Adapting the Adaptive Toolbox

The Computational Cost of Building Rational Behavior

Thesis submitted in partial fulfillment of
the requirements for the degree of
MASTER OF SCIENCE IN ARTIFICIAL INTELLIGENCE

Author:
Marieke SWEERS
Student number: 3046907

Supervisors:
Maria OTWOROWSKA, MSc.
Dr. H. Todd WAREHAM
Dr. Iris VAN ROOIJ

August 31, 2015

Radboud University

Abstract

One of the main challenges in cognitive science is to explain how people make reasonable inferences in daily life. Theories that attempt to explain this either fail to capture inference in its full generality or they seem to postulate intractable computations. One account that seems to aspire to achieve generality without running into the problem of computational intractability is "the adaptive toolbox" by Gigerenzer and Todd (1999b). This theory proposes that humans have a toolbox, adapted through learning and/or evolution to the environment. Such a toolbox contains heuristics, each of them computationally tractable, and a selector which selects a heuristic for every situation so that the toolbox can solve the type of inference problems that people solve in their daily life. In this project we investigate whether such a toolbox can have adapted and under what circumstances. We propose a formalization of an adaptive toolbox and two formalizations of the adaptation process and analyze them with computational complexity theory. Our results show that applying a toolbox is doable in reasonable amount of time, but adapting a toolbox can only be done efficiently when certain restrictions are placed on the environment. If these restrictions occur in the environment and the adaptation processes exploit them humans could have indeed adapted an adaptive toolbox.

Acknowledgements

The process of making my thesis was long and often difficult. At multiple points during the one-and-a-half year process have I wondered whether obtaining a master's degree was all worth the trouble, but at the end of it all I am happy that I went through all of it. I would like to thank some people who helped me during the process.

First of all, my supervisors, who all had the best interest and who all took a great deal of their time for me. They have helped me at all stages of the thesis. Maria was my daily supervisor and I would like to thank her for being there almost every week to have deep thoughts about the toolbox theory. Todd I would like to thank for the long complexity analysis sessions we had (over mail and in person). My friends used to joke that half the work of my thesis was writing e-mails. Also, I felt comforted by his words, quoted from Douglas Adams (don't panic!), which were necessary at a few points in time. I would like to thank Iris for helping me develop my research skills and going beyond that by telling me about matters like the downsides of perfectionism. I am grateful that I got a chance to work with you three.

Then my family: my parents and my sister, for being there when I needed to talk to them and supporting me unconditionally. Furthermore, my friends, who supported me too. Especially Thomas who sat next to me in the tk (the AI computer room) for countless hours and whom I interrupted from his own thesis by making him proofread some text or small parts of the analyses. I would also like to thank all the people in the tk who kept me company during the long process, mostly Franc who also provided cookies :), and lastly all the other people who helped me during the process in one way or another whom I haven't mentioned above.

Contents

Chapter 1

Introduction

During a lifetime humans make millions of decisions. These can be decisions with little impact such as choosing what to eat for dinner or whether or not to do the laundry today or decisions with large impact such as choosing whom to marry or choosing what educational degree to pursue. Gerd Gigerenzer, Peter Todd and the ABC research group have proposed a theory which is intended to capture how humans make such decisions (2008; 2015; 1999b).

Different models of decision making have been proposed previously (Anderson & Milson, 1989; Laplace, 1951). According to Gigerenzer et al., these models assume that humans can make perfect decisions and can use unlimited time, knowledge and computational power for this while humans do not have these resources to make decisions (Gigerenzer & Todd, 1999a). Taking unlimited time to decide what to eat at a restaurant would results in the restaurant closing before dinner was served or worse, in starvation. Moreover, humans do not have unlimited knowledge. One cannot see into the future and know the exact consequences of marrying someone. Lastly, even if humans were omniscient creatures, they still do not have the computational power to integrate all that knowledge.

1.1 The adaptive toolbox

The theory of Gigerenzer et al., called the adaptive toolbox, takes into account the constraints on time, knowledge of the individual and computational power, which makes it a more plausible theory of human decision making. The term 'adaptive toolbox' is a

metaphor for a set of cognitive mechanisms, the tools, each of them adapted to different situations. The tools are called heuristics, which are rules of thumb. They are all fast and frugal, meaning that they can make decision quickly using little information. As such, the heuristics do not pose an unrealistic computational demand on humans. Gigerenzer and colleagues have shown that the heuristics work well (Czerlinski, Gigerenzer, & Goldstein, 1999; Gigerenzer, 2008), and claim that this is so because they have been adapted to fit to the environment through evolution and/or learning (Wilke & Todd, 2010) by changing the heuristics successively in small steps.

1.2 Motivation

The adaptive toolbox account is promising since it does not seem to propose unfeasible resources and puts high emphasis on the environment. It may therefore be able to explain resource-bounded human decision making. If the account is accurate it can be used in numerous ways, for example to facilitate rational thinking by presenting information in a format to which the cognitive system is adapted (Chase, Hertwig, & Gigerenzer, 1998; Gigerenzer & Todd, 1999a) or to build human-level rationality into AI systems.

However, to date, the adaptive toolbox account has not been completely worked out. For example, it is stated that in each situation one heuristic is used but it is unclear how such a heuristic is selected from the entire set of heuristics. In this research we put forth one possible formalization of the adaptive toolbox which includes a selection mechanism and determine whether under this formalization the adaptive toolbox is fast (uses little time resources) with computational complexity analyses. This will potentially advance the theory by rekindling the debate on how heuristics are selected.

Furthermore, there has been very little research on the adaptation process (through evolution and/or learning) of an adaptive toolbox. Toolbox adaptation is not trivial, for there is a large number of configurations a toolbox might have. For example, the number of heuristics, their configuration (in terms of decisions that can be made depending on the information that they use) and the selection mechanism may all vary. Therefore the number of possible toolboxes (which may or may not perform well) is huge and adapting a toolbox such that it performs well may not be easy. Schmitt and Martignon (2006) looked into the time resources needed for the adaptation process of one heuristic, called Take The Best, but did not look into adapting an entire toolbox. However,

it is very important for the adaptive toolbox account to determine whether a complete adaptive toolbox can have adapted, because humans cannot posses an adaptive toolbox if it has not been adapted. This research is a first step in determining whether the toolbox can have adapted. Using computational complexity analyses, we investigated the time resources needed to adapt a toolbox and determined under what circumstances a plausible amount of time is needed, where a plausible amount is some time which is polynomial with respect to the size of the environment. We found that under certain restrictions on the environment a toolbox is indeed adaptable in polynomial time.

1.3 Outline

The thesis is structured as follows. First we give an overview of the adaptive toolbox account and review some prior research (Chapter 2). The methods section (Chapter 3) includes an introduction to the formal concepts and tools of computational complexity theory, the threefold research question and our formalization of an adaptive toolbox. We use computational complexity theory to answer the research questions in the results section (Chapter 4) and end with a general discussion (Chapter 5).

Chapter 2

Background

In this chapter we give an overview of the adaptive toolbox account. First, we briefly explain how the mind and the environment shape human decision making. We continue with an overview of the adaptive toolbox itself, which includes an explanation of how heuristics work, the necessity of a fast and frugal selection mechanism and the notion of ecological rationality and adaptability. Lastly, we discuss some prior research into the adaptive toolbox.

2.1 The mind and the environment

Gigerenzer and colleagues state that humans cannot always make perfect decisions. They are not completely rational, but instead exhibit bounded rationality. This bounded rationality is shaped by both the limitations of the mind and the structure of the environment. Gigerenzer et al. adopted this idea from Herbert Simon (Simon, 1956, 1990), who used the metaphor of a pair of scissors. One blade represents the mind while the other represents the environment, and both together shape human behavior.

The heuristics in the toolbox should be fast and frugal (using little information) because the mind, the first blade, has a limited speed and computational power and the heuristics should not take long to execute. Moreover, they should be frugal in their use of information, because of limitations in the size of memory. Nevertheless, the heuristics are able to work well because they exploit the structure of the environment, the second blade (Todd, 2001).

While other models use internal criteria to measure performance—for example con-

sistency (always prefer a over b) and transitivity (if a is preferred over b and b is preferred over c than a is preferred over c—bounded rationality is measured with correspondence criteria—accuracy, frugality (use of little information), and speed—which measure performance relative to the external world. This is deemed a more appropriate performance measure, as humans need to perform well in the environment in which they live, not perform perfect internally.

2.2 The adaptive toolbox

The adaptive toolbox is described as "the collection of specialized cognitive mechanisms that evolution has built into the human mind for specific domains of inference and reasoning" (Gigerenzer & Todd, 1999a, pg.30). These cognitive mechanisms are heuristics and they form the set of tools in the toolbox.[1]

2.2.1 Heuristics

A heuristic is a mechanism which uses little information in order to make fast decisions which are still accurate. As to date, Gigerenzer and colleagues have proposed nearly a dozen heuristics (Gigerenzer, 2008; Gigerenzer & Gaissmaier, 2011). Each heuristic in the toolbox fits to a certain part of the environment, where the environment is a set of all the situations in which a decision needs to be made that an individual may encounter. If there would be a separate heuristic for every situation, the toolbox could perform very well, since every heuristic could be fit precisely to its own situation. However, since the number of situations a human can come across in its lifetime is near infinite, the number of heuristics needed to cover all possible situations would not be encodable in a brain. Gigerenzer et al. avoid this by stating that the heuristics are able to generalize well over different situations because they are so simple, using little information. Due to this generalization the heuristics cannot match any situation precisely (Gigerenzer & Todd, 1999a, pg.19), but instead give 'good enough' decisions (Gigerenzer, 2008, pg.23).

An example heuristic is Take The Best. This heuristic has been proposed as a description of the processes by which people determine which of two alternatives has a higher value of some variable based on an ordered list of pieces of information. It does not

[1] Gigerenzer et al. state that in some situations other systems, like logic and probability theory are used instead of the toolbox (Gigerenzer, 2008). In this thesis we focus on those situations in which the toolbox is used.

combine all information to come to a decision. Instead, Take The Best searches through the list successively, deciding which alternative to choose based only on the first piece of information that distinguishes the two. For example, it can be used to determine which of two cities is larger based on a list of information which states whether the cities have a train station, have soccer teams or are capitals. If both have a train station, this piece of information cannot be used to decide and the second piece is used, whether the cities have a soccer team. The first information piece which differentiates the two alternatives is used to make a decision.

2.2.2 The selection mechanism

In each situation only one heuristic is applied. This heuristic must somehow be chosen from the entire set of heuristics. It is not clearly stated how this is done, but the mechanism for selecting a heuristics should be fast and frugal (Gigerenzer & Todd, 1999a, pg. 32). If this were not the case, then a human would take a long time to ponder which heuristic would be best. Even if the decision making with heuristics itself is fast and frugal, the entire process would not be fast and frugal.

Gigerenzer suggests that there are multiple simple mechanisms instead of one universal algorithm. He names that in many situations only one or a few heuristics will be applicable, because of the type of the problem (Gigerenzer & Brighton, 2009; Gigerenzer & Sturm, 2012; Gigerenzer & Todd, 1999a). For example, if one recognizes all alternatives, the recognition heuristic cannot be used.[2] It has also been proposed that this mechanism is some higher order heuristic (Todd, Fiddick, & Krauss, 2000; Todd, 2001) or that strategy selection learning is used (Rieskamp & Otto, 2006). The strategy selection learning (SSL) theory states that people learn via reinforcement learning which strategy is best to use in a situation, and the empirical study by Rieskamp and Otto (2006) suggested that humans indeed apply this strategy. However, no clear commitments are made in the account and as such there is no complete theory of how the adaptive toolbox works. This hiatus has been pointed out by some as a reason why the adaptive toolbox account be adequately tested (Cooper, 2000; Newell, 2005).

[2]The recognition heuristic is used to decide which of two alternatives to choose based on whether the alternatives are recognized.

2.2.3 Ecological rationality

If a heuristic performs well in a certain environment, it is fit to that environment and is said to be ecologically rational in that environment. The degree to which a heuristic exploits the structure of the environment is called its ecological rationality (Gigerenzer & Todd, 1999a, pg.13). It is claimed by Gigerenzer et al. that heuristics, which have bounded rationality because they are so simple, can perform well because they have a high ecological rationality.

Gigerenzer states that "Heuristics aim at satisficing solutions (i.e., results that are good enough), which can be found even when optimizing is unfeasible" (Gigerenzer, 2008). So, in order for a toolbox to have a high ecological rationality a satisfactory solution must always be found. The term 'satisfactory solution' in this context is not clearly defined by Gigerenzer and colleagues, although it should be a solution which is good enough.

2.2.4 Adaptation

It is stated that an adapted toolbox is ecologically rational because it is adapted to the environment through evolution and/or learning (Gigerenzer & Goldstein, 1999; Todd, 2001; Wilke & Todd, 2010). The term 'adapted' is used as both the process of changing the toolbox to fit to the environment and the property of the toolbox.

Both the heuristics themselves and their specific configuration (e.g. ordering of cues in Take The Best) are supposedly adapted. By exploiting the structure of the environment the adaptive toolbox is postulated to have a high accuracy even though it is fast (Czerlinski et al., 1999; Gigerenzer & Todd, 1999a, pg.18).

Adapting heuristics

The heuristics are presumably constructed by recombining heuristics and building blocks, small principles that for example determine how information is searched for or how a decision is made based in the information. The building blocks may be evolved capacities, such as recognition memory, forgetting unnecessary information, imitating others (Gigerenzer & Todd, 1999a; Wilke & Todd, 2010), and emotions (Gigerenzer, 2001; Gigerenzer & Todd, 1999a).

2.3 Previous research

2.3.1 Research on toolbox heuristics

Take The Best has been evaluated with empirical studies and computer simulations. Empirical studies provided evidence that Take The Best is used by humans (Bergert & Nosofsky, 2007; Bröder, 2000; Dieckmann & Rieskamp, 2007), but this has been questioned by others (Hilbig & Richter, 2011; Newell, 2005; Newell & Shanks, 2003; Newell, Weston, & Shanks, 2003). With computer simulations Take The Best has been compared to other heuristics and methods like multiple linear regression in real-world environments where one had to decide which of two alternatives (e.g. persons) had a higher value (e.g. attractiveness) based on cues. The tests indicated that Take The Best predicts new data as well as, or better than, other methods such as multiple linear regression (Czerlinski et al., 1999; Gigerenzer & Goldstein, 1999). Other heuristics, such as the recognition heuristic (used to decide between alternatives based only on recognition information), have also been evaluated (Borges, Goldstein, Ortmann, & Gigerenzer, 1999; Goldstein & Gigerenzer, 1999; Pohl, 2006).

2.3.2 Research on toolbox adaptation

Previous research on the adaptation process is the work by Schmitt and Martignon (2006). The authors determined the complexity of adapting Take The Best. Depending on the order of the list of pieces of information, the amount of correct inferences can change. In order to be correct more often in a certain environment, one can adapt the ordering to what works best in that environment. Schmitt and Martignon found that determining this optimal order of the pieces of information cannot be done in polynomial time with respect to the amount of information pieces. Even determining some sub optimal order close enough to the optimum is not doable in polynomial time. This shows that it is not plausible that Take The Best is adapted to have a high ecological rationality.

