* formalization is potentially strong enough to encode all problems. We could reach this result because an algorithm solving *eRCT* is capable of simulating any computable function, and thus will surely be able to simulate also IPS cognitive functions.

This is an important result given that various IPS reviews [Batchelder and Alexander, 2011][Ohlsson, 2011] doubted such formal possibility. Moreover this result guarantees that eRCT could be applied to virtually all insight problems, and thus assuring that this theory could be evaluated against many other examples of insight problems. For example, thanks to this result we have the certainty that we could use eRCT to model not only matchstick problems, but potentially any problem we would want to, provided that we input to the model a representation capable of solving the problem. In this way we could use an eRCT-solving algorithm to provide a baseline for human empirical studies.

Indeed one such class of algorithms could for example be ideal-observer models algorithms. These are algorithms that explore a problem space and find out an optimal sequence of moves that leads from the starting state toward a goal state [Batchelder and Alexander, 2011, Page 81]. So we potentially already have some algorithmic options that could be used to provide a best-case scenario depiction for problem solving any insight problem.

Another conclusion from the Generalization Argument is that the N&S-PS system is a Turing complete model of computation. This is interesting, because the N&S-PS system is the basic search space mechanism of problem solving models, without insight. So this result hints that probably not much else in computational terms is needed to describe insight problem solving. This is an interesting aspect, maybe already partially noted in [Kaplan and Simon, 1990], which suggests that the old models based on the Simon and Newell architecture could also be applied for IPS.

Yet, the line of reasoning from the Generalization Argument makes us also notice how in the eRCT theory the initial problem representation must be given in input together with search and restructuring operators capable of finding a solution. The origin of this input is not explained and assumed as coming from lower-level problem perception cognitive abilities. Thus, even if eRCT can potentially be applicable to all problems, we asked ourselves if there was a more general way to produce those insight restructuring mechanisms. This is what we investigated with the Minimality Argument.

7.2 Discussion of the Minimality Argument

With the Minimality Argument we discovered that the restructuring seems to be built in eRCT. This means that it is the restructuring information that permits to reach a problem representation such that the search operators starting from that problem representation can reach a solution. Yet no explanation is given for the origin of this restructuring operators, as this information is given in input to the computational description. This causes the problem of explaining away the insight ad infinitum, since the restructuring information might in turn be searched for by other types of operators.

This is in line with a previous review [Ash et al., 2009] that concluded there was too little scientific progress to justify restructuring mechanisms (such as the chunking decomposition theory from [Knoblich et al., 1999]) as unique explanations for IPS. Citing from their article:

> "We believe that equating “insight” and “restructuring” has led to an underlying assumption in the literature that different theories of “insight” are in some way competing or mutually exclusive, when in fact there may be multiple processes which can lead to new discoveries or “Aha!” experiences."[Ash et al., 2009, page 251]

Indeed with the Minimality Argument we proposed a framework that could lead to a solution to the issue of defining insight properly. We propose that insight can be represented with an innate set of computable functions that can be provably complete (that is, can create any possible function with their combination). Then this basic innate set can build all necessary insights functions, and among those the restructuring that were previously directly built-in in the computational description.

This line of thought may have important implications for the future of the problem solving field. First of all, it begs for the search of these basic innate insights that could form any possible insight. This question could be approached theoretically (as we have done in part in this paper) by looking at already existing models of computation that have some basic functions that could provably form all possible computable functions (examples of these could be mu-recursive functions [Kleene, 1968] or SKI combinator calculus [Van Heijenoort, 1967]). Then one could hypothesize that some basic functions similar to those are implemented by the human brain. For example, a recursive function is common in many models of computation, and there is some evidence for a modality independent recursive cognitive function, implemented in the left inferior frontal gyrus [Bahlmann et al., 2008] [Friederici et al., 2011] .

On the other hand the question about the existence of innate insights might be approached also from a more empirical prospective. One could try to look experimentally at some basic human problem solving processes, and then try to prove that they would indeed be powerful enough to generate computable functions. For example, some neuroimaging data supports the existence of two models of reasoning about problems [Goel, 2007] one linguistic-temporal and the other geometrical-parietal. The linguistic-temporal system seems more active for tasks that involve problems related to syntax understanding (for example, given that A>C, and that C>B, what is the relation among A and B?), while the geometrical-parietal is more active in tasks such as mental imagery manipulation. From these results one might speculate that the geometrical system is the one used to combine simple representations of search operators into more complex ones. While the linguistic system, given its ability to generate grammar structures recursively, might be implementing a recursive function capable of generating a set of search functions different from the geometrical system.

Furthermore, other empirical results could stem from the Minimality Argument. We proposed the existence of a computational description that could be able to generalize to all possible problems, without having to build-in specific search operators capable of solving those problem. This theoretical investigation would thus be useful for future algorithmic attempts at problem solving.

Historically, finding the correct search operators combination that could solve a problem was always tackled with brute force methods [Simon, 1974] exploring a tree-like search space. In this framework, one of the innovations of the Simon and Newell's framework [Newell et al., 1959] was the addition of a heuristic system that would direct the search in the search space, and make the search quicker. Similarly, given that the search space of algorithms solving our computational description would be massive, other heuristical processes would have to be devised.

