The Plausibility of Evolving an Adaptive Toolbox

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

Author:
Robin WELLNER
Student number: 4046323

Supervisors:
Maria OTWOROWSKA, MSc.
Dr. Iris VAN ROOIJ

November 19, 2015

Radboud Universiteit

Abstract

To explain the mechanics of decision making strategies, Gigerenzer, Todd, and the ABC research group (1999) put forth a model they called the adaptive toolbox. The tools in this adaptive toolbox are heuristics. Each of those heuristics is suited for specific situations and have been adapted to the environment by evolution.

This project explores by means of evolutionary algorithms the plausibility of the toolbox being evolvable in two separate conditions: a fixed sized population and a variable sized population that can die out. In neither condition the adaptive toolbox was found to be plausibly evolvable.

Acknowledgements

Writing this thesis has been a long and arduous journey, one that I could not have completed without the help of the following people I would like to thank.

First, my supervisors, for believing in me when I did not, as well as for their incredible patience. In particular, I would like to thank Iris for keeping me focused on the things that really matter instead of the flashy details, and Maria for spending so much time and attention on giving feedback on my thesis, as well as on regular meetings to discuss every part of my thesis as well as my well-being.

Second, Marieke, who has helped me to gain a better understanding of the adaptive toolbox, gave me thorough and valuable feedback, and has been a good friend, even when I was at my worst.

Third, my parents, for always being supportive, and for calling me every day when I needed it.

Fourth, everyone from the university who advised, counseled or guided me, whose services I relied on to not stand still.

Finally, everyone whom I have failed to mention who have helped me in completing my thesis.

Chapter 1

Introduction

Humans make decisions ranging from snap judgments and decisions made more deliberately. These decisions are based on intuition or more explicit concerns. Imagine a person choosing a spouse. Which one will this person choose? There are a number of traits one could desire in a spouse, based on the previous experience, the general attitude towards life and the character of this person. Which of those does the subject value most, and how do various desirability of traits compare? All of these considerations could influence one to make a decision.

Humans make these kinds of decisions each day, ranging from choosing a parking spot or whether or not to accept a second piece of cake, to purchasing real estate or choosing a career. Humans, however, only have limited resources for making decisions. For example, they only have limited information available about the problem domain, cognitive capacity might be tied up in other matters, and time available to solve a problem is often constrained, e.g. because it may be too late to take action if the decision-making process lasts too long. Current probabilistic theories about decision assume unrealistic time and cognitive resources, and as such do not satisfactorily explain this aspect of human cognition (Gigerenzer, 2008).

According to Gigerenzer et al. (1999), human decision making is better explained by heuristics, which are algorithms for a specific problem that trade off accuracy in solving that problem for resources or time. These heuristics often make good decisions for many situations, even though they may fail to do so for many other situations (Rooij, Wright, & Wareham, 2012). Despite this limitation, heuristics are still useful, because they can give the right output (or output that is similar enough to it) for many inputs.

The next question is what theory can explain the use of these heuristics, and how each heuristic is chosen. Gigerenzer et al. (1999) answer this by proposing a model called the adaptive toolbox which is "a Darwinian-inspired theory that conceives of the mind as a modular system that is composed of heuristics, their building blocks, and evolved capacities." This adaptive toolbox is a collection of heuristics (the "tools") which are each only useful for specific situations, and a selection mechanism (not specified by Gigerenzer et al.), that in each situation is supposed to select the most useful or appropriate tool. Gigerenzer and Todd argue that the adaptive toolbox is fast and frugal, which means that the heuristics have early stopping rules and use only a small subset of available of knowledge for the chosen heuristic. This would solve the problem of the requirement of unrealistic time and resources. These constraints apply to the selector of the heuristics as well, because a sufficiently complex selector with very simple heuristics could solve any problem perfectly. For the purpose of this thesis, a

specific formalization is used, which is explained in the next chapter.

The adaptive nature of the toolbox is both developmental and evolutionary. This thesis is concerned only with evolutionary adaptation as a mechanism for the adaptive toolbox to arise. As of yet, the plausibility of a toolbox as described by Gigerenzer et al. (1999) evolving by means of natural selection is mostly unexplored (Otworowska et al., 2015). To explore this area, this thesis investigates the plausibility of the evolvability of the adaptive toolbox, using evolutionary algorithms. This thesis provides an exploratory first step in this area by evaluating the evolvability of the toolbox under several specific conditions. This will hopefully provide future research with a framework to start from, to help find evidence either for or against the adaptive toolbox theory.

