Having your cake and eating it too

Towards a fast and optimal method for analogy derivation

Tijl Grootswagers

July 24, 2013

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE

Supervisors:

dr. T. Wareham	Memorial University of Newfoundland, St.John's, NL Canada
dr. I. van Rooij	Donders Centre for Cognition, Radboud University Nijmegen

External examiner:

dr. ir. J. Kwisthout	Donders Centre for Cognition, Radboud University Nijmegen

Student number:
0439789

Abstract

The human ability for forming analogies – the ability to see one thing as another – is believed to be a key component of many human cognitive capacities, such as language, learning and reasoning. Humans are very good at forming analogies, yet it is non-trivial to explain how they achieve this given that the computations appear to be quite time consuming. For instance, one of the most influential theories of analogy derivation, Structure-Mapping Theory (SMT) (Gentner, 1983) characterizes analogies as optimally systematic mappings from one representation to another. This theory has been previously proven to be intractable (formally, $\mathcal{NP}$-hard), meaning that computing SMT analogies requires unrealistic amounts of time for all but trivially small representations. However, a large body of empirical research supports the optimality assumption of SMT. This poses the question: If SMT is indeed descriptive of human performance, then how can we explain that humans are able to derive optimal analogies in feasible time? A standard explanation is that humans use a heuristic, which has also been proposed in the literature. A novel explanation is that humans exploit representational parameters to achieve efficient computation.

This thesis provides the first systematic controlled test of the heuristic explanation and a systematic comparison of its performance with that of the parameter explanation. The results establish two main findings: (1) The extent to which the heuristic is capable of computing (close to) optimal analogies is considerably worse than what was previously believed; and (2) an exact algorithm exploiting a key parameter of SMT can compute optimal analogies in a time that matches that of the heuristic. Based on these results we conclude that, in its current form, the heuristic explanation is lacking validity, and the parameter explanation provides a viable alternative which motivates new experimental investigations of analogy derivation.

Acknowledgements

I would like to thank my supervisors, Todd Wareham and Iris van Rooij, who have contributed to this thesis on many (if not all) levels. I very much enjoyed our email conversations, phone meetings, and live discussions. Thank you for your enthusiasm, guidance, support, contributions and for including me in the TCS group, the CogSci poster, and the AIfternoon session. I thank Johan Kwisthout for his helpful suggestions on earlier versions of this document. Furthermore, I appreciated the interesting discussions in the TCS-meetings on, amongst many other things, this project.

I thank my sister and brothers, who are awesome, and my parents, who have always encouraged the four of us in every way they can. Special thanks to Denise for her endless support during this project. Finally, I would like to thank all the people who have contributed in any way to this thesis, as well as those who provided the necessary distractions.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

"The ships hung in the sky in much the same way that bricks don't."

(Douglas Adams (1979): *The Hitchhikers Guide to the Galaxy*)

The above is an example of an analogy between ships and bricks (describing a very complex relation between the two). Humans are very good at understanding and forming such analogies (and simpler ones such as "school is like a prison" or "my father is like a rock"), and we use them in everyday life, even implicitly.

The ability to derive analogies is believed to be a key factor for human intelligence, underlying many cognitive functions like language (e.g. the use of metaphors), learning (e.g. generalization), reasoning (e.g. case-based reasoning) and other high level cognitive skills that make humans so smart (Gentner, Holyoak, & Kokinov, 2001; Gentner, 2003; Hofstadter, 2001; Kurtz, Gentner, & Gunn, 1999; Penn, Holyoak, & Povinelli, 2008). It is argued that analogy derivation also plays a part in low level processes like representation (Blanchette & Dunbar, 2002) and language comprehension (Day & Gentner, 2007). For example, there is support for analogy in similarity judgement (Gentner & Markman, 1997) and spatial learning (Smith & Gentner, 2012). In the case of language comprehension, people seem to focus on relational matches when comparing sentences (Gentner & Kurtz, 2006; Gentner & Christie, 2008; Rasmussen & Shalin, 2007). Analogies are used in creative processes, like scientific reasoning and problem solving, which use an analogy between the unknown and previously encountered situations to transfer knowledge about the known situation to the new situation (Blanchette & Dunbar, 2001; Dunbar, 1995; Gentner, 1983; Gick & Holyoak, 1980; Holyoak & Thagard, 1996).

It is however hard to explain how humans are so good at the non-trivial task of making analogies. For example, one of the most influential theories of human analogising, Structure-Mapping Theory (SMT) (Gentner, 1983), characterizes analogy derivation as finding the most systematic common structure between two representations. Computational complexity analyses have shown that SMT is computationally intractable, which implies that there can not exist a method that derives optimal analogies in feasible time for all representations. Humans, however, seem able to make analogies very quickly, and these analogies fit the optimal systematic analogies described by SMT (Clement and Gentner (1991); see also Section 2.3 of Gentner and Colhoun (2010)). This seems to contradict the intractability of SMT, assuming that human cognition is bound by computational resources.

Two explanations have been proposed to deal with this question of how humans are able to compute optimal solutions in feasible time. The first claims that humans are deriving close-to optimal analogies by using heuristics (Gentner & Markman, 1997). This is a standard explanation in cognitive science (Chater, Tenenbaum, & Yuille, 2006; Gigerenzer, 2008). A novel explanation argues that humans are only fast (and optimal) when the analogy has certain characteristics (van Rooij, Evans, Müller, Gedge, & Wareham, 2008). To date, there have been no controlled studies (outside of a limited test of the heuristic implementation of SMT (Forbus & Oblinger, 1990)) which compare the differences between these proposed explanations. This study aims to assess the viability of these explanations for dealing with the intractability of analogy derivation under SMT by means of a systematic comparison of the performance of the proposed methods.

This thesis is organized as follows: Chapter 2 describes SMT, computational complexity and the two explanations in more detail. Chapter 3 begins by specifying the hypothesis and research questions of this study more formally, and describes the methodology used in order to assess the viability of the two explanations. The results of our study are presented in Chapter 4. Finally, Chapter 5 discusses how the interpretations, implications and limitations of these results lead to the conclusion that the heuristic explanation in its current form lacks validity and that a parameterized exact algorithm provides a viable alternative.

2 Background

This chapter introduces the key concepts used in this thesis, starting with Structure-Mapping Theory and then moving in to complexity theory, its use in cognitive science and its applications for Structure-Mapping Theory. Finally the two methods of explaining how humans could be making optimal analogies in Structure-Mapping Theory are more formally described.

2.1 Structure Mapping Theory

According to one of the most influential theories of analogy derivation, Structure-Mapping Theory (SMT), analogy derivation involves finding maximal mappings between representations (Gentner, 1983, 1989). SMT describes the analogical process in three steps:

1. Retrieval
2. Mapping
3. Inference

When trying to make analogies, the first step is to scan long-term memory for candidate structures. The second step, mapping, is the process of aligning two structures in a maximal way (deriving the analogy). Given a mapping, it is then possible to make inferences (the third step) by projecting relations from one structure to another.

The most studied process is mapping, and this will also be the focus of this thesis. The analogy derivation process will now be introduced using the example in Figures 1 & 2. For more details and a complete overview of the other two steps, the reader is referred to Gentner and Colhoun (2010) and Gentner and Smith (2012).

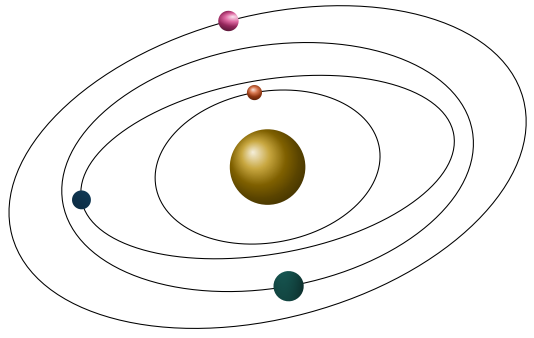
Mapping involves the aligning of predicate structures, collections of statements about the world which are commonly used in knowledge or language representation. For example, the fact that a planet orbits the sun can be expressed in a predicate structure:

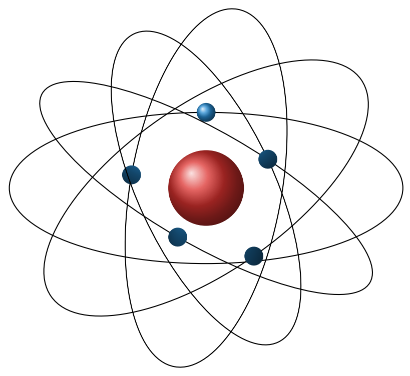
- ATTRACTS(SUN, PLANET)

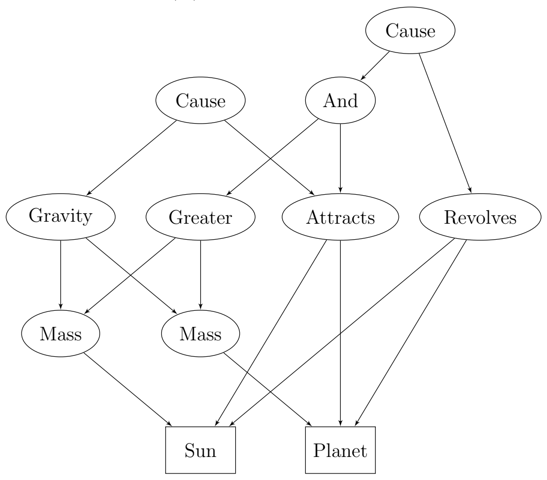
Here, SUN and PLANET are *objects*, physical entities in the world. Note that objects can represent many things, e.g. situations or concepts. ATTRACTS is a *predicate* describing the relation between the two (the sun attracts the planet). The predicate has two arguments (in other words, the *arity* of the predicate is two), PLANET and SUN, which themselves have no arguments. Predicates can be ordered or unordered depending on whether or not argument order matters, e.g., GREATER is ordered ("X is greater than Y" means something different as "Y is greater than X"), but AND is unordered ("X and Y" means the same as "Y and X"). Arity-1 predicates are called *attributes* and can be used to describe specific attributes of the objects, for example:

- MASS(SUN)

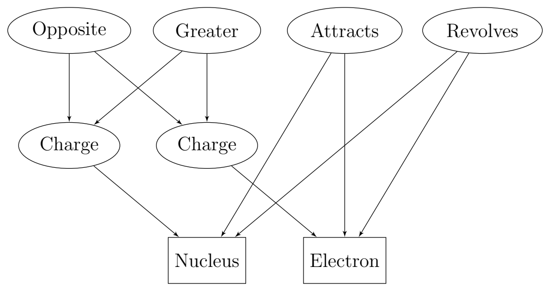
The MASS predicate represents the mass of its argument and is a special kind of attribute, a *function*, as it has a resulting value. Adding more knowledge to this structure, more complex relations can be symbolized:


(a) Solar system


(b) Atom



(c) Solar system



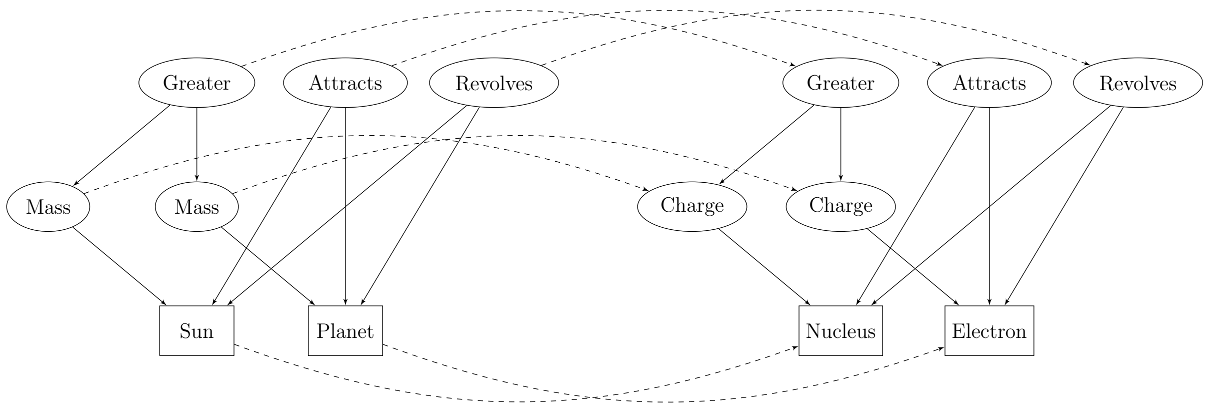
(d) Atom

Figure 1: Example of predicate-structure representations of the solar system and an atom:
(a) Simplified model of the Solar system, with planets orbiting the sun
(b) Simplified model of an atom, with electrons orbiting a nucleus
(c) Predicate structure representation of the Solar system
(d) Predicate structure representation of an atom

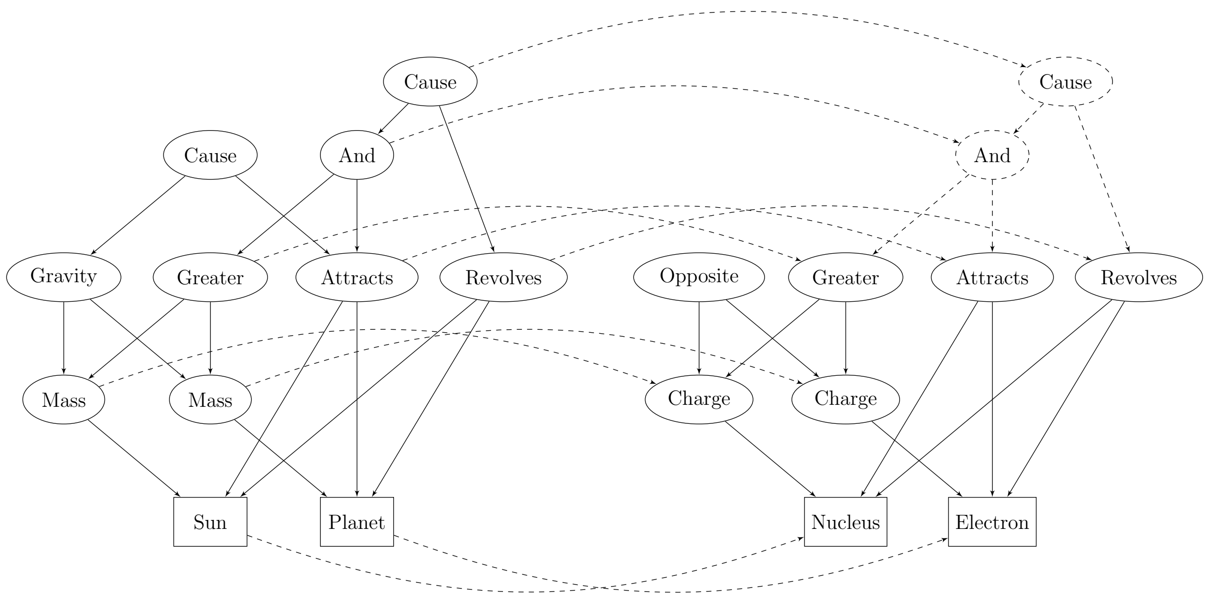
- Cause(
  And(Greater(Mass(sun), Mass(planet)), Attracts(sun,planet)),
  Revolves(planet,sun)
)

Such structures can be visualized using directed graphs, where predicates, attributes, functions and objects are nodes (objects are leaves) and arcs between nodes indicate predicate arguments. The complete sun and planet example is illustrated as predicate structures in Figure 1.

Given the two predicate structures like the ones in Figure 1, SMT defines an analogy to be a mapping from nodes in the base (solar system) to the nodes in the target (atom) structure (Figure 2a). Both base and target in this example have two objects, which have functions and relations. In the example, sun, planet, nucleus and electron are all objects, and mass and charge are examples of functions. Note that relations can be defined on all nodes in the structure (objects, attributes



(a) Analogy



(b) Inference

Figure 2: Example of an analogy between the solar system and an atom. (a) The analogy (dashed line) between the representations of the solar system (base) to the atom (target). (b) How the analogy from (a) can be used to project relations from one predicate structure to another (e.g. the CAUSE relation).

or other relations); this way, high-level knowledge structures can be represented.

The goal of analogy derivation is to find the mapping between base and target, connecting objects in the base with objects in the target, and connecting the relevant predicates and attributes. Predicates and attributes can only map to other predicates and attributes of the same type (e.g. GREATER can map to GREATER, but not to ATTRACTS). In the example, SUN is mapped to NUCLEUS and PLANET to ELECTRON. Their common predicates are mapped (REVOLVES and ATTRACTS), as well as their attributes MASS and CHARGE. Note that functions, like MASS and CHARGE, can map to other functions of different types. A mapping is a valid analogy when it is *consistent* (it contains no many to one mappings), and *supported* (if predicate A in base maps to predicate B in target, all arguments of A must map to the corresponding arguments of B as well.). Gentner (1989) refers to these constraints as "one-to-one correspondence", and "parallel connectivity". Whether the arguments must match in the same order differs per type: When matching two ordered predicates (e.g. GREATER), their arguments must match in the same order, i.e. the first argument of GREATER in the base must match to the first argument of GREATER in the target. When matching two unordered predicates (e.g. AND), their arguments may match in any order.

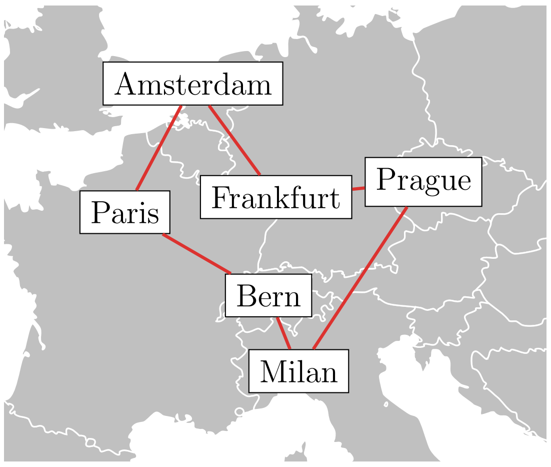
Many possible mappings can be created, and according to the *systematicity principle* humans prefer mappings with many high-level connections (Gentner, 1983). Structural Evaluation Score (SES) is a measure to score the quality (systematicity) of analogies by rewarding the interconnectivity and deepness of the analogy. SES works by giving a value (e.g. 1, although different values can be assigned for e.g. functions or objects) to each match in an analogy, and then this value is passed on down to the arguments of the match (adding it to their value). The score of an analogy is the sum of the scores of all its matches (e.g the analogy in Figure 2a has a SES of 17). This mechanism results in deep, interconnected analogies having a higher SES than for example flat structures with few connections (Forbus & Gentner, 1989). For example, the analogy "my father is like a rock" would have deeper structure than e.g. "my father is like the Eiffel Tower". This method of scoring has been shown to fit human data very well. For example, Clement and Gentner (1991) and Gentner and Toupin (1986) found that people prefer larger shared systematic structures when making analogies.