For example, methods such as the Monte Carlo tree search [Chaslot and Bouzy, 2008] could be used to optimize this search space. These methods are based on a random sampling of the search space, capable of probabilistically inferring the best combinations of operators. These type of algorithms could thus recreate the randomness of a problem solving process and may be a stepping stone in connecting IPS research to Bayesian brain accounts.

Furthermore, recently a new set of algorithms based on neural networks and reinforcement learning have been used to effectively explore search spaces of game moves. The first of those algorithms, AlphaGO [Silver et al., 2016], was composed by a tree search procedure guided by a deep learning network, capable of learning over many trials an appropriate exploration procedure for a given search space.

These algorithmic advances could be used to solve the computational description proposed in this article. The really basic search operators proposed in our computational description would necessitate the search of a really deep search tree to find the solution. Some basic algorithms trying to exhaustively explore the search space would immediately have problems of combinatorial explosion of the search space. Yet the aforementioned algorithms are devised to infer useful patterns of combinations of operators in a search space, that could be progressively inferred and saved into memory for later use. So given enough training they could become efficient problem solvers by generating solutions inspired by combinations of operators that were previously found to be successful

As a final note, the work presented with the Minimality Argument could also be important for other areas of cognitive science research. The issue of "in-building" in IPS is similar to open issues in predictive processing. A problem that Bayesian models might have is the pre-specification of a built-in hypothesis space [Perfors et al., 2011]. This is because Bayesian models usually have already in input (and thus build-in) the hypothesis space used in their inference tasks, which may be seen as an unfair advantage given to the model. Similarly to IPS, the origin of these hypotheses spaces might cause an infinite regress trap. Indeed, where could these hypotheses spaces be taken from? They will also have to be built from some parameters, which will have in turn to be built-in. So since a pre-specification of the hypothesis space will always be necessary, then the actual question should be shifted to the level of in-building in the model, and not its actual necessity. [Perfors, 2012]

Additionally, the idea of generating the set of new search operators that are needed to solve a problem seems to be very similar to the field of study of models capable of generating new hypothesis [Gentner and Colhoun, 2010] [Tenenbaum et al., 2006]. This field is home to some very different scientific lines of research. For example, the work of [Goodman et al., 2012] tried to build a probabilistic programming language, Church, capable of probabilistic inference in pre-specified generative models. Interestingly, given that Church is constructed upon lambda calculus, a Turing complete method of computation, the possible programs that can be built with it coincides with computable functions. Otherwise, hypothesis generation can also be explained in the analogical inference framework [Gentner and Smith, 2013] by a computational model of abduction proper [Blokpoel et al., 2018]. Finally, as argued before, some of the issues pointed out by this work seem to be similar in nature to other open problems in the fields of linguistics (as pointed out with the concept acquisition example described in the Minimality Argument). Given all these examples we conclude that the methodologies and insights presented in this article could have some utility also for the aforementioned areas of research.

7.3 Discussion of the Unbounded Argument

In this argument we showed how one of the bounds of present day IPS theory (and PS in general) is critical for making the theory implementable by an algorithm, and thus critical for making it a plausible hypothesis for a cognitive function.

Informally speaking, the necessity of a bound in the number of search operators represents how it is impossible to know if a given approach we have chosen to solve a problem will actually lead to a solution. Hence when an algorithm is programmed to solve *eRCT* it could explore the search space that is spanned by the application of the search operators to the problem representation, but the depth of this space will be defined by the bound k_s of maximum search operators application.

This result informs us that to reach a more complete understanding of PS and IPS, some empirical investigation of this bound might be warranted. For example, one hypothesis might be that motivational aspects could influence the bound k_s, that is, the depth of the search tree that we are inclined to explore might be directly influenced by the degree of motivation we have for solving the problem at hand. A problem solver with low motivation (and so low k_s) might give up the search almost immediately after having explored just a bit of the search space spanned by the application of few k_s search operators. While a more motivated problems solver (someone with a higher k_s) will explore more of the search space and maybe find a solution. Indeed the fact that motivational aspects influence problem solving skills is a known results in the literature [Mayer, 1998], and thus the computational description here proposed might be a useful tool to better formalize that line of research.

Additionally, working memory limits are a known variable that potentially influences problem solving [Sweller, 1988]. One explanation for this might be that higher working memory might speed up mental processing, but this hypothesis is not translated without difficulties to computational models. In this prospective, the bound k_s might be a good candidate to represent working memory, since a bigger k_s would represent a bigger search space exploration. Akin to the difference between a chess player that can calculate 2 moves ahead and one that can calculate 3 moves ahead, a difference in k_s might be a good representation of different individuals' working memory span.

Furthermore, the bound k_s could also be one candidate for defining the boundary between unsuccessful PS and a first attempt at IPS. That is, when we are looking unsuccessfully for a solution to a problem, the number k_s might inform us about when our search needs to be revised by an insight. If no solution is found within the bound k_s, even if the solution is a step away from this bound, *eRCT* would still stop the search operators combination and start a restructuring process.

Finally, the results of this argument impose to also redefine the computational description of IPS presented in the Minimality Argument. Given that the computable functions of *IP S* are as powerful as the search operators and problem representation of *eRCT* then the computational description of IPS would also need the same bounds to be computable, i.e.