This leads into the research question: which conditions would allow an adaptive toolbox to evolve?

We will look at two basic conditions. One where the population has a fixed size, and one where the population can grow or shrink, based on the performance of each individual.

The rest of this paper contains a formalization of the adaptive toolbox based on the original proposal (Gigerenzer et al., 1999) and the formalization from Otworowska et al. (2015), a description of the methods used and the different conditions for the evolutionary algorithms, the results of the simulations and finally a discussion, which includes interpretation of the results and suggestions for future research.

Chapter 2

Formalization

This chapter describes the formalization of the various components of the adaptive toolbox and of the environment used in this thesis.

The formalization makes use of fast and frugal decision trees. Those are a kind of binary decision tree. Binary decision trees are trees that allow a choice of two options at each step, until reaching a terminal node. Fast and frugal refers to early stopping and frugal use of knowledge, and so a fast and frugal decision tree is a binary decision tree where the left child node is always a terminal node.

2.1 Environment

The environment in which the toolboxes operate is formalized as a set of discrete, independent events $E = \{e_1, e_2, \ldots, e_n\}$. A situation is a truth assignment $s : E \to \{T, F\}$ of these events. We denote the set of all possible situations by $S = \{T, F\}^n$, where S is the set of all possible n-length vectors of truth-values. This means that $|S| = 2^{|E|}$. For every situation there is a certain favored action a to perform, where a is an element of the set of all possible actions $A = \{a_1, a_2, \ldots, a_m\}$. A function $D : S \to A$ maps each situation $s \in S$ to an action $a \in A$. (Otworowska et al., 2015)

2.2 Cues

A cue is a function that maps a situation to a truth value. We denote this $C : E \times S \to \{T, F\}$. Each cue asserts the truth or falsehood of a single event, evaluating whether a specific event e is true in situation s. A cue only looks at a single event in each situation. We use c_i to denote the cue that returns T iff $s(e_i) = T$ and $\neg c_i$ to denote the cue that returns T iff $s(e_i) = F$.

2.3 Actions

Actions abstractly model the actual, physical decisions that humans make, e.g. buying a house, not eating a piece of food you just found, or prescribing your patient a specific drug. Actions are selected by heuristics.

2.4 Heuristics

We formalized heuristics in the toolbox as a *fast and frugal tree* (Martignon, Vitouch, Takezawa, & Forster, 2003; Gigerenzer & Gaissmaier, 2011). Here each heuristic is a binary decision tree, which only branches to the right. These heuristics are fast, as there is an opportunity to stop early at every step. They are also frugal, as they only use a limited subset of available knowledge of the agent and the knowledge extractable from the world. See figure 2.1 for an example. For these heuristics, the inner nodes are cues and the terminal nodes are actions.

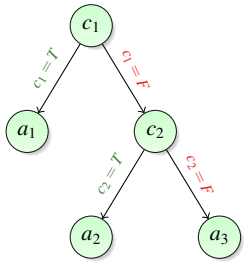


Figure 2.1: An example heuristic. a_1 is chosen if $c_1(e,s) = T$, a_2 is chosen if $c_1(e,s) = F \wedge c_2(e,s) = T$ and a_3 is chosen if $c_1(e,s) = F \wedge c_2(e,s) = F$. One could imagine c_1 to be "is it hailing outside?", c_2 to be "is it raining outside?", a_1 to be "stay indoors", a_2 to be "go outside with an umbrella" and a_3 to be "go outside without an umbrella"

2.5 Selector

A selector is a series of cues, where each cue is associated with a heuristic. Like in a fast and frugal decision tree, the cues are ordered, are always checked in that order and their left child node is a terminal node (in this case, a heuristic). The difference between a selector and a fast and frugal decision tree is that if the last cue is false, the heuristic associated to the first cue is chosen.

A further difference between a selector and a heuristic is that instead of actions, the terminal nodes of a selector are heuristics.

As an example, if we look at figure 2.2, then in the case that both c_4 and c_2 are false, the second heuristic is used to select an action.

2.6 Toolbox

A toolbox consists of a selector and the heuristics associated to each of the cues in that selector.