However, the results of Schmitt and Martignon only hold in a very general environment, because they did not make any assumptions about the structure of the environment. The authors did not look into restrictions on the environment or restrictions on the mind. Furthermore, only the complexity of adapting one heuristic, not the complexity of adapting the entire toolbox, was investigated. Results from Otworowska et

al. (2015) suggest that the processes of evolution alone could not produce ecologically rational toolboxes. For their argumentation they used a simple environment which was structured as a toolbox as formalized in this thesis. That is, there was only one correct action in each situation and it was determined by this 'environment toolbox'. They did a mathematical analysis and computer simulations and found that even in this simple environment, it seemed that the toolbox has too many degrees of freedom to have been created by a random process, such as evolution. It was not ruled out that the toolbox is adapted through the combination of evolution and learning, or learning alone.

In this thesis, we evaluate a part of the adaptive toolbox account, the process of adapting the toolbox to fit to the environment, by evolution and/or learning. The complexity of adapting the entire toolbox is analyzed using computational complexity theory and we determine under what circumstances it is tractable.

Chapter 3

Methods

In this section we present the used methods. We give a short background in computational complexity theory (Section 3.1), present the three-fold research question (Section 3.2) and give the formalization of an adaptive toolbox (Section 3.3).

3.1 Computational complexity theory

Computational complexity theory can be used to determine the resources needed to compute the output of a function, like time. By analyzing the computational complexity of a function one tries to answer the question whether that function is computationally feasible, i.e., whether the output of the postulated function can be computed in a reasonable amount of computational time. It is assumed by cognitive psychologists that cognitive functions, functions that model some aspect of human behaviour, are computationally feasible (van Rooij, 2008). For a cognitive psychologist, computational complexity analyses can be a valuable companion. When studying a cognitive capacity (such as object recognition), a cognitive psychologist can disregard models that are not computationally plausible without having to do empirical tests (van Rooij, 2003, pg. 39-42). Furthermore, computational complexity theory can be used to analyze the adaptation process of a cognitive capacity to determine whether it can have evolved or been learned during a lifetime. Learning is a cognitive capacity and is thus also bounded by the computational constraints of humans, but evolution needs to be computationally feasible as well. Although evolution works on a larger timescale, we explain in Section 3.1.1 that the same boundaries can be applied to evolution as to cognitive

capacities.

When analyzing the complexity of a model, one first defines a function F that represents the model as a mapping from input I to output O: $F : I \rightarrow O$. The function specifies this mapping, without stating how this output is computed from the input. In computational complexity theory, one tries to determine the resources required to compute a function by any algorithm. The traveling salesman problem is an example function and its task is to find a short route through a set of cities. Note that the TSP function does not specify how this route is to be found. During the rest of the overview we use the traveling salesman problem (TSP) as an example function.

A function can be defined as a search function or a decision function. The search version asks for an object called a solution that satisfies some criterion if there is such an object. The decision version merely ask whether a solution exists (van Rooij, 2003). The output for this function is thus either 'yes' or 'no'. An instance of the function for which the output is 'yes' is called a yes-instance; if the output is 'no', the instance is called a no-instance. A function $F : I \rightarrow O$ is solved by an algorithm A if it gives the correct output $o \in O$ for any instance $i \in I$, i.e. if A outputs 'yes' if i is a yes-instance and 'no' if i is a no-instance.

The search version of TSP is to find a route shorter than length k (the output) given a set of cities and a pairwise distance between them (the input). Here, any route shorter than length k is a solution. The decision function takes the same input, but asks whether there is a solution (a route shorter than length k). Formally the TSP search and decision functions are defined as follows:

TRAVELING SALESMAN PROBLEM (SEARCH VERSION)
Input: A set of n cities $C = \{c_1, c_2, \ldots, c_n\}$ with a cost of travel $d(c_i, c_j)$ for all pairs of cities $c_i, c_j \in C$ and a positive integer B, the budget.
Output: A tour that visits every city in the set C, starting and ending in the same city, such that the total cost of the tour is smaller than or equal to B, or the symbol $\emptyset$ if no such tour exists.

TRAVELING SALESMAN PROBLEM (DECISION VERSION)
Input: A set of n cities $C = \{c_1, c_2, \ldots, c_n\}$ with a cost of travel $d(c_i, c_j)$ for all pairs of cities $c_i, c_j \in C$ and a positive integer B, the budget.
Question: Does there exist a tour that visits every city in the set C, starting

and ending in the same city, such that the total cost of the tour is smaller than or equal to B?

In computational complexity theory, the decision version of a function is analyzed, but a cognitive psychologist is often not interested in a function which only outputs the answer 'yes' or 'no'. Nevertheless, these analyses can still provide useful information because properly formulated decision functions are at least as hard as the search version (Garey & Johnson, 1979, pg.19). One can determine the output of a decision version by determining the output for the search version and then checking whether or not there is a solution as output. If a decision function is intractable, the search version of the function cannot find a solution in tractable time; this is so because if a solution can be found in tractable time, the decision function can be solved in tractable time, which contradicts the given intractability of the decision function.

As the size of the input of a function grows, e.g. the number of cities in the TSP increases, the time to solve the function increases. However, there are many instances for each single input size and their computing time may differ. In computational complexity theory, the computing time of a certain input size is measured as the computing time of the instance of that input size that requires the largest amount of time, i.e., the worst-case computing time. Computational complexity theory studies how the computing time grows with its input size. The Big-Oh notation, $O(.)$, is used to describe this time. It represents an upper bound on the complexity, ignoring constants and lower order polynomials. For example, if n is the input size of a function F and algorithm A solves the function in time $4n^2 + n$, the complexity of the algorithm is $O(n^2)$. The time complexity of a function F is defined as the time it takes the fastest algorithm to solve F, described with the Big-Oh notation.

Below, we define in more detail what we mean by 'tractable time' in classical computational complexity theory (Section 3.1.1) and in parameterized computational complexity theory (Section 3.1.2).

3.1.1 Classical computational complexity theory

In classical complexity theory, a function is seen as tractable if it can be solved in polynomial time and as intractable if this is not the case (exponential time or worse). Table 3.1 (column one to four) presents how the running time grows as a function of the input for polynomial time solvable functions and exponential time solvable functions. As the

running time for intractable functions grows so quickly for larger inputs, it is assumed that people cannot solve those functions. Even if humans can do a high number of computations a second, say ten thousand computations, it would take over a day to solve an instance of input size 30 of an intractable function. It is even assumed that only a subset of all tractable functions is solvable by humans (van Rooij, 2008, pg.948). Tractability of a function is then a necessary but not sufficient condition for computational plausibility. We assume that evolution cannot solve intractable functions either, because with reasonably large input size, for example a hundred, the time to solve an intractable function exceeds the time that earth has existed (Dalrymple, 2001). Therefore we say that modeling a cognitive function or the evolution of a cognitive function must be tractable.

The class of functions which can be solved in polynomial time is called P. To prove that a function is tractable, one must give an algorithm which can solve that function in polynomial time.

Definition 3.1. *P is the class of decision problems which are solvable in polynomial time.*

To explain how we can prove that a problem is intractable, we need the class NP, where NP stands for non-deterministic polynomial time.[1] The definition for this class is abstract. A function is in the class NP if there is an algorithm A which can determine in polynomial time whether a given candidate solution of a function $F : I \rightarrow O$ for a yes-instance i_{yes} is correct.

Definition 3.2. *NP is the class of decision problems for which the solution of a yes-instance can be verified in polynomial time.*

The traveling salesman problem is in NP, because there exists an algorithm which can determine in polynomial time whether a given route is shorter than some budget and whether it visits all cities. All functions which are in P are also in the class NP: $P \subseteq NP$. It is strongly believed that the class NP contains functions which are not in P and thus cannot be solvable in polynomial time, i.e., $P \neq NP$ (Garey & Johnson, 1979).

A function F can be proven to be intractable, not solvable in polynomial time, by showing that it is at least as hard as any function in NP. We say that such a function

[1] See for example Garey and Johnson (1979) for a formal definition of the class NP, which makes use of Turing machines.

F is NP-hard. Assuming that the hardest function in NP is not solvable in polynomial time, an NP-hard function F is not solvable in polynomial time either. One can prove that a function F is NP-hard with a polynomial time reduction.

Definition 3.3. *A function $F_1 : I_1 \rightarrow O_1$ polynomial time reduces to $F_2 : I_2 \rightarrow O_2$ if:*

1. *An algorithm A transforms an instance $i_1 \in I_1$ into an instance $i_2 = A(i_1) \in I_2$ in polynomial time;*
2. *instance $i_1 \in I_1$ is a yes-instance then $i_2 \in I_2$ is a yes-instance; and*
3. *instance $i_2 \in I_2$ is a yes-instance then $i_1 \in I_1$ is a yes-instance.*

A reduction is a way of transforming a function F_1 to another, F_2. With a reduction, F_1 can be solved by function F_2 indirectly. First an instance of F_1 is transformed in polynomial time into an instance of F_2, then an algorithm which solves F_2 can solve the instance. To prove a function is NP-hard, it must be reduced from every function in NP. Alternatively, it can be reduced from a function which is known to be NP-hard.

If an NP-hard function F can be solved in polynomial time then the classes P and NP are equal. This is so because any function F_1 in NP can be solved in polynomial time by reducing F_1 to F in polynomial time and then solving the reduced function F. The reduction and solving F both have a polynomial running time, thus solving F_1 would then take polynomial time. However, to date no polynomial time running algorithm has been found that solves an NP-hard function, so it is believed that $P \neq NP$ (Fortnow, 2009).

What to do when a function is NP-hard.

It cannot be that an NP-hard function correctly models a cognitive capacity, because we assume that humans (can) only solve tractable functions. Neither can an NP-hard function correctly model evolution, since, as explained, we assume that evolution can only solve tractable functions. But maybe this intractability can be dealt with without discarding the function. We show two possible ways of dealing with intractability and explain why these do not work.

- Could a probabilistic algorithm solve the function faster than any deterministic algorithm? Such an algorithm makes guesses and outputs the correct answer for a high number of inputs. This notion of probabilistic algorithms is captured by

Input	$O(n^2)$	$O(2^n)$		$O(2^k n^2)$ (10,000 steps/sec)		
size n	100 steps/sec	100 steps/sec	10,000 steps/sec	k=2	k=10	k=25
2	0.04 sec	0.04 sec	0.02 msec	0.0016 sec	0.41 sec	3.7 hrs
5	0.25 sec	0.32 sec	0.19 msec	0.01 sec	2.56 sec	23.3 hrs
10	1.00 sec	10.2 sec	0.10 sec	0.04 sec	10.2 sec	3.9 days
15	2.25 sec	5.46 min	3.28 sec	0.09 sec	23 sec	8.7 days
20	4.00 sec	2.91 hrs	1.75 min	0.16 sec	41 sec	15.5 days
30	9.00 sec	4.1 mths	1.2 days	0.36 sec	1.5 min	5.0 wks
50	25.0 sec	8.4 $\times 10^4$ yrs	8.4 centuries	1.0 sec	4.3 min	3.2 mths
100	1.67 min	9.4 $\times 10^{19}$ yrs	9.4 $\times 10^{17}$ yrs	4.0 sec	17 min	1.1 yrs
1000	2.78 hrs	7.9 $\times 10^{290}$ yrs	7.9 $\times 10^{288}$ yrs	6.7 min	28 hrs	106 yrs

Table 3.1: The computing time for algorithms which run in polynomial time (n^2), exponential time (2^n) and fixed-parameter tractable time as a function of the input size n and parameter k (in the case of the fixed-parameter tractable algorithms). In column 2 and 3 it is assumed a hundred computing steps per second are taken, while in column 4 to 7 ten thousand computing steps per second are taken. Adapted from Table 2.1 and 2.2 in van Rooij (2003).

the class BPP, bounded-error probabilistic polynomial. It is the class of functions which give the correct output in at least a fraction $\frac{2}{3}$ of the possible inputs.

It is assumed that the class BPP is equal to P, so that functions which can be solved probabilistically in polynomial time, can also be solved deterministically in polynomial time (Wigderson, 2006). Because of our assumption that $P \neq NP$, an NP-hard function cannot be solved (with bounded error) by any probabilistic polynomial time running algorithm.

- Could the output be approximated by some polynomial time running algorithm? We could loosen the criterion, not always demanding a correct solution, but rather asking an output that is close to the solution (e.g. some constant value) or an output that is often correct (e.g. in 95% of the cases). Regarding the TSP, we could ask either for routes whose lengths are close to k or routes which are often shorter than or equal to k.

  For many types of approximation, it is often not possible to approximate the solution of an NP-hard problem with a polynomial time running algorithm unless $P = NP$ (Ausiello et al., 1999; Garey & Johnson, 1979; van Rooij, Wright, & Wareham, 2012). Approximation may thus not help in dealing with the intractability

either.

Even though the above ways of trying to deal with intractability do not work, there is a useful and widely-applicable way of coping with intractability. In the next section we show how this can be done.

3.1.2 Parameterized computational complexity theory

When a function is deemed intractable by classical computational complexity theory, one can still analyze whether that function is tractable under some restrictions by using parameterized computational complexity theory (Downey & Fellows, 1999, 2013). The restrictions are posed on the input of a function by bounding certain parameters of the input of the function. If the function is easy to compute for the instances that have those restrictions, the function is said to be fixed-parameter tractable for those parameters.

For example, the traveling salesman problem is NP-hard and thus intractable in the classical sense of complexity theory. However, there are certain types of instances which are easier to solve than others. For example, if the cities form a convex hull the fastest route is easy to find: follow the cities around the convex hull.[2] The number of inner cities, i, (cities lying within the convex position) is a parameter of TSP and the TSP function is easy to compute as long as that parameter is small (Deineko, Hoffmann, Okamoto, & Woeginger, 2006).

We now define fixed-parameter tractable time formally. A function F is fp-tractable (fixed-parameter tractable) for parameter set κ if it can be solved in time $f(\kappa)n^c$, where f is an arbitrary function of κ, which may thus be a super-polynomial function, n is the input size of F and c is some constant. A function F so parameterized relative to a parameter-set κ is denoted by κ-F. As long as the parameters in set κ are small, F is easy to compute. This is because the main input size requires polynomial computing time, while only parameter set κ makes the computing time grow super polynomial. In Table 3.1 (column 5 to 7) the computing time for a function which is fp-tractable for k is shown for three values for k. As can be seen, as long as the value of k is small, the function needs little computing time. For an input size of a hundred only four seconds are needed when ten thousand steps per second are taken. There is no strict boundary

[2] A set of points in a 2D-plane form a convex hull if you can put a rubber band stretching around all of the points such that there are no points of the set outside the 'ring' that the rubber band forms. Points inside the rubber band are called inner points.

for the parameter size, but if k is 25, the function is already unfeasible for input size 2, taking over 3 hours to calculate. The class of functions which can be solved in fixed parameter tractable time is called FPT. To prove that a function is in FPT, one must give an algorithm which solves that function in fp-tractable time.

Definition 3.4. *A function κ-F parameterized relative to a parameter-set $\kappa = \{k_1, k_2, \ldots, k_m\}$ is in FPT if it can be solved in time $f(\kappa)n^c$, where f is an arbitrary function of parameter set κ, n is the input size and c is some constant.*

The class FPT is the fixed-parameter analogue of the class P and there is also an analogue to NP-hardness, called $W[x]$-hardness, where $W[x]$ is a class in the W-hierarchy $= \{W[1], W[2], \ldots, W[P], ...XP\}$ (see Downey & Fellows, 2013, for the detailed definition of these classes). Parameterized functions that are $W[x]$-hard are at least as hard to solve as any problem in $W[x]$. It is assumed that $FPT \neq W[x]$, and thus $W[x]$-hard functions are postulated not to be solvable in fixed-parameter tractable time. $W[x]$-hard functions are said to be fixed-parameter intractable. To prove that a parameterized function is $W[x]$-hard we can do a parameterized reduction from another $W[x]$-hard parameterized problem.