> *IP S*
> **Input:** A problem representation input p (an input), an initial mu-recursive function set O (the initial search operators), an insight mu-recursive function set I (comprised of the basic set of functions that can span all possible computable functions), constraint-set $C = C_G$ (constraints in what the solution looks like), integers k_s, k_d.
> **Output:** A sequence s consisting in the application of maximum k_s computable functions from O and of maximum k_d from I in any order that transforms p into a goal state consistent with C_G, if such an s exists, and special symbol $\perp$ otherwise.

7.4 Final remark

In conclusion, we hope that this thesis has helped in achieving a more complete understanding of IPS. The major finding was that the Turing completeness of some elements of eRCT implies that trying to encode with them any problem is indeed possible. Yet, great care must be given to properly define the computational definition of problem solving, as the Turing completeness of N&S-PS might make the definition uncomputable. Furthermore, we hope to have shown an alternative definition of IPS that doesn't fall into the shortcomings of eRCT, and that could be a possible candidate for the origin of any insight. Finally, as a higher level remark, we hope to have showed how the use of computational-level analysis might be of use in inspecting the definitions and the possible implications of human cognitive functions.

A Appendix: Turing completeness proof

We first prove that search operators operating on a predicate structure are Turing complete. Then, in section A.2 we will prove, by extension, that also search operators operating on a predicate structure and informed by a constraint C_G are Turing complete

A.1 Search operators acting on predicate structures are Turing complete

Informally speaking, a Turing machine can be described as a list of operations that can be carried on an unbounded tape, divided in cells. The operation is made by a head that reads the symbol in the cell it is positioned upon. This head is also in a particular state, that dictates what to do based upon the symbol being read. The head can change the symbol it is currently reading in another symbol, or leave it be, and then it shifts left or right (this instruction is again in the state) and changes the state it is in.

Defined formally [Hopcroft, 2013], a Turing machine is a 7-tuple $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$ where:

- Q is a non-empty and finite set of states;
- Γ is a finite, non-empty set of alphabet symbols;
- $b \in \Gamma$ is the blank symbol (this symbol is the only one allowed to occur on the tape infinitely often);
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of input symbols;
- $q_0 \in Q$ is the initial state;
- $F \subseteq Q$ is the set of accepting states, the ones that halt the computation. The initial input (that is, the initial tape contents) is said to be accepted by M if it eventually halts.
- $\delta : (Q \setminus F) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function, where L is left shift, R is right shift. It represents the "head" of the machine that, depending on the state it is in, reads the current symbol, overwrites with another symbol or leaves it as it is, then shifts left or right and changes the state it is in. If δ is not defined on the current state and the current tape symbol, then the machine halts

To prove that a data-manipulation system is Turing complete we need to prove that it is capable of simulating any Turing machine [Sipser, 2006] We will show that a predicate structure can encode any tape, head position, and current state. Likewise, we will show that search operators can act to shift the head of a Turing machine constructed with a predicate structure. In the following we describe a computable method, that is, an algorithm that transforms any Turing machine and its input in the relative predicate structure+search operators. This is because predicate states can be described as vertex labelled directed acyclic graph, with objects as leaves and predicates as internal vertices. Thus we can create the following predicate structure:

- The initial cells of the tape are coded as predicates, with each predicate being assigned an object denoting the symbol written on the tape taken from Σ, and the blank symbol b placed at the edges of the predicate structure to represent the unboundedness of the tape (see Figure 4A). Additionally, the adjacency of the cells is represented with a predicate labelled "tape". Finally, another predicate with label q_0 denotes that the head of the machine is placed in the starting cell and in state q_0).
- The search operators are defined as substructure replacement rules of the form $X \to Y$ that operate on predicate- structures. We will use the search operators to define the behavior of the head. The state of the head and the symbol being read correspond to one specific search operator that is used to change the predicate structure by shifting the head left or right, change the symbol that was being read if necessary, and eventually creating a new predicate with the blank symbol if the computation exceeded the initial tape. These search operators are created from the transition function δ. For each transition in δ we create a number of search operators based on the possible surroundings of the head (for example, in the case of a Turing machine with tape symbols "1", "0", and blank b, the surroundings will be $1 - head - 1$, $1 - head - 0$, $0 - head - 1$, $0 - head - 0$, $b - head - 1$, $1 - head - b$...). This number of search operators will always be $|\Gamma|^2$ and so finite. These search operators will then transform the predicate structure accordingly to δ, by changing the symbol, shifting the head, and changing the state. Additionally, in the finite cases where a search operators is applied to the boundaries of the tape, those search operators will also add the blank symbols as a new predicate at the edge of the predicate structure (This is made to guarantee the unboundedness of the tape). (see Figure for example)

So the search operators will be applied one after another to the predicate structure and they will simulate the behavior of a Turing machine.