In a specific situation s, the action that a toolbox takes is the action chosen by applying the heuristic that is chosen by the selector in that situation.

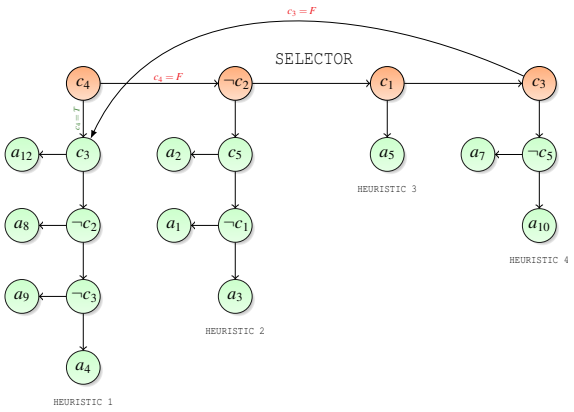


Figure 2.2: An example selector with heuristics of different sizes. Heuristic 1 is chosen if $c_4(e,s) = T \vee (c_2(e,s) = T \wedge c_1(e,s) = F \wedge c_3(e,s) = F)$. Heuristic 2 is chosen if $c_4(e,s) = F \wedge c_2(e,s) = F$. Heuristic 3 is chosen if $c_4(e,s) = F \wedge c_2(e,s) = T \wedge c_1(e,s) = T$. Heuristic 4 is chosen if $c_4(e,s) = F \wedge c_2(e,s) = T \wedge c_1(e,s) = F \wedge c_3(e,s) = T$.

Chapter 3

Methods

The formalization of the adaptive toolbox formed the basis of the simulations constructed to answer the research question: which conditions would allow an adaptive toolbox to evolve? To answer the research question we used computer simulations of the process of evolution, namely evolutionary algorithms. The performance and survival of the population in each simulation is used as an indicator for the evolvability of the adaptive toolbox.

Evolutionary algorithms are inspired by natural selection, and as such evolve a population of individuals. Some of those individuals are then selected and mutated to form the next generation. Many generations are made in this way, either for a fixed number of iterations or until a specific condition is reached. This cycle of generations is depicted in figure 3.1.

In general, there are two different kinds of evolutionary algorithms: those that are focused on finding the best solutions to a computational problem that are hard to solve with exact algorithms, and those that are focused on plausible simulations of natural selection. In this case, we are interested in the simulation aspect, which means we chose for realism over optimal solutions.

Evolutionary algorithms have a population of individuals. Each individual can be said to have a genotype and a phenotype. These terms are derived from evolutionary biology, where a phenotype is a characteristic of an individual and a genotype is the genetic sequence responsible for that characteristic. In evolutionary algorithms, the genotype is the object that is mutated according to the rules of the simulation, while the phenotype is the behavior resulting from the genotype. The phenotype is what determines the fitness of an individual. The fitness is a measure of how well the individual performs on a simulation-specific task. The higher the fitness, the better the individual did, and the more likely it is to pass its genes on to the next generation.

Populations of these individuals are grouped in discrete, separate generations. Generations are simply deleted once they have had the opportunity to pass on their genes. Each generation consists of three phases, as pictured in figure 3.1. This cycle within each generation starts with a generation of individuals, in the group labeled “children” in figure 3.1. Then the simulation performs survivor selection on them. Here, individuals with higher fitness have a larger chance of surviving. This simulates the chance of dying before reproduction. Those that are selected here, end up in the group labeled “survivors”. The precise mechanism of selection differs between the conditions and is explained in section 3.2. From the survivors, the parents of the next generation are chosen, with parent selection. Here, individuals with higher fitness have a larger

chance of having children, and have a larger number of children on average. Those that are selected here, end up in the group labeled “parents”, which gives rise to the next child generation. The precise mechanism of selection differs between conditions here as well and is again explained in section 3.2. The parents are subjected to mutation and cross-over to result in the next generation in the variation phase. Mutation involves changing a small part of the genotype of the parent individual in a (pseudo-)random way. Cross-over involves mixing the genotypes of two parent individuals, which makes it possible for multiple sets of beneficial mutated sections of the genotypes that originated in different lineages to be present in a single individual, increasing the fitness of that individual. The precise mechanisms with which individuals are mutated is explained in section 3.1.