To summarize, many possible mappings can be created between two structures, and humans are very good at finding those mappings that are systematic analogies. The following sections will describe how computational complexity theory can be used to show how hard it is to find optimal analogies under SMT and the implications of those results for SMT as a model of human cognition.

2.2 Computational Complexity

Computational complexity theory can be used to assess the inherent difficulty of computational problems. This section will broadly introduce relevant definitions used in complexity theory. For a more complete introduction to complexity theory, refer to (Garey & Johnson, 1979), (Cormen, Leiserson, Rivest, & Stein, 2001) and (Papadimitriou, 1993).

A *computational problem* is a specification of the input and the corresponding output for this problem. An example would be the problem of sorting a list of integers, where the input is a list of integers ($L =< x_1, x_2, \ldots, x_n >$) and the output



From	To	Distance
Amsterdam	Paris	430
Amsterdam	Frankfurt	360
Amsterdam	Prague	700
Amsterdam	Bern	630
Amsterdam	Milan	830
Paris	Frankfurt	480
Paris	Prague	880
Paris	Bern	430
Paris	Milan	640
Frankfurt	Prague	410
Frankfurt	Bern	360
Frankfurt	Milan	520
Prague	Bern	620
Prague	Milan	650
Bern	Milan	210

Figure 3: Example of a TSP instance. Distances between cities listed on the right. The goal is to find the shortest possible route (solid line) visiting all cities exactly once. In this small example, already 360 possible routes exist and adding another city would cause 2520 possible routes, illustrating that the amount of time required to check every route increases exponentially with the input size.

is a permutation of L such that for all $x_i, x_{i+1} \in L, x_i \leq x_{i+1}$. A computational problem can be solved by an *algorithm* (i.e. a finite sequence of steps relative to some computer (e.g. a Turing machine)) if and only if this algorithm gives the correct output for all possible inputs.

The difficulty of computational problems can be characterized by how the worst-case required time (in terms of the number of performed steps) to solve the problem grows with the input size of a problem. This is expressed using the *Big-Oh* notation, which describes the upper bound of a function. Some problems can be solved in polynomial time, meaning the worst-case required time grows as a polynomial function of the input size (e.g. $O(1)$, $O(n)$ or $O(n^4)$, but not $O(2^n)$). The class of all problems that are computable in polynomial time is denoted by $\mathcal{P}$.

Take, for example, the sorting problem introduced earlier. There exist algorithms that sort lists in $O(n \; log \; n)$ time (with n being the size of the list) in the worst case, which means that the upper bound on the required time is a polynomial function of the input size (Cormen et al., 2001). Therefore, sorting is in $\mathcal{P}$.

Another class of problems is $\mathcal{NP}$, which is the class of *decision* problems (i.e. problems for which the output is either "yes" or "no") for which "yes"-answers are easy to check, meaning that verifying whether a candidate solution is a valid solution to the problem can be done in polynomial time (Note that as this is the case for all problems in $\mathcal{P}$, $\mathcal{P} \subseteq \mathcal{NP}$).

An example of a problem for which its decision version is in $\mathcal{NP}$ is the classic *Travelling Salesperson problem* (TSP) (see Figure 3 for an example instance of TSP). TSP can be defined as follows:

- Given a list of cities and the distances between these cities, what is the shortest route that visits all cities exactly once?

And its decision version:

- Given a list of cities and the distances between these cities and a value k, does there exist a route that visits all cities exactly once and has a length of at most k?

It is easy to see that checking whether a candidate solution (a route) is a valid solution for the decision problem can be done in polynomial time, by verifying that every city is part of the route, computing the sum over all distances in the route and verifying that this sum $\leq k$. Therefore, TSP is in $\mathcal{NP}$.

In the case of TSP, to solve the decision problem, in the worst case all possible routes have to be examined, of which there are $\frac{(n-1)!}{2}$ (with n being the number of cities). The fastest algorithms for TSP also require exponential time (Cormen et al., 2001; Woeginger, 2003), so it seems likely that there does not exist an algorithm solving TSP in polynomial time (i.e. TSP is not in $\mathcal{P}$). This is confirmed by the result that TSP is so-called $\mathcal{NP}$-complete (Garey, Graham, & Johnson, 1976; Papadimitriou, 1977). To explain $\mathcal{NP}$-completeness, first $\mathcal{NP}$-hardness has to be introduced:

For certain problems it can be proven that all problems in $\mathcal{NP}$ are *polynomial time reducible* to it. A decision problem A is polynomial-time reducible to a decision problem B if every instance of problem A can be transformed into an instance of problem B such that this transformation runs in polynomial time and the answer to the created instance of B is "yes" if and only if the answer the given instance of A is "yes". With such a transformation, any instance of A can be solved by

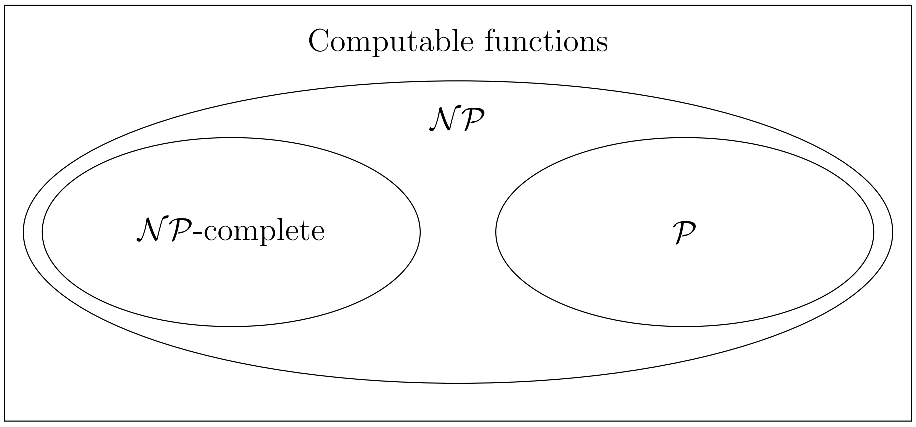


Figure 4: The relation between complexity classes within the domain of all computable functions. Figure adapted from van Rooij (2008).

using the transformation and an algorithm for B. These problems are called $\mathcal{NP}$-hard (they are at least as hard as the hardest problems in $\mathcal{NP}$). Problems that are both in $\mathcal{NP}$ and $\mathcal{NP}$-hard are called $\mathcal{NP}$-complete. Figure 4 illustrates these relations between the complexity classes. It is generally believed that $\mathcal{P} \neq \mathcal{NP}$, i.e. there exist problems in $\mathcal{NP}$ for which no polynomial time algorithm exists (Garey & Johnson, 1979; Fortnow, 2009). This implies that all $\mathcal{NP}$-hard problems are not in $\mathcal{P}$. Returning to the TSP example; the finding that TSP is $\mathcal{NP}$-complete confirms that there does not (and cannot ever) exist an algorithm that solves TSP in polynomial time (unless $\mathcal{P} = \mathcal{NP}$). How computational complexity theory and these complexity classes are of interest to cognitive science will be explained in the next section.

2.3 Computational Complexity and Cognition

Marr (1982) distinguishes three levels in the computational analysis and explanation of cognitive functions. The computational level is the top level, describing what a cognitive process does in terms of the high-level input-output mapping, i.e. a function. Next is the algorithmic level, which specifies how the function converts the input to the output (e.g. which sequences of steps it takes). Finally, on the implementation level the physical realization of the algorithm is described (e.g. neuronal activities).

Cognitive processes can thus be analysed at the computational level by treating the cognitive function as a computational problem, specifying the input and the output of the function (defining *what* the function does). Computational complexity theory can then in turn help in the process of creating or adapting such computational level models of cognition. As human cognition is restricted by com-

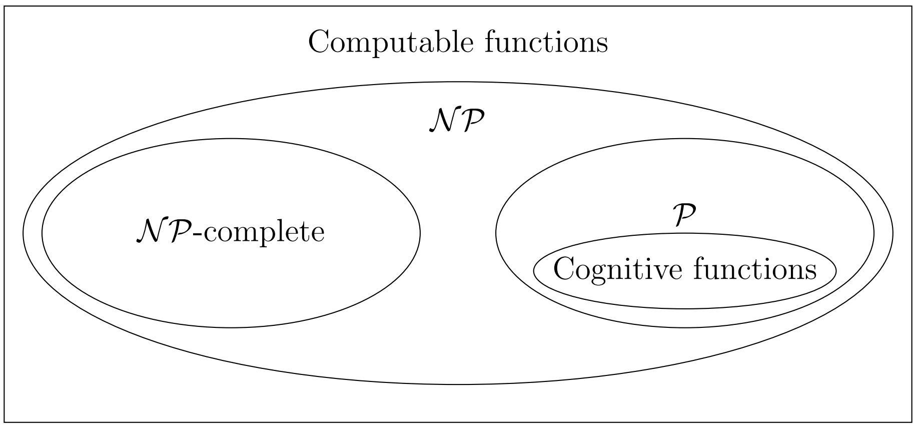


Figure 5: The relation between cognitive functions and the complexity classes under the P-cognition thesis (Frixione, 2001). Figure adapted from van Rooij (2008).

n	n^2	2^n	$n!$
1	1	2	1
5	25	32	120
10	100	1024	3628800
25	625	33554432	1.6×10^{25}
50	2500	1.1×10^{15}	3.0×10^{64}
100	10000	1.3×10^{30}	9.3×10^{157}
1000	1000000	1.1×10^{301}	4.0×10^{2567}

Table 1: A comparison of example polynomial and exponential algorithm runtimes. Listed are the number of steps needed for polynomial (n,n^2) and exponential (2^n,$n!$) algorithms for input size n.

putational resources, it is important to know how hard computational models of cognitive functions are. This has been formalized in the P-cognition thesis, which states that all cognitive functions are in $\mathcal{P}$ (computable in polynomial time), and therefore computational models of cognition should adhere to this limit (Frixione, 2001). The P-cognition thesis is illustrated in Figure 5.

When computational level theories of cognition are not in $\mathcal{P}$, it is unlikely that, as such, they characterize the cognitive process. The reason for this is illustrated in Table 1. When exponential time is needed to compute a function, it is only feasible for really small input and would take years to compute for slightly larger input (even when thousands of steps can be performed per second).

Returning to SMT, complexity analyses have shown that SMT is $\mathcal{NP}$-complete (and thus not in $\mathcal{P}$, unless $\mathcal{P} = \mathcal{NP}$) (Veale & Keane, 1997; van Rooij et al., 2008). The finding that SMT is $\mathcal{NP}$-complete means that there does not exist an algorithm that derives optimal analogies under SMT in polynomial time for all possible inputs (unless $\mathcal{P} = \mathcal{NP}$). This explains why the Structure-Mapping Engine (SME), the implementation of SMT by Falkenhainer, Forbus, and Gentner (1989), has a worst case complexity of $O(n!)$ (where n is the total number of objects and predicates), even though it uses many optimization techniques to reduce the number of substructure matches that it considers.

2.4 Dealing with Intractability in Cognitive Models

The P-cognition thesis, together with the intractability of SMT, suggests that humans are not computing optimal analogies (as specified in SMT). On the other hand, as stated in Section 1, SMT's optimality assumption fits human performance data very well. This contradiction has led to two explanations for how humans could be forming optimal analogies so quickly: The standard explanation is that humans use a heuristic to derive optimal, or otherwise close-to-optimal solutions (Forbus, 2001; Markman & Gentner, 2000). A novel explanation is that humans are only computing the optimal solution when the input has certain characteristics (van Rooij et al., 2008; Wareham, Evans, & van Rooij, 2011). These explanations for dealing with SMT's intractability and their limitations will be described in the next two sections.

2.4.1 Approximation: Heuristic SME

A commonly used method in computer science when faced with intractable problems is to use heuristics (Garey & Johnson, 1979). A heuristic is an algorithm that in short (polynomial) time can come up with a solution that is often reasonably good, however, it does not guarantee the optimality of its solutions (Gonzalez, 2007). Heuristics are often used in the same way for explaining intractable cognitive theories (van Rooij, Wright, & Wareham, 2012), such as SMT (Forbus, 2001; Markman & Gentner, 2000).

In the case of SMT, Forbus and Oblinger (1990) implemented a heuristic for SMT. This heuristic has been claimed to be very successful; it returned the optimal solution in 90% of the cases and when it was not optimal, its solution was in the worst case 67% of the optimal (see Table 2).

There are two criticisms to this approach: First, the empirical tests of the quality of the heuristic have only been performed on a small set of 56 manually encoded

Type of input	Object	Physical Systems	Stories
Number of tests	8	20	28
Percentage of cases the heuristic is optimal	100%	85%	96%
Lowest ratio heuristic score / optimal score	100%	67%	91%

Table 2: Empirical results of the quality of the solutions returned by the heuristic for SMT. Table adapted from Forbus & Oblinger (1990)

examples (See Table 2). One question that needs to be answered is whether these examples are representative of the full input space of SMT. Second, Grootswagers, Wareham, and van Rooij (2013) showed that SMT can not be approximated within a fixed value from the optimal; more specifically, the authors demonstrated that:

- There cannot be a polynomial-time algorithm that returns analogies whose systematicity is within an arbitrarily small specified additive factor from the optimal analogy (unless $\mathcal{P} = \mathcal{NP}$).
- There cannot be a polynomial-time algorithm that returns analogies whose systematicity is within an arbitrarily small specified multiplicative factor of the optimal analogy (unless $\mathcal{P} = \mathcal{NP}$).

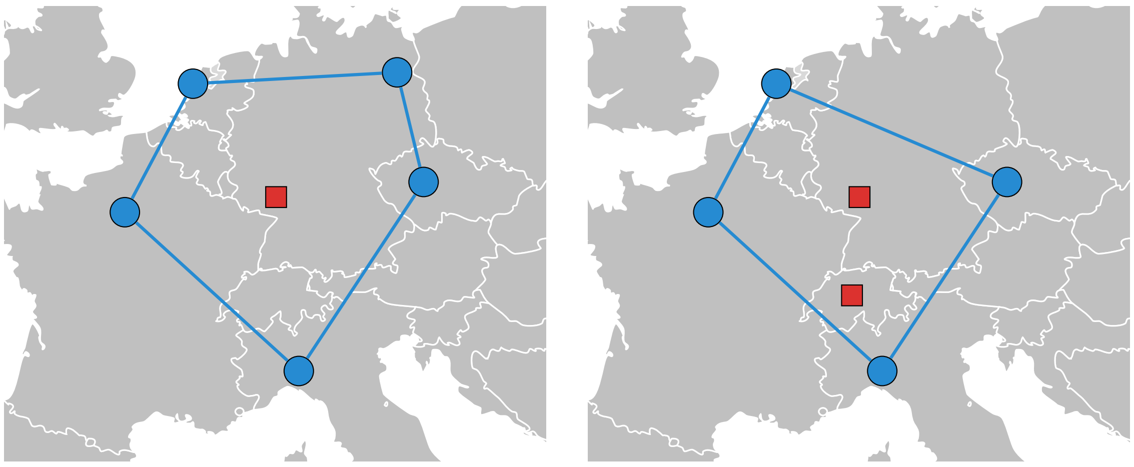
These results suggest that while the heuristic has been shown to be successful on this subset, it can not be close to optimal (within a fixed factor of optimal) for all inputs.

2.4.2 Parameterizing the input: Fixed-parameter tractable algorithms

Section 2.2 explained the complexity classes $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-hard and $\mathcal{NP}$-complete for computational problems. To further investigate what aspect of the problem makes some problems $\mathcal{NP}$-hard, parameterized complexity theory can be used to find so-called sources of complexity. Parameterized complexity (Downey & Fellows, 1999; Fellows, 2002; Flum & Grohe, 2006; Niedermeier, 2006) focuses on investigating

		$2^k \cdot n$		
n	2^n	$k = 1$	$k = 5$	$k = 10$
1	2	2	32	1024
5	32	10	160	5120
10	1024	20	320	10240
25	33554432	50	800	25600
50	1.1×10^{15}	100	1600	51200
100	1.3×10^{30}	200	3200	102400
1000	1.1×10^{301}	2000	32000	1024000

Table 3: A comparison of example polynomial, exponential, and fp-tractable algorithm runtimes. Listed are the number of steps needed for an fp-tractable algorithm with a complexity of $2^k \cdot n$ relative to input size n and parameter k for different values of k compared to the number of steps needed for an exponential algorithm (2^n).



(a) 1 Inner point (b) 2 Inner points

Figure 6: Illustration of inner points in TSP. Inner points (squares) are points that do not lie on the convex hull (solid line). The convex hull is the smallest boundary that encapsulates all points.

the complexity of parameterized problems, problems for which the input has an additional parameter set (a set of parameters describing some characteristics of the input). For example in the case of TSP (introduced in section 2.2), the minimum, maximum or average distance between cities in the graph or the number of cities that have a close neighbor (e.g. with distance < 100) are all different parameters.

For some problems, it is possible to prove that for some parameter sets, the problem can be solved in time that is polynomial in the input size and non-polynomial in those sets. The class of these problems is called FPT (*Fixed-parameter tractable*), and algorithms for these problems are fixed-parameter (fp-) tractable algorithms, as they are tractable with respect to input size per parameter slice (fixed values for all parameters in the parameter set).[1] More specifically, for a problem Q with some parameter set k, $\{k\}$-Q is in FPT if (and only if) its runtime can be described as a polynomial function of input size (n) multiplied by or added to some function of k. For example, relative to parameter k, runtimes of $O(2^k \cdot 2n^3)$ or $O(k! + 24n)$ are fp-tractable but $O(2^{k+n})$ and $O(k + n^k)$ are not. This shows that fp-tractable algorithms can still need only a small number of steps when the input size (n) is large, as long as the parameter (k) is small (see Table 3).