To finalize the proof that predicate structures+search operators are Turing complete, we need to show that the search operators applied to a predicate structure will have a finite number of

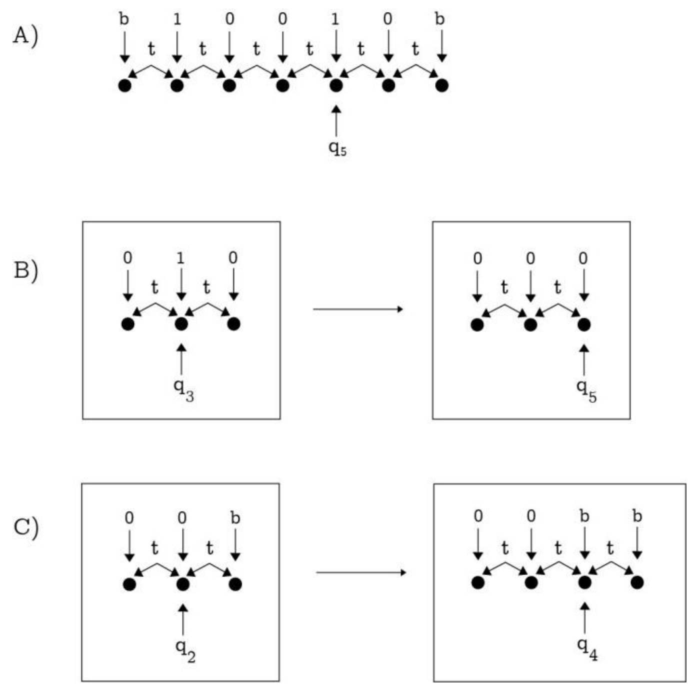

Figure 4: An example of how to encode a Turing machine with predicate structure+search operators. We can see a predicate structure encoding the tape (labelled "t"), with the input symbols encoded as predicates, and also one extra predicate denoting where the head is and in what state (q_3). Then the search operators transform a predicate structure into another by changing the symbol being read, shifting the head, and changing the state. In the example we see a search operator that encodes the case when the head is in state q_3 and over a "1" symbol, with the specific case of having adjacent tape symbols as "0"s. This predicate then encodes how the head is shifted right and set at state q_5 after changing the symbol into a "0". At the bottom, we see one last example of search operator, where the head, when reaching the boundaries of the predicate structure, must add a blank symbol predicate structure to account for possible blank tape cells.

applications (that is, the application of search operators to a predicate structure "halts") if and only if the Turing machine they are simulating halts on the given input.

First we show that if a simulation of a predicate structures+search operators is finite then also the corresponding Turing machine halts. The search operators will be specified by the transition function δ and just one search operator will be available at a given time to transform the predicate structure representing the tape into another predicate structure representing the tape after a head move. Accordingly, this simulation of the computation of a Turing machine will end if a search operator writes an object labelled with an accepting state in the predicate structure (In fact, note that the domain of δ doesn't have the set F as a subset, so there will be no search operators available to change the predicate structure anymore). And the other condition for the halting of a Turing machine is also covered: if there are no search operators for the current predicate structure, it means that the δ function wasn't defined for the respective head configuration, and so the computation will halt.

Similarly, if a Turing machine halts then also the corresponding predicate structures+search operators simulation will finish. This is because a Turing machine halts if an accepting state is reached with a given step of the machine. Given that the transition function δ is translated exactly by the search operators, then the search operators will exactly follow the steps of the Turing machine. So when the Turing machine finds an accepting state and halts then also the search operators will have printed an accepting state on the predicate structure and have no further application. Similarly, also if the Turing machine state is not defined in δ then there will be no search operators applicable to continue the search, and the simulation will finish.

Furthermore we also notice that the translation from Turing machines to predicate structures and search operators presented above has no time or space overhead blow-up. There is a polynomial space overhead increase because the Turing machine tuple with an input long n cells will be translated into a $n + 2$ number of cells predicate structure, while the number of operators will grow in a Γ^2 fashion. Furthermore, as we noted above, the number of executed TM transitions exactly equals the number of search operator applications. This proves there is no time overhead increase.

A.2 Extra proof with C_{tt}

Since in the text we also talk about a N&S-PS system, here we will prove that also predicate structures and search operators informed by a constraint C_G (we will call this a N&S-PS system) are Turing complete models of computation.

The proof is similar to the one above: the cells are coded as predicates and search operators are acting as a Turing machine head based on the transition function. Furthermore we model also the halting condition with a logical constraint C_G, this will be a first order logic formula $H(p) \vee D(p)$ that is satisfied when anywhere on the predicate structure is present an accepting state or if the predicate structure represents a state that is not defined in δ. $H(p)$ is intuitively possible since it just checks if anywhere on the predicate structure is a symbol from the finite set F. Also $D(p)$ is possible, since it is a formula that compares the current predicate structure to $(Q \setminus F) \times \Gamma$, which is a finite set, and is satisfied when the comparison fails.

The N&S-PS system operates as the system defined above, with the search operators manipulating the predicate structures, until $H(p) \vee D(p)$ is satisfied.

Now the proof is the same as the previous proof since we can notice that $H(p) \vee D(p)$ represents the halting condition of a Turing machine by definition. So a N&S-PS system simulating a Turing machine will reach a solution that satisfies the constraint $H(p) \vee D(p)$ if and only if the simulated Turing machine halts.