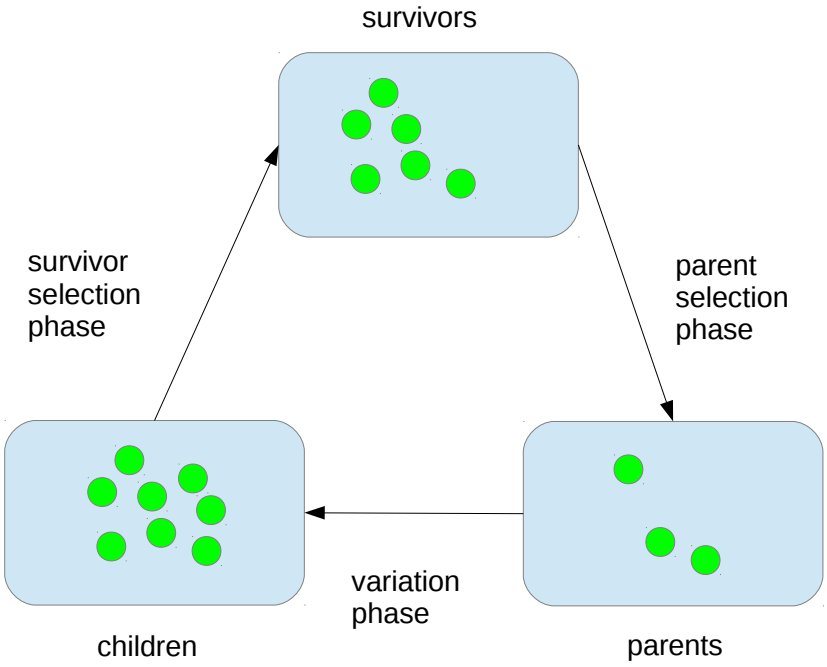


Figure 3.1: A high-level illustration of evolutionary algorithms. One generation is a full cycle, from a population of children to the next population of children.

These steps continue on until a terminating condition is reached. Examples of terminating conditions are a set number of generations have passed, the fitness of the best or average individual has surpassed a pre-chosen level, a specific solution has been reached, or the whole simulation died out. The terminating conditions used are explained in section 3.2.

3.1 Implementation

In this section, the specific set-up used for this thesis is presented. Source code used for the evolutionary algorithm can be found in appendix A.

In this set-up, each individual's genotype is a toolbox equivalent to the formalization described in chapter 2.

The heuristics were implemented as a list of tuples. The first element of such a tuple is a cue function, and the second element is an action. Actions were kept symbolic, and are simply integers. Cue functions take a list of boolean values which represent the environment and return a single boolean. The action a heuristic will take in situation s is the first action where the corresponding cue returns $True$. If all cues return $False$, the action from the *last* tuple is taken.

The selector is implemented as a list of tuples as well. The first element of such a tuple is a cue function, the same kind as the heuristics use. The second element is a heuristic. The heuristic a selector will take in situation s is the first heuristic where the corresponding cue returns $True$. If all cues return $False$, the heuristic from the *first* tuple is taken.

This implementation was chosen for its practicality in evolutionary algorithms, but is equivalent to the formalization. The example heuristic in figure 2.1 is equivalent to $[(c_1, a_1), (c_2, a_2), (c_?, a_3)]$. The last cue can be anything, since if it is reached a_3 is chosen no matter what the cue returns.

Note that in this implementation, a toolbox and its selector were represented by the same object in memory, because a selector contains references to its heuristics.

The world was implemented as a toolbox as well, containing five heuristics, where each of those heuristics contains five (cue, action) pairs. This implementation was chosen because while in the formalization the world can be an arbitrary function mapping situations to actions, by implementing the world as a toolbox we ensure it is at least possible for individuals to evolve that match the world perfectly, making no mistakes. The world had a size of five by five because that was already computationally non-trivial, but at the same time it was small enough such that a simulation not evolving an adaptive toolbox close to the world could not be caused by the complexity of the world. The actions in the world were pseudo-randomly chosen, with a uniform distribution, on each evolutionary simulation. The same was true for the cues, with the exception that the function that generated the world kept track of the cues used earlier, which were not used again for the rest of that heuristic. This exception prevents the world from having a heuristic like $[(c_1, a_1), (\neg c_1, a_2), (c_3, a_3), (c_4, a_4), (c_?, a_5)]$, which will only ever result in a_1 or a_2, and where c_3 and c_4 are not even evaluated, which means that heuristic is functionally the same as $[(c_1, a_1), (c_?, a_2)]$. Much more problematic would have been the same happening on the selector, potentially making all but the first two heuristics irrelevant. Such a world would be far more trivial to solve for toolboxes, which would make the results hard to compare with simulations where that did not occur, because those would essentially have worlds of different sizes.