Fixed-parameter tractability can be illustrated using the TSP example. Let *inner points* be the cities that do not lie on the boundary that encapsulates all cities in the graph (the convex hull), as illustrated by Figure 6. The example TSP instance from Figure 3 has two inner points (Frankfurt and Bern). If k is the number of inner points, Deineko, Hoffmann, Okamoto, and Woeginger (2004) give an algorithm for $\{k\}$-TSP that runs in time $O(k!kn)$,[2] and therefore $\{k\}$-TSP is in FPT.[3]

[1]Downey and Fellows (1999, p. 8) refer to this as "tractable by the slice".

[2]Note that this result only holds for TSP in 2-dimensional Euclidean planes, as was the case in our TSP example.

[3]Interestingly, this result was derived independently from earlier empirical results which indicated that the number of inner points is also a factor in human performance on TSP (MacGregor & Ormerod, 1996).

Similar to the P-Cognition thesis discussed earlier, the FPT-Cognition thesis states that all cognitive functions are in FPT and the parameters in their parameter set are small in practice (van Rooij, 2004; van Rooij, 2008). The FPT-Cognition thesis can be viewed as a relaxation of the constraint that the P-cognition thesis puts on models, as for all problems in P, their parameterized versions are in FPT. However, both constrain the models in such a way that they must be computable in feasible time with regards to the input size.

To date, two parameters have been identified for which SMT is fp-tractable, namely o (the number of objects) and p (the number of predicates).[4] The corresponding algorithms $\{o\}$-SMT and $\{p\}$-SMT are described in van van Rooij et al. (2008) and Wareham et al. (2011), respectively. These parameterized algorithms allow SMT to be solved optimally in polynomial time for cases where the number of objects or predicates is small. By showing that parameterizing SMT can lead to tractable algorithms, the authors made the case that a parameterized version of SMT can explain how humans could be fast and optimal, when these parameters are small in practice. However, how small these parameters need to be for the algorithms to have a feasible runtime is unclear, as the algorithms have never been implemented and tested in practice.

To summarize, this chapter has introduced one of the most influential theories of analogy derivation, namely, Structure Mapping Theory (SMT). While SMT's optimality assumption seems to fit human analogy making very well, it does not explain how humans are deriving optimal analogies in feasible time, as computing such analogies under SMT requires infeasible amounts of computational resources. Two explanations have been proposed in the literature of how humans are able to form optimal analogies in reasonable time: The standard explanation claims that humans use heuristics, while a novel explanation argues that humans only make analogies when the input has certain characteristics. The next chapter begins by introducing the main goal of this thesis: to investigate which of these explanations is the most viable. It then continues to describe the simulation experiments created to answer this question.

[4]Note that $\{n\}$-SMT with n being the input size $(o + p)$ is also in FPT, but this is trivial as all problems parameterized by their input size are in FPT.

3 Methods

With the concepts introduced in the previous chapter, the main hypothesis and research questions of this study can be more formally described. This chapter then continues by describing the methods and experiment set-ups used to answer these questions.

3.1 Research question and hypothesis

As explained in Section 2.4.1, the standard heuristic implementation of SMT has been evaluated only by using a small set of manually encoded predicate structures, which explore only a small and possibly not representative subset of the possible input space. The proposed fp-tractable algorithms for the novel explanation have never been implemented and validated or compared to the heuristic. There have been no controlled studies which analysed and compared differences between these proposed explanations. This research aimed to fill this void by testing the following hypothesis:

- FP-tractable algorithms can be a viable alternatives to the heuristic explanation of how humans are making optimal analogies (as defined by SMT) in feasible time.

The objectives of this study were to systematically compare the performance of the standard (exhaustive and heuristic) implementations of SMT with the proposed fp-tractable algorithms, in terms of runtime and quality of produced analogies using different inputs (e.g. randomly generated or manually encoded examples) and parameters (e.g. number of objects or predicates in the predicate structures). In particular, the main questions were:

Q1: How often (and under which conditions) does the heuristic find the optimal solution, and how far off (in terms of systematicity score) are its solutions?

Q2: How do the running times of the heuristic and the exact algorithms compare under varying conditions and how do these times differ from the projected theoretical running times?

Q3: How do results for the first two questions differ between manually encoded and randomly generated predicate structures?

For the purposes of this study, a predicate structure generator was created to provide a large set of examples for analogy derivation with specified characteristics. This method allowed for making systematic comparisons between algorithms for analogy derivation. Also, by varying parameters, the influence of single parameters could be investigated. While many of these parameters (such as structure depth or flatness (Falkenhainer et al., 1989)) have been conjectured to influence the speed of the analogy derivation process, these conjectures have not yet been validated.

The approach for this study can be roughly described in three parts. The first part deals with the implementation of algorithms for analogy derivation under SMT. It will then go on to describe the random predicate structure generator. Finally, the set-up that was used to investigate each research question is listed. These parts are described below.

3.2 Implementing Structure-mapping theory (SMT)

For this study, we considered and thus required implementations of the following four algorithms for SMT:

1. SME-exhaustive (Falkenhainer et al., 1989)
2. Heuristic SME (Forbus & Oblinger, 1990)
3. $\{o\}$-SMT (van Rooij et al., 2008)
4. $\{p\}$-SMT (Wareham et al., 2011)

For the purposes of this study, all algorithms were (re-)implemented in python[5] (van Rossum & Drake Jr, 1995), which was chosen because it is modern, modular and easily extendible. In addition, for graph and predicate structure functionality, the *networkx* python library[6] was used (Hagberg, Schult, & Swart, 2008). The following two sections will describe the algorithms, starting with SME-exhaustive and the heuristic.

3.2.1 SME (Exhaustive and heuristic)

The first implementation of SMT was SME-exhaustive, which uses a number of optimization strategies to reduce the number of combinations to check (Falkenhainer et al., 1989). The algorithm for SME-exhaustive, in pseudo code, is listed in Algorithm 1. The algorithm works by creating *match hypotheses* (mhs, matching a node in base to a node in target) from all combinations of relations of the same type in the base and target predicate structures, and creating mhs for objects, attributes and functions that are arguments of these combinations of relations (lines 1-11). SME-exhaustive then goes on to compute which mhs are inconsistent with each other by mapping the same node in the base to different nodes in the target or vice-versa. Then a collection of *gmaps* (collections of mhs with all mhs that map their arguments) is created for all root mhs (mhs that are not arguments of other mhs) that are supported (for all their arguments there exists mhs, recursing all the way down to the objects) and internally consistent (no many-to-one mappings). If (and only if) the root mhs is not supported or inconsistent, mhs that map its arguments are recursively evaluated the same way and added to the collection of gmaps (lines 12-21).

So far, all these steps are done in polynomial time, and results in a collection of gmaps that map substructures of base to substructures of targets. Note that each gmap on itself is an analogy between base and target. However, to achieve the optimal solution, the gmaps need to be combined to create larger (more systematic) analogies: The final step is to merge the collection of gmaps into one gmap, by exhaustively combining all possible combinations of initial gmaps. The combination of gmaps with the highest score (i.e. SES) that is internally consistent is the optimal analogy match between the two predicate structures (lines 22-28). This final step reflects the intractability of SMT, as its complexity is $O(n!)$ (Falkenhainer et al., 1989).

The only difference between SME-exhaustive and the heuristic, is the method used to merge the gmaps in the final step (see Algorithm 2). Where, after initial

[5] `http://python.org/`
[6] `http://networkx.github.io/`

Algorithm 1 The SME-exhaustive algorithm in pseudo code.

```
 1: $mhs = \emptyset$
 2: for all combinations $x, y$ of relations in base and target do
 3:    if $type(x) = type(y)$ then
 4:       add $(x, y)$ to $mhs$
 5:       for all $c_x, c_y \in Children(x, y)$ do
 6:          if $c_x$ and $c_y$ are both either functions, entities or attributes of the same type
            then
 7:             add $(c_x, c_y)$ to $mhs$
 8:          end if
 9:       end for
10:    end if
11: end for
12: $gmaps = \emptyset$
13: $roots = mh \in mhs$ if $mh$ is not a child of any $mh \in mhs$
14: while $roots \neq \emptyset$ do
15:    take one $mh$ from $roots$
16:    if consistent($mh$) and supported($mh$) then
17:       add $mh$ to $gmaps$
18:    else
19:       update $roots$ with $mh$'s children that map relations
20:    end if
21: end while
22: $solution = \emptyset$
23: for all $gmapset \subseteq gmaps$ do
24:    if $gmapset$ is internally consistent and systematicity($gmapset$) >
       systematicity($solution$) then
25:       $solution = gmapset$
26:    end if
27: end for
28: return $solution$
```

Algorithm 2 The heuristic merging in pseudo code. The initial gmaps are created using steps 1-21 from SME-exhaustive (Algorithm 1).

```
1: $solution = \emptyset$
2: while $gmaps \neq \emptyset$ do
3:    take $gmap$ with highest systematicity from $gmaps$
4:    if $gmap$ is consistent with $solution$ then
5:       add $gmap$ to $solution$
6:    end if
7: end while
8: return $solution$
```

gmap creation, SME-exhaustive checks every possible subset of gmaps, the heuristic starts by selecting the gmap with the highest score. It then goes on to add the next highest scoring gmap if it is consistent with the selection. This is repeated for all gmaps, and is essentially a greedy merging strategy (Forbus & Oblinger, 1990). The result is a substantial increase in speed, as the algorithm now runs in polynomial time ($O(n^2)$). However, the solutions will not always be optimal, and are not guaranteed to be within any distance from the optimal.

Even though SME-exhaustive is advertised as an optimal algorithm, there exist two cases where it is not clear how SME-exhaustive (as described in (Falkenhainer et al., 1989)) yields the optimal solution, which were discovered when comparing the optimal solution returned by SME-exhaustive with the solution returned by the fp-tractable algorithms. First, unordered predicates (where their arguments may match in any order) do not appear to be dealt with correctly in the mapping process. Second, SME-exhaustive does not consider all possible sub-structures, while these sub-structures are needed to form the optimal solution. These two cases are described in more detail in Appendix A. In this study, rather than making unfounded assumptions about the algorithm, cases where the analogies derived by SME-exhaustive are not optimal were excluded from the analyses.

3.2.2 Fixed-parameter tractable algorithms

A very different approach from SME is the parameterized method, which exploits parameters from the input to achieve efficient computation using fp-tractable algorithms. As mentioned in the Introduction, there exist two proposed fp-tractable algorithms, one that proves that SMT is fp-tractable for parameter o (number of objects), and another that proves that SMT is fp-tractable for parameter p (number of predicates). The first algorithm, using the number of objects, is given in van Rooij et al. (2008). The algorithm in pseudo code is listed in Algorithm 3. In short, $\{o\}$-SMT works by exhaustively considering all possible mappings of sets of objects in the base to sets of objects of the same size in the target, and for each such mapping, growing predicate matches upwards (lines 4-11) from the objects to create maximal mappings.

The second algorithm is $\{p\}$-SMT (Wareham et al., 2011), which exhaustively combines predicate sets in the base and target in maximal ways. $\{p\}$-SMT is listed in pseudo code in Algorithm 4. For every consistent and supported maximal mapping between predicate-sets, a mapping for the relevant objects can be found (lines 5-8) in polynomial time.

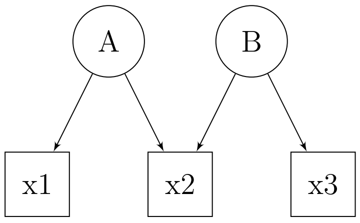
Two problems were encountered when implementing these algorithms, the first being that the algorithms allowed objects and attributes to match without a higher order relation supporting or licensing this match. Such matching is not allowed under SMT and the algorithms therefore were slightly adjusted by simply removing these unsupported matches (lines 9 & 12 in Algorithm 3 & 4, respectively) from its solutions (this is done in polynomial time). The second problem was that the description of $\{o\}$-SMT did not involve the matching of functions (recall that functions are allowed to match to all other functions). This proved problematic as it seems that the only way of dealing with this is to exhaustively combine all possible function mappings as well, resulting in an additional parameter f (number of functions), i.e. $\{o, f\}$-SMT. Because function matching was not included in the specification of $\{o\}$-SMT, predicate structures with functions were not included in this study.

Algorithm 3 The $\{o\}$-SMT algorithm in pseudo code.

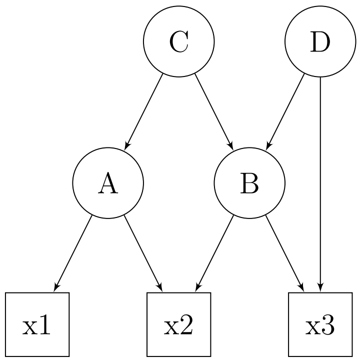
```
1: $solutions = \emptyset$
2: for all possible combinations $o_{base}, o_{target}$ of objects in base and target do
3:    let $map$ be the mapping from $o_{base}$ to $o_{target}$
4:    let $eval$ be the set of predicates in $base$ that are not in $map$ and have all their
      children in $map$
5:    while $eval \neq \emptyset$ do
6:       take a predicate $p_{base}$ from $eval$
7:       if there exist a predicate $p_{target} \in target$ that is not in $map$ and $type(p_{base}) =$
         $type(p_{target})$ then
8:          add $p_{base} \rightarrow p_{target}$ to $map$
9:          update $eval$ with parents from $p_{base}$ that are not in $map$ and have all their
            children in $map$
10:      end if
11:   end while
12:   remove objects, functions and attributes that are not children of a predicate in
      $map$ from $map$
13:   add $map$ to $solutions$
14: end for
15: return $map$ in $solutions$ with the highest SES score
```

Algorithm 4 The $\{p\}$-SMT algorithm in pseudo code.

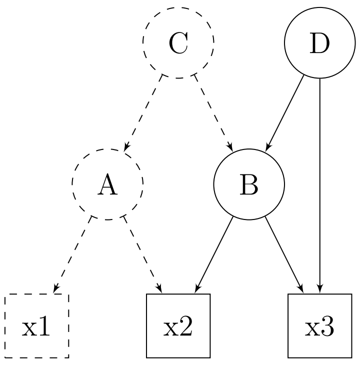
```
1: $solutions = \emptyset$
2: for all possible combinations of subsets $p_{base}, p_{target}$ of predicates in base and target
   do
3:    let $map$ be the mapping from $p_{base}$ to $p_{target}$
4:    if $map$ contains no many-to one mappings and for every predicate $p \in map$, all
      children of $p$ that are predicates are in $map$ then
5:       let $L$ be the set of leaves in $base$ that are a children of $p_{base}$ in the mapping
6:       let $L'$ be the set of leaves in $target$ that are a children of the mappings of $(p_{base})$
7:       if $L = L'$ and the mapping $L$ to $L'$ does not contain many-to-one mappings
         then
8:          extend $map$ with the mapping $L$ to $L'$
9:          remove objects, functions and attributes that are not children of a predicate
            in $map$ from $map$
10:         add $map$ to $solutions$
11:      end if
12:   end if
13: end for
14: return $map$ in $solutions$ with the highest SES score
```



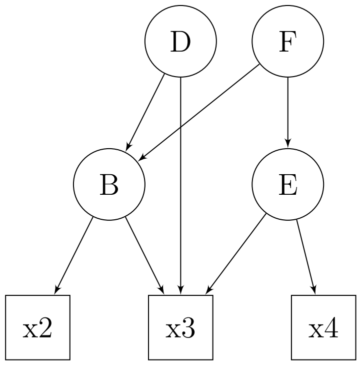
(a) Creating base: Adding the first layer of predicates (A & B), connecting to objects (x1, x2 & x3).



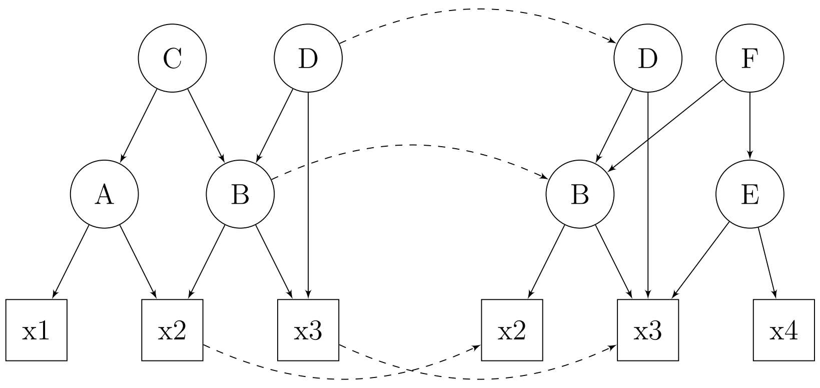
(b) Creating base: Adding a second layer of predicates (C & D), connecting to nodes from all layers below.



(c) Removing predicates and objects from the base, leaving the core.



(d) Creating target: adding new objects (x4) and predicates (E & F) to the core



(e) The resulting pair with the analogy it contains.

Figure 7: Illustration of predicate structure pair generation. The generation parameters that resulted in this predicate structure pair are given in Table 4.

3.3 Generation of test inputs

Dimension	Description	Fig. 2	Fig. 7
predicates	Number of predicates	9	4
objects	Number of objects	2	3
height	Longest path from predicate to object	4	2
types	Number of predicate types	7	6
max_arity	Maximum predicate arity	2	2
chance_function	Chance of arity-1 predicates being functions	-	0
preservation	Percentage that is kept when extracting the core analogy	-	0.6
decay	The decay of the preservation parameter per layer	-	0.0
scaling	Size of the target predicate structure compared to the base predicate structure	0.6	1.0

Table 4: Parameters for predicate structure generation. When possible, the values of these parameters in the predicate structures of Figure 2 and Figure 7 are listed as illustration.