Definition 3.5. *A parameterized function κ_1-$F_1 : I_1 \rightarrow O_1$ parameterized reduces to κ_2-$F_1 : I_2 \rightarrow O_2$ if:*

1. *An algorithm A transforms an instance $i_1 \in I_1$ into an instance $i_2 = A(i_1) \in I_2$ in time $f(\kappa_1)|i_1|^c$ (where c is a constant);*
2. *instance $i_1 \in I_1$ is a yes-instance then $i_2 \in I_2$ is a yes-instance;*
3. *instance $i_2 \in I_2$ is a yes-instance then $i_1 \in I_1$ is a yes-instance; and*
4. *$k \leq g(\kappa_1)$ for some function g and for each parameter $k \in \kappa_2$.*

With a parameterized reduction from κ_1-F_1 to κ_2-F_2, κ_1-F_1 can be solved indirectly by κ_2-F_2 by reducing an instance of κ_1-F_1 in fp-tractable time to an instance of κ_2-F_2 and then solving it with an algorithm which solves κ_2-F_2.

Using Lemma 3.1 and 3.2 (taken from Wareham, 1999), additional results can be obtained from fp-(in)tractability results. That is, given some fp-(in)tractability results for a parameterized function, usually more parameter sets can be obtained for which the function is fp-(in)tractable.

Lemma 3.1. *If function F is fp-tractable relative to parameter-set κ then F is fp-tractable for any parameter-set κ' such that $\kappa \subset \kappa'$.*

Lemma 3.2. *If function F is fp-intractable relative to parameter-set κ then F is fp-intractable for any parameter-set κ' such that $\kappa \subset \kappa$.*

If a parameterized function models a cognitive capacity and is in FPT, it has to be determined whether the restrictions posed by bounding the parameters are plausible, i.e. whether only the instances with these restrictions are solved tractably by humans. If that is the case, the parameterized function is, seen from the computational perspective, a plausible model of the cognitive capacity. The same holds if the parameterized function models evolution of a cognitive capacity: if the instances with the restrictions are the only ones solved tractably by evolution, the parameterized function is a plausible model. A $W[x]$-hard parameterized function cannot model any cognitive capacity or evolution of a cognitive capacity. However, one can still investigate whether other parameter sets make the function tractable.

3.2 The research questions

In Chapter 2 we explained that Gigerenzer et al. assume that the toolbox has adapted (see e.g. Wilke & Todd, 2010). We can define this adaptation process as a function, and in order for their assumption to be computationally plausible this function must be tractable. This is because, as explained in Section 3.1, the time to solve intractable functions is so big for relatively small inputs that it is unlikely that they can model the adaptation process, either through learning or even through evolution. Previous research by Schmitt and Martignon (2006) already showed that adapting a single heuristic is intractable. However, they proved only that this holds when no restrictions are made. Is a whole set of heuristics, including a selection mechanism, tractably adaptable? If not, under what restrictions? This is an important question, because no adaptive toolbox can exist if it cannot have been adapted to what it is now according to Gigerenzer and colleagues. In this thesis we take a first step in answering this question, using computational complexity theory to analyze the complexity of the adaptation process. The question is answered in three sub questions.

We wish to find out whether it is possible that humans use an adaptive toolbox, by determining whether it is possible to have adapted one. Before analyzing the adapta-

tion process, we need to know whether there is a tractable formalization of the toolbox. If there is no such formalization of the toolbox, there does not exist a fast(-and-frugal) toolbox and thus determining whether or not it can have adapted becomes redundant, because humans cannot use an intractable toolbox. The first question therefore states whether there is a formalization of the adaptive toolbox which is tractable. This question is answered using the adaptive toolbox formalization from Section 3.3.2. If there is indeed a tractable adaptive toolbox, the main topic can be addressed. Is the adaptation process tractable? The second and third question cover this question in two steps. The second question concerns the tractability of the formalization of the adaptive toolbox in general. No restrictions are posed, not on the mind, nor on the environment. However, it is plausible that some parameters need to be restricted in order to make the adaptive toolbox tractably adaptable, e.g. posing size constraints on the toolbox or constraining the value of the required ecological rationality. Schmitt and Martignon showed that adapting Take The Best is intractable, which is a strong indication that adapting the entire toolbox is intractable as well. The third question addresses this: If the toolbox is not tractably adaptable in general, are there restrictions which do make it tractably adaptable?

Is the adaptive toolbox tractably adaptable?

1. Is the application of the adaptive toolbox tractable?
2. If so, is the adaptive toolbox tractably adaptable in general?
3. If not, are there restrictions under which the adaptive toolbox is tractably adaptable?

In the next section the environment, the toolbox and ecological rationality are formalized. These formalizations are used in Chapter 4 where the research questions are addressed.

3.3 Formalizing the adaptive toolbox

In this section the Adaptive Toolbox theory is formalized. First the formal environment is given, which is described as a set of situations and a set of actions which are satisfac-

tory in these situations. Our formalization of the adaptive toolbox is described next and the term ecological rationality is formalized last.

3.3.1 The environment

The environment, E, contains a set of situations S, and a set of possible actions to take, A. It is in this environment E that the adaptive toolbox is used. The environment implicitly contains a set of information, which is defined as Boolean pieces of information $i \in I$ which can be either true or false. Examples of pieces of information are 'the sun is shining', 'I am hungry' and 'the capital city of the Netherlands is Amsterdam'. Note that the information may come from from the world (external) and from the self (internal). The actions are anything that a human might be able to do, e.g. 'run away' or 'eat a sandwich'.

As a simplification we assume that the user of the adaptive toolbox, the agent, is omniscient. In other words, all information in the environment is known to the agent. Further, we assume that the agent's knowledge is always correct, i.e., that this knowledge is always the same as the information in the environment. However, it is not plausible that any person would know everything there is to know. Not only are there limitations in what one can remember, there are also limitations to what one can perceive, either due to limitations of the senses (one cannot see infrared) or because there simply is no time to perceive everything. We opted for omniscience of agents nevertheless because we are interested in the complexity and performance of the adaptive toolbox and its adaptation, not so much in the influence of the lack of knowledge or presence of false knowledge on the performance. This has already been addressed partly in previous research by the ABC research group (see for example Goldstein and Gigerenzer (1999). It is of course a simplification to assume people are all-knowing, but it allows us to determine how the toolbox performs without confounding the result with lower performance due to incorrect or missing information.

A situation $s \in S$ is defined as a truth assignment to all pieces of information in the world; $s : I \rightarrow \{T, F\}$. Thus, in a small world where the only information is 'the sun is shining' and 'I am hungry', a possible situation is: {'the sun is not shining', 'I am hungry'}. The set of all possible situations is $\{T, F\}^{|I|}$.

For each situation, it is denoted which actions are satisfactory and which are unsatisfactory. There is at least one satisfactory action in each situation and it is possible that

The sun is shining	I am hungry	I am tired	I am outside	It is raining	Eat ice cream	Eat Sandwich	Run Away	Sleep	Go outside
T	T	F	T	F	1	1	0	0	0
F	F	F	T	T	0	0	1	0	0
F	T	F	T	F	0	1	0	0	0
F	F	T	F	T	0	0	0	1	0
F	T	T	F	T	0	1	0	1	0
T	F	F	F	F	0	0	0	0	1

Table 3.2: An example environment. Each row is one situation. Column one to five denote whether an information piece is true (T) or false (F) in the situation; column six to ten denote whether an action is satisfactory (1) or unsatisfactory (0) in the situation. Row one states that in the situation when the sun is shining, an agent is hungry, not tired and outside and it is not raining, the satisfactory actions are eating an ice cream and eating a sandwich.

multiple actions are satisfactory. The set of actions which are satisfactory in at least one situation in S is called A.

An environment $E = (S, A)$ is a set of situations and the set of satisfactory actions for those situations. The environment does not need to contain all possible situations ($\{T, F\}^{|I|}$), only those that an agent would come across. Thus, $S \subseteq \{T, F\}^{|I|}$. A multi-valued function $D_E : S \to A$ maps a situation to a set of actions in environment E. An example environment is shown in Table 3.2. It contains 5 pieces of information and 6 of all 32 possible situations. For each situation the satisfactory actions are listed.

3.3.2 The adaptive toolbox

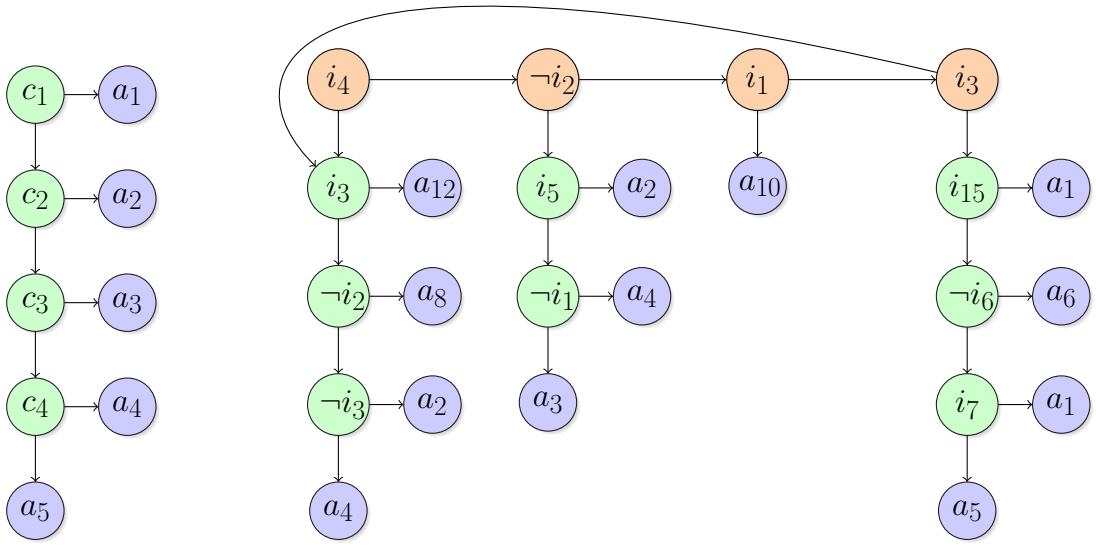
To formalize the complete adaptive toolbox, both the selector which selects a heuristic and the heuristics are defined. Our formalization makes extensive use of fast-and-frugal trees. The heuristics are all formalized as such trees and, in addition to that, the selector is formalized as such as well. First, a detailed explanation of fast-and-frugal trees is given and then it is explained what the adaptive toolbox looks like and why it is formalized as such.

Fast and Frugal Trees One of the heuristics that Gigerenzer and colleagues propose as a tool in the toolbox is the fast-and-frugal tree (Gigerenzer & Gaissmaier, 2011; Martignon, Vitouch, Takezawa, & Forster, 2003). This tree is a one-reason decision mechanism, because it decides what to do based on just one piece of information.

A fast-and-frugal tree contains internal nodes and leaf nodes. Each internal node has exactly two children, of which at least one is a leaf node. Only the last internal node has two leaves as children. The size of a fast-and-frugal tree is defined as the number of internal nodes it contains. See Figure 3.1a (left) for an example tree. Here, the green nodes are internal and the blue nodes are leaf nodes. The children of cue-node c_1 are cue c_2 and action a_1.

To make a decision, an agent uses a fast-and-frugal tree by metaphorically walking through the tree over internal nodes until she arrives at a leaf node, an exit. The route of traversal is determined by the situation in which the agent finds herself and the values of the internal nodes. The internal nodes are cues, functions which evaluate whether a piece of information is true or false in a situation. For example, for the information piece: 'I am hungry' a cue can either ask: 'Am I hungry?' or 'Am I not hungry?'. We call a cue positive if it evaluates whether an information piece is true and negative if it evaluates whether it is false. If a cue evaluates to true in the situation the agent proceeds to the leaf child, otherwise she proceeds to the cue child. The leaf nodes are actions and reaching an action is equivalent to deciding to perform that action. If none of the cues evaluates to true, the last action of the heuristic is performed. We call this the default action. The tree is called fast because there is an exit node at each internal node and thus a decision can be made very quickly. It is frugal in its use of information, because each cue evaluates only single piece of information.

The adaptive toolbox Instead of formalizing the toolbox where only one heuristic which is a fast-and-frugal tree, as Gigerenzer et al. propose, we formalized all heuristics as such. We found that at least some of the heuristics that Gigerenzer has proposed that can be rewritten as fast-and-frugal trees (see Appendix A), so that the set of heuristics in our formalization represents more than just one heuristic. Moreover, if we find that the toolbox adaptation is already intractable with this smaller toolbox (the toolbox containing a subset of all heuristics), it is likely that toolbox adaptation for a toolbox which includes other heuristics is also intractable. We cannot make a similar gener-



(a)

(b)

Figure 3.1: a) A fast-and-frugal tree. The tree contains cues (in green; c_1, c_2, c_3 and c_4) and actions associated to them (in blue; a_1, a_2, a_3, a_4 and a_5). Each cue $c_i \in C$ is a simple Boolean function which asks whether one piece of information $i_i \in I$, part of the information in the environment, is true or false. If the cue evaluates to true, the action attached to that cue is executed; otherwise the next cue function is tried. In this example tree, the tree traversal stops at latest when c_4 returns false. In that case action a_5 is executed. The tree has size four as it contains four cues.
b) A toolbox. Cues are named by the function they evaluate. For example, the first cue of the selector evaluates whether i_4 is true, while the second cue evaluates whether i_2 is false. The selector is represented in orange. The selector is traversed from left to right. If a selector-cue is evaluated to true, the corresponding heuristic is executed. When the last cue of the selector returns false, the first heuristic is executed.

alization when toolbox adaptation is tractable with this subset of tools, because then the extra tools might make toolbox adaptation intractable. Lastly, this is a first step in the investigation of the tractability of the toolbox. Later research can include a higher diversity of heuristics, including those that cannot be rewritten as fast-and-frugal trees.

As explained in Section 2.2.2, no formal definition of a selector has been given by Gigerenzer et al., although they have stated that the selector should be some fast-and-

frugal mechanism in order for the whole toolbox to work fast and frugal. We formalized the selector as a fast-and-frugal tree as well because this is a simple mechanism[3]. Again, if we find that toolbox adaptation is intractable with this simple version of a selector, it will probably also be intractable for a more complicated selector.

The selector's leaf nodes are heuristics instead of actions. If none of the cues evaluates to true in a situation, the first heuristic is performed.[4] We call this the default heuristic, as it is executed when no other heuristic is applicable.

An example toolbox can be seen in Figure 3.1b. Here, an agent starts at the top left node, which evaluates i_3, and traverses the selector to the right until a cue evaluates to true. Then a heuristic is traversed until a decision is made.

3.3.3 Ecological rationality

Ecological rationality defines how well a heuristic is adapted to the environment. If a heuristic is adapted to the environment, it performs well. So, indirectly, ecological rationality defines how well a heuristic performs and is thus a measure of performance.