In each generation, one thousand situations were pseudo-randomly chosen, with a uniform distribution. The fitness of each individual was the fraction of situations in which the toolbox belonging to that individual and the world toolbox would select the same action. Every situation an individual selected a different action than the world toolbox is called a mistake.

Mutation happened on the level of cues, heuristics and actions. In each selector, there was a fixed chance of mutation for each single (cue, heuristic) pair (see table 3.1). If a pair was to be mutated, one of the following would be chosen from a uniform

distribution: either the whole (cue, heuristic) pair would be deleted, the cue would be changed for a random cue, a new (cue, heuristic) pair would be inserted in front of the current pair (where the heuristic only contains a single (cue, action) pair), or the pair would be swapped with a random other pair in the same selector. In each heuristic, there was similarly a fixed chance of mutation for each single (cue, action) pair (see table 3.1). If a pair was to be mutated, one of the following would be chosen from a uniform distribution: either the whole (cue, action) pair would be deleted, the cue would be changed for a random cue, the action would be changed for a random action, a new (cue, action) pair would be inserted in front of the current pair, or the pair would be swapped with a random other pair in the same heuristic.

3.2 The two conditions

The two different conditions used in this thesis used a fixed size population and a variable sized population respectively. The conditions were chosen for their different strategies, as explained later.

In the fixed size populations, only parent selection was used. The method of parent selection used is called tournament selection. Tournament selection involves randomly choosing a small, fixed number of candidates (listed in table 3.1) from the old population. From those candidates, the one with the highest fitness was chosen. Two individuals were selected in that way to form a pair of parents. Then, copies of those individuals were mutated, and optionally crossed over with each other to result in two new individuals that were put in the next generation. This process was repeated until the new generation contains exactly as many individuals as the old generation.

Instead of survivor selection, every individual survived in the fixed size condition. The terminating condition was to run until a set number of generations had passed, generally 1000, but some trial simulations of 2000 were also included.

Tournament selection is an effective way to evolve solutions when that is the goal, and can serve as an "upper bound" for the evolvability of the adaptive toolbox.

In the variable size populations, only survivor selection was used. The precise method of selection was as follows: each individual had a survival chance of $(1-p_d)^m$, where p_d was the chance of death for a single mistake (see table 3.1 for the precise probability used), and m was the number of mistakes made by the individual. Populations could die out or grow to 120% of their original size, which were the terminating conditions. The 120% was chosen because the population would likely start out by shrinking, because no toolbox had evolved yet, and since the change in population size only depended on the fitness of the individuals, if the individuals started performing better, the population would start to grow and likely keep growing, so once the original population size was surpassed, it told us the population will survive. As a measure to prevent simulations with stable population sizes from never ending, there was a set number of generations in this condition as well. Every surviving individual had a fixed chance of getting either one or two children (listed in table 3.1). Since that chance did not depend on fitness, there was no parent selection.

This condition more closely mimics the expected conditions of human evolution, and can be used as a "lower bound" for the evolvability of the adaptive toolbox to compare the fixed size condition to.

Number of actions	50
Selector size of world	5
Heuristic size of world	5
Number of cues (excluding negative cues)	10
Starting population size	1000
Stopping population size	1200
Default number of generations	1000
Maximum toolbox size	30
Tournament selection size	5
Number of situations tested per generation	1000
Probability of point mutation	0.11
Probability of death for a single mistake	0.00045
Probability of two children in variable population	0.474

Table 3.1: Settings used for simulation

Chapter 4

Results

This section shows the results of the fixed size population condition and variable size population separately.

In every evolutionary simulation, for every generation, the best fitness, median fitness and population size (where relevant) were recorded.

The fixed size population condition simulations ran for a fixed number of generations and population size remained constant, while the variable size population condition simulations ended when the population died out or exploded. Therefore, different metrics are relevant for the different conditions, although both conditions can be directly compared to each other when it comes to the best and mean fitness in the population over time.