To systematically investigate algorithm performance on inputs with specific characteristics, a predicate structure pair generator was created for this study. This method allows control over predicate structure generation by various parameters, which are listed in Table 4. Especially the ability to control the size difference and similarity between predicate structures (by controlling how much is preserved between predicate structures and how many new predicates are created), allowed us to specify characteristics for predicate *pairs*, which could not be achieved using existing single predicate structure or directed graph generators.

Note that not all combinations of parameters are possible; for example the number of predicates must always be larger than the height of the structure and there must be enough predicates to support all objects. Also, if the number of types is too small, it might not be possible to fit enough different predicates in the predicate structure. However, these constraints only apply to a limited number of cases, and this method allows us to randomly create (infinitely) many different predicate structure pairs.

The generation of predicate structure pairs can be described in the following four steps (which are also illustrated in Figure 7):

1. **Generate a pool of predicates:** To create a pool of predicates, first a pool of predicate types is created using the number of predicate types and maximum predicate arity. For each type a random arity (up to the maximum arity) is assigned, and it is added to the pool. The distribution of arity over predicate types can be controlled by an optional parameter, for example, to create more arity-1 predicates. Next, using the number of predicates and total height of the structure, the number of predicates at each height level is computed.[7] Arity-1 predicates (i.e. functions and attributes) are only created

[7]Note that shape of the structure can be controlled by an optional shape parameter. For example, predicate structures could be 'square'-shaped, with the same number of predicates on each height, or they could be 'triangle'-shaped, with a decreasing the number of predicates per height level.

at the first height layer (connecting only to objects), as specified in SMT (Gentner, 1983).

2. **Generate the base predicate structure:** With these predicates, the base predicate structure is grown layer by layer (of height), starting with the specified number of objects. Predicates are taken from the predicate pool and connected randomly to nodes in the layers below. This is also described in pseudo code in Algorithm 5 (See also Figures 7a & 7b).
3. **Generate the core analogy:** The process then continues to extract the *core* analogy from the base (Figure 7c), using the preservation and decay parameters. Extracting the core is done by deleting nodes from the base in each layer starting at the objects, and deleting structure connecting to these nodes in layers above. The preservation parameter defines the percentage of nodes that are preserved in the bottom layer (objects), then for every layer, the preservation parameter is multiplied by the decay parameter, to allow control over how much structure is preserved in higher layers of the core. This is also described in pseudo code in Algorithm 6.
4. **Generate the target predicate structure:** Finally, from the core, the target is grown by adding other nodes from the predicate pool. For each layer is computed (using the scaling parameter) how many predicates or objects should be added (Figure 7d). This works in the same way as generating the base, except that now there exists an initial structure to grow on. This is also described in pseudo code in Algorithm 7. Scaling defines how large the target structure should be relative to the base and allows to control size differences in predicate structure pairs.

The steps are repeated until a connected (no unconnected substructures) base and target predicate structure are found (i.e. first create a connected base structure, then continue to create the core etc.).

Algorithm 5 Pseudo code for generating the base predicate structure.

```
1: let base be an empty predicate structure
2: add the specified ($o$) number of objects to base
3: for all height layers $d$, starting at one layer above the objects do
4:   repeat
5:     take a random predicate $p$ from the predicate pool
6:     let $a$ be the arity of predicate $p$
7:     let $C = \emptyset$
8:     for all $a$-sized combinations of nodes in base do
9:       if at least one of the nodes is at height layer $d - 1$ and all nodes are at height
         $< d$ and a predicate of the same type as $p$ which connects to the nodes does
         not already exist in base then
10:        add this combination of nodes to $C$
11:      end if
12:    end for
13:    randomly take one combination of nodes from $C$ and connect $p$ to these nodes
14:  until the specified number of predicates in this layer has been reached
15: end for
```

Algorithm 6 Pseudo code for extracting the core analogy structure.

```
1: let $core$ be an empty predicate structure
2: for all layers $d$ in base, starting at 0 (the objects) do
3:     let $C$ be the set of nodes in layer $d$ in $base$ that have all their successors in $core$
4:     let $x = |C| \cdot (preservation - d \cdot decay)$
5:     randomly take $x$ nodes from $C$ and add them to $core$ (preserving their connections)
6: end for
```

Algorithm 7 Pseudo code for growing the target predicate structure.

```
1: $target = core$
2: $n$ objects to $target$ where $n = scaling \times |objects_{base}|$
3: for all layers $d$, starting at 1 (one layer above the objects) do
4:     repeat
5:         Take a random predicate $p$ from the predicate pool
6:         let $a$ be the arity of predicate $p$
7:         let $C = \emptyset$
8:         for all $a$-sized combinations of nodes in $target$ do
9:             if at least one of the nodes is at height layer $d - 1$ and all nodes are at height
               $< d$ and a predicate of the same type as $p$ which connects to the nodes does
               not already exist in $base$ or $target$ then
10:                add this combination of nodes to $C$
11:            end if
12:        end for
13:        randomly take one combination of nodes from $C$ and connect $p$ to these nodes
14:    until $|predicates_{target}| = scaling \times |predicates_{base}|$ for height level $d$
15: end for
```

3.4 Simulation

This section describes the simulation experiments that were performed in order to answer the main questions (introduced in Section 3.1). For each question, the parameters used for predicate structure pair generation are listed. In addition, the measures used to compare the performance are described.

3.4.1 Q1 set up: Quality of the heuristic

To systematically assess the quality of the heuristic, both the heuristic and the exact algorithms were run on randomly generated pairs (Like the example in Figure 7e) and their solutions were compared using two measures:

1. The total percentage of trials where the heuristic was optimal; and
2. The normalized distance from the optimal, defined as $\frac{SES_{optimal} - SES_{heuristic}}{SES_{optimal}}$, which is generally used for computing heuristic solution quality when the optimal solution is known (Barr, Golden, Kelly, Resende, & Stewart Jr, 1995).

Pairs were randomly generated in the dimensions listed in Table 5. If it was possible to generate a pair using the given parameter values (e.g. constraints like $p > o$ were satisfied), which was the case for 80% of the input (70866 pairs), the algorithms

Dimension	Values
predicates	5, 10, 15, 20, 25, 30, 35, 40, 60, 80, 120, 160
types	2, 10, 20
objects	2, 4, 6, 8, 10, 15, 20
height	2, 4, 6, 8, 10, 15, 20
chance_function	0.0
preservation	0, 0.25, 0.5, 0.75, 1.0
max_arity	2
preservationdecay	0.0
scaling	1.0

Table 5: The values used to assess the overall quality of the heuristic. On each combination of values in the dimensions (8820 combinations), 10 pairs were randomly generated as input for the heuristic (88200 pairs).

were run on this pair. If (at least) one of the exact algorithms yielded a solution in reasonable time (under 2 minutes on a 3.8 Ghz CPU) for this pair, the heuristic could be compared on this pair. This was true for 78% of the cases, leaving 55249 trials.

While the first set-up gave an overview of the range of quality of the heuristics solutions, specific dimensions were also explored. The set-up was similar, although only the dimension of interest was varied and all other dimensions were to set to fixed values, to see more specifically how this dimension influenced the quality. The dimensions that were investigated were chosen based on conjectures about the difficulty of SMT, some of which are mentioned at the beginning of this chapter (Section 3.1). The following dimensions were individually manipulated:

- Closeness: To manipulate closeness, the preservation parameter was varied, as this defines how much the pairs overlap, thus how similar they are.
- Height: To compare flat structures with deep structures, height was varied while fixing the number of predicates and other parameters. The options here were limited, as it is hard to generate deep structures with few predicates and shallow structures with many predicates at the same time.
- Size (number of predicates), with the number of objects fixed at a low value allowing to get the optimal solutions from $\{o\}$-SMT.
- Types: The number of possible different predicate types.
- Objects: The number of objects in the predicate structures.

The specific parameter settings for these manipulations are listed in Table 6

3.4.2 Q2 set up: Speed of the fp-tractable algorithms

The same individual manipulations used to investigate the quality of the heuristic were used to investigate the effect of these dimensions on the runtimes of the algorithms. Runtimes were measured by the time that the algorithms spent on the CPU (in this study, a 3.8GHz AMD processor), to get an indication of the number of instructions executed. Note that the CPU-time measure does not reflect human

Dimension	Closeness	Height	Predicates	Types	Objects
predicates	25	25	(20:200:20)	25	25
types	10	10	10	(2:20:2)	100
objects	5	5	5	5	(2:10:2)
height	5	(2:10:1)	5	5	5
chance_function	0.0	0.0	0.0	0.0	0.0
preservation	(0:1:0.25)	0.5	0.5	0.5	0.5
typeshape	random	random	random	random	random
heightshape	square	square	square	square	square
max_arity	2	2	2	2	2
decay	0.0	0.0	0.0	0.0	0.0
scaling	1.0	1.0	1.0	1.0	1.0

Table 6: The values used to assess influence of individual dimensions on quality and complexity. Value range is shown as (start:stop:step)

runtime, but it is used to compare the relative difference in speed between algorithms. The CPU time was measured using the *psutil* library[8] for python (Rondolà, 2013). Of the dimensions that were investigated, the number of objects and predicates are of special interest for the fp-tractable algorithms, as they can be compared to the theoretical projections. As SME-exhaustive was likely to be influenced by the same parameters as the heuristic, they were also included to get a better idea of the cases in which SME-exhaustive was faster. Besides the worst-case runtime of the algorithms, the average runtimes of the algorithms was also computed, to get an indication of how these differ.

As the runtime increases exponentially with the size of the predicate structures, the sizes of their search spaces are computed before performing the exhaustive step with the exact algorithms. If a search space is too large (larger than 10^5), computing the optimal solution would become infeasible,[9] and the search space size was used as predictor for the runtimes. This allowed us to compare the runtimes of the algorithms on larger structures as well, although caution must be applied when interpreting those results.

3.4.3 Q3 set up: Comparing manually encoded predicate structures

To evaluate the difference between randomly generated and manually encoded structure pairs, predicate structures from the THNET library[10] (which was created to evaluate connectionist analogy mapping engines) were used (Holyoak & Thagard, 1989). Specifically, the plays and fables from this library were extracted and parsed to the predicate structure format used in this study. Figure 8 shows two examples of fables in predicate structure format and Figure 9 shows a part of one play (as the plays are very large). Table 7 lists the properties of the manually encoded predicate structures. Note that not all parameters that are controllable

[8]`http://code.google.com/p/psutil/`

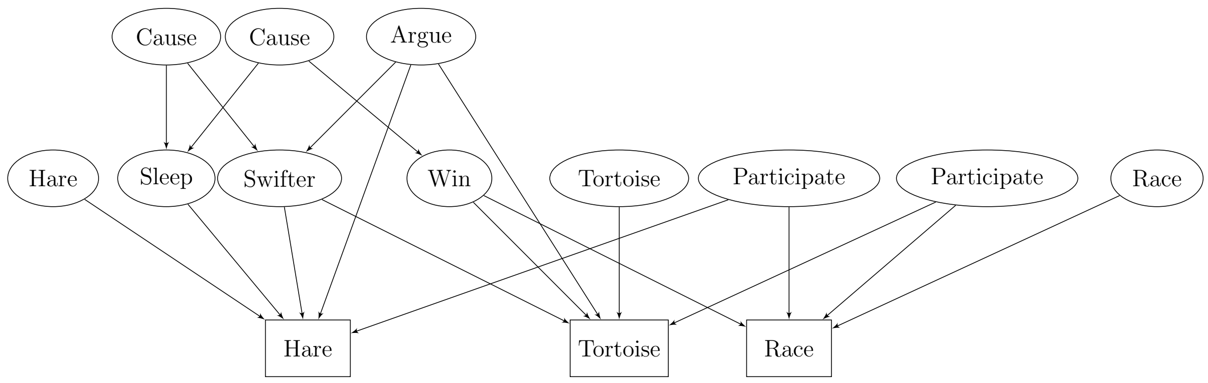
[9]For the purpose of simulating many trials, runtimes in the order of minutes are still feasible. Larger search spaces would already need hours (or even days) to compute.

[10]Available online: `http://www.cs.cmu.edu/Groups/AI/areas/neural/systems/thnet/0.html`

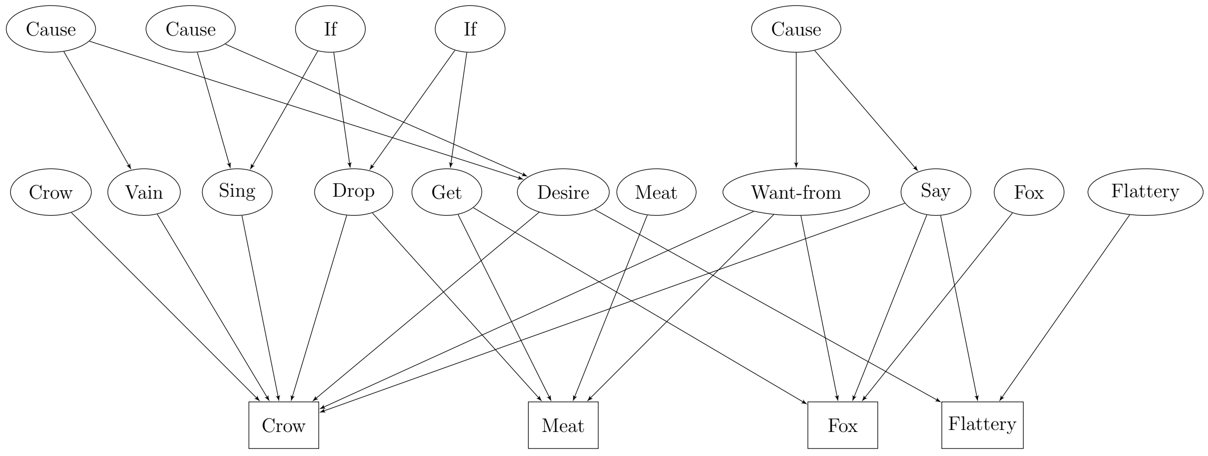
in the random pair generator can be derived from the manually encoded input. For both plays and fable categories, all (almost 10000) combinations of structures were tested using all algorithms and algorithm runtime and heuristic quality were reported.

	Plays (253 pairs)				Fables (9506 pairs)				Combined (9759 pairs)			
Dimension	min	max	avg	std	min	max	avg	std	min	max	avg	std
predicates	36	86	58.60	13.83	6	33	23.43	4.30	6	86	24.34	7.36
types	31	89	58.82	13.08	9	48	31.27	5.73	9	89	31.98	7.46
objects	9	24	16.29	3.92	3	14	9.15	1.90	3	24	9.34	2.28
height	2	5	3.75	0.80	1	5	3.21	0.74	1	5	3.23	0.75

Table 7: Details of the manually encoded predicate structures (Holyoak & Thagard, 1989).



(a) The Tortoise and the Hare.



(b) The Crow and the Fox (Le Corbeau et Le Renard).

Figure 8: Examples of manually encoded predicate structures from the fables category (Holyoak & Thagard, 1989).

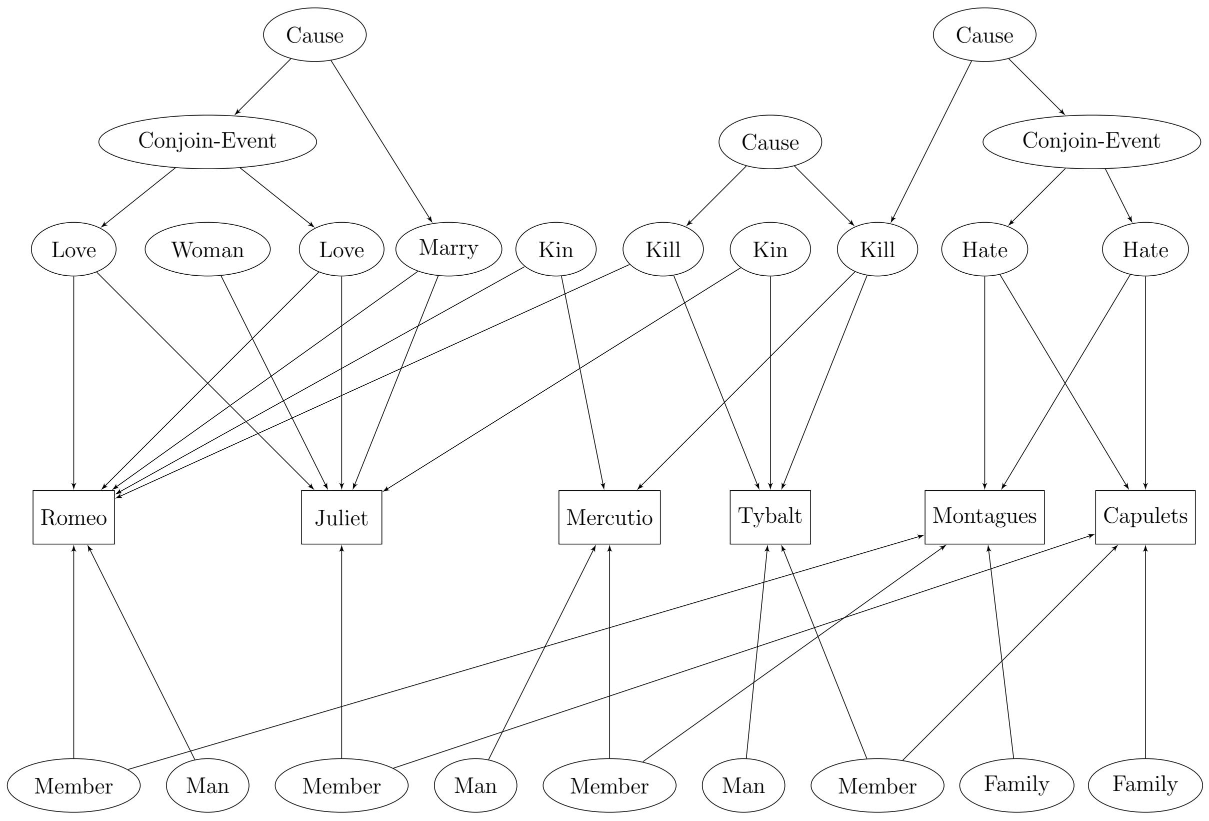


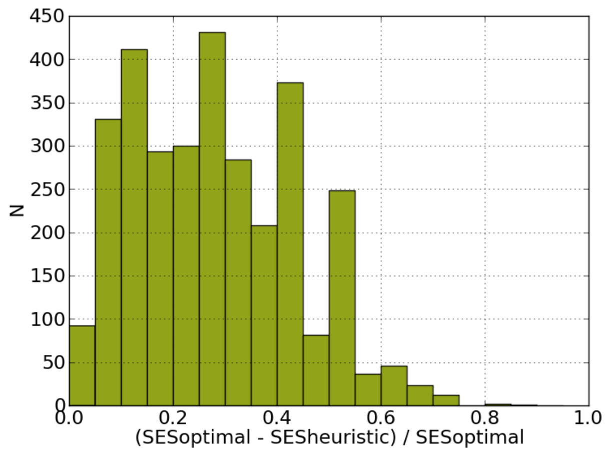
Figure 9: Example of a manually encoded predicate structure from the plays category (Shakespeare's *Romeo and Juliet*). Note that only part of the full structure is shown (Holyoak & Thagard, 1989).