The environment in our formalization contains only satisfactory and unsatisfactory actions. Only if a satisfactory action is given, will it add to the ecological rationality. We define the ecological rationality of a toolbox as the fraction of situations in S of an environment $E = (S, A)$ in which a satisfactory action is given, such that

$$er = \sum_{\substack{s \in S, \\ T(s) \in D_E(s)}} \frac{1}{|S|} \tag{3.1}$$

is the ecological rationality of a toolbox T. Here, $T(s)$ is the action chosen by toolbox T in situation s and $D_E(s)$ is the set of satisfactory actions according to environment E. If $T(s)$ is in the set $D_E(s)$ a satisfactory action is given. We say that a toolbox is ecologically rational when its ecological rationality er is greater or equal to some minimal ecological rationality er_{min}, $er \geq er_{min}$, where er_{min} is some value between zero and one.
The formalizations of the environment, the adaptive toolbox and the ecological ratio-

[3]Technically, although the heuristics and the selector are trees, the entire toolbox is not a tree, since the two nodes (the first and the last of the selector) are parents for the first heuristic node.

[4]We chose to have the toolbox execute the first heuristic rather than executing the last heuristic. The first heuristic can be seen as the most important heuristic. For example, one may first want to check whether there is a predator. Finding yourself in a situation where no heuristic was picked by the selector, it is a safe bet to use the most important heuristic.

nality given above are used in the functions with which we model toolbox application and adapting a toolbox. In the next section these functions are defined and used to formulate the research questions.

Chapter 4

Results

In this section, we present both our three research questions and results of complexity analyses answering these questions. For each of the first two research question we introduce the functions that we will subsequently analyze with computational complexity theory. For the last research question, the functions derived for the second research question are analyzed with parameterized complexity theory.

4.1 RQ1: Is the application of the adaptive toolbox tractable?

4.1.1 Introducing TOOLBOX APPLICATION

We define the process of applying the toolbox formally as a function called TOOLBOX APPLICATION. As input, the agent receives an adaptive toolbox and the current situation. The first part of the input is a toolbox, which the agent can use to determine what action to take in a situation. The second part of the input is a situation, which contains all information in the environment. Information retrieval is not included in the function, i.e., we assume that the information has already been perceived or retrieved from memory, because we are only interested in the complexity of applying the toolbox, not the complexity of retrieving information. Though information retrieval is necessary for any decision making process, it is not an integral part of the Adaptive Toolbox theory. The output is the action which an agent should take according to the adaptive toolbox in the given situation.

The function TOOLBOX APPLICATION is as follows.

TOOLBOX APPLICATION
Input: A toolbox T, a situation s.
Output: The action a which is associated with situation s according to T.

Toolbox T is a fast-and-frugal adaptive toolbox, formalized in Section 3.3.2; situation s is an element of set S in environment $E = (S, A)$, defined in Section 3.3.1. Implicitly, the toolbox contains a set of actions A', the leaf nodes of the heuristics in the toolbox. The set of actions A in the environment and the set A' do not have to be equal, i.e., there may be actions in A which are not in A' and vice versa. As the action set A is defined as all the actions which are satisfactory in at least one situation in the environment, this means that if an action a is in A' but not in A, a is never a satisfactory action. As there may be multiple satisfactory actions per situation, a satisfactory action may even be chosen with T for a situation s if one (or multiple) satisfactory actions for that situation are not in A', but it may also be that none of the satisfactory actions for s are in A'.

Research question 1 can be formalized as follows:

RQ1: Is the function TOOLBOX APPLICATION in P?

4.1.2 Analyzing TOOLBOX APPLICATION

In order to assess and consequently prove the tractability of TOOLBOX APPLICATION first an algorithm is given (see Algorithm 1 below). Then it is proved that the algorithm runs in polynomial time and that it gives the correct output for all inputs.

We assume that the toolbox does not contain redundant cues on one path, because they make the toolbox larger without adding useful evaluations. A path is defined as the chain of cues an agents evaluates—both in the selector and in the heuristic—when determining what action to take in a situation. A cue c_i is called redundant if it is equal to a cue c_j higher up on the path (where c_i and c_j are equal if $c_i = i_k$ then $c_j = i_k$, or if $c_i = \neg i_k$ then $c_j = \neg i_k$). In any path, there are at most $|I| + 1$ useful cues, where $|I|$ is the number of information pieces. If both cue c_i and $\neg c_i$ are on one path, then one of those will evaluate to true. There are at most $|I|$ cues before the complement of a cue is found on the path. As Gigerenzer assumes heuristics are frugal in their information

use (see Chapter 2), it is valid to assume there will not be double cues as the second of the two cues will simply be redundant.

Algorithm 1: ToolboxApplication(Toolbox, situation)

```
1  for i ← 0 to size (selector) do
2      if selector (i)(situation) = TRUE then            /* Cue i of selector   */
3          for j ← 0 to size (heuristic_i) do
4              if heuristic_i (j)(situation) = TRUE then  /* Cue j of heuristic_i  */
5                  return action_j;
6              end
7          end
8      end
9  end
10 for j ← 0 to size (heuristic_1) do
11     if heuristic_1 (j)(situation) = TRUE then         /* Cue j of heuristic_1  */
12         return action_j;
13     end
14 end
```

Running time of the algorithm

The code in the first for-loop (line 1) is executed at most $|selector|$ times, which is bounded by the maximum number of selector cues ($|I|+1$). The first if-statement (line 2) takes at most time $|I|$, since the agent has to find the piece of information in I. The second for-loop (line 3) takes, as the first for-loop, at most time $|I|+1$.[1] As the first if-statement, the second (line 4) also takes time $|I|$. Returning the action (line 5) takes one time-step. Lines 10 to 14 take the same time of execution as line 3 to 7. They will only be executed if no action has been returned before.

The executing time of this algorithm is $(|I|+1)\times|I|+(|I|+1)\times|I|+1 = 2|I|(|I|+1)+1$. As we take $|I|$ as input size for the function, the function's complexity is then: $O(|I|^2)$.[2]

[1] The longest path without redundant cues is $|I|+1$. As we assume that there are no redundant cues on a path and a path includes both the cues on the selector and the heuristic, the maximum size of the heuristics become shorter the further along a selector it is found. To simplify calculating the running time of the algorithm, an maximum size of $|I|+1$ is assumed for each heuristic.

[2] The input consists of a toolbox and a situation. The size of this input can be written as a polynomial function of $|I|$. The situation has a size of $|I|$. As we assume that the toolbox does not contain redundant cues the maximum size of the toolbox, the number of cues and actions combined, is: $|I|+1$ (selector cues) $+|I|+1$ (#heuristics) $\times(|I|+1)$ (#heuristic cues) $\times 2$ (cues and actions) $+|I|+1$ (the single actions

The function can thus be solved in polynomial time which is tractable.

Proof of correctness

When executing Algorithm 1, an agent goes through the selector cues of the given toolbox one at a time, which takes at most time $|selector|$. For each cue, the agent determines whether a cue is true by looking up the truth value of the piece of information that the cue evaluates in the given situation. If the cue is true, the corresponding heuristic is executed. The agent goes through at most $|heuristic|$ cues of the heuristic in the same manner as the selector: if a cue evaluates to true in the situation, the corresponding action is chosen. If none of the cues in the selector evaluates to true, the first heuristic is executed. Using this algorithm, the agent applies the toolbox in the exact manner as proposed in our formalization. Thus, it gives the action belonging to the given situation according to our formalized adaptive toolbox which is the output which should be given according to the function TOOLBOX APPLICATION.

Given the above the answer to research question 1 is yes, the application of the adaptive toolbox is in P. We continue with the results from the second research question.

4.2 RQ2: Is the adaptive toolbox tractably adaptable in general?

4.2.1 Introducing TOOLBOX ADAPTATION and TOOLBOX READAPTATION

In this section, two functions are introduced which will be used to answer research questions 2 and 3. These functions need to be solved to adapt a toolbox, either by making it from scratch or by readapting it to a slightly changed environment. Because we analyze functions and not an algorithm solving the functions, the functions are generic, i.e., they model both evolution and learning.

at bottom of a heuristic) $= |I| + 1 + 2(|I| + 1)^2 + |I| + 1 = 2(|I|^2 + 3|I| + 2$. Thus, the entire input is: $2(|I|^2 + 3|I| + 2 + |I|$.

Toolbox Adaptation

This function models adapting a toolbox to an environment from scratch. The input is an environment and a minimal ecological rationality; the output is an adaptive toolbox which performs good enough, i.e., has an ecological rationality equal to or higher than the value defined in the input. This function is defined as follows:

> TOOLBOX ADAPTATION
> **Input**: An environment $E = (S, A)$, the positive integers $\#h$, $|h|$ and nc denoting the maximum number of heuristics, the maximum size of a heuristic and the maximum number of negative cues in the entire toolbox respectively, and the minimal ecological rationality $er_{min} \in [0, 1]$.
> **Question**: Is there an adaptive toolbox T with at most nc negative cues, with at most $\#h$ heuristics each at most of size $|h|$ and with an ecological rationality $er \geq er_{min}$ for environment E?

Environment E, the ecological rationality er and toolbox T are defined as in Section 3.3.

Toolbox Readaptation

This function models adapting an existing toolbox to a new, slightly changed environment, which we call readapting. This new environment is mostly the same as the prior environment, that is, one new situation is added to the old environment. We analyze this function to determine how hard it is to adapt the toolbox in small steps. It could very well be that adapting a toolbox relative to a large set of environmental changes (and in the extreme a whole new environment) is intractable but that incrementally adapting to each of the individual environmental changes in that set is tractable relative to each individual change.

The input of the function is an environment, a minimal ecological rationality, a toolbox which has at least that ecological rationality in the given environment, a new situation-action pair and the changes which can be made to the toolbox. Readaptation will be done via the following toolbox-structure changes:

- Deleting a cue-action pair
- Adding a cue-action pair,
- Change a cue

- Change an action

One can make more heuristics by adding a cue-action pair to the selector, thereby adding a heuristic with just one action. No more complicated changes, such as switching two heuristic, are in the list. Although it is plausible such mechanisms exist—think of learning to change the order without having to switch the heuristic bit by bit, or cross-over of (bits of) genes in evolution—each such mechanism can be simulated by using a constant-length sequence of actions drawn from the set of four given above. For example, a switch in the order of two heuristics A and B could be simulated by deleting heuristic A and then adding it again after B. Note that such simulations will not change the complexity of a function from something not solvable in polynomial time to something that is solvable in polynomial time.

The output of the function is a toolbox which is made from the changes C and which performs good enough in the new environment, i.e., has an ecological rationality equal to or higher than the given value in the input.

> TOOLBOX READAPTATION
> **Input**: An environment $E = (S, A)$, the positive integers $\#h$, $|h|$ and nc denoting the maximum number of heuristics, the maximum size of a heuristic and the maximum number of negative cues in the entire toolbox respectively, the minimal ecological rationality $er_{min} \in [0, 1]$, an adaptive toolbox T which has at most $\#h$ heuristics where each heuristic is at most size $|h|$, with at most nc negative cues and ecological rationality $er \geq er_{min}$, a new s-a pair e and a set of changes which can be made $C = \{$delete a cue-action pair, add a cue-action pair, change a cue, change an action$\}$.
> **Question**: Is there an adaptive toolbox T' reconfigured with changes from C which has at most $\#h$ heuristics where each heuristic is at most size $|h|$, with at most nc negative cues and ecological rationality $er \geq er_{min}$ in the new environment $E' = E \cup e$?

Environment E, situation-action pair e, the ecological rationality and toolboxes T and T' are defined in Section 3.3. Research question 2 can now be formulated as follows.

RQ2: Is the function TOOLBOX ADAPTATION in P, and if not, is TOOLBOX READAPTATION then in P?

4.2.2 Analyzing TOOLBOX ADAPTATION

In this section, we show that TOOLBOX ADAPTATION is NP-hard with a polynomial time reduction from DOMINATING SET (DS). First, DOMINATING SET is introduced, followed with the reduction for TOOLBOX ADAPTATION, which follows the steps as put forth in Section 3.1.1.

Dominating set

A vertex set $V' \subseteq V$ is a dominating set for a graph $G = (V, E)$ if for every vertex $v \in V$ either $v \in V'$ or v has a neighbor $u \in V'$, where vertex u is a neighbor if it is directly connected to v with an edge. The entire set of neighbors of a vertex v is called the neighborhood of v and the neighborhood combined with v is called the closed neighborhood of v. The size of an instance of DOMINATING SET is $|V|$.

DOMINATING SET
Input: A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Question: Is there a dominating set V' for G with $|V'| \leq k$?

DOMINATING SET is a known NP-hard problem (Garey & Johnson, 1979). Therefore reducing DOMINATING SET to a function F proves that F is at least as hard and is thus also NP-hard.

We use an example yes-instance of DOMINATING SET, $i_{DSx} = (G, k)$, throughout the two reductions to illustrate the transformations. Parameter k, the maximum size of a dominating set, is 3 and graph G is displayed in Figure 4.1.

Step 1. Polynomial time transformation algorithm A_{TA}

The transformation assumes a minimal ecological rationality $er_{min} = 0.5$. As we explain in the discussion of this section, a similar reduction can be made for certain other values of er_{min}. We start with an instance i_{DS} from DOMINATING SET, which has a graph $G = (V, E)$ and an integer k. The instance is transformed with algorithm A_{TA} to an instance of TOOLBOX ADAPTATION with $er_{min} = 0.5$. Algorithm A_{TA} set the parameters of TOOLBOX ADAPTATION as follows:

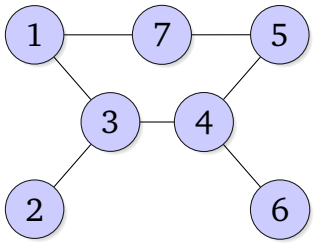


Figure 4.1: An example instance, $i_{DSx} = (G, k)$, of DOMINATING SET, where parameter k is 3. As there is a dominating set with size 3 (nodes 2, 6 and 7), this is a yes-instance. Note that no dominating set of size 2 exists.

- $\#h$ is set to 1.
- $|h|$ is set to $k - 1$.
- nc is set to k.
- er_{min} is set to 0.5

Algorithm A_{TA} constructs the environment $E = (S, A)$ of TOOLBOX ADAPTATION by specifying situations, information pieces and actions.