Finally we have to note that an algorithm based on the N&S-PS system that solves the *eRCT* description (with the restructuring input set to zero) will have to have also in input the bound k_s. This is important since we will show in the next appendix that a computational definition lacking that bound is uncomputable. This means that our N&S-PS system is Turing complete just for an arbitrarily large k_s, in a similar sense as a home computer is Turing complete just if it has an arbitrarily expandable memory. Yet we argue this is not a problem for our argumentation, as cognitive functions of human individuals necessarily have a time and space limit (being the brain a limited physical system, just like a home computer). So the k_s bound can be set to be arbitrarily large but in practice will never be required to be infinite, and still an N&S-PS system will be able to simulate any cognitive function.

B Appendix: proof that $unbounded - eRCT$ is uncomputable

We will give two proofs for the result that $unbounded - eRCT$ is uncomputable. The first gives a reduction from $unbounded - eRCT$ to the Post's correspondence problem, an undecidable problem [Post, 1946]. The second, more trivial, directly uses the result from Appendix A to show that $unbounded - eRCT$ could solve the Halting problem.

B.1 Proof 1

We will show that $unbounded - eRCT$ is not computable by proving that an algorithm for $unbounded - eRCT$ could solve Post's correspondence problem, an undecidable problem [Post, 1946].

Post's problem is defined as follows:

> *Correspondence − problem*
> **Input:** a 2 symbol alphabet A, and two finite lists $a_1, a_2, \ldots a_n$ and $b_1, b_2, \ldots b_n$ of combination of symbols over A.
> **Output:** a sequence of indices $(i_k)_{1 \leq k \leq K}$ with $K \geq 1$ and $1 \leq i_k \leq n$ for all k (intuitively, this says that there are as many indexes as we want but they are all smaller than the number n of list length) such that:
> $a_{i_1} a_{i_2} \ldots a_{i_K} = b_{i_1} b_{i_2} \ldots b_{i_K}$
> if such sequence exists.

An example follows: let's set $A = \{x, y\}$ and $a_1 = x$, $a_2 = xy$, $a_3 = yyx$, $b_1 = yxx$, $b_2 = xx$, $b_3 = yy$. Then a solution is the sequence $(3, 2, 3, 1)$, since $a_3, a_2, a_3, a_1 = b_3, b_2, b_3, b_1 = yyxxyyyxx$
Now let's recall the formal definition of *unbounded − eRCT* :

unbounded − eRCT
Input: A problem representation p (a predicate structure) with chunk-structure D (a grouping of predicates forms a chunk); a chunk-type set T (possible chunks that can make a chunk-structure), a search-operator set O, a constraint-set $C = (C_G, C_O)$ (constraints in using the operators and in what the solution looks like), and integers k_C, k_D.
Output: A sequence s consisting of the application of $\leq k_C$ constraint relaxation and $\leq k_D$ chunk decomposition operators and an unspecified number of search operators from O in any order that transforms p into a goal state consistent with C_G, if such an s exists, and special symbol $\perp$ otherwise.

We will give an algorithm that transforms each instance of the *Correspondence− problem* into an instance of the *unbounded− eRCT* . Thus, if *unbounded −eRCT* is computable, so too the *Correspondence problem* is computable. But since that is a contradiction, we conclude that *unbounded eRCT* is uncomputable as well. The algorithm is as follows: given a 2 symbol alphabet A, and two finite lists $a_1, a_2, ...a_n$ and $b_1, b_2, ...b_n$, we create a problem representation p comprised of just one object, labelled "I" (for the initial state). Then we create a number n of search operators based on the pairs (a_1, b_1), $(a_2, b_2)...(a_n, b_n)$ such that the search operators transform the simple predicate structure "I" into a "a_i, b_i-I" predicate structure (see Figure 5). Then the only constraint C_G is set as the first order formula $E(a, b)$ that is satisfied only when the a and b lists of symbols are equal (this is trivially possible since the terms of first order logic formulas can also be functions, and it is possible to devise a function that takes in input the objects of the predicates, forms two lists a and b, and outputs 1 if they are equal). The other inputs of the *unbounded eRCT* are given as empty (that is, no chunk structure or chunk-type set T , no constraints C_O, and the integers k_D and k_D set equal to zero).

If an algorithm that solved *unbounded−eRCT* existed then, together with the aforementioned transformation algorithm, it could also solve the *Correspondence−problem*. This is because there is a one-to-one map between the possible predicate structures reachable with those search operators and the possible lists $a_{i_1} b_{i_1}, a_{i_2} b_{i_2}, ...a_{i_K} b_{i_K}$, that can then be recognized by $E(a, b)$.

More formally, we can prove that an instance of the *Correspondence − problem* will have a solution if and only if the corresponding instance of the *unbounded eRCT* has a solution.
We first prove that if an instance of the *Correspondence−problem* has a solution then the corresponding instance of the *unbounded eRCT* has a solution. if a *Correspondence problem* has a solution then there is a sequence of indexes such that $a_{i_1} a_{i_2} ...a_{i_K} = b_{i_1} b_{i_2} ...b_{i_K}$. This means that there exists an application of the search operators corresponding to the translated instance such that $E(a, b)$ is satisfied. This is because the search operators defined above can continuously transform the "I" predicate to add new couples a, b to the predicate structure, and so each possible configuration of $a_1, b_1, a_2, b_2...a_n, b_n$ can be checked by $E(a, b)$.
Now we prove that if an instance of the *unbounded−eRCT* has a solution then the corresponding instance of the *Correspondence problem* has a solution. if an *unbounded eRCT* instance has a solution then the $E(a, b)$ first-order logic formula is satisfied by some configuration $a_1, b_1, a_2, b_2...a_n, b_n$. This configuration corresponds to a sequence of indexes such that $a_{i_1} a_{i_2} ...a_{i_K} = b_{i_1} b_{i_2} ...b_{i_K}$ and thus also the *Correspondence problem* has a solution.