4 Results

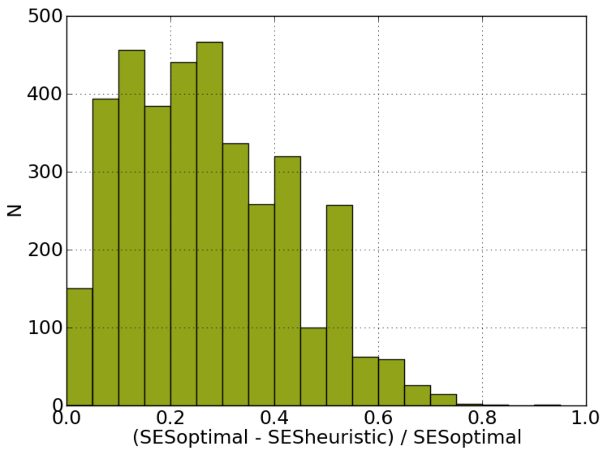
In this chapter, the results of the simulations are described, split up for each research question that was listed in Section 3.1

4.1 Quality of solutions (Optimality)

To assess the quality of the solutions returned by the heuristic, two investigations were performed. The first tried to explore the performance of the heuristic on the whole search space by combining many different values for dimensions (see Table 5). The percentage of trials where the heuristic found the correct solution was 87.50%. Of all trials where it was not optimal the normalized distance from optimal was computed as $\frac{SES_{optimal}-SES_{heuristic}}{SES_{optimal}}$, which has a value of 0 if the heuristic returns an optimal analogy and 1 if the heuristic returns no analogy at all. On average this ratio was 0.272, with a standard deviation of 0.158. The lowest ratio encountered was 0.009 (almost optimal), and the highest 0.934 (far from optimal). The distribution of these distances, separated for small and large predicate structures, is shown in Figure 10. Two observations can be made from these distributions: (1) the distributions are spread very wide, with a large portion of the results between 0.0 (close to optimal) and 0.5 (half as good as optimal); and (2) the distributions are similar for small and large predicate structures.



(a) Number of predicates smaller than 50.



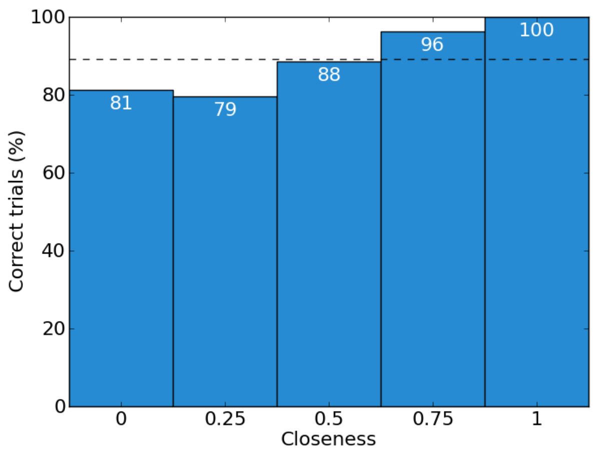
(b) Number of predicates between 50 and 200.

Figure 10: The distribution of normalized distance from optimal of the heuristics non-optimal solutions over all inputs. The graphs are split up for small (a) and large (b) predicate structures.

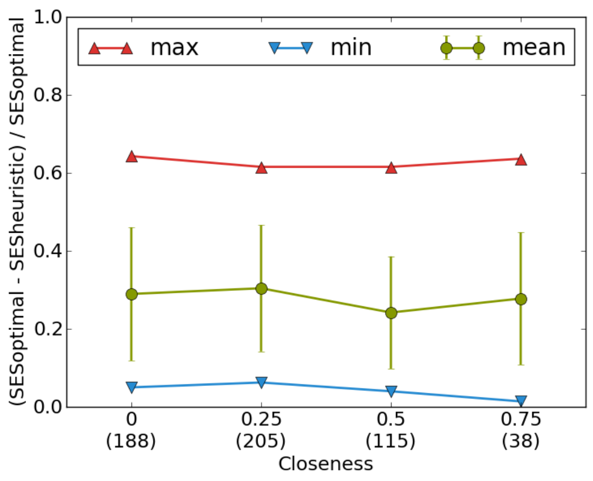
The second investigation looked at the influence of various dimensions on the solution quality. For these dimensions (see Table 6 for the specific parameters), the following results were found:

- Closeness (Figure 11): The performance of the heuristic is better on closer predicate structures (predicate structures that are more similar). However, non-optimal solutions are not guaranteed to be close to optimal. Note that for closeness 1.0, the heuristic was optimal in 100% of the cases and therefore this value is not included in the graph showing the distance of non-optimal scores.

- Height (Figure 12): Manipulating the number of height levels shows that the heuristic performs better on deeper structures in terms of percentage of correct trials. This was also reflected in the maximum distance of non-optimal solutions.
- Predicates (Figure 13): These graphs show that the quality of the solutions by the heuristic drop from being optimal on 93% of the trials on small predicate structures (20 predicates), down to 24% of the trials on large predicate structures. Interestingly, the average distance from optimal does not seem to vary with the number of predicates.
- Types (Figure 14): Manipulating the number of types resulted in better performance on predicate structures with more types in terms of percentage of optimal solutions. However, the ratio of non-optimal solutions does not improve in the same way.
- Objects (Figure 15): Manipulating the number of objects suggests that the performance increases with more objects in terms of maximum distance of non-optimal solutions. The percentage of optimal solutions does not seem to change.

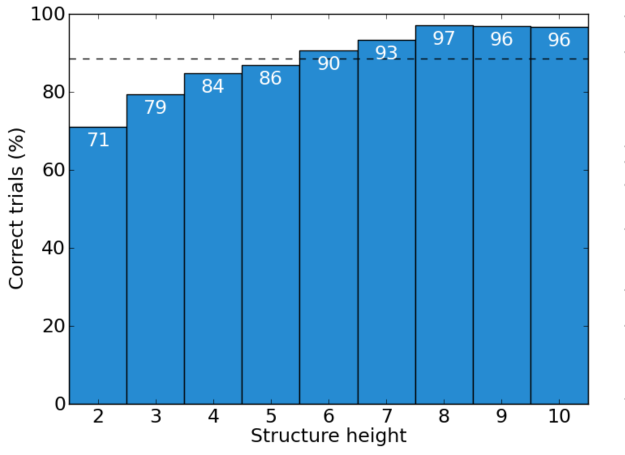


(a) Percentage of trials for which the heuristic found the optimal solution. The average over all trials (dashed line) was 89.0%. 1000 trials were done per value (5000 total).

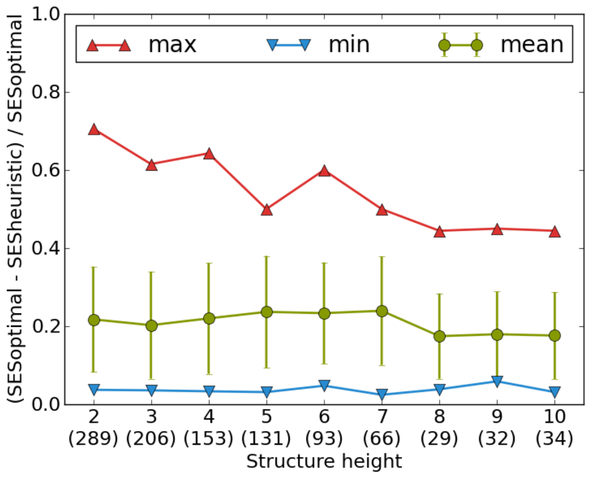


(b) Distance from optimal of the non-optimal solutions. The number of trials the values are based on are listed between brackets.

Figure 11: Heuristic solution quality when manipulating the closeness with the preservation parameter.

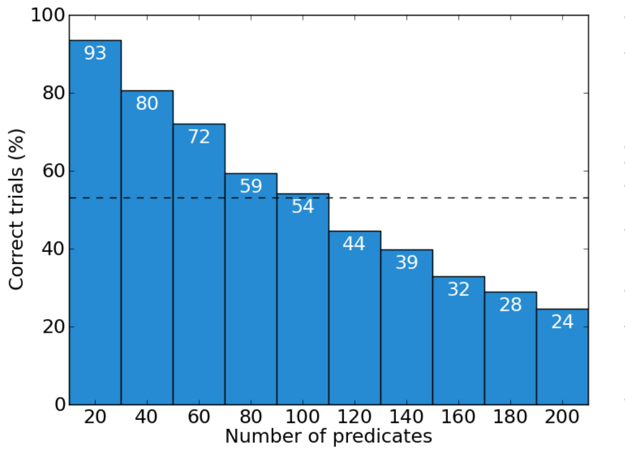


(a) Percentage of trials for which the heuristic found the optimal solution. The average over all trials (dashed line) was 88.5%. 1000 trials were done per value (9000 total).

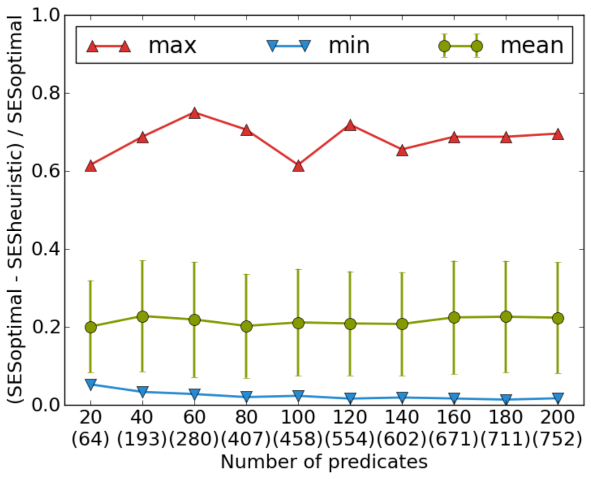


(b) Distance from optimal of the non-optimal solutions. The number of trials the values are based on are listed between brackets.

Figure 12: Heuristic solution quality when manipulating the number of height levels.

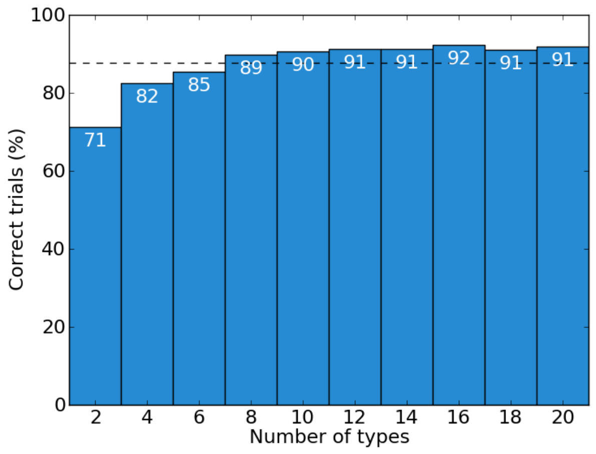


(a) Percentage of trials for which the heuristic found the optimal solution. The average over all trials (dashed line) was 53.1%. 1000 trials were done per value (10000 total).

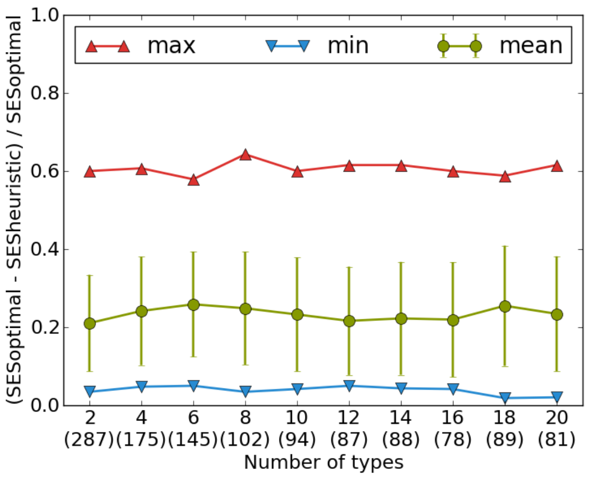


(b) Distance from optimal of the non-optimal solutions. The number of trials the values are based on are listed between brackets.

Figure 13: Heuristic solution quality when manipulating the number of predicates.

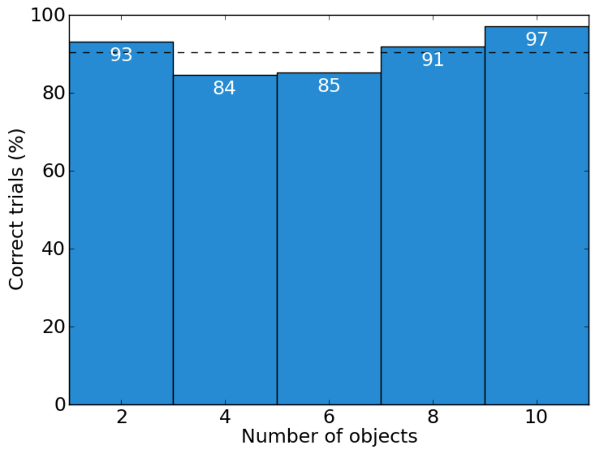


(a) Percentage of trials for which the heuristic found the optimal solution. The average over all trials (dashed line) was 87.7%. 1000 trials were done per value (10000 total).

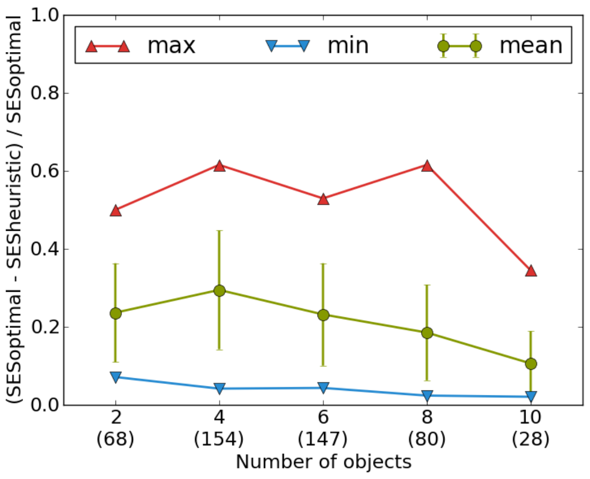


(b) Distance from optimal of the non-optimal solutions. The number of trials the values are based on are listed between brackets.

Figure 14: Heuristic solution quality when manipulating the number of types.



(a) Percentage of trials for which the heuristic found the optimal solution. The average over all trials (dashed line) was 90.5%. 1000 trials were done per value (5000 total).



(b) Distance from optimal of the non-optimal solutions. The number of trials the values are based on are listed between brackets.

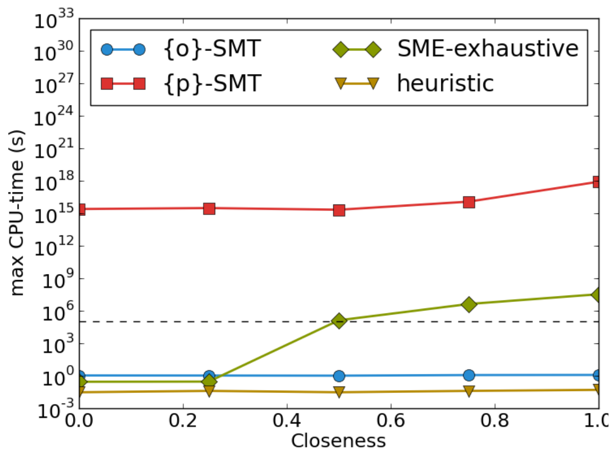
Figure 15: Heuristic solution quality when manipulating the number of objects.

4.2 Runtime (Complexity)

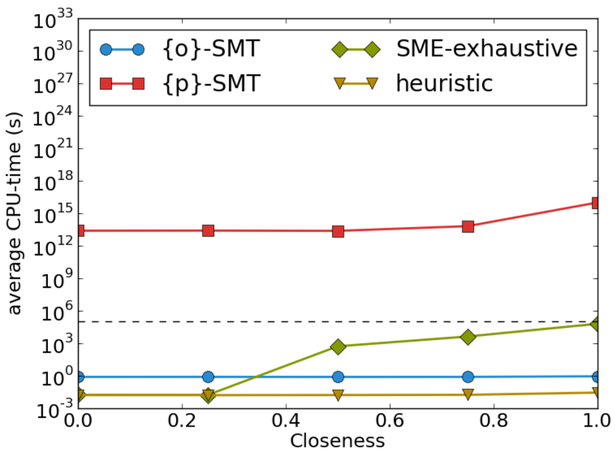
The second research question was how the algorithms compare in terms of running time. The following dimensions were investigated by running 1000 trials for each value (see Table 6 for the specific parameters). The following results were found:

- Closeness (Figure 16): Interestingly, SME-exhaustive is slower on closer predicate structures, while for the other algorithms closeness does not seem to influence running time.
- Height (Figure 17): This indicates that SME-exhaustive is performing better on structures as the number of layers increases, while algorithm $\{p\}$-SMT seems to get slower with the number of layers.
- Predicates (Figure 18): This figure shows the exponential growth with regard to the number of predicates for SME-exhaustive and $\{p\}$-SMT. The most striking observation here is that $\{o\}$-SMT is performing in the same range as the heuristic, and $\{o\}$-SMT's runtime does not grow much with the increasing number of predicates.
- Types (Figure 19): Except for $\{o\}$-SMT, all algorithms appear to benefit from more types in the predicate structures. With many types, SME-exhaustive seems to perform in the same range as the heuristic.
- Objects (Figure 20): While the other algorithms are not influenced by the number of objects, $\{o\}$-SMTs search space and runtime increase exponentially with the number of objects.