- Two types of situations are made, neighborhood-situations and x-situations. A neighborhood-situation, $n(v)$, codes for the closed neighborhood of v. There is one such situation for each vertex $v \in V$ so that there are $|V|$ such situations. An x-situation does not code for anything in i_{DS}. There are $|V|$ of those and the set of x-situations is called X.
- A set of $2|V|$ information pieces is made. For each vertex $v_i \in V$ one information piece is made, called v_i. These are set to true in a neighborhood-situation $n(v)$ if v_i is in the closed neighborhood of v. For all x-situations these information pieces are set to false. For each x-situation $x_i \in X$ there is one information piece, x_i, which is true only in situation x_i. Each information piece x_i is thus true only for one situation.
- Lastly, $2|V|$ actions are made. There are $|V|$ vertex-actions, denoted as a_v. An action a_{vi} is called a vertex-action and is satisfactory in a neighborhood-situation $n(v)$ if v_i is in the closed neighborhood of the vertex v, otherwise it is unsatisfactory. For the x-situations, all a_{vi} actions are unsatisfactory. The other $|V|$ actions

Situation	v_1	v_2	v_3	v_4	v_5	v_6	v_7	x_1	x_2	x_3	x_4	x_5	x_6	x_7	a_{v1}	a_{v2}	a_{v3}	a_{v4}	a_{v5}	a_{v6}	a_{v7}	a_{x1}	a_{x2}	a_{x3}	a_{x4}	a_{x5}	a_{x6}	a_{x7}
$n(v_1)$	T	F	T	F	F	F	T	F	F	F	F	F	F	F	1	0	1	0	0	0	1	0	0	0	0	0	0	0
$n(v_2)$	F	T	T	F	F	F	F	F	F	F	F	F	F	F	0	1	1	0	0	0	0	0	0	0	0	0	0	0
$n(v_3)$	T	T	T	T	F	F	F	F	F	F	F	F	F	F	1	1	1	1	0	0	0	0	0	0	0	0	0	0
$n(v_4)$	F	F	T	T	T	T	F	F	F	F	F	F	F	F	0	0	1	1	1	1	0	0	0	0	0	0	0	0
$n(v_5)$	F	F	F	T	T	F	T	F	F	F	F	F	F	F	0	0	0	1	1	0	1	0	0	0	0	0	0	0
$n(v_6)$	F	F	F	T	F	T	F	F	F	F	F	F	F	F	0	0	0	1	0	1	0	0	0	0	0	0	0	0
$n(v_7)$	T	F	F	F	T	F	T	F	F	F	F	F	F	F	1	0	0	0	1	0	1	0	0	0	0	0	0	0
x_1	F	F	F	F	F	F	F	T	F	F	F	F	F	F	0	0	0	0	0	0	0	1	0	0	0	0	0	0
x_2	F	F	F	F	F	F	F	F	T	F	F	F	F	F	0	0	0	0	0	0	0	0	1	0	0	0	0	0
x_3	F	F	F	F	F	F	F	F	F	T	F	F	F	F	0	0	0	0	0	0	0	0	0	1	0	0	0	0
x_4	F	F	F	F	F	F	F	F	F	F	T	F	F	F	0	0	0	0	0	0	0	0	0	0	1	0	0	0
x_5	F	F	F	F	F	F	F	F	F	F	F	T	F	F	0	0	0	0	0	0	0	0	0	0	0	1	0	0
x_6	F	F	F	F	F	F	F	F	F	F	F	F	T	F	0	0	0	0	0	0	0	0	0	0	0	0	1	0
x_7	F	F	F	F	F	F	F	F	F	F	F	F	F	T	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 4.1: The environment of the instance of TOOLBOX ADAPTATION with $er_{min} = 0.5$ which is constructed from the example instance i_{DSx} of DOMINATING SET (Figure 4.1). Each row is one situation and its name is displayed in the first column, where $n(v_i)$ is a neighborhood-situation and x_i is an x-situation. Columns 2 to 15 denote the information pieces I, where each $v_i \in I$ (column 2 to 8) is a vertex information piece and each $x_i \in I$ (column 6 to 15) is an x information piece. A value T or F in column i and row s denotes that information piece i is true or false, respectively, in situation s. Columns 16 to 29 denote the set A of actions that are correct in at least one situation, where each a_{vi} is a vertex-action and each a_{xi} is an x-action. A value 1 or 0 in column a and row s denotes that action a is correct or incorrect, respectively, in situation s.

are x-actions, denoted as a_x. An a_{xi} action is unsatisfactory for all situations, except for situation x_i. In short, the action a_z of any situation s is satisfactory if the information piece i_z is true.

From instance i_{DSx}, shown in Figure 4.1, we can make instance i_{TAx} using transformation algorithm A_{TA}. The parameters belonging to this instance are: $\#h = 2$, $|h| = 3$, $nc = 0$, $er_{min} = 0.5$. See Table 4.1 for the constructed environment.

Running time of algorithm A_{TA}: Setting the parameters takes one step for each parameter (4 steps in total). Each of the $|V|$ situations $n(v)$ of the environment is set by looking up the neighbors of v in G. This takes time $|V|$. Setting the value of all information pieces takes one step for each piece ($2|V|$ steps). Setting an information piece for each of the $|V|$ x-situations takes one step as well ($2|V|$ steps). The action a_z of any situation s is set to satisfactory if the information piece i_z is true ($2|V|$). Making the whole instance takes time $4 + 5|V|^2 + 4|V|^2$, which has complexity $O(|V|^2)$. The transformation thus runs in polynomial time with respect to the size of the instance of

DOMINATING SET.

Step 2. If i_{DS} is a yes-instance then $i_{TA} = A_{TA}(i_{DS})$ is a yes-instance

Given a If yes-instance $i_{DS} = (G(V, E), k)$, there is a set of vertices $V' \in V$ which dominates all of the vertices $v \in V$. That is, for every vertex $v \in V$, a vertex in its closed neighborhood is in V'. For instance i_{TA}, which is constructed from i_{DS} using algorithm A_{TA}, a toolbox can be constructed with one heuristic with size $k - 1$. The selector cue is set to v_1. This could have been any cue, as there is only one heuristics and as this heuristic is also the default and thus the only one which is used. All vertices in dominating set V' except one, called v_k, are cues in the heuristic. That is, for each $v_i \neq v_k \in V'$ there is an information piece v_i as cue. Each cue v_i has a corresponding action a_{vi} and the default action is a_{vk}, the action belonging to v_k. Figure 4.2a illustrates the constructed toolbox for instance i_{TAx}.

There are $|V| + |X| = 2|V|$ situations in the environment, so in order for the toolbox to obtain an ecological rationality ≥ 0.5, a satisfactory action needs to be given in at least $|V|$ situations. For all $|V|$ neighborhood-situations $n(v)$ a satisfactory action is given because the actions of the heuristic are set as the dominating set. That is, in any neighborhood-situation $n(v)$, a cue v_i is true if and only if an action a_{vi} is satisfactory and if and only if vertex v_i is in the closed neighborhood of v. Because there is a dominating set for i_{DS}, there is at least one such v_i cue and action for $n(v)$. Either this v_i is a cue-action pair in the heuristic, and then a satisfactory action is chosen for situation v, or none of the cues v_j in the heuristic are v_i and the default action is performed. However, as there is at least one vertex v_i for each v in the dominating set, and none of the cue pairs corresponding to the dominating set (except v_k) were true, the last vertex, v_k, must dominate v and thus vertex v_k corresponding to action a_{vk} must dominate v. As such, for any neighborhood-situation a satisfactory action is chosen. There are no a_x actions in this toolbox and thus no actions which satisfy any x-situation. The ecological rationality of this toolbox is thus exactly

$$er = \frac{|V|}{2|V|} = er_{min} = 0.5. \tag{4.1}$$

The single heuristic has the correct length ($k - 1$). Thus, if instance i_{DS} of DOMINATING SET is a yes-instance, the transformed instance i_{TA} of TOOLBOX ADAPTATION is also a

yes-instance.

Step 3. If $i_{TA} = A_{TA}(i_{DS})$ is a yes-instance then i_{DS} is a yes-instance

Given a yes-instance $i_{TA} = (E', er_{min} = 0.5)$, there is a toolbox T such that the ecological rationality is equal to or higher than 0.5 in environment E' with one heuristic of length $k - 1$. This means that T gives a satisfactory action for at least $|V|$ situations as there are $2|V|$ situations in E'. We show that it is only possible to satisfy $|V|$ situations if there is a dominating set of size at most k in graph G of $i_{DS} = (G, k)$. The selector cue may evaluate any information piece, since the default heuristic is always applied because there are no other heuristics. We go through three toolbox types:

1. **Only satisfying x-situations:** Each x-situation has a single satisfactory action. Since there are at most k actions in the toolbox, at most k x-situations can be satisfied. If the toolbox would only satisfy x-situation it would have an ecological rationality
$$er = \frac{k}{2|V|} < er_{min} = 0.5 \tag{4.2}$$
as k is always smaller than $|V|$.[3] Hence, no toolbox of this type is viable, i.e., has $er \geq 0.5$ and heuristic size at most $k - 1$.

2. **Only satisfying neighborhood-situations:** The only way each neighborhood-situation $n(v)$ is satisfied is if at least one satisfactory action of $n(v)$ is in A', the set of actions of the toolbox. If i_{TA} is a yes-instance (while satisfying only neighborhood-situations), the toolbox satisfies all $|V|$ neighborhood-situations such that
$$er = \frac{|V|}{2|V|} = er_{min} = 0.5 \tag{4.3}$$
and there are at most k actions in A'. The k actions in set A' must correspond directly to the vertices of a k-dominating set of V, because any situation $n(v)$ is satisfied only if the vertex v_i of action a_{vi} is in the closed neighborhood of v. It does not matter what the value of the cues in the toolbox are, as long as the toolbox is traversed such that for each neighborhood-situation a satisfactory action is chosen.

[3] In those cases where $k = |V|$, this would be a yes-instance, but then there would also be a trivial dominating set which includes all vertices in the graph.

This means that we do not even need to assume there may only be positive cues, as it does not matter what value the cues have.

3. **Satisfying neighborhood-situations and x-situations:** It is possible to satisfy m x-situations and have an ecological rationality of at least 0.5 if at least $|V| - m$ neighborhood-situations can be satisfied. The ecological rationality is then

$$er \geq \frac{m + |V| - m}{2|V|} = er_{min} = 0.5. \tag{4.4}$$

The m x-situations are satisfied only if there are m x-actions in the toolbox. The $|V| - m$ neighborhood-situations must then be satisfied by the $k - m$ actions which are left, because none of the x-actions are associated with any neighborhood-situations. If this is possible there is a dominating set of size $\leq k$ for i_{DS}. This is so because there are $k - m$ v-actions which correspond to vertices, a partial dominating set, with which at least $|V| - m$ vertices are covered. There are at most m unsatisfied neighborhood-situations (corresponding to the vertices not dominated by this partial dominating set). The partial dominating set may be m pieces larger, since the dominating set is allowed to be size k. The at-most-m not-dominated vertices can be covered by choosing those m vertices as the rest of the dominating set. The total size of the dominating set is then $\leq k$ and all $|V|$ vertices are covered. Here too, it does not matter how many negative cues there are in the toolbox, because as long as for all situations a satisfactory action is chosen, it does not matter how one arrives at that action.

The three types of toolboxes above form the only possible toolboxes that can be made given a transformed instance of TOOLBOX ADAPTATION. Thus, if an instance of TOOLBOX ADAPTATION is a yes-instance, there is at least one viable toolbox of one of the toolbox types, i.e., a toolbox with $er \geq er_{min}$ with one heuristic of size $k - 1$. We have shown that if a viable toolbox of any of those types can be made for an instance, there is a possible dominating set of at most size k for the instance of DOMINATING SET. Thus, we have proved that i_{DS} is a yes-instance if $i_{TA} = A_{TA}(i_{DS})$ is a yes-instance.

We have now proven that all three conditions of a correct polynomial time reduction hold for our reduction from DOMINATING SET to TOOLBOX ADAPTATION (with $er_{min} =$

0.5), thereby proving that DS polynomial time reduces to TA. This means that TOOLBOX ADAPTATION is at least as hard as DOMINATING SET and because DOMINATING SET is NP-hard we prove that TOOLBOX ADAPTATION is NP-hard as well. The answer to the first part of research question 2 is no, TOOLBOX ADAPTATION is not in P.

Discussion

The reduction above proves that adapting a toolbox is not tractable if the toolbox needs to have an ecological rationality of at least 0.5. This proof can be adapted for any minimal ecological rationality writable as

$$er_{min} = \frac{|V|}{|V| + |X|}, \tag{4.5}$$

where $|X|$ is the number of x-situations, x-information pieces and x-actions which can be any positive integer including zero. Note that Equations 4.1 to 4.4 in step 2 and 3 of the proof can be replaced by their generalized versions in Equations 4.6 to 4.9. The reduction still holds relative to these generalized versions.

$$er = \frac{|V|}{|V| + |X|} = er_{min} \tag{4.6}$$

$$er = \frac{k}{|V| + |X|} < er_{min} \tag{4.7}$$

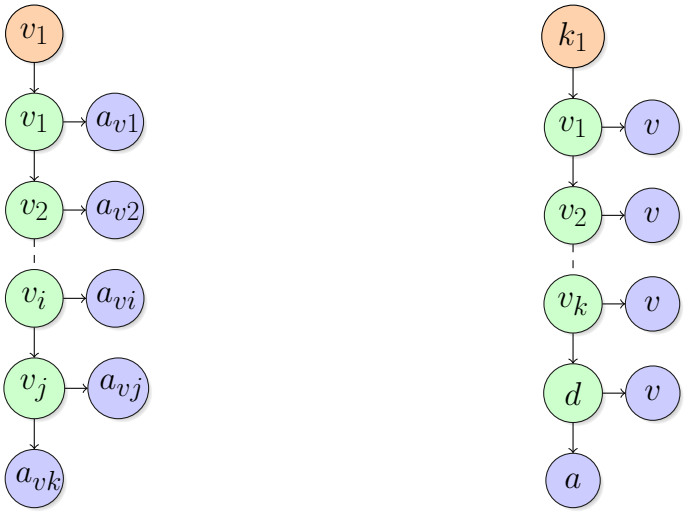
$$er = \frac{|V|}{|V| + |X|} = er_{min} \tag{4.8}$$

$$er \geq \frac{m + |V| - m}{|V| + |X|} = er_{min} \tag{4.9}$$

We have shown that adapting a toolbox is not tractable in general for a minimal ecological rationality writable as Equation 4.5. However, although this means for each such er_{min} at least one instance exists which is not solvable in polynomial time, there exist also instances for at least some such values of er_{min} where adapting a toolbox can be done in polynomial time. This is true as long as the number of situations for which a satisfactory action needs to be given is small. For example, in an instance where the toolbox needs to perform a satisfactory action in only one situation in an entire environment $E = (S, A)$, the minimal ecological rationality is $\frac{1}{|S|}$. A toolbox can

be constructed in polynomial time which contains one heuristic with one action $a \in A$ which is associated with at least one situation $s \in S$ (Figure 4.3a). Constructing a toolbox which performs a satisfactory action in at least two situations in E ($er_{min} = \frac{2}{|S|}$) can also be done in polynomial time, even if these situations need different actions (Figure 4.3b). The first cue in the only heuristic in a toolbox is set to cue i_i and an action a which is satisfactory in a situation where i_i is true is paired with the cue. The default action of the heuristic is set to an action b which is satisfactory in a situation where i_i is false.

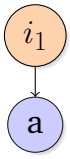
It is not clear how constructing a toolbox in such a way generalizes to having to give a satisfactory action in a higher number of situations. Since we know that for any er_{min} writable as Equation 4.5 there exists an instance which is not solvable in polynomial time, we know for certain that, unless $P = NP$, a generalized polynomial time construction of a toolbox with high enough ecological rationality is impossible.



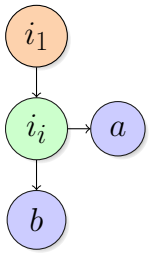
(a) The toolbox which is constructed for a yes-instance of TOOLBOX ADAPTATION. All actions a_{vz} correspond to a vertex v_z in a dominating set V'.

(b) The toolbox which is constructed for a yes-instance of TOOLBOX READAPTATION. All actions k_i correspond to a vertex v_i in a dominating set V'.

Figure 4.2: Example toolboxes for TOOLBOX ADAPTATION (a) and TOOLBOX READAPTATION (b).



(a) A toolbox which always performs action a, no matter what the situation is.



(b) A toolbox which performs action a in all situations where i_i is true and b in all situations where i_i is false.

Figure 4.3: Simple toolboxes with $er \geq er_{min}$ for sufficiently small values of er_{min}.

4.2.3 Analyzing TOOLBOX READAPTATION

We proved that TOOLBOX ADAPTATION is intractable for any value of er_{min} writable as $\frac{|V|}{|V|+|X|}$. Making a toolbox from scratch is thus intractable, but what if we already have a preexisting toolbox and wish to adapt it to a slightly changed environment? In this section we show that TOOLBOX READAPTATION is NP-hard as well with a reduction from DOMINATING SET.