This concludes the proof. Interestingly, there exists a bounded version of the *Correspondence−problem* that is actually decidable, but NP-complete. This version can be reduced to the bounded version of *eRCT*, and, given that the above transformation algorithm is polynomial in time, it can be used as an alternative proof method for the computational complexity of *eRCT*.

B.2 Proof 2

A simple proof that *unbounded − eRCT* in not computable can be derived by showing that an algorithm solving the *unbounded − eRCT* could also solve the *Halting − problem*.

Halting problem
Input: a Turing machine M and an input i
Output: 1 if the computation of M on i halts, 0 otherwise

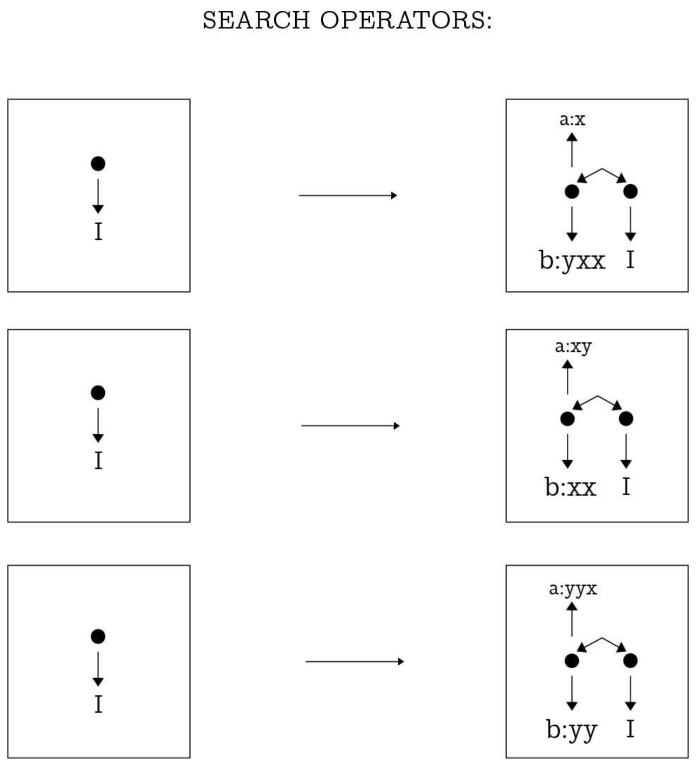

Figure 5: an example of the search operator rules that encode the list $a_1 = x$, $a_2 = xy$, $a_3 = yyx$, $b_1 = yxx$, $b_2 = xx$, $b_3 = yy$. With these search operators, one could form a chain of predicates each specifying a pair (a, b). Then multiple applications of the search operators can lead to any sequence $a_{i_1} b_{i_1}, a_{i_2} b_{i_2}, \ldots a_{i_K} b_{i_K}$

This is a well known undecidable problem [Sipser, 2006]. Hence if there is a way to convert an instance of the *Halting −problem* into an instance of the *unbounded −eRCT* , then we could solve the *Halting −problem* by solving *unbounded −eRCT* , but this is impossible so *unbounded− eRCT* is uncomputable as well.

We show an algorithm that transforms an instance of *Halting− problem* into an instance of *unbounded−eRCT* . This is done by keeping the chunk-structure D and the chunk-type set T equal to zero, and also $k_c = k_d = 0$ while the problem representation p and search-operators set O are created based on M and i accordingly to Appendix A.1. The only constraint $C = C_G$ (C_0 is given as null, as there are no constraints in the use of search operators) is the first order logic formula $H(p) \vee D(p)$ created as in Appendix A.2, that will be satisfied if either one of the accepting states of M is printed on the predicate structure or if some not specified cases from the domain of the δ functions (these will be finite in number) is printed on the predicate structure. Then from Appendix A.2 this constraint is satisfied if and only if M halts, and the algorithm solving *unbounded−eRCT* outputs the sequence of search operators that satisfy the constraints. Otherwise the algorithm outputs the special symbol $\perp$. This algorithm evidently solves the *Halting − problem*, and this completes this proof.

C Appendix: mu-recursive functions

The mu-recursive functions [Kleene, 1968] are a class of functions on the natural numbers that are provably Turing complete. They are partial functions taking in input a finite tuple of natural number and outputting a natural number. They are defined from a starting set of 3 functions classes.

1. The constant function class is defined as

$$f(x_1, \ldots, x_k) = n$$

That is, for each number of inputs k and for each number n, it returns the constant n.

2. The successor function

$$S(x) = x + 1$$

outputs the successor of the input number x.

3. The projection function

$$P_i^k(x_1, ..., x_k) = x_i$$

for any k and i such that $1 \leq i \leq k$, it takes in input a number k of inputs and returns the i-th input.