One striking general observation that can be made from these results is that the average runtime is similar to the worst case runtime; The average lies within a factor 10^3 of the worst-case, which is not that much when looking at ranges up to 10^{33}.

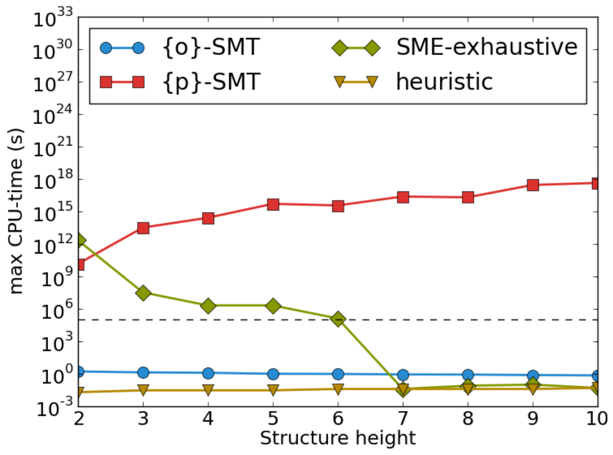


(a) Max CPU time.

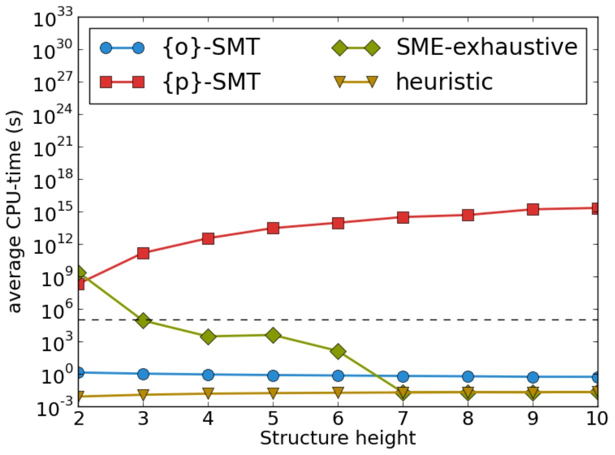


(b) Average CPU time.

Figure 16: Algorithm runtime when manipulating the closeness with the preservation parameter. Points above the dashed line depict search space size instead of time.

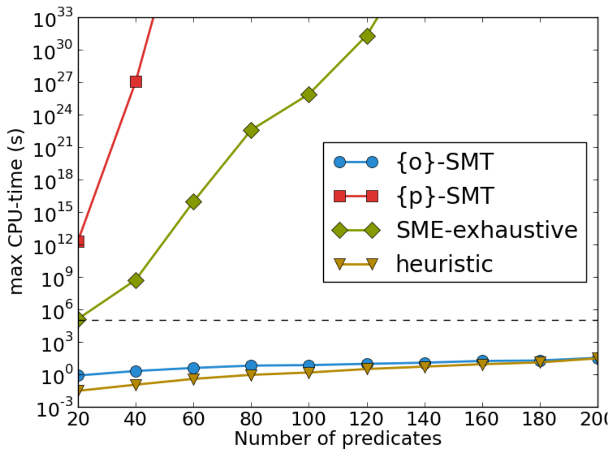


(a) Max CPU time.

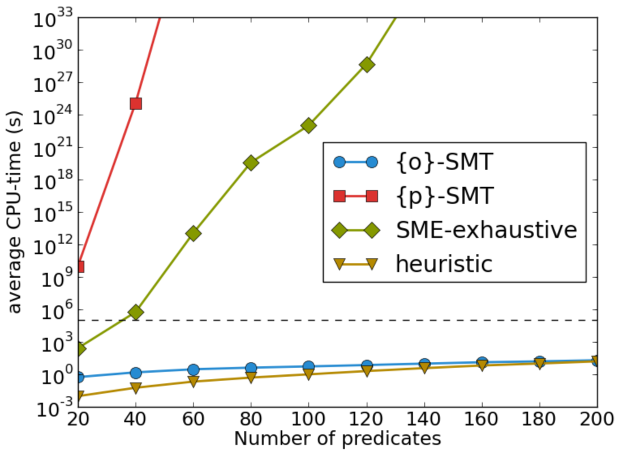


(b) Average CPU time.

Figure 17: Algorithm runtime when manipulating the number of height levels. Points above the dashed line depict search space size instead of time.

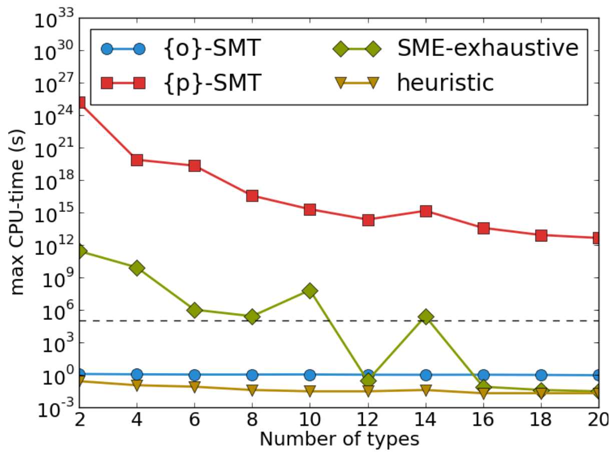


(a) Max CPU time.



(b) Average CPU time.

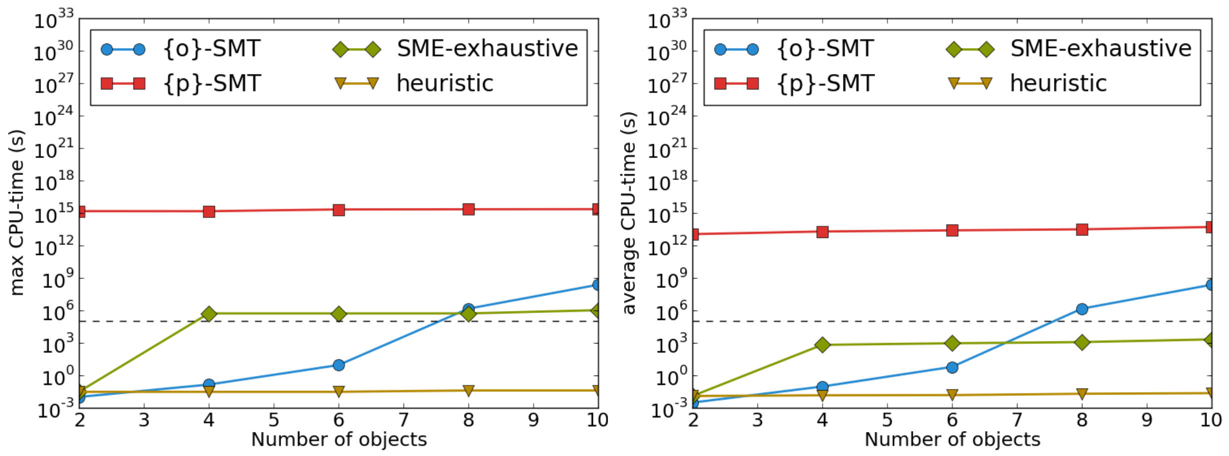
Figure 18: Algorithm runtime when manipulating the number of predicates. Points above the dashed line depict search space size instead of time.



(a) Max CPU time.

(b) Average CPU time.

Figure 19: Algorithm runtime when manipulating the number of types. Points above the dashed line depict search space size instead of time.



(a) Max CPU time.

(b) Average CPU time.

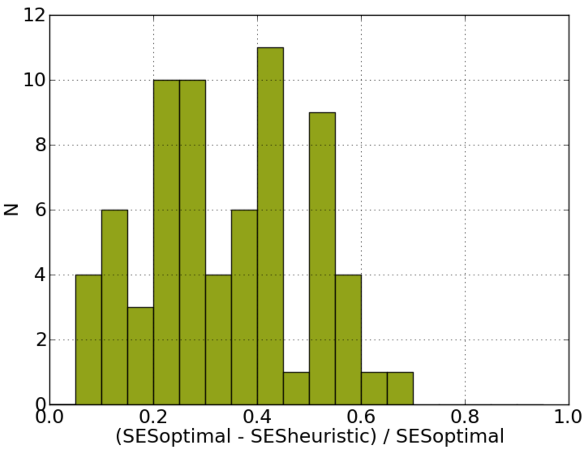
Figure 20: Algorithm runtime when manipulating the number of objects. Points above the dashed line depict search space size instead of time.

4.3 Manually encoded input

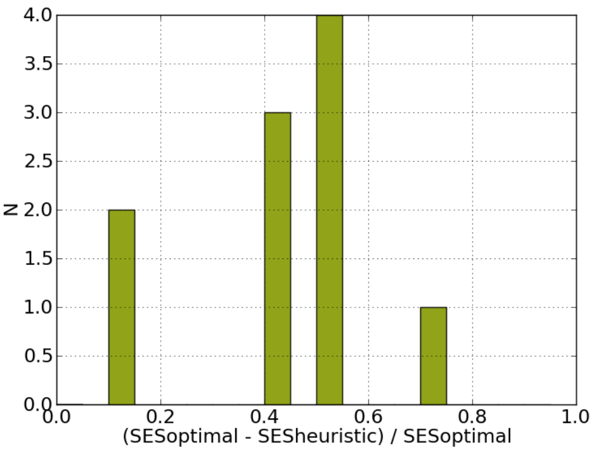
From the 253 combinations of plays (see Table 7), an optimal solution could be computed for 197 of them (others were too large). From these 197, the heuristic gave the optimal solution in 122 trials (64.47%). For the 70 trials where it was not optimal, the average quality (percentage of optimal solution) was 66.56% (standard-deviation was 15.59%), and the minimum and maximum respectively 33.33% and 94.44%. The distribution of these scores are shown in Figure 21a, and show a similar spread as was found for randomly generated input.

All 9506 combinations of fables were small enough to compute the optimal solution. On these pairs, the heuristic was optimal in 9496 trials (99.89%). For the 10 trials where it was not optimal, the average quality was 57.66% (standard-deviation was 17.30%), and the minimum and maximum respectively 29.41% and 88.89%. The distribution of these scores are shown in Figure 21b, although not much can be inferred from this, as the number of non-optimal solutions was very low.

The small number of non-optimal solutions for the fables category is partly caused by the observation that only 14% of the combinations contained an analogy, leaving 1351 combinations with a non-zero solution. The heuristic still performed very good on these non-zero cases, returning the optimal solution for all but 10 examples, which explains the small numbers in Figure 21b. Note that, on the other hand, all combinations of plays contained a non-zero analogy.



(a) Plays



(b) Fables

Figure 21: Distributions of the quality of non-optimal solutions for the heuristic on manually encoded predicate structure pairs.

The influence of height, predicates, types and objects of manually encoded input on the performance of the heuristic were also analysed. The results of these dimensions are listed in Table 8 and showed similar results as was seen for randomly generated input for objects and predicates, but different trends were observed for the other two, as for depth levels performance didn't vary and performance was worse with more predicate types. The latter may be a result of the minimum number of types in the manually encoded cases being larger than the ones tested with the randomly generated input, and the fact that this effect was not isolated. Finally, the influence of these dimensions on the algorithm runtime were compared to the results on the random graphs. These results are listed in Table 9. The algorithms behave more wildly on these examples, as there was less control of input parameters. The worst case runtimes show the expected exponential increase for the fp-tractable algorithms, when their parameters increase.

	Plays					Fables					Combined				
H	% opt	min	max	avg	std	% opt	min	max	avg	std	% opt	min	max	avg	std
1	-	-	-	-	-	100.00	-	-	-	-	100.00	-	-	-	-
2	90.91	0.26	0.26	0.26	0.00	99.92	0.42	0.42	0.42	0.00	99.84	0.26	0.42	0.34	0.08
3	57.14	0.08	0.67	0.35	0.15	99.87	0.11	0.71	0.40	0.20	99.50	0.08	0.71	0.36	0.17
4	60.58	0.06	0.62	0.33	0.16	99.91	0.50	0.50	0.50	0.00	98.07	0.06	0.62	0.34	0.16
5	78.79	0.10	0.58	0.33	0.17	100.00	-	-	-	-	98.85	0.10	0.58	0.33	0.17

(a) Structure height

	Plays					Fables					Combined				
P	% opt	min	max	avg	std	% opt	min	max	avg	std	% opt	min	max	avg	std
0-10	-	-	-	-	-	100.00	-	-	-	-	100.00	-	-	-	-
10-20	-	-	-	-	-	100.00	-	-	-	-	100.00	-	-	-	-
20-30	-	-	-	-	-	99.87	0.11	0.71	0.42	0.17	99.87	0.11	0.71	0.42	0.17
30-40	76.92	0.21	0.58	0.39	0.15	100.00	-	-	-	-	99.49	0.21	0.58	0.39	0.15
40-50	73.13	0.08	0.50	0.34	0.13	-	-	-	-	-	73.13	0.08	0.50	0.34	0.13
50-60	68.18	0.10	0.67	0.37	0.17	-	-	-	-	-	68.18	0.10	0.67	0.37	0.17
60-70	48.72	0.06	0.58	0.33	0.15	-	-	-	-	-	48.72	0.06	0.58	0.33	0.15
70-80	56.52	0.06	0.58	0.28	0.18	-	-	-	-	-	56.52	0.06	0.58	0.28	0.18
80-90	54.55	0.17	0.44	0.33	0.12	-	-	-	-	-	54.55	0.17	0.44	0.33	0.12

(b) Number of predicates

	Plays					Fables					Combined				
T	% opt	min	max	avg	std	% opt	min	max	avg	std	% opt	min	max	avg	std
0-10	-	-	-	-	-	100.00	-	-	-	-	100.00	-	-	-	-
10-20	-	-	-	-	-	100.00	-	-	-	-	100.00	-	-	-	-
20-30	-	-	-	-	-	99.91	0.44	0.71	0.53	0.12	99.91	0.44	0.71	0.53	0.12
30-40	66.67	0.21	0.58	0.41	0.12	99.86	0.11	0.50	0.38	0.17	99.75	0.11	0.58	0.39	0.15
40-50	68.97	0.08	0.50	0.31	0.12	100.00	-	-	-	-	98.82	0.08	0.50	0.31	0.12
50-60	72.06	0.10	0.67	0.37	0.16	-	-	-	-	-	72.06	0.10	0.67	0.37	0.16
60-70	63.64	0.08	0.62	0.32	0.14	-	-	-	-	-	63.64	0.08	0.62	0.32	0.14
70-80	43.75	0.06	0.58	0.32	0.17	-	-	-	-	-	43.75	0.06	0.58	0.32	0.17
80-90	66.67	0.12	0.20	0.16	0.04	-	-	-	-	-	66.67	0.12	0.20	0.16	0.04

(c) Number of types

	Plays					Fables					Combined				
O	% opt	min	max	avg	std	% opt	min	max	avg	std	% opt	min	max	avg	std
0-5	-	-	-	-	-	100.00	-	-	-	-	100.00	-	-	-	-
5-10	100.00	-	-	-	-	99.91	0.44	0.71	0.52	0.10	99.91	0.44	0.71	0.52	0.10
10-15	70.00	0.08	0.62	0.34	0.16	99.88	0.11	0.50	0.33	0.18	99.29	0.08	0.62	0.34	0.16
15-20	59.14	0.06	0.67	0.33	0.16	-	-	-	-	-	59.14	0.06	0.67	0.33	0.16
20-25	65.22	0.17	0.50	0.36	0.11	-	-	-	-	-	65.22	0.17	0.50	0.36	0.11

(d) Number of objects

Table 8: Heuristic solution quality on the manually encoded predicate structure pairs. The table shows the percentage of optimal solutions (% opt), and min,max,mean and standard deviation of the quality of non-optimal solutions on different dimensions. Values that were not present or could not be computed are depicted with a dash.