Step 1. Polynomial time transformation algorithm A_{TR}

Let's assume an instance i_{DS} of DOMINATING SET, consisting of a graph $G = (V, E)$ and an integer k. The instance $i_{TR} = (E, e, er_{min}, T, C)$ from TOOLBOX READAPTATION is constructed from i_{DS} with A_{TR} as follows. Algorithm A_{TR} sets the parameters of TOOLBOX READAPTATION as follows.

- $\#h$ is set to 1.
- $|h|$ is set to k.
- nc is set to 0.
- er_{min} is set to 1.

Algorithm A_{TR} constructs environment E of TOOLBOX READAPTATION as follows.

- A neighborhood-situation, $n(v)$, is constructed for each vertex $v \in V$. An extra situation, $\emptyset$, is constructed, which does not correspond to any neighborhood.

- For each vertex $v_i \in V$ one information piece called v_i is made, and an additional piece, called d. For each neighborhood-situation $n(v)$, all information pieces $v_i \in I$ are set to false, except those information pieces corresponding to the closed neighborhood of v and piece d which are set to true. For situation $\emptyset$, all information pieces are set to false.
- Action set A contains three actions (v, a and b). For each neighborhood-situation $n(v)$, the action v is satisfactory, the other actions are unsatisfactory. For the $\emptyset$-situation all actions except a are unsatisfactory; a is satisfactory.

Algorithm A_{TR} constructs the new situation e by setting all information pieces to false except piece d, which is set to true. Action b is satisfactory, the others are unsatisfactory.

Algorithm A_{TR} constructs toolbox T as a toolbox with one heuristic and selector cue v_1. The first k cues of the heuristic are random cues (other than d) and the last cue is d. It has action v for all actions except for the default action, which is a. See Figure 4.2a for the toolbox. This prior toolbox performs perfect in E: if none of the cues above d evaluates to true cue d will catch the neighborhood-situation so that a satisfactory action, v, is still performed. Situation $\emptyset$ is also satisfied, because none of the cues is true for $\emptyset$.

Finally, the set of changes C can be found in Section 4.2.1.

From instance i_{DSx}, shown in Figure 4.1, we can make instance i_{TRx} using transformation algorithm A_{TR}. The parameters belonging to this instance are: $\#h = 1$, $|h| = 3$, $nc = 0$ and $er_{min} = 1$. The environment E and new situation e are shown in Table 4.2 and toolbox T can be found in Figure 4.2a.

Running time of algorithm A_{TR}: Setting the parameters takes one step each (4 steps in total). For each vertex v of the $|V|$ vertices in DOMINATING SET, a situation with $|V|+1$ information pieces and three actions is constructed by determining the neighbors of v in $|V|$ steps and then setting the $|V|+1$ information pieces and 3 actions. Constructing the situations $\emptyset$ and e takes in total time $2(|V|+1+3)$. Constructing an instance i_{TA} takes time $4+|V|(2|V|+1)+(2|V|+1+3)$, which has complexity $O(|V|^2)$. The transformation thus runs in polynomial time with respect to the size of the instance of DOMINATING SET.

Situation	v_1	v_2	v_3	v_4	v_5	v_5	v_6	d	v	a	b
$n(v_1)$	T	F	T	F	F	F	T	T	1	0	0
$n(v_2)$	F	T	T	F	F	F	F	T	1	0	0
$n(v_3)$	T	T	T	T	F	F	F	T	1	0	0
$n(v_4)$	F	F	T	T	T	T	F	T	1	0	0
$n(v_5)$	F	F	F	T	T	F	T	T	1	0	0
$n(v_6)$	F	F	F	T	F	T	F	T	1	0	0
$n(v_7)$	T	F	F	F	T	F	T	T	1	0	0
$\emptyset$	F	F	F	F	F	F	F	F	0	1	0
e	F	F	F	F	F	F	F	T	0	0	1

Table 4.2: The environment of the example instance i_{TRx} of TOOLBOX READAPTATION which is constructed from the example instance, i_{DSx}, of DOMINATING SET (Figure 4.1). Situation $n(v_i)$ denotes the closed neighborhood of v_i, information piece v_j denotes whether v_j is in the closed neighborhood $n(v_i)$. d is a dummy information piece.

Step 2. If i_{DS} is a yes-instance then $i_{TR} = A_{TR}(i_{DS})$ is a yes-instance

Instance i_{TR} contains a toolbox with k random cues and the last cue, d and needs to be readapted to accommodate situation e to perform perfect. Situation $\emptyset$ can only be accommodated by keeping the default action a. Situation e can only be accommodated by changing the action paired with d to b. But then all the neighborhood-situations need to be accommodated by the cues above d, because they all have action v. Since i_{DS} is a yes-instance, there is a dominating set V' of size k for graph G of i_{DS}. All k cues in toolbox T are changed so that for each $v_i \in V'$ there is a cue v_i. At least one cue v_i in the toolbox is true for each situation $n(v)$, because an information piece v_i is true if v_i is in the closed neighborhood of v. Thus each neighborhood-situation is satisfied with an action v.

All situations in E and situation e can be accommodated. Furthermore, the toolbox has not exceeded the size parameters nor does it contain any negative cues. Thus, if an instance for DOMINATING SET is a yes-instance, the transformed instance of TOOLBOX READAPTATION is also a yes-instance.

Step 3. If $i_{TR} = A_{TR}(i_{DS})$ is a yes-instance then i_{DS} is a yes-instance

Given a yes-instance i_{TR} of TOOLBOX RECONFIGURATION, there is a toolbox which satisfies all situations of i_{TR} (those in E and e), so that its ecological rationality is 1.0, which means that all situations are satisfied. Situations $\emptyset$ and e can only be accommodated

when action a is the default action and cue d paired with action b is directly above. The only way a neighborhood-situation $n(v)$ is satisfied is if at least one of the k cues is an information piece which is true in situation $n(v)$. Since 'true' for information piece v_i directly codes that v_i is in the closed neighborhood of v, the vertex is dominated. This holds for all vertices, since all have a corresponding situation. Thus, if an instance of TOOLBOX RECONFIGURATION is a yes-instance, the corresponding instance of DOMINATING SET is also a yes-instance.

We have proven that all three conditions for a correct polynomial time reduction hold for our reduction from DOMINATING SET to TOOLBOX READAPTATION, thereby proving that DS polynomial time reduces to TA. As DOMINATING SET is NP-hard, this proves that TOOLBOX READAPTATION is NP-hard as well. The answer to the second part of research question 2 is no, TOOLBOX READAPTATION is not in P.

Discussion

We have proven that apart from TOOLBOX ADAPTATION, even the seemingly simpler function TOOLBOX READAPTATION is NP-hard. However, this second result only holds when the ecological rationality is 1.0, which means that the toolbox must perform perfectly.

4.3 RQ3: Are there restrictions under which the adaptive toolbox is tractably adaptable?

Both TOOLBOX ADAPTATION and TOOLBOX READAPTATION are NP-hard, which means that they are intractable in general. But what happens if we restrict some parameters in the functions? In this section we determine whether (re)adapting the toolbox is tractable under restrictions. First we propose a set of parameters (Section 4.3.1) after which we show the intractability results (Section 4.3.2) and tractability results (Section 4.3.3) for the parameterized functions TOOLBOX ADAPTATION and TOOLBOX READAPTATION .

4.3.1 Introducing the parameters

We propose the following parameters: $\#h$, $|h|$, nc, pc, er_{min}, $|I|$, $|A|$, a_{max} and $corr$ (see Table 4.3 for a description). Both $\#h$, the number of heuristics, and $|h|$, the maximum

size of a heuristic (defined as the maximum number of cues in a heuristic), are parameters for the size of the toolbox. Together they completely constrain the size of the toolbox. It seems plausible that restraining these parameters makes it easier to adapt a toolbox, because this restricts the search space. The parameters nc and pc represent the number of negative and positive cues in the toolbox, respectively. Combined, they are the total number of cues in the selector and heuristics in a toolbox. The minimal ecological rationality, er_{min}, states how well a toolbox needs to perform. It seems plausible that it is easier to find a toolbox if er_{min} is restricted because if the minimal ecological rationality is low, there will probably be many toolboxes which reach this value and thus perform good enough. The parameters $\#h$, $|h|$, nc, pc and er_{min} are all related to the toolbox by either restricting the size or the performance of a toolbox.

The other four parameters involve the environment. The number of information pieces in the environment, $|I|$, restricts the number of toolbox candidates by restricting the number of cues that can be made. Parameters $|A|$ and a_{max} are the number of actions and the maximum number of actions per situation, respectively. Lastly, we look at the correlation in the environment. This restriction is suggested by Gigerenzer et al. to explain why a toolbox can perform well (Martignon & Hoffrage, 1999). It is stated that little information is needed in order for the toolbox to perform well, because information is often highly correlated (Todd, 2001). However, it might also be used to explain why a toolbox is able to adapt quickly. An information piece is fully correlated with a second if it has the same value (true or false) as the second for all situations. If two pieces of information are fully correlated, this means that one of the pieces is redundant, because all the information can be obtained by attending to only one of the two pieces. We define parameter $corr$ as the number of groups of fully correlated information pieces.

4.3.2 Fixed-parameter intractability results

In this section the fp-intractability results are presented for both TOOLBOX ADAPTATION and TOOLBOX READAPTATION. We made use of the reductions in Section 4.2 and the knowledge that k-DOMINATING SET is W[2]-hard (Downey, Fellows, & Stege, 1999).

Name	Definition
$\#h$	The number of heuristics in a toolbox
$\|h\|$	The length of a heuristic counted in the number of cues
nc	The number of negative cues in a toolbox
pc	The number of positive cues in a toolbox
er_{min}	The minimal ecological rationality a toolbox should have
$\|I\|$	The number of information pieces in an environment
$\|A\|$	The number of actions
a_{max}	The number of satisfactory actions per situation
$corr$	The number of groups of fully correlated information pieces.

Table 4.3: The parameters which may be causing the intractability of TOOLBOX ADAPTATION and TOOLBOX READAPTATION.

TOOLBOX ADAPTATION

We show that the polynomial time reduction from DOMINATING SET to TOOLBOX ADAPTATION (Section 4.2.2) is also a parameterized reduction with respect to parameters $\#h$, $|h|$, nc, pc and er_{min}. To prove this we need to prove that the four points in Definition 3.5 hold. We already proved that the transformation runs in polynomial time. This means it also runs in fp-tractable time, $O(f(k)n^c)$. We can set $f(k)$ to a constant function, independent of k, so that $O(f(k)n^c)$ becomes $O(n^c)$, which is some polynomial time function. Points 2 and 3 are already proved for the polynomial time reduction. This leaves us with proving point 4, whether the parameters $\#h$, $|h|$, nc, pc and er_{min} are dependent only on k. That is, whether the parameters can be written as some function of k, independent of the input size of DOMINATING SET.

Parameter $|h|$ is set to $k-1$, a function of k alone ($f(k) = k-1$). Parameter nc is also dependent only on k, as $nc = k$. Since any number of positive cues is allowed, any number of negative cues is allowed. Thus, pc is also set to k in this reduction. Parameters $\#h$ and er_{min} are constants and thus trivial functions of k ($g(k) = 1$ and $h(k) = 0.5$). This means that point 4 holds as well and thus the reduction in Section 4.2.2 is also a parameterized reduction from k-DOMINATING SET to $\{\#h, |h|, nc, pc, er_{min}\}$-TOOLBOX ADAPTATION.

This means that $\{\#h, |h|, nc, pc, er_{min}\}$-TOOLBOX ADAPTATION is at least as hard as k-DOMINATING SET, and thus W[2]-hard as well. Given Lemma 3.2, TOOLBOX ADAPTATION is also fp-intractable for any subset of the parameter set.

TOOLBOX READAPTATION

We show that the polynomial time reduction from DOMINATING SET to TOOLBOX READAPTATION (Section 4.2.3) is also a parameterized reduction with respect to parameters $\#h$, $|h|$, nc, pc and er_{min}. To prove this we need to that prove the four points in Definition 3.5 hold. We proved that the transformation runs in polynomial time which means it also runs in fp-tractable time, $O(f(k)n^c)$, which proves point 1. We also proved points 2 and 3. Both parameter $|h|$ and pc are k, and the other parameters are constants in this reduction. Parameter $\#h$ is 1, nc is 0, er_{min} is 1, a_{max} is 1 and $|A|$ is 3. Thus, point 4 also holds. Thus, $\{\#h, |h|, nc, pc, er_{min}, a_{max}, |A|\}$-TOOLBOX READAPTATION is W[2]-hard. TOOLBOX READAPTATION is also fp-intractable for any subset of the parameter set.

4.3.3 Fixed-parameter tractability results

In this section we prove fixed-parameter tractability of TOOLBOX ADAPTATION and TOOLBOX READAPTATION for parameter sets $\{|I|, |A|\}$, $\{corr, |A|\}$, $\{|I|, a_{max}\}$ and $\{corr, a_{max}\}$. Given Lemma 3.1 both functions are also fixed-parameter tractable for all super sets of any of the four parameter sets. First we give the proof for $\{|I|, |A|\}$ which we will extend to prove that the other three parameter sets make both functions fp-tractable as well.

Parameters $|I|$ and $|A|$

We prove that $\{|I|, |A|\}$-TOOLBOX ADAPTATION and $\{|I|, |A|\}$-TOOLBOX READAPTATION are in FPT by giving an algorithm and proving that it solves the functions in fixed-parameter tractable time $O(f(|I|, |A|)$.

The algorithm consists of two steps. First, all possible toolboxes are generated and then for each toolbox it is determined whether it has a high enough ecological rationality ($er \geq er_{min}$) with the size constraints, less than $\#h$ heuristics, each with a size smaller than $|h|$ and less than nc negative cues. If there is a toolbox T with these constraints, the output of function TOOLBOX ADAPTATION is 'yes', otherwise it is 'no'.

The function TOOLBOX READAPTATION receives a prior toolbox T as input. Therefore, for this function we do not only need to determine whether there is a toolbox T' with these constraints, but also whether it can be constructed from prior toolbox T. The 'delete cue-action pair' change can be applied until the toolbox is completely empty. Thereafter, it can be rebuilt with the 'add cue-action pair' change. Thus, any toolbox T'

can be constructed from a prior T. Therefore, if a toolbox T' with $er \geq er_{min}$ can be found, the output of function TOOLBOX READAPTATION is also 'yes'.

Step 1: Generating toolbox set $\mathcal{T}$ from I and A. One can generate all possible toolboxes by simply constructing all combinations. We assume that any toolbox only contains useful cues so that the maximum size of a heuristic and that of a selector are both $|I| + 1$.[4] In our further calculations of the number of toolboxes, we ignore this assumption for simplicity. As such, the outcome of the number of generated toolboxes is an upper bound.

A heuristic of length $|I| + 1$ contains $|I| + 1$ cues and $|I| + 2$ actions. Each cue can have $2|I|$ different values and each action can have $|A|$ different values. The number of possible heuristics is thus: $(2|I|)^{|I|+1} \times |A|^{|I|+2}$. A toolbox contains at most $|I| + 1$ heuristics, using the same reasoning as for the size of a heuristic. Further, a toolbox contains a selector with $|I| + 1$ cues, each of them can have $2|I|$ different values. The number of possible selectors is thus: $(2|I|)^{|I|+1}$. In total, there are $(2|I|)^{|I|+1} \times |A|^{|I|+2} \times (|I| + 1) \times (2|I|)^{|I|+1}$ possible toolboxes. We call this set of toolboxes $\mathcal{T}$ and write its size as a function $f_1(|I|, |A|)$.