These basic function classes are then combined by 3 operators.

1. The Composition operator $\circ$, defined on a m-ary function $h(x_1, . . . , x_m)$, and on m k-ary functions $g_1(x_1, . . . , x_k), . . . , g_m(x_1, . . . , x_k)$ as $h \circ (g_1, ..., g_m) = f$ where

$$f(x_1, .., x_k) = h(g_1(x_1, ...x_k), ..g_m(x1, ..., x_k)$$

2. The primitive recursion operator ρ, defined over a k-ary function $g(x_1, . . . , x_k)$ and a k+2-ary function $h(y, z, x_1, . . . , x_k)$ as $\rho(g, h) = f$ with

$$f(0, x_1, ..x_k) = g(x_1, ..., x_k)$$
$$f(y + 1, x_1, ..x_k) = h(y, f(y, x_1, ..x_k), x_1, ..., x_k)$$

Intuitively this operator defines a new function recursively starting from $f(0) = g()$, with

$$f(1) = h(0, f(0))$$
$$f(y + 1) = h(y, f(y))$$

3. The Minimization operator μ defined over a $(k + 1)$-ary total function $f(y, x_1, . . . , x_k)$ as $\mu(f)(x_1, ..., x_k) = z$ where

$$f(z, x_1, ..., x_k) = 0$$
$$f(i, x_1, ..., x_k) > 0 \text{ for all } i = 0, ...z - 1$$

Intuitively, the minimization operator searches a z from 0 and returns the smallest argument z that causes the function $f(z)$ to return zero.

For an example, let's see how we could derive the addition function $f(y, x) = x + y$: we first define some initial functions

$$\begin{aligned} S(x) &= x + 1 \\ C(x) &= x \\ P(y, z, x) &= z \\ S'(z) &= S(P(y, z, x)) = z + 1 \end{aligned}$$

Then we have $\rho(C, S^j) = f$ with $f(0, x) = C(x) = x$ and $f(y+1, x) = S^j(y, f(y, x), x)$. This finally gives us

$$\begin{array}{lllllll}
f(0, x) & = & C(x) & = & x & & \\
f(1, x) & = & S^j(0, f(0, x), x & = & S^j(0, x, x) & = & x + 1 \\
f(2, x) & = & S^j(1, f(1, x), x) & = & S^j(1, x + 1, x). & = & x + 2 \\
\ldots & & & & & & \\
f(y, x) & = & x + y & & & &
\end{array}$$

References

[Anderson, 1990]Anderson, J. R. (1990). *The adaptive character of thought*. Lawrence Erlbaum Associates, Hillsdale, NJ.

[Ash et al., 2009]Ash, I. K., Cushen, P. J., and Wiley, J. (2009). Obstacles in investigating the role of restructuring in insightful problem solving. *The Journal of Problem Solving*, 2(2):3.

[Bahlmann et al., 2008]Bahlmann, J., Schubotz, R. I., and Friederici, A. D. (2008). Hierarchical artificial grammar processing engages Broca's area. *Neuroimage*, 42(2):525–534.

[Batchelder and Alexander, 2011]Batchelder, W. H. and Alexander, G. E. (2011). Insight problem solving: A critical examination of the possibility of formal theory. *Journal of Problem Solving*, 3(2):51–100.

[Blokpoel et al., 2018]Blokpoel, M., Wareham, T., Haselager, P., Toni, I., and van Rooij, I. (2018). Deep analogical inference as the origin of hypotheses. *Accepted in The Journal of Problem Solving*.

[Chaslot and Bouzy, 2008]Chaslot, G.M.J., W. M. H. H. U. J. and Bouzy, B. (2008). Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation (NMNC)*, 4(03):343–357.

[Chu, 2009]Chu, Y. (2009). *Human insight problem solving: Performance, processing, and phenomenology*. Vdm Verlag, Saarbrücken, Germany.

[Danek et al., 2016]Danek, A. H., Wiley, J., and Öllinger, M. (2016). Solving classical insight problems without Aha! experience: 9 dot, 8 coin, and matchstick arithmetic problems. *Journal of Problem Solving*, 9(1).

[Einhard, 1960]Einhard, V. K. M. (1960). *The life of Charlemagne*. University of Michigan Press, Ann Arbor, MI.

[Fodor, 1975]Fodor, J. A. (1975). *The language of thought*, volume 5. Harvard University Press, Cambridge, MA.

[Fodor, 2001]Fodor, J. A. (2001). *The mind doesn't work that way: The scope and limits of computational psychology*. The MIT press, Cambridge, Massachusetts.

[Friederici et al., 2011]Friederici, A. D., Bahlmann, J., Friedrich, R., and Makuuchi, M. (2011). The neural basis of recursion and complex syntactic hierarchy. *Biolinguistics*, 5(1-2):087–104.

[Gentner and Colhoun, 2010]Gentner, D. and Colhoun, J. (2010). Analogical processes in human thinking and learning. In *Towards a theory of thinking, Albrecht von Müller*, pages 35–48. Springer, New York City, New York.

[Gentner and Smith, 2013]Gentner, D. and Smith, L. A. (2013). *Analogical learning and reasoning.* Oxford University Press.