	Heuristic			SME-exhaustive			$\{o\}$-SMT			$\{p\}$-SMT		
H	max	avg	std	max	avg	std	max	avg	std	max	avg	std
1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	58.56	12.64	19.37	< 0.1	< 0.1	< 0.1
2	< 0.1	< 0.1	< 0.1	$6 \cdot 10^{14}$	$5 \cdot 10^{11}$	$2 \cdot 10^{13}$	$3 \cdot 10^{19}$	$2 \cdot 10^{16}$	$8 \cdot 10^{17}$	$2 \cdot 10^{09}$	$3 \cdot 10^{06}$	$8 \cdot 10^{07}$
3	0.11	< 0.1	< 0.1	$7 \cdot 10^{16}$	$1 \cdot 10^{13}$	$1 \cdot 10^{15}$	$2 \cdot 10^{26}$	$3 \cdot 10^{22}$	$2 \cdot 10^{24}$	$7 \cdot 10^{16}$	$1 \cdot 10^{13}$	$9 \cdot 10^{14}$
4	< 0.1	< 0.1	< 0.1	$1 \cdot 10^{11}$	$1 \cdot 10^{08}$	$3 \cdot 10^{09}$	$7 \cdot 10^{22}$	$4 \cdot 10^{19}$	$1 \cdot 10^{21}$	$3 \cdot 10^{19}$	$1 \cdot 10^{16}$	$6 \cdot 10^{17}$
5	< 0.1	< 0.1	< 0.1	$1 \cdot 10^{15}$	$2 \cdot 10^{12}$	$5 \cdot 10^{13}$	$8 \cdot 10^{20}$	$2 \cdot 10^{18}$	$3 \cdot 10^{19}$	$3 \cdot 10^{22}$	$5 \cdot 10^{19}$	$1 \cdot 10^{21}$

(a) Structure height

	Heuristic			SME-exhaustive			$\{o\}$-SMT			$\{p\}$-SMT		
P	max	avg	std	max	avg	std	max	avg	std	max	avg	std
0-10	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	0.22	< 0.1	0.10	< 0.1	< 0.1	< 0.1
10-20	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	$8 \cdot 10^{08}$	$5 \cdot 10^{06}$	$5 \cdot 10^{07}$	< 0.1	< 0.1	< 0.1
20-30	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	$7 \cdot 10^{11}$	$1 \cdot 10^{09}$	$2 \cdot 10^{10}$	$2 \cdot 10^{06}$	$2 \cdot 10^{03}$	$4 \cdot 10^{04}$
30-40	< 0.1	< 0.1	< 0.1	$2 \cdot 10^{06}$	$4 \cdot 10^{03}$	$9 \cdot 10^{04}$	$1 \cdot 10^{11}$	$1 \cdot 10^{09}$	$8 \cdot 10^{09}$	$5 \cdot 10^{14}$	$9 \cdot 10^{11}$	$2 \cdot 10^{13}$
40-50	< 0.1	< 0.1	< 0.1	$6 \cdot 10^{14}$	$7 \cdot 10^{12}$	$6 \cdot 10^{13}$	$3 \cdot 10^{16}$	$4 \cdot 10^{14}$	$3 \cdot 10^{15}$	$3 \cdot 10^{14}$	$8 \cdot 10^{12}$	$4 \cdot 10^{13}$
50-60	< 0.1	< 0.1	< 0.1	$2 \cdot 10^{07}$	$5 \cdot 10^{05}$	$3 \cdot 10^{06}$	$5 \cdot 10^{15}$	$3 \cdot 10^{14}$	$1 \cdot 10^{15}$	$2 \cdot 10^{11}$	$6 \cdot 10^{09}$	$3 \cdot 10^{10}$
60-70	< 0.1	< 0.1	< 0.1	$2 \cdot 10^{15}$	$4 \cdot 10^{13}$	$3 \cdot 10^{14}$	$2 \cdot 10^{22}$	$4 \cdot 10^{20}$	$3 \cdot 10^{21}$	$1 \cdot 10^{15}$	$3 \cdot 10^{13}$	$2 \cdot 10^{14}$
70-80	< 0.1	< 0.1	< 0.1	$1 \cdot 10^{15}$	$3 \cdot 10^{13}$	$2 \cdot 10^{14}$	$3 \cdot 10^{20}$	$7 \cdot 10^{18}$	$4 \cdot 10^{19}$	$3 \cdot 10^{22}$	$8 \cdot 10^{20}$	$5 \cdot 10^{21}$
80-90	0.11	< 0.1	< 0.1	$7 \cdot 10^{16}$	$4 \cdot 10^{15}$	$1 \cdot 10^{16}$	$2 \cdot 10^{26}$	$8 \cdot 10^{24}$	$4 \cdot 10^{25}$	$7 \cdot 10^{16}$	$3 \cdot 10^{15}$	$1 \cdot 10^{16}$

(b) Number of predicates

	Heuristic			SME-exhaustive			$\{o\}$-SMT			$\{p\}$-SMT		
T	max	avg	std	max	avg	std	max	avg	std	max	avg	std
0-10	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
10-20	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	$2 \cdot 10^{05}$	$2 \cdot 10^{03}$	$2 \cdot 10^{04}$	< 0.1	< 0.1	< 0.1
20-30	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	$1 \cdot 10^{10}$	$2 \cdot 10^{07}$	$3 \cdot 10^{08}$	$2 \cdot 10^{06}$	$3 \cdot 10^{03}$	$6 \cdot 10^{04}$
30-40	< 0.1	< 0.1	< 0.1	$6 \cdot 10^{14}$	$1 \cdot 10^{11}$	$8 \cdot 10^{12}$	$2 \cdot 10^{12}$	$1 \cdot 10^{09}$	$4 \cdot 10^{10}$	$5 \cdot 10^{14}$	$2 \cdot 10^{11}$	$9 \cdot 10^{12}$
40-50	< 0.1	< 0.1	< 0.1	$1 \cdot 10^{08}$	$2 \cdot 10^{05}$	$5 \cdot 10^{06}$	$3 \cdot 10^{16}$	$4 \cdot 10^{13}$	$1 \cdot 10^{15}$	$7 \cdot 10^{13}$	$1 \cdot 10^{11}$	$3 \cdot 10^{12}$
50-60	< 0.1	< 0.1	< 0.1	$1 \cdot 10^{12}$	$2 \cdot 10^{10}$	$1 \cdot 10^{11}$	$3 \cdot 10^{20}$	$4 \cdot 10^{18}$	$3 \cdot 10^{19}$	$7 \cdot 10^{16}$	$9 \cdot 10^{14}$	$8 \cdot 10^{15}$
60-70	< 0.1	< 0.1	< 0.1	$2 \cdot 10^{15}$	$6 \cdot 10^{13}$	$3 \cdot 10^{14}$	$3 \cdot 10^{18}$	$1 \cdot 10^{17}$	$5 \cdot 10^{17}$	$1 \cdot 10^{15}$	$2 \cdot 10^{13}$	$1 \cdot 10^{14}$
70-80	< 0.1	< 0.1	< 0.1	$7 \cdot 10^{16}$	$2 \cdot 10^{15}$	$1 \cdot 10^{16}$	$2 \cdot 10^{22}$	$4 \cdot 10^{20}$	$3 \cdot 10^{21}$	$3 \cdot 10^{22}$	$7 \cdot 10^{20}$	$5 \cdot 10^{21}$
80-90	0.11	< 0.1	< 0.1	$5 \cdot 10^{15}$	$6 \cdot 10^{14}$	$1 \cdot 10^{15}$	$2 \cdot 10^{26}$	$2 \cdot 10^{25}$	$5 \cdot 10^{25}$	$7 \cdot 10^{16}$	$7 \cdot 10^{15}$	$2 \cdot 10^{16}$

(c) Number of types

	Heuristic			SME-exhaustive			$\{o\}$-SMT			$\{p\}$-SMT		
O	max	avg	std	max	avg	std	max	avg	std	max	avg	std
0-5	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	0.11	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1
5-10	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	$2 \cdot 10^{07}$	$1 \cdot 10^{06}$	$3 \cdot 10^{06}$	$9 \cdot 10^{05}$	$1 \cdot 10^{03}$	$3 \cdot 10^{04}$
10-15	< 0.1	< 0.1	< 0.1	$1 \cdot 10^{15}$	$4 \cdot 10^{11}$	$2 \cdot 10^{13}$	$4 \cdot 10^{12}$	$9 \cdot 10^{09}$	$1 \cdot 10^{11}$	$3 \cdot 10^{15}$	$1 \cdot 10^{12}$	$5 \cdot 10^{13}$
15-20	< 0.1	< 0.1	< 0.1	$2 \cdot 10^{15}$	$2 \cdot 10^{13}$	$2 \cdot 10^{14}$	$2 \cdot 10^{18}$	$5 \cdot 10^{16}$	$3 \cdot 10^{17}$	$3 \cdot 10^{22}$	$3 \cdot 10^{20}$	$3 \cdot 10^{21}$
20-25	0.11	< 0.1	< 0.1	$7 \cdot 10^{16}$	$2 \cdot 10^{15}$	$1 \cdot 10^{16}$	$2 \cdot 10^{26}$	$4 \cdot 10^{24}$	$3 \cdot 10^{25}$	$7 \cdot 10^{16}$	$2 \cdot 10^{15}$	$1 \cdot 10^{16}$

(d) Number of objects

Table 9: Algorithm runtime on the manually encoded predicate structure pairs. The table shows the max,mean and standard deviation of the runtimes on all combinations of manually handcoded inputs on different dimensions. The reported runtime is in CPU-seconds, values larger than 10^5 reflect search space size.

5 Discussion

In this research, the aim was to assess the validity of proposed explanations for dealing with intractability of analogy derivation under SMT by means of systematic simulation experiments. The proposed explanations were implemented and their quality of solutions and time complexity were systematically tested using a random predicate structure pair generator. The current section will discuss the results of these experiments along with their implications and limitations.

The first goal in this research was to assess the quality of solutions given by the heuristic. When evaluating the performance on a large number of cases, taken from a large range of dimensions, the heuristic was optimal in 88% of the cases. This number depends highly on the number of predicates in the predicate structure, starting at 93% correct on small predicate structures (20 predicates), but dropping to being optimal only 24% on predicate structures with 200 predicates (see Figure 13). Results also showed that there is no guarantee of near-optimal solutions with the heuristic. For all non-optimal solutions, the average distance from the optimal was around 0.27 with a high standard deviation (0.16). While the majority of non-optimal solutions were less than 0.50 away from the optimal, cases were encountered where the solution was as far off as 0.93 of the optimal (Figure 10). These results show that the heuristic did not perform as well as previously suggested in Forbus and Oblinger (1990), where the algorithm was optimal between 85% and 96% of the cases and close-to-optimal in the other cases (see also Table 2 in the Introduction). While for small predicate structures the heuristic finds the optimal in most cases, this number drops very quickly with larger structures. The distance from the optimal solution was found to be anywhere between 0.0 and 0.93, which is much worse than was believed. It is interesting that while the number of optimal solutions drops with the number of predicates, the average quality of non-optimal solution did not change (as seen in Figure 13). This could be caused by a property of the input, if the minimum possible score increases with the size of the predicate structures as well as the maximum score. Some dimensions appeared to affect the quality of the solutions as well, with the solution quality being better on close, deep structures with more types (Figures 11, 12 & 14). The reason for this is not exactly clear but it may have something to do with deeper structures having fewer root predicates than shallow structures, and in close pairs (and pairs with more predicate types) there are fewer possible consistent and supported combinations of root predicates, resulting in fewer structures to merge. It must be taken into account that the number of predicates was kept constant in this run; structures with more levels also had less predicates per level, the same holds for number of types, where more predicate types also implied less predicates of the same type in the predicate structures. The number of predicates of the same type directly influence the number of initial predicate combinations that can be made, from which later on gmaps are formed. With more layers of predicates, the chance of initial combinations of predicates on the same height level is smaller, and it is less likely that these combinations have matching arguments all the way down to the objects. Closeness influences the chance of initial combinations resulting in supported gmaps as well, with closer predicate structures allowing for less possibilities. The number of initial gmaps is directly related to the combinations in the exhaustive (exponential) merging step, which the heuristic solves by greedy merging. Fewer gmaps means that fewer decisions have to be made in the greedy merging, thus in that case it is less likely that the wrong gmaps are chosen for merging.

These findings suggest that in general the heuristic returns many optimal solutions in small predicate structures and few in large predicate structures. Additionally, the returned solution can be anywhere between the optimal value and very far from the optimal, with the closeness, height and number of predicate types influencing the quality of the solutions.

One important limitation of this quality analysis is that the heuristic and exact solutions were only compared by values (systematicity), and not on structural similarity (van Rooij & Wareham, 2012). It is unknown how far of the heuristic is in terms of structure, as a distance of e.g. 0.10 from the optimal value does not necessarily imply that the structure of the solution is close to the structure of the optimal analogy. On the other hand, when the value is far off, it can be the case that the structure is rather similar (e.g. by just missing one deeply connected predicate in the solution). Evaluation of structure approximation performance was left out in this study for two reasons: First, for at least some cognitive processes, the value of the solutions is all that is important. Judging similarity is a good example of this (Gentner & Markman, 1997). Furthermore, it is not clear how to measure predicate structure similarity, as there are many different methods (Bunke, 2000) and no consensus exists on which would be the best for SMT. Future research could therefore focus on investigating the structural approximation capabilities of heuristics for analogy derivation under SMT (using various measures), for a more complete understanding of the solution quality of the heuristic.

Another limitation lies in the fact that while this study investigated a large range of values for the parameters, optimal solutions can still not be derived for the whole possible input space, thus restricting the examples for which the heuristic can be assessed, as predicate structures with a large number of objects *and* a large number of predicates, the exact algorithms would take too long to compute the optimal solution. A solution to this problem might be to modify the generator in a way to make sure that the core structure that the base and target predicate structure have in common also the most optimal analogy. However, to which extent this is possible is currently unknown.

On the question of runtime of the algorithms, this study found that the fp-algorithms behave as expected (as seen in Figures 18 & 20), with their runtimes increasing exponentially with higher values for their respective parameters. It is somewhat surprising that $\{p\}$-SMT algorithm is generally very slow, even for small numbers of predicates. This is not helped by the fact that restricting predicates is very hard to do as larger and deeper predicate structures automatically have more predicates. SME-exhaustive on the other hand behaves wildly different per example, and shows a wide fluctuation in runtimes, being faster on deep structures (Figure 16) with few types (Figure 19). It is interesting to note that both the heuristic and SME-exhaustive perform better on the same dimensions, and worse on others. This finding raises questions about the practical benefits of the heuristic, as in those cases where it is known to perform better (high structures with few types), SME-exhaustive will be fast (and optimal). However, one important difference was that the heuristic seems to perform better on close predicate structures, while SME-exhaustive becomes slower when the closeness increases. One reason for this could be that close predicate structures could still have many possible substructures to merge, but the greedy merging picks the best order of merging with a higher chance. For example, the largest initial gmap already contains most of the final analogy on closer predicate structures, which benefits the heuristic but there can still be many smaller gmaps that need to be exhaustively combined by SME-exhaustive.

The most interesting finding was that $\{o\}$-SMT performs very well on larger predicate structures with a small number of objects and is comparable in speed to the heuristic (with many predicates, it even performs on-par, see Figure 18), making $\{o\}$-SMT a strong competitor for the heuristic in terms of the trade-off between quality of solutions and speed.

It must be noted that it is highly likely that $\{o\}$-SMT and $\{p\}$-SMT can be improved upon to gain more speed, as the proposed versions of the algorithms were only used to prove the fixed-parameter tractability of SMT. Various techniques from parameterized algorithm design could possibly be applied to improve (parameterized) algorithms for SMT. To illustrate, the worst case runtime of algorithms for the *Vertex Cover* problem has seen dramatic improvements (Balasubramanian, Fellows, & Raman, 1998; Chen, Kanj, & Jia, 2001; Chen, Kanj, & Xia, 2010) since the first FPT results by Buss and Goldsmith (1993). Such improvements have been made for other problems in FPT as well, and are called FPT-races[11]. It seems therefore likely that the runtimes of fp-tractable algorithms for SMT can be improved to a great extent as well.

When comparing the average and worst runtimes of the algorithms, only small differences ('only' a factor 1000, which is small in the range of 10^{30}) can be observed, indicating that the average runtime is well estimated by the worst case runtime. Even if the worst case runtime would never occur in practice, the average still approaches this scenario. This finding provides support for the tractable cognition thesis, as it is sometimes argued that instead of analysing worst case scenarios of computational level models of cognition, average case scenarios that are more likely to occur in practice should be targeted.[12]

The third question in this research was how results for randomly generated input differ from manually encoded examples. The percentage of heuristic solutions that are optimal is very high (99%) for the fables category; this is likely due to the relatively small number of predicates in the predicate structures (refer to Table 7), as it was earlier shown that the quality of the heuristic is better on predicate structures with few predicates. For the other category (plays), the average number of predicates was higher, which was also reflected in the results, dropping the percentage of heuristic optimal solutions to 64%. The quality of non-optimal solutions was similar for both categories (Figure 21), compared to the results on randomly generated pairs presented in Figure 10. However, with the small number of non-optimal solutions for the manually encoded input, this statement must be interpreted with caution. There are also no surprising differences when comparing the trends of the different dimensions (Table 8 and 9), except for the effect of the number of predicate types on heuristic solution quality. The number of types was generally large in manually encoded predicate structures, and larger than the largest number

[11]Refer to the parameterized complexity wiki (`http://fpt.wikidot.com/fpt-races`) for an overview of FPT-races (Parameterized Complexity Community, 2013)

[12]Note that this argument already contradicts itself as the definition of a computational level theory of cognition *should* limit its input description to match the worst case input encountered in practice (van Rooij, 2008)

of types in randomly generated predicate structures. Note that these results also do not reflect the effect in isolation, as was the case with the randomly generated predicate structures. It is likely that the manually encoded predicate structures with many types also had more predicates and this would cause a drop in heuristic performance, as observed with the randomly generated predicate structures. Together, these findings indicate that the randomly generated predicate structures are not very different from manually encoded predicate structures in terms of structure, resulting in more support for our interpretation of our findings.

One question that remains is how representative the manually encoded and random examples are for the actual input space that humans are dealing with. While a large space of possible inputs was examined, it is likely that, in practice, only a subset of this space is encountered. In line with the parameterized approach, this study did illustrate that the dimensions of real world examples are of high importance to the complexity of the problem, and investigating these dimensions should therefore be the target of future research.

5.1 Future work

This research has raised many questions in need of further investigation. Most importantly, as mentioned before, future research should focus on validating the results of this study against human performance in analogy derivation. For instance, investigating whether the input encountered by humans in practice generally has few objects would validate the use of the $\{o\}$-SMT algorithm to model human analogy derivation. It would also be of interest to analyse whether humans perform better (faster) when the input has few objects.

Future studies could investigate the effect of dimensions that were not included in this study on the quality and complexity of the algorithms, for example the influence of functions or the number of ordered predicates. However, this study raised some issues regarding the exact algorithms which first need to be addressed (see Appendix A). Additionally, as was mentioned in the previous section, various measures of structural solution quality could be included in the analysis, for a more complete understanding of the distance of the heuristic solutions from the optimal solutions (van Rooij & Wareham, 2012). It is also possible to explore different measures for solution quality, for example the distance in value from a random search solution could be used (Barr et al., 1995). As for this measure the optimal solution is not needed, the heuristic could be assessed on even larger predicate structures than the ones used in this study. Besides different measures for solution quality, different measures for complexity could also be applied. This study considered time-complexity, while humans are also likely to be constrained by space (memory). However, no theoretical results of the space complexity of SMT have been found in the literature. Additionally, while the memory usage of the algorithms could be monitored, memory usage depends highly on the implementation and characteristics of the programming language.