A toolbox can be created by adding cue-action pairs. The number of cue-action pairs is at most $|I| + 2 \times |I| + 1$. Thus, $|I| + 2 \times |I| + 1$ steps need to be taken to create one toolbox. We write the time to create one toolbox as a function $f_2(|I|)$. The time to generate the entire toolbox set $\mathcal{T}$ is then $f_1(|I|, |A|) \times f_2(|I|)$ and thus depends only on $|I|$ and $|A|$.

Step 2: Finding a viable toolbox in set $\mathcal{T}$. In this step all toolboxes in $\mathcal{T}$ are evaluated. To evaluate the ecological rationality of one toolbox one has to go through all situations $s \in S$ and determine whether or not a satisfactory action is given by that toolbox. This is done by evaluating at most $2(|I| + 1)$ information pieces, in the selector and chosen heuristic combined. Thus for each toolbox we need time: $|S| \times 2(|I| + 1)$. As explained in Section 3.3.1, the total set of situations is $\{T, F\}^{|I|}$, so that there are $2^{|I|}$ situations. $|S| \times 2(|I| + 1)$ can be rewritten as: $2^{|I|} \times 2(|I| + 1)$, which is some function $f_3(I)$, dependent only on $|I|$. To evaluate whether the a toolbox follows the constraints $\#h$, $|h|$ and nc, one has to go through all cues in

[4] See Section 4.1.2 for an explanation of useful cues.

the toolbox and keep score of the amount of heuristics, the number of negative cues and the largest heuristic. This takes time at most time $f_2(|I|)$, as the time to create one toolbox is the number of cues in the toolbox.

The whole algorithm thus takes time: $f_1(|I|,|A|) \times f_2(|I|) + f_1(|I|,|A|) \times f_3(|I|) + f_2(|I|) = f_4(|I|,|A|)$. The running time of this algorithm is thus dependent only on the number of information pieces and actions. The complexity of the algorithm can be written as: $O(f_4(|I|,|A|) \times |I|^0)$.

Parameter $corr$

We prove that the functions TOOLBOX ADAPTATION and TOOLBOX READAPTATION that are in FPT for parameter set $\{|I|,|A|\}$ are also in FPT when the parameter set includes $corr$ instead of $|I|$. We do this by giving an algorithm which solves the functions in time $f_1(corr,|A|) \times |I|)$. This algorithm makes use of the algorithm above by inserting Step 0 into the above algorithm before Step 1 and 2.

Step 0: Finding a set of uncorrelated information pieces I_c. In this step the groups of correlated information pieces are determined. First, two information pieces are compared in each situation. If they have the same value in each situation they are correlated and therefore put in one group, otherwise they are put in separate groups.

Every other information piece i_i is appointed to a group by comparing it to the existing groups. This is done by picking one information piece i_j from a group I' and determining whether i_i and i_j are correlated. If they are, i_i is assigned to I'; otherwise i_i is compared to another group that has not been compared to i_i. If no such group is available, i_i is assigned to a new group. All $i \in I$ are assigned to a group in this manner.

All information pieces in one group always have the same value in any situation. If the value of one information piece is known, the values of the other pieces on the group can be directly inferred. Therefore there is no information loss if only one information piece is used. One random information piece is chosen from each group. These pieces form a new set, called I_c and its size is $corr$, the number of groups with fully correlated information pieces.

Comparing two information pieces takes time $2|S|$ and each information piece needs to be compared with at most the number of correlated groups in the environment, $corr$. Above we defined the maximum number of situations as $2^{|I|}$. Only one information piece per group of correlated pieces can contribute to the number of situations as each situation is different from the next. The number of situations is therefore at most 2^{corr}. For each information piece at most $corr$ comparisons are made and there are $|I|$ information pieces. Picking one information pieces from each group takes time $corr$. The total time of this step is then $2 \times 2^{corr} \times corr \times |I| + corr$ which can be written as some function $f_5(corr) \times |I|$.

Step 1 and 2. The two steps from above now are run. However, as input they do not receive I and A, but I_c and A. In the running time we can simply replace the size of I with the size of I_c ($corr$). The running time for step 1 and 2 is then $f_5(corr, |A|)$.

The whole algorithm takes time $f_5(corr) \times |I| + f_3(corr, |A|)$, which is of complexity $O(f(corr, |A|) \times |I|)$ for some function f. This is a fixed-parameter tractable running time with respect to $corr$ and $|A|$. Thus, $\{corr, |A|\}$-TOOLBOX ADAPTATION and $\{corr, |A|\}$-TOOLBOX READAPTATION are in FPT.

Parameter a_{max}

We prove that the functions TOOLBOX ADAPTATION and TOOLBOX READAPTATION that are in FPT for parameter set $\{|I|, |A|\}$ and $\{corr, |A|\}$ are also in FPT when the parameter sets includes a_{max} instead of $|A|$.

The total number of actions, $|A|$, is dependent on the number of situations and on the number of satisfactory actions per situation a_{max}. There can be at most a_{max} different actions for each situation, $|A| = a_{max} \times |S|$.

The first algorithm (Steps 1 and 2) takes time $f_3(|I|, |A|)$ and solves both $\{|I|, |A|\}$-TOOLBOX ADAPTATION and $\{|I|, |A|\}$-TOOLBOX READAPTATION in fp-tractable time. The number of situations is $2^{|I|}$ in this algorithm and we can therefore rewrite $|A|$ as $a_{max} \times 2^{|I|}$ such that we rephrase the time to run the algorithm as being dependent only on a_{max} and $|I|$. Therefore, $\{|I|, a_{max}\}$-TOOLBOX ADAPTATION and $\{|I|, a_{max}\}$-TOOLBOX READAPTATION are in FPT as well.

The second algorithm (Steps 0, 1 and 2) takes time $f_1(corr, |A|) \times |I|$ and solves

both $\{corr, |A|\}$-TOOLBOX ADAPTATION and $\{corr, |A|\}$-TOOLBOX READAPTATION in fp-tractable time. The number of situations in this algorithm is 2^{corr} and thus we can substitute $|A|$ for $a_{max} \times 2^{corr}$ so that we can rewrite f_1 as being dependent only on a_{max} and $corr$: $f_1(corr, a_{max})$. The entire time to run the second algorithm is then: $f_1(corr, a_{max}) \times |I|$. Therefore, $\{corr, a_{max}\}$-TOOLBOX ADAPTATION and $\{corr, a_{max}\}$-TOOLBOX READAPTATION are in FPT as well.

4.4 Summary of the results

In Table 4.4 and 4.5 we have summarized the results for TOOLBOX ADAPTATION and TOOLBOX READAPTATION that can be obtained from the (parameterized) reductions and the two fp-tractable algorithms, and those we extrapolated from the results by using Lemmas 3.1 and 3.2. Table 4.4 shows what parameter sets make TOOLBOX ADAPTATION fp-tractable and (fp-)intractable. We have shown that TOOLBOX ADAPTATION is $W[2]$-hard for the parameter set $\{\#h, |h|, nc, pc, er_{min}\}$. From Lemma 3.1 we know that no subset of these parameters can make the function fp-tractable. Furthermore, we know that the four sets $\{|I|, |A|\}$, $\{corr, |A|\}$, $\{|I|, a_{max}\}$ and $\{corr, a_{max}\}$ make the function fixed-parameter tractable. From Lemma 3.2 we know that any super set of these parameter sets makes the function fp-tractable as well. This leaves some sets (constructed from combinations of the parameters in Table 4.3) for which we cannot determine the parameterized complexity with these results. These are the sets which include parameters of only one of the two sets $\{|I|, corr\}$ and $\{|A|, a_{max}\}$ and possibly any of the other parameters. Table 4.5 shows these results for TOOLBOX READAPTATION. As we have proven fixed-parameter intractability for a larger set of parameters, we know for more parameter set whether the function is tractable or not. Only for those sets which include the parameters $|I|$ and/or $corr$, but not $|A|$ or a_{max} do we not have complexity results.

Only in those cases where certain parameters related to the structure of the environment are restricted is (re)adapting a toolbox fixed-parameter tractable. This shows that there are circumstances under which a toolbox can be (re)adapted in relatively small amount of time. The question is now whether these circumstances are plausible in the real world. This is discussed in Section 5.

	$\emptyset$	$\#h$	$\lvert h\rvert$	$\#h$, $\lvert h\rvert$	nc, pc	nc, pc, $\#h$	nc, pc, $\lvert h\rvert$	nc, pc, $\#h$, $\lvert h\rvert$
$\emptyset$	NP-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
er_{min}	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
$\lvert A\rvert$	?	?	?	?	?	?	?	?
er_{min}, $\lvert A\rvert$	?	?	?	?	?	?	?	?
a_{max}	?	?	?	?	?	?	?	?
er_{min}, a_{max}	?	?	?	?	?	?	?	?
a_{max}, $\lvert A\rvert$	?	?	?	?	?	?	?	?
er_{min}, a_{max}, $\lvert A\rvert$	?	?	?	?	?	?	?	?
$\lvert I\rvert$	?	?	?	?	?	?	?	?
er_{min}, $\lvert I\rvert$	?	?	?	?	?	?	?	?
$\lvert I\rvert$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $\lvert I\rvert$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$\lvert I\rvert$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $\lvert I\rvert$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$\lvert I\rvert$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $\lvert I\rvert$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$	?	?	?	?	?	?	?	?
er_{min}, $corr$	?	?	?	?	?	?	?	?
$corr$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, $\lvert I\rvert$	?	?	?	?	?	?	?	?
er_{min}, $corr$, $\lvert I\rvert$	?	?	?	?	?	?	?	?
$corr$, $\lvert I\rvert$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, $\lvert I\rvert$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, $\lvert I\rvert$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, $\lvert I\rvert$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, $\lvert I\rvert$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, $\lvert I\rvert$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT

Table 4.4: A summary of the results for TOOLBOX ADAPTATION. An entry in a certain column and row denotes whether the parameters of that column and row together make the function TOOLBOX ADAPTATION FPT, NP-hard (only in the case when the set is empty) or $W[2]$-hard. For some sets, this cannot be inferred from the results. This is denoted with a question mark.

	$\emptyset$	$\#h$	$\lvert h\rvert$	$\#h$, $\lvert h\rvert$	nc, pc	nc, pc, $\#h$	nc, pc, $\lvert h\rvert$	nc, pc, $\#h$, $\lvert h\rvert$
$\emptyset$	NP-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
er_{min}	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
$\lvert A\rvert$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
er_{min}, $\lvert A\rvert$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
a_{max}	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
er_{min}, a_{max}	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
a_{max}, $\lvert A\rvert$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
er_{min}, a_{max}, $\lvert A\rvert$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard
$\lvert I\rvert$	?	?	?	?	?	?	?	?
er_{min}, $\lvert I\rvert$	?	?	?	?	?	?	?	?
$\lvert I\rvert$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $\lvert I\rvert$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$\lvert I\rvert$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $\lvert I\rvert$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$\lvert I\rvert$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $\lvert I\rvert$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$	?	?	?	?	?	?	?	?
er_{min}, $corr$	?	?	?	?	?	?	?	?
$corr$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, $\lvert I\rvert$	?	?	?	?	?	?	?	?
er_{min}, $corr$, $\lvert I\rvert$	?	?	?	?	?	?	?	?
$corr$, $\lvert I\rvert$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, $\lvert I\rvert$, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, $\lvert I\rvert$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, $\lvert I\rvert$, a_{max}	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
$corr$, $\lvert I\rvert$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT
er_{min}, $corr$, $\lvert I\rvert$, a_{max}, $\lvert A\rvert$	FPT	FPT	FPT	FPT	FPT	FPT	FPT	FPT

Table 4.5: A summary of the results for TOOLBOX READAPTATION. An entry in a certain column and row denotes whether the parameters of that column and row together make the function TOOLBOX READAPTATION FPT, NP-hard (only in the case when the set is empty) or $W[2]$-hard. For some sets, this cannot be inferred from the results. This is denoted with a question mark.

Chapter 5

Discussion

Gigerenzer and colleagues assume that the adaptive toolbox has been created through evolution and/or learning (Wilke & Todd, 2010). To determine whether this assumption is true, we analyzed the computational complexity of adapting a toolbox. With these analyses we could determine whether the adaptation process is tractable, which is a necessary condition for computational plausibility. These analyses are an important addition to determine whether humans can have an adaptive toolbox. First, we presented our fast-and-frugal toolbox formalization, in which both the selector and the heuristics are formalized as fast-and-frugal trees. We analyzed the complexity of applying and adapting this type of toolbox and found that although applying the toolbox is tractable, adapting a toolbox is intractable in general. Yet when certain restrictions are placed on the environment the adaptation process can be tractable. If the adaptation processes do exploit these restrictions, it is possible to adapt the adaptive toolbox in a reasonable amount of time. Thus, it is possible that humans have an adaptive toolbox.

We first discuss our toolbox formalization in Section 5.1 and then continue with the plausibility of the restrictions in Section 5.2. Lastly, we discuss some parameters which might be interesting to explore in Section 5.3.

5.1 The toolbox formalization

We are not aware of any previous formalization of an entire toolbox. Previous research has been done on the complexity of adapting a single heuristic (Schmitt & Martignon, 2006), but no selector had been formalized, nor was the complexity of adapting other

heuristics analyzed. As such, our research is a first attempt to analyze the complexity of a complete toolbox. We chose to represent the selector as a fast-and-frugal tree because this is a fast mechanism and, as Gigerenzer et al. state, the selector should be fast and frugal in order for the entire toolbox to be fast and frugal. However, there is probably a high number of other possible selector formalizations. These other possible selection mechanisms must still be fast and frugal and will therefore have to run in (fixed-parameter) tractable time. If they are rewritable to our formalization in (fixed-parameter) tractable time, our (fp-) tractability results will hold for a toolbox with that other selector formalization. For selector formalizations are not rewritable in polynomial time to our formalization, new analyses must be done. We leave the investigation of these selector formalizations open to future research.

In our formalization, the heuristics were all formalized as fast-and-frugal trees. We proved that most of the heuristics proposed by Gigerenzer et al. can be written as fast-and-frugal trees (see Appendix A). Thus our results hold for this subset of heuristics as well. Most likely the intractability results hold when the other heuristics, the ones not included in this subset, are included in the toolbox, because these heuristics are about as simple as fast-and-frugal trees and thus do not seem to contain any computational power which can overcome the intractability of adapting the toolbox. However, we cannot be certain. We leave the investigation of the heuristics which are not rewritable as fast-and-frugal trees to future research.

5.2 Plausibility of current tractability results

We have proven that under certain restrictions it is possible to adapt the toolbox. It is important to look further into these restrictions to determine whether they are present in the environment in which humans live and whether evolution and learning mechanisms exploit them. If this holds for any of the parameter sets $\{|I|, |A|\}$, $\{|I|, |a_{max}|\}$, $\{corr, |A|\}$ or $\{corr, |a_{max}|\}$ humans could have indeed adapted an adaptive toolbox. We leave this to future research, but already have some suggestions about which of the parameters are most likely to be small in the world.