4.1 Fixed size population

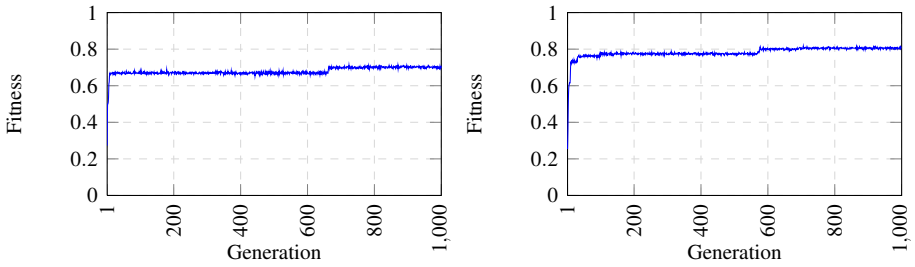
In figure 4.1 the progressions of fitness over time are shown for the different simulations. They show the fitness of the fittest individual in each generation. The horizontal axis is the generation number, and the vertical axis the fitness. A fitness of 0 means the individual got every situation wrong, and one of 1 means they got every situation right.

Here we see a quick growth in fitness, from near 0.25 to 0.65 or 0.78, after which the growth mostly stops, except for one or two minor jumps further on. Median fitness closely follows the best fitness, mirroring most jumps in best fitness.

4.2 Variable size population

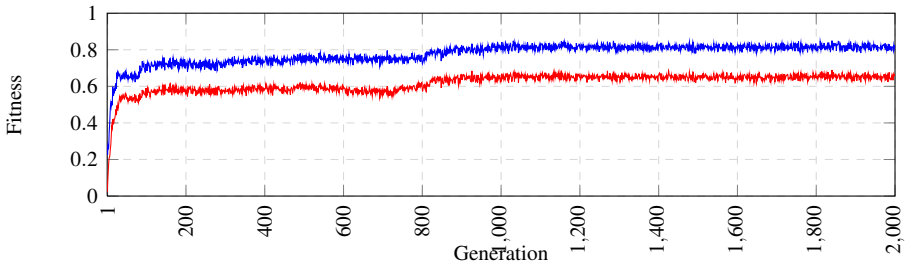
In figure 4.2, the population size over time for the different simulations is shown. This includes both surviving populations and ones that died out. On the horizontal axis the generation number is shown, and the vertical axis the number of individuals in that generation. The red line is the number of individuals before survivor selection and the blue line the number of individuals after survivor selection.

Despite large fluctuations in population sizes, a clear trend appears, where population tends to shrink, up to the point where individuals evolve with a good enough toolbox to sustain the population, at which point the population starts growing until the simulation stops. The exception is when the population dies out before such individuals evolved.