Finally, the methods used in this study can be applied to other (heuristic) algorithms used for analogy derivation, and provides a platform to systematically compare new methods against SME-exhaustive and the heuristic, which are some of the most influential algorithmic level implementations for SMT. For example, as closeness was found to highly influence the performance of the heuristic and SME-exhaustive, it would be interesting to investigate more specifically what role closeness plays in the complexity of SMT, and how this could be exploited by a possible fp-tractable algorithm. As pointed out in the previous section, there is much room for improvement of the algorithms. For example, the heuristic can be improved by adding other greedy searches, starting with different gmaps, which would likely increase its performance (while only making it a little slower). Furthermore, the fp-tractable algorithms implemented in our study were only proposed to prove fixed-parameter tractability, and can be optimized further (e.g. by adding search pruning methods), to make the handling of slightly larger input more feasible.

5.1.1 Artificial Intelligence

Besides the insights into explanations in Cognitive Science, this thesis makes several noteworthy contributions to the field of (Cognitive) Artificial Intelligence (AI). First, analogy derivation as described by SMT is often used in AI systems (see Gentner and Forbus (2011) for an overview), and the results of this study provide a systematic analysis of the performance of SME, which is one of the most influential algorithms for analogy derivation. Another important contribution for future research on SMT is the implementation of (heuristic) SME, the novel fp-tractable algorithms, and the predicate structure pair generator in a modern and popular programming language. The software is made available online under an open-source license (GPL) to allow and promote further research on this topic.[13] Second, the findings of this study showed the importance of rigorous tests of algorithm performance. For example, these tests showed that the popular heuristic was not as good as was believed, and we discovered cases that are not handled correctly by the exact algorithms, which could possibly be overlooked in the design of such algorithms. Finally, we showed that the parameterized complexity approach to intractable models of human cognition can result in algorithms that are both fast (like heuristics) and optimal, promoting the use of such algorithms for AI systems, where speed and optimality are often important factors as well.

[13]`https://github.com/Tijl/ANASIME`

5.2 Conclusion

This study has shown that $\{o\}$-SMT outperforms the heuristic in terms of speed versus solution quality when restricting the number of objects. Returning to the question posed in the beginning of this study, these findings support the idea that restricting the input space (i.e. limiting the number of objects) is the most viable explanation of how humans could be both fast and optimal when deriving analogies. Besides analogy derivation, the findings presented in this study can be extrapolated to explanations in cognitive science in general, as speed and optimality are a factor in many theories of cognitive functions and heuristics are often used as standard explanations (van Rooij et al., 2012). The same holds for the simulation approach, as this method allowed us to gain specific insights into algorithm performance. Additionally, the benefit of rigorous testing of performance can point out issues with the algorithms that may have been overlooked in the design of these algorithms, for example the cases where the exhaustive algorithms did not return optimal solutions (see Appendix A).

In conclusion, this thesis makes three main contributions. First, this study contributes to a better understanding of the range of options for dealing with intractability of SMT in particular, and explanations in cognitive science in general. Second, the simulation approach taken in this thesis has revealed that many important insights can be achieved only through rigorous tests of algorithm performance. Finally, it has produced a software framework that can be used by other researchers to perform novel tests of algorithms for analogy derivation under SMT, for purposes of both psychological explanation and cognitive engineering.

References

Balasubramanian, R., Fellows, M., & Raman, V. (1998). An improved fixed-parameter algorithm for vertex cover. *Information Processing Letters*, *65*(3), 163–168.

Barr, R., Golden, B., Kelly, J., Resende, M., & Stewart Jr, W. (1995). Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics*, *1*(1), 9–32.

Blanchette, I., & Dunbar, K. (2001). Analogy use in naturalistic settings: The influence of audience, emotion, and goals. *Memory & Cognition*, *29*(5), 730–735.

Blanchette, I., & Dunbar, K. (2002). Representational change and analogy: How analogical inferences alter target representations. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *28*(4), 672–685.

Bunke, H. (2000). Graph matching: Theoretical foundations, algorithms, and applications. In *Proc. Vision Interface 2000* (pp. 82–88).

Buss, J., & Goldsmith, J. (1993). Nondeterminism within p. *SIAM Journal on Computing*, *22*(3), 560–572.

Chater, N., Tenenbaum, J., & Yuille, A. (2006). Special issue: probabilistic models in cognition. *Trends in Cognitive Science*, *10*(7).

Chen, J., Kanj, I., & Jia, W. (2001). Vertex cover: further observations and further improvements. *Journal of Algorithms*, *41*(2), 280–301.

Chen, J., Kanj, I., & Xia, G. (2010). Improved upper bounds for vertex cover. *Theoretical Computer Science*, *411*(40), 3736–3756.

Clement, C., & Gentner, D. (1991). Systematicity as a selection constraint in analogical mapping. *Cognitive Science*, *15*(1), 89–132.

Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2001). *Introduction to algorithms.* MIT Press; Cambridge, MA.

Day, S., & Gentner, D. (2007). Nonintentional analogical inference in text comprehension. *Memory & Cognition*, *35*(1), 39–49.

Deineko, V., Hoffmann, M., Okamoto, Y., & Woeginger, G. (2004). The traveling salesman problem with few inner points. In *Computing and Combinatorics* (pp. 268–277). Springer; Berlin, Germany.

Downey, R., & Fellows, M. (1999). *Parameterized complexity.* Springer; Berlin, Germany.

Dunbar, K. (1995). How scientists really reason: Scientific reasoning in real-world laboratories. In J. Davidson & R. Sternberg (Eds.), *The nature of insight* (pp. 365–395). MIT Press; Cambridge, MA.

Falkenhainer, B., Forbus, K., & Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, *41*, 1–63.

Fellows, M. (2002). Parameterized complexity: the main ideas and connections to practical computing. In *Experimental Algorithmics* (pp. 51–77). Springer; Berlin, Germany.

Flum, J., & Grohe, M. (2006). *Parameterized complexity theory.* Springer; Berlin, Germany.

Forbus, K. (2001). Exploring analogy in the large. In D. Gentner, K. Holyoak, & B. Kokinov (Eds.), *The analogical mind: Perspectives from cognitive science* (pp. 23–58). MIT Press; Cambridge, MA.

Forbus, K., & Gentner, D. (1989). Structural evaluation of analogies: What counts. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society* (Vol. 34, pp. 341–348).

Forbus, K., & Oblinger, D. (1990). Making SME greedy and pragmatic. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society* (pp. 61–68).

Fortnow, L. (2009). The status of the P versus NP problem. *Communications of the ACM*, *52*(9), 78–86.

Frixione, M. (2001). Tractable competence. *Minds and Machines*, *11*(2001), 379–397.

Garey, M., Graham, R., & Johnson, D. (1976). Some NP-complete geometric problems. In *Proceedings of the eighth annual ACM symposium on Theory of computing* (pp. 10–22).

Garey, M., & Johnson, D. (1979). *Computers and intractability.* W.H. Freeman; New York, NY.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, *7*(2), 155–170.

Gentner, D. (1989). The mechanisms of analogical learning. In S. Vosniadou & A. Ortony (Eds.), *Similarity and analogical reasoning* (pp. 197–241). Cambridge University Press; Cambridge, UK.

Gentner, D. (2003). Why we're so smart. In D. Gentner & S. Goldin-Meadow (Eds.), *Language in mind: Advances in the study of language and thought* (pp. 195–235). MIT Press; Cambridge, MA.

Gentner, D., & Christie, S. (2008). Relational language supports relational cognition in humans and apes. *Behavioral and Brain Sciences*, *31*(02), 136–137.

Gentner, D., & Colhoun, J. (2010). Analogical processes in human thinking and learning. In B. Glatzeder, V. Goel, & von Mèuller A. (Eds.), *On Thinking: Vol. 2. Towards a theory of thinking* (pp. 35–48). Springer; Berlin, Germany.

Gentner, D., & Forbus, K. (2011). Computational models of analogy. *Wiley Interdisciplinary Reviews: Cognitive Science*, *2*(3), 266–276.

Gentner, D., Holyoak, K., & Kokinov, B. (2001). *The analogical mind: Perspectives from cognitive science.* MIT Press; Cambridge, MA.

Gentner, D., & Kurtz, K. (2006). Relations, objects, and the composition of analogies. *Cognitive Science*, *30*(4), 609–642.

Gentner, D., & Markman, A. (1997). Structure mapping in analogy and similarity. *American Psychologist*, *52*(1), 45–56.

Gentner, D., & Smith, L. (2012). Analogical reasoning. In V. Ramachandran (Ed.), *Encyclopedia of human behavior (2nd ed.)* (pp. 130–136). Elsevier; Oxford, UK.

Gentner, D., & Toupin, C. (1986). Systematicity and surface similarity in the development of analogy. *Cognitive science*, *10*(3), 277–300.

Gick, M., & Holyoak, K. (1980). Analogical problem solving. *Cognitive Psychology*, *12*(3), 306–355.

Gigerenzer, G. (2008). Why heuristics work. *Perspectives on Psychological Science*, *3*, 20–29.

Gonzalez, T. (2007). *Handbook of approximation algorithms and metaheuristics* (T. Gonzalez, Ed.). Chapman and Hall/CRC; London, UK.

Grootswagers, T., Wareham, T., & van Rooij, I. (2013). *Closer than you think?: Options for efficiently approximating optimal analogies under structure mapping theory.* Poster presented at the 35th Annual Conference of the Cognitive Science Society.

Hagberg, A., Schult, D., & Swart, P. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th Python in Science Conference (SciPy2008)* (pp. 11–15). Pasadena, CA.

Hofstadter, D. (2001). Analogy as the core of cognition. In D. Gentner, K. Holyoak, & B. Kokinov (Eds.), *The analogical mind: Perspectives from cognitive science* (pp. 499–538). MIT Press; Cambridge, MA.

Holyoak, K., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science*, *13*(3), 295–355.

Holyoak, K., & Thagard, P. (1996). *Mental leaps: Analogy in creative thought.* MIT Press; Cambridge, MA.

Kurtz, K., Gentner, D., & Gunn, V. (1999). Reasoning. In M. Bly & D. Rumelhart (Eds.), *Handbook of perception and cognition: Cognitive science* (p. 144-200). Academic Press; San Diego, CA.

MacGregor, J., & Ormerod, T. (1996). Human performance on the traveling salesman problem. *Perception & Psychophysics*, *58*(4), 527–539.

Markman, A., & Gentner, D. (2000). Structure mapping in the comparison process. *The American Journal of Psychology*, *113*(4), 501–538.

Marr, D. (1982). *Vision: A computational investigation into the human representation and processing of visual information.* W.H. Freeman; New York, NY.

Niedermeier, R. (2006). *Invitation to fixed-parameter algorithms.* Oxford University Press; Oxford, UK.

Papadimitriou, C. (1977). The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, *4*(3), 237–244.

Papadimitriou, C. (1993). *Computational complexity.* Addison-Wesley; Reading, MA.

Parameterized Complexity Community. (2013). Table of FPT races. In *the Parameterized Complexity Community Wiki.* Retrieved from `http://fpt.wikidot.com/fpt-races` ([Online; accessed 14-July-2013])

Penn, D., Holyoak, K., & Povinelli, D. (2008). Darwin's mistake: Explaining the discontinuity between human and nonhuman minds. *Behavioral and Brain Sciences*, *31*(2), 109–129.

Rasmussen, L., & Shalin, V. (2007). Is Conversation an Example of Analogical Reasoning? In A. Schwering, U. Krumnack, K. Kühnberger, & H. Gust

(Eds.), *Analogies: Integrating Multiple Cognitive Abilities* (Vol. 5, pp. 57–62).

Rondolà, G. (2013). *psutil: A cross-platform process and system utilities module for Python.* Retrieved from `http://code.google.com/p/psutil/`

Smith, L., & Gentner, D. (2012). Using spatial analogy to facilitate graph learning. In *Spatial Cognition VIII* (pp. 196–209). Springer; Berlin, Germany.

van Rooij, I. (2008). The tractable cognition thesis. *Cognitive Science*, *32*, 939–984.

van Rooij, I., Evans, P., Müller, M., Gedge, J., & Wareham, T. (2008). Identifying sources of intractability in cognitive models: An illustration using analogical structure mapping. In *Proceedings of the 30th Annual Conference of the Cognitive Science Society* (pp. 915–920).

van Rooij, I. (2004). *Tractable cognition: Complexity theory in cognitive psychology.* British Columbia Canada: Department of Psychology, University of Victoria Unpublished doctoral thesis.

van Rooij, I., & Wareham, T. (2012). Intractability and approximation of optimization theories of cognition. *Journal of Mathematical Psychology*, *56*, 232–247.

van Rooij, I., Wright, C., & Wareham, T. (2012). Intractability and the use of heuristics in psychological explanations. *Synthese*, *187*(2), 471–487.

van Rossum, G., & Drake Jr, F. (1995). *Python reference manual.* Centrum Wiskunde & Informatica; Amsterdam.

Veale, T., & Keane, M. (1997). The competence of sub-optimal theories of structure mapping on hard analogies. In *Proceedings of the 1997 International Joint Conference on Artificial Intelligence* (Vol. 15, pp. 232–237).

Wareham, T., Evans, P., & van Rooij, I. (2011). What does (and doesn't) make analogical problem solving easy? a complexity-theoretic perspective. *The Journal of Problem Solving*, *3*(2), 30–71.

Woeginger, G. (2003). Exact algorithms for np-hard problems: A survey. In *Combinatorial Optimization - Eureka, You Shrink!* (pp. 185–207). Springer; Berlin, Germany.

A SME and non-optimal solutions

During the process of implementing the algorithms, it was found that the description of the handling of unordered predicates by SME-exhaustive, as described in Falkenhainer et al. (1989), does not seem to work in practice, which is illustrated with an example in Appendix A.1. Furthermore, while comparing the solutions returned by SME-exhaustive with the solutions from the other exact algorithms, differences were found in the optimal solution (the solution from SME-exhaustive had a lower score than the optimal). It seems that the reason for these differences is that SME-exhaustive does not consider enough sub-structures. This case is also illustrated with an example in Appendix A.2.

A.1 Unordered predicates

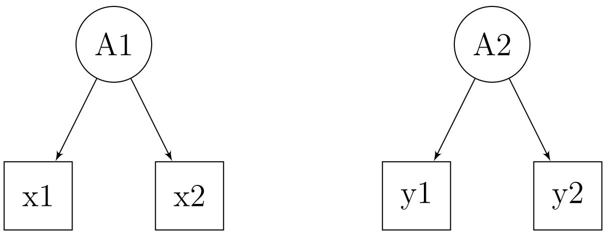


Figure 22: Predicate structure pair illustrating a case with unordered relations. SME-exhaustive is not able to deal with structures containing unordered relations of the same type (In this example, A1 and A2 are both unordered relations of the same type).

SME-exhaustive, as described in (Falkenhainer et al., 1989), does not appear to handle unordered predicates correctly, and it is not clear how the optimal solution would be achieved when predicate structures contain unordered predicates. This is illustrated by the example from Figure 22. Following the description of SME-exhaustive, one initial mhs $(A1, A2)$ will be created with the following mappings, where *emaps* are the mappings between leaves/objects:

$$mh : \{root = (A1, A2); emaps = \{(x1, y1), (x1, y2), (x2, y1), (x2, y2)\}\}$$

Computing the conflicting sets (the set of mhs that is inconsistent with a mhs) for this mhs gives:

- Conflicting$(x1, y1) = \{(x1, y2), (x2, y1)\}$
- Conflicting$(x1, y2) = \{(x1, y1), (x2, y2)\}$
- Conflicting$(x2, y1) = \{(x2, y2), (x1, y1)\}$
- Conflicting$(x2, y2) = \{(x2, y2), (x1, y2)\}$

It is clear that the emaps are always conflicting within this structure, and therefore no consistent solution is possible according to the algorithm as given in Falkenhainer et al. (1989). One way of correctly dealing with this problem is to create different match hypotheses for all possible pairings of the arguments when mapping two unordered relations. This would greatly increase the number of initial gmaps though, which would lead to more combinations to check in the exhaustive combining process.

A.2 Important sub-structures

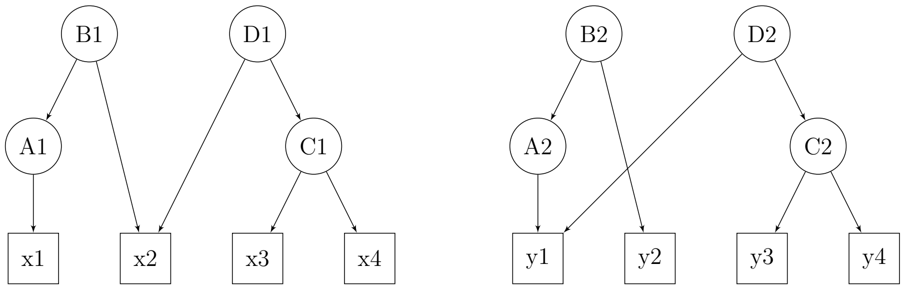


Figure 23: Predicate structure pair illustrating a case with important subgraphs. SME-exhaustive does not consider smaller subgraphs when combining *gmaps*, leading to a sub-optimal solution.

The second case where SME-exhaustive does not appear to give the optimal solution is a case where it does not consider all possible sub-structures. Consider the predicate structures in Figure 23. Now, SME-exhaustive creates two initial gmaps which conflict with each other as they map object x2 to different objects in the target:

- $gmap1 : (B1, B2), mhs = \{(A1, A2), (x1, y1), (x2, y2)\}, conflicting = \{(x2, y1)\}$
- $gmap2 : (D1, D2), mhs = \{(C1, C2), (x2, y1), (x3, y3), (x4, y4)\}, conflicting = \{(x1, y1), (x2, y2)\}$

As these gmaps conflict, only one can be given as solution, while the optimal solution combines gmap1 with a substructure of gmaps:

$$\text{Optimal} = \{(A1, A2), (B1, B2), (C1, C2), (x1, y1), (x2, y2), (x3, y3), (x4, y4)\}$$

$(C1, C2)$ is not considered as an initial gmap as it is a child of $(D1, D2)$ and only the largest structure is used as initial gmap if it is supported and consistent, which is the case for $(D1, D2)$. To deal with cases like this, SME-exhaustive would have to recurse on all children of consistent roots and consider them all as initial gmap in the merging process, which would again lead to much more combinations to check when exhaustively combining gmaps.