[Goel, 2007]Goel, V. (2007). Anatomy of deductive reasoning. *Trends in Cognitive Sciences*, 11(10):435–441.

[Goodman et al., 2012]Goodman, N., Mansinghka, V., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. (2012). Church: a language for generative models. *arXiv preprint arXiv:1206.3255*.

[Hopcroft, 2013]Hopcroft, J. E. (2013). *Introduction to Automata Theory, Languages and Computation: For VTU, 3/e*. Pearson Education India, Noida, Uttar Pradesh 201301, India.

[Kaplan and Simon, 1990]Kaplan, C. A. and Simon, H. A. (1990). In search of insight. *Cognitive Psychology*, 22:374–419.

[Kleene, 1968]Kleene, S. C. (1968). *Introduction to metamathematics*. Ishi Press, Bronx, NY.

[Knoblich et al., 1999]Knoblich, G., Ohlsson, S., Haider, H., and Rhenius, D. (1999). Constraint relaxation and chunk decomposition in insight problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 25(6):1534–1555.

[Köhler, 2013]Köhler, W. (2013). *The mentality of apes*. Read Books Ltd, Redditch, Worcestershire, England.

[Laird et al., 1987]Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64.

[Marr, 1981]Marr, D. (1981). *Vision: A computational investigation into the human representation and processing visual information*. W. H. Freeman, San Francisco, CA.

[Massaro and Cowan, 1993]Massaro, D. W. and Cowan, N. (1993). Information processing models: Microscopes of the mind. *Annual Review of Psychology*, 44(1):383–425.

[Mayer, 1998]Mayer, R. E. (1998). Cognitive, metacognitive, and motivational aspects of problem solving. *Instructional Science*, 26(1-2):49–63.

[Newell et al., 1959]Newell, A., Shaw, J. C., and Simon, H. A. (1959). Report on a general problem solving program. In *IFIP Congress*, volume 256, page 64. Pittsburgh, PA.

[Newell and Simon, 1972]Newell, A. and Simon, H. A. (1972). *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ.

[Ohlsson, 1992]Ohlsson, S. (1992). Information-processing explanations of insight and related phenomena. In Keane, M. and Gilhooly, K., editors, *Advances in the Psychology of Thinking*, pages 1–44, London. Harvester-Wheatsheaf.

[Ohlsson, 2011]Ohlsson, S. (2011). The problems with problem solving: Reflections on the rise, current status, and possible future of a cognitive research paradigm. *Journal of Problem Solving*, 3(2):101–128.

[Öllinger et al., 2014]Öllinger, M., Jones, G., and Knoblich, G. (2014). The dynamics of search, impasse, and representational change provide a coherent explanation of difficulty in the nine-dot problem. *Psychological Research*, 78(2):266–275.

[Peebles and Cooper, 2015]Peebles, D. and Cooper, R. P. (2015). Thirty years after marr's vision: levels of analysis in cognitive science. *Topics in Cognitive Science*, 7(2):187–190.

[Penrose, 1994]Penrose, R. (1994). *Shadows of the Mind*, volume 4. Oxford University Press.

[Perfors, 2012]Perfors, A. (2012). Bayesian models of cognition: what's built in after all? *Philosophy Compass*, 7(2):127–138.

[Perfors et al., 2011]Perfors, A., Tenenbaum, J. B., Griffiths, T. L., and Xu, F. (2011). A tutorial introduction to bayesian models of cognitive development. *Cognition*, 120(3):302–321.

[Pinker, 1999]Pinker, S. (1999). How the mind works. *Annals of the New York Academy of Sciences*, 882(1):119–127.

[Post, 1946]Post, E. L. (1946). A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268.

[Silver et al., 2016]Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484.

[Simon, 1974]Simon, H. A. (1974). How big is a chunk? *Science*, 183(4124).

[Sipser, 2006]Sipser, M. (2006). *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology,Boston, MA.

[Sternberg and Davidson, 1995]Sternberg, R. J. and Davidson, J. E. (1995). *The nature of insight.* The MIT press, Cambridge, Massachusetts.

[Sweller, 1988]Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2):257–285.

[Tenenbaum et al., 2006]Tenenbaum, J. B., Griffiths, T. L., and Kemp, C. (2006). Theory-based Bayesian models of inductive learning and reasoning. *Trends in Cognitive Sciences*, 10(7):309–318.

[Vallée-Tourangeau, 2018]Vallée-Tourangeau, F. (2018). *Insight: on the origins of new ideas*. Routledge, Abingdon-on-Thames, United Kingdom.

[Van Gelder, 1995]Van Gelder, T. (1995). What might cognition be, if not computation? *The Journal of Philosophy*, 92(7):345–381.

[Van Heijenoort, 1967]Van Heijenoort, J. (1967). *From Frege to Gödel: a source book in mathematical logic, 1879-1932*. Harvard University Press, Cambridge, Massachusetts.

[Wallas, 1926]Wallas, G. (1926). *The art of thought*. Harcourt, Brace, New York.

[Wareham, 2017]Wareham, T. (2017). The roles of internal representation and processing in problem solving involving insight: A computational complexity perspective. *The Journal of Problem Solving*, 10(1):3.