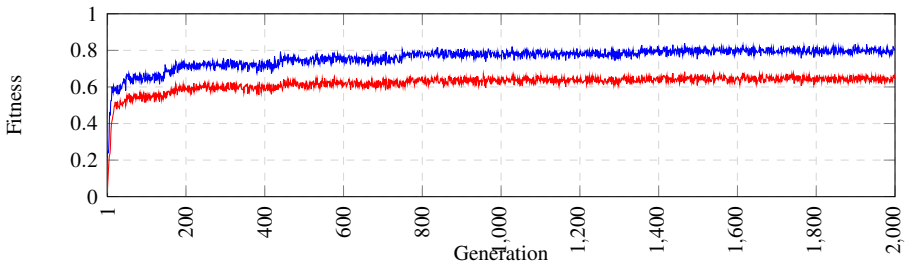


(a) Running for 1000 generations

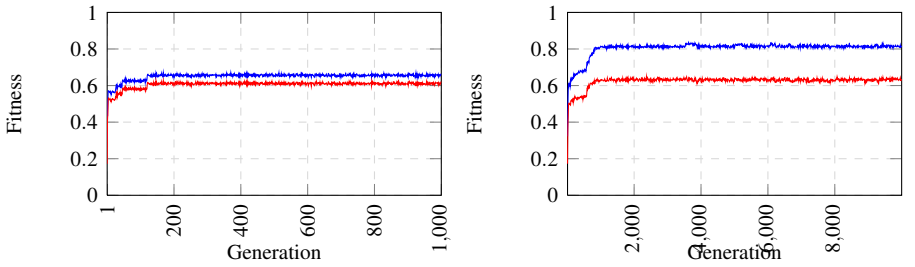
(b) Running for 1000 generations



(c) Running for 2000 generations



(d) Running for 2000 generations



(e) Running for 1000 generations

(f) Running for 10000 generations

Figure 4.1: Fitness per generation in simulations with fixed population size. The blue line shows the best fitness in a generation, and the red line shows the mean fitness in a generation. The fitness in every generation fluctuated, due to a different sample of situations being used to measure fitness in each generation. Both best and mean fitness shoot upwards in the first couple of generations of all these simulations, slowing down to a more moderate growth until around generation 100, after which both best and mean fitness remain relatively stable, with the exception of a few advantageous mutations causing minor improvements in fitness. The mean fitness mirrors major changes in best fitness, smaller improvements are not recognizable.

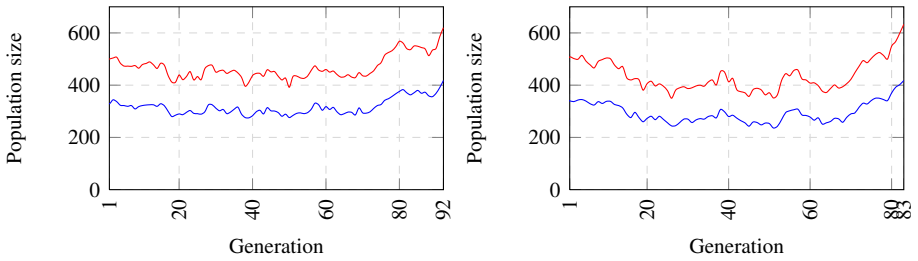
The exception seems to be the simulation pictured in figure 4.2e, which seems to reach a stable population around generation 400, but doesn't start to show any strong growth until after generation 850.

In figure 4.3, the progressions of fitness over time are shown for the surviving populations. They show the fitness of the fittest individual in each generation. The horizontal axis is the generation number, and the vertical axis the fitness.

These figures show a mostly constant level of fitness, except for regular fluctuations. There is rarely any strong growth in fitness like with the fixed size condition. The changes in fitness were subtle, and subtle changes turn out the only thing the population needs to grow instead of shrink. Especially compare the minor changes in figure 4.3e after generation 850 with the enormous change in population growth in figure 4.2e.

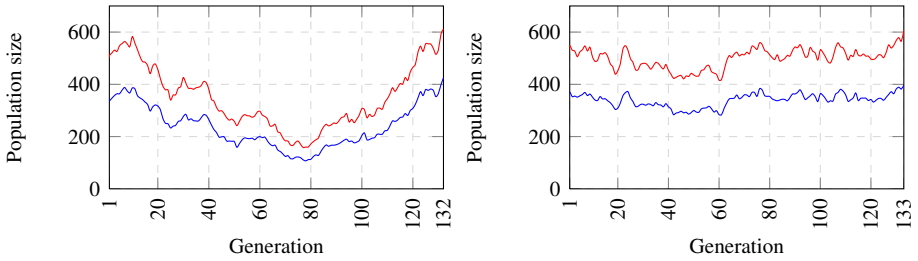
In figure 4.4, the progressions of fitness over time are shown for the populations that died out. They show the fitness of the fittest individual in each generation. The horizontal axis is the generation number, and the vertical axis the fitness.

Instead of the mostly constant fitness of the populations that grew and survived, the populations that died out actually showed fitness decreasing over time, so it seems the smaller a population is, the less it is able to grow and the better chances deleterious mutations have of surviving in the population.



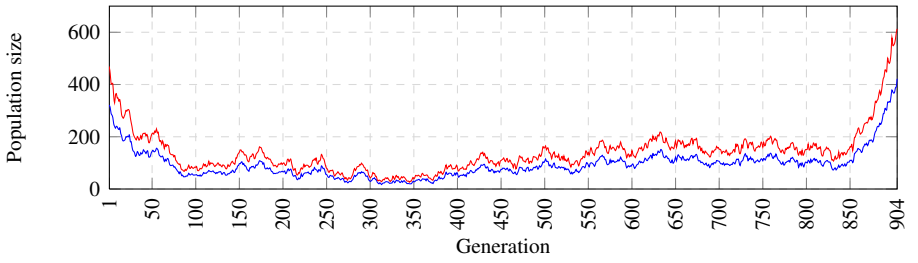
(a) Running for 92 generations, surviving