Information: $|I|$ and $corr$ The number of information pieces, $|I|$, is very large in the real world. Just by reading the front page of a single newspaper we find more than a hundred Boolean pieces of information. Therefore, parameter $|I|$ is not

restricted to some small number. A more plausible candidate is parameter $corr$, number of groups of fully correlated information, because the number of information pieces can be very large while $corr$ is small and, as Gigerenzer et al. point out, information is often correlated in the world (Czerlinski et al., 1999). In this thesis we chose to analyze only the case of full correlation of information and leave the case of partial correlation as a topic for future research. As information in the world is often not fully correlated, it is important to know whether adapting a toolbox is fp-tractable when in the parameter sets $corr$ is replaced by some parameter for partial information. We chose not to pursue this topic in this thesis, as it is uncertain how choosing only one information piece of each group will decrease the ecological rationality. The pieces of information chosen from each group are probably correlated with one another, so that there could still be redundant information. Moreover, picking one information piece from a group means ignoring the extra information that other pieces in that group provide.

Actions: $|A|$ and a_{max} The number of actions that are satisfactory in at least one situation, $|A|$, is very high in reality. For example, a high number of food types can be chosen to satisfy a person's hunger, even though choosing a specific dish is just one type of action. A multitude of other types of actions are possible, for example choosing jobs, or choosing a house to live in. Thus, parameter $|A|$ is not restricted to some small number. However, it is more likely that a_{max}, the maximum number of satisfactory actions in one situation, is restricted. For example, if a person is hungry and it is warm then a salad or an ice cream may be two of the few types of food which will satisfy him.

The parameters $corr$ and a_{max} should be further explored to determine whether they are indeed small. Afterwards, it needs to be determined whether these (possible) restrictions of the environment are exploited by the adaptation processes, i.e., whether these processes have endowed the adaptive toolbox with a limited number of information pieces and actions.

5.3 Other parameters

In this research a small set of parameters was analyzed, but there may be more restrictions which make adapting the toolbox fp-tractable. Gigerenzer et al. propose

non-compensatory information and scarce information as environmental structures in which Take The Best might work better than some other algorithms (multiple linear regression) (Czerlinski et al., 1999; Martignon & Hoffrage, 1999). An example of non-compensatory information is the ordering of a dictionary, where the first letter is more important than the second letter in ordering the words, the second letter is more important than the third, etc. The information of a letter cannot be compensated with the information from the letters which follow that one, even if the information of the other letters is combined. For example, the word 'azz' would always come before the word 'zaa' in the dictionary, even though 'zaa' has two a's instead of one. Although these structures were proposed to explain why applying a heuristic works well, they may be interesting for adapting a toolbox.

Other environmental structures that are proposed by Gigerenzer et al. are very specific. No general restrictions, but rather single rules were proposed, for example, the rule that people are attracted to large cities (see for more examples Todd & Gigerenzer, 2007). How these solitary rules can be included in our adaptation formalization is not clear.

5.4 The role of a_{max} in intractability

We found a fixed-parameter tractable algorithm which solves TOOLBOX ADAPTATION and TOOLBOX READAPTATION. From this we proved that a combination of two parameters can make the functions fixed-parameter tractable. Furthermore, we proved that TOOLBOX READAPTATION is fixed-parameter intractable for a_{max} alone. It might be that TOOLBOX ADAPTATION is fp-intractable as well when only parameter a_{max} is restricted. The results from Schmitt and Martignon (2006) suggest that this is the case. Schmitt and Martignon proved that optimal cue ordering of Take The Best is intractable. This holds when only one action (choosing one alternative) is satisfactory in any comparison between two alternatives. Take The Best and fast-and-frugal trees are closely related heuristics. The alternatives between which one can choose in Take The Best are closely related to the situations in the environment in TOOLBOX ADAPTATION, while the cues are closely related to information pieces. Therefore, it may be that TOOLBOX ADAPTATION is intractable as well when only one action is satisfactory in any situation ($a_{max} = 1$). This is an interesting topic to look into.

5.5 Some last remarks

We have taken a first step in analyzing the process of adapting a toolbox. We have determined that it is not trivial to adapt a toolbox, but that there are certain restrictions on the environment under which it can be adapted. What now remains is determining whether these restrictions on the environment actually hold and testing whether adaptive processes, evolution and/or learning, make and have made use of them. If the restrictions are not used, we have suggested some others which may be interesting to investigate.

By determining the complexity of the adaptation process, we can prove whether it is computationally plausible that the adaptive toolbox has adapted. Only if the process is tractable, can the toolbox have adapted. If the process were intractable, the time to adapt a toolbox would be so long that the adaptation process would have had to start before the earth existed, even for relatively small input sizes. Only if the adaptive toolbox has adapted can it exist as a mechanism for resource-bounded decision making in humans, because only then can a toolbox have been created.

References

Anderson, J. R., & Milson, R. (1989). Human memory: An adaptive perspective. *Psychological Review*, *96*(4), 703-719.

Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A., & Protasi, M. (1999). *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer-Verlag, Berlin Heidelberg.

Bergert, F. B., & Nosofsky, R. M. (2007). A response-time approach to comparing generalized rational and take-the-best models of decision making. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *33*(1), 107–129.

Borges, B., Goldstein, D. G., Ortmann, A., & Gigerenzer, G. (1999). Can ignorance beat the stock market? In G. Gigerenzer, P. M. Todd, & ABC Research Group (Eds.), *Simple heuristics that make us smart* (p. 59-72). Oxford University Press.

Bröder, A. (2000). Assessing the empirical validity of the "Take-the-best" heuristic as a model of human probabilistic inference. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *26*(5), 1332-1346.

Chase, V. M., Hertwig, R., & Gigerenzer, G. (1998). Visions of rationality. *Trends in cognitive sciences*, *2*(6), 206–214.

Cooper, R. (2000). Simple heuristics could make us smart; but which heuristics do we apply when? *Behavioral and Brain Sciences*, *23*(05), 746–746.

Czerlinski, J., Gigerenzer, G., & Goldstein, D. G. (1999). How good are simple heuristics? In G. Gigerenzer, P. M. Todd, & ABC Research Group (Eds.), *Simple heuristics that make us smart* (p. 97-118). Oxford University Press.

Dalrymple, G. B. (2001). The age of the earth in the twentieth century: a problem (mostly) solved. *Geological Society, London, Special Publications*, *190*(1), 205–221.

Deineko, V. G., Hoffmann, M., Okamoto, Y., & Woeginger, G. J. (2006). The traveling

salesman problem with few inner points. *Operations Research Letters*, *34*(1), 106–110.

Dieckmann, A., & Rieskamp, J. (2007). The influence of information redundancy on probabilistic inferences. *Memory & Cognition*, *35*(7), 1801–1813.

Downey, R. G., & Fellows, M. R. (1999). *Parameterized complexity*. Springer-Verlag.

Downey, R. G., & Fellows, M. R. (2013). *Fundamentals of parameterized complexity* (Vol. 4). Springer, Berlin.

Downey, R. G., Fellows, M. R., & Stege, U. (1999). Parameterized complexity: A framework for systematically confronting computational intractability. In *Contemporary trends in discrete mathematics: From dimacs and dimatia to the future* (Vol. 49, pp. 49–99).

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

Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: a guide to the theory of NP-completeness.

Gigerenzer, G. (2001). The adaptive toolbox. In G. Gigerenzer & R. Selten (Eds.), *Bounded rationality: The adaptive toolbox* (p. 37-50). MIT Press.

Gigerenzer, G. (2008). Why heuristics work. *Perspectives on Psychological Science*, *3*(1), 20–29.

Gigerenzer, G. (2015). *Simply rational: Decision making in the real world*. Oxford University Press.

Gigerenzer, G., & Brighton, H. (2009). Homo heuristicus: Why biased minds make better inferences. *Topics in Cognitive Science*, *1*(1), 107–143.

Gigerenzer, G., & Gaissmaier, W. (2011). Heuristic decision making. *Annual review of psychology*, *62*, 451–482.

Gigerenzer, G., & Goldstein, D. G. (1999). Betting on one good reason: The take the best heuristic. In G. Gigerenzer, P. M. Todd, & ABC Research Group (Eds.), *Simple heuristics that make us smart* (p. 75-95). Oxford University Press.

Gigerenzer, G., & Sturm, T. (2012). How (far) can rationality be naturalized? *Synthese*, *187*(1), 243–268.

Gigerenzer, G., & Todd, P. M. (1999a). Fast and frugal heuristics: The adaptive toolbox. In G. Gigerenzer, P. M. Todd, & ABC Research Group (Eds.), *Simple heuristics that make us smart* (p. 3-34). Oxford University Press.

Gigerenzer, G., & Todd, P. M. (1999b). *Simple heuristics that make us smart*. Oxford University Press, USA.

Goldstein, D. G., & Gigerenzer, G. (1999). The recognition heuristic: How ignorance makes us smart. In G. Gigerenzer, P. M. Todd, & ABC Research Group (Eds.), *Simple heuristics that make us smart* (p. 37-58). Oxford University Press.

Hilbig, B. E., & Richter, T. (2011). Homo heuristicus outnumbered: Comment on Gigerenzer and Brighton (2009). *Topics in Cognitive Science*, *3*(1), 187–196.

Laplace, P. S. (1951). *A philosophical essay on probabilities* (F. W. Truscott & F. L. Emory, Trans.). J. Wiley.

Martignon, L., & Hoffrage, U. (1999). Why does one-reason decision making work? A case study in ecological rationality. In G. Gigerenzer, P. M. Todd, & ABC Research Group (Eds.), *Simple heuristics that make us smart* (p. 119-140). Oxford University Press.

Martignon, L., Vitouch, O., Takezawa, M., & Forster, M. R. (2003). Naive and yet enlightened: From natural frequencies to fast and frugal decision trees. In D. Hardman & L. Macchi (Eds.), *Thinking: Psychological perspective on reasoning, judgment, and decision making* (pp. 189–211). Wiley.

Newell, B. R. (2005). Re-visions of rationality? *Trends in Cognitive Sciences*, *9*(1), 11 - 15.

Newell, B. R., & Shanks, D. R. (2003). Take the best or look at the rest? factors influencing "one-reason" decision making. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *29*(1), 53-63.

Newell, B. R., Weston, N. J., & Shanks, D. R. (2003). Empirical tests of a fast-and-frugal heuristic: Not everyone "takes-the-best". *Organizational Behavior and Human Decision Processes*, *91*(1), 82–96.

Otworowska, M., Sweers, M., Wellner, R., Uhlmann, M., Todd, W., & Van Rooij, I. (2015). How did *Homo Heuristicus* become ecologically rational? In *Proceedings of the euroasianpacific joint conference on cognitive science* (p. 324–329).

Pohl, R. F. (2006). Empirical tests of the recognition heuristic. *Journal of Behavioral Decision Making*, *19*(3), 251–271.

Rieskamp, J., & Otto, P. E. (2006). SSL: a theory of how people learn to select strategies. *Journal of Experimental Psychology: General*, *135*(2), 207–237.

Schmitt, M., & Martignon, L. (2006). On the complexity of learning lexicographic

strategies. *The Journal of Machine Learning Research*, *7*, 55–83.

Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological review*, *63*(2), 129-138.

Simon, H. A. (1990). Invariants of human behavior. *Annual review of psychology*, *41*(1), 1–20.

Todd, M., Fiddick, L., & Krauss, S. (2000). Ecological rationality and its contents. *Thinking & Reasoning*, *6*(4), 375–384.

Todd, P. M. (2001). Fast and frugal heuristics for environmentally bounded minds. In G. Gigerenzer & R. Selten (Eds.), *Bounded rationality: The adaptive toolbox* (p. 51-70). MIT Press.

Todd, P. M., & Gigerenzer, G. (2007). Environments that make us smart ecological rationality. *Current Directions in Psychological Science*, *16*(3), 167–171.

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

van Rooij, I. (2008). The tractable cognition thesis. *Cognitive science*, *32*(6), 939–984.

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

Wareham, H. T. (1999). *Systematic parameterized complexity analysis in computational phonology* (Unpublished doctoral dissertation). University of Victoria, Canada.

Wigderson, A. (2006). P, NP and mathematics a computational complexity perspective. In *Proceedings of the international congress of mathematicians: Madrid, august 22-30, 2006: invited lectures* (pp. 665–712).

Wilke, A., & Todd, P. M. (2010). Past and present environments: The evolution of decision making. *Psicothema*, *22*(1), 4–8.

Appendices

Appendix A

Heuristics as fast-and-frugal trees

Gigerenzer et al. have proposed a list of heuristics, including the recognition heuristic, the fluency heuristic, Take The Best, tallying, satisficing, 1/N, default heuristic, tit-for-tat, imitate the majority and imitate the successful (Gigerenzer, 2008; Gigerenzer & Sturm, 2012). Here we discuss in turn whether these heuristics can be rewritten as fast-and-frugal trees. All heuristics in this list can be represented as fast-and-frugal trees, except for the fluency heuristic and tallying. They require the presence of non-Boolean information and trees with non-binary branching.

The recognition and fluency heuristics work by choosing the alternative which is recognized over the unrecognized one or by choosing the alternative which is recognized faster, respectively. The recognition heuristic can be implemented as a fast-and-frugal tree. One can add a cue which states 'I have seen this alternative before'. However, the fluency heuristic contains information about how quickly an alternative is recognized, which is a specific time value. This information cannot be represented as a Boolean variable without a loss of information. As such the fluency heuristic cannot be implemented as a fast-and-frugal tree, which can only contain Boolean information.

Take The Best can be described as a fast-and-frugal tree, although some information pieces need to be combined. Take The Best compares the value of some information piece for two alternatives. For example, it compares whether two cities have a train station. This can be encoded in two cues. The first cue asks whether the first alternative has a higher value than the second (where higher means the alternative has a train station while the other does not, or it is not known for the other). The second

cue asks whether the second alternative has a higher value. The action associated with the first cue is then 'pick alternative one', while action associated with the second cue is 'pick alternative two'. However, this encoding relies on the fact that multiple information pieces can be evaluated in one cue, because two information pieces need to be compared. In our present formalization, cues are formalized as always evaluating only one information piece. Alternatively, there may be information pieces which are combinations of other information pieces. In that way, a cue would only have to evaluate one information piece. In our formalization we made no assumptions whether or not this is the case.

Tallying is used to choose one of multiple alternatives. This is done by assigning each alternative a value by adding 1 to the value if an information piece is true and subtracting 1 if the information piece is false. The alternative with the highest value is chosen. This is, to our knowledge, not representable by a fast-and-frugal tree, because the subtotal needs to be kept in memory and tallying does not stop until it has passed every cue.

Satisficing is used when one would like to know which alternative has the highest value of some variable. For example, one would like to get the house which the biggest garden. Satisficing searches through the alternatives and chooses the first one that exceeds a certain aspiration level. This could be represented as a fast-and-frugal tree, where each cue evaluates one alternative. If that alternative has a value higher than the aspiration level, that alternative is chosen.

1/N is used when resources need to be allocated to multiple alternatives. This heuristic states that resources need to be allocated equally among the N options. There is always only one action to take, and thus it can be implemented as a trivial fast-and-frugal tree, containing no cues and only one action.

The default heuristic is used for choosing between different alternatives. It simply takes the default one if there is such an alternative. This can be implemented by simply starting a fast-and-frugal tree with a cue 'is there a default action?' and if that is the case, the default action is taken.

Tit-for-tat is used for choosing between (mostly two) alternatives when interacting with someone. It was used for the prisoner's dilemma. First the cooperative choice is

picked and then thereafter the choice the other person made. This can be implemented as a fast-and-frugal tree with a first cue 'this is the first move'. The associated action is 'the cooperative action'. The other cues are 'the other person one chose alternative a last time' with associated action a, for all alternatives a.

Imitate the majority or the successful In imitate the majority the action that the majority of people has done is chosen. It can be implemented in one cue-action pair. The cue is 'the majority of people (I know) chooses action a, while the action is a. Imitate the successful works in a similar way, only the most successful person is imitated.