(b) Running for 83 generations, surviving

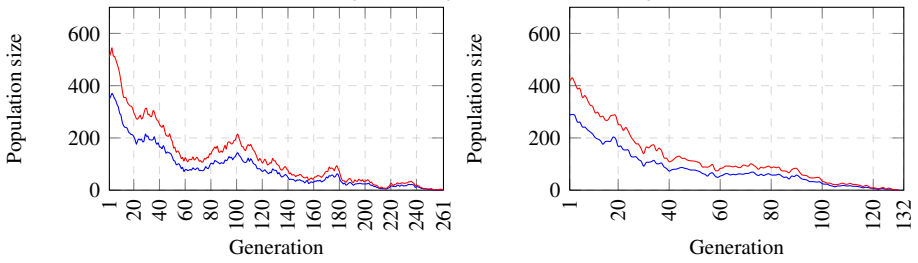


(c) Running for 132 generations, surviving

(d) Running for 133 generations, surviving



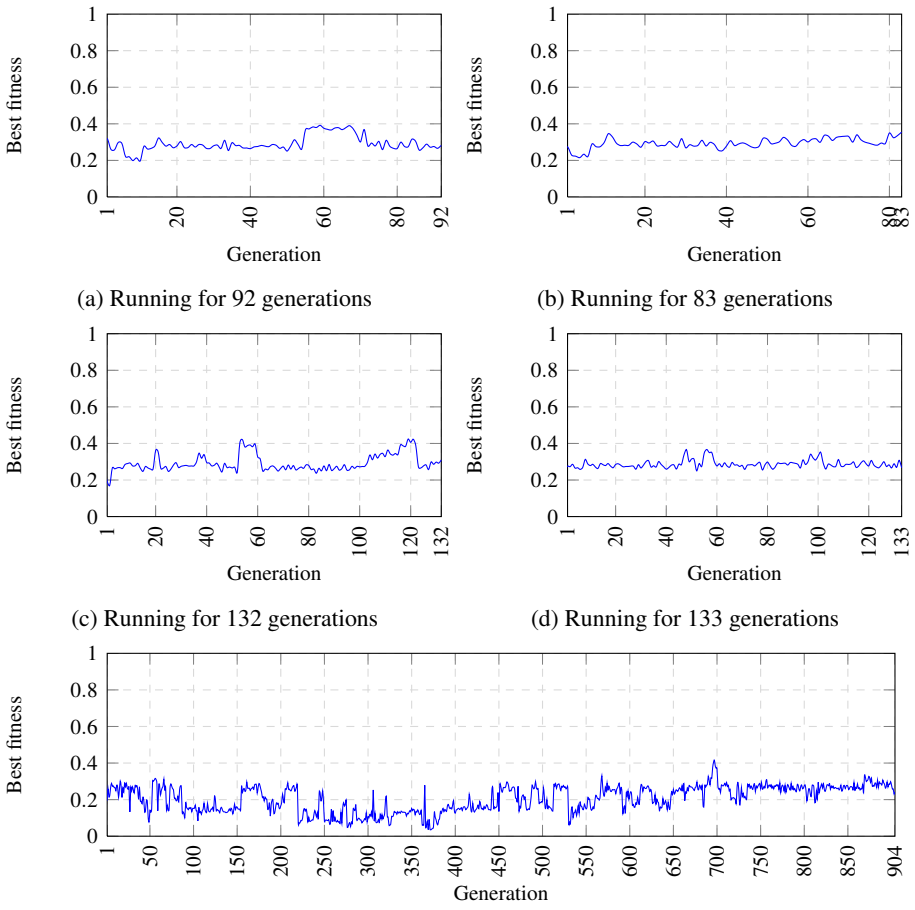
(e) Running for 904 generations, surviving



(f) Running for 261 generations, dying

(g) Running for 132 generations, dying

Figure 4.2: Population size per generation in simulations with variable population size. The red line shows the population size before survivor selection, while the blue line shows the population size after survivor selection. The population sizes first shrink for all simulations, but recover later for some of the simulations, while others die out completely.



(a) Running for 92 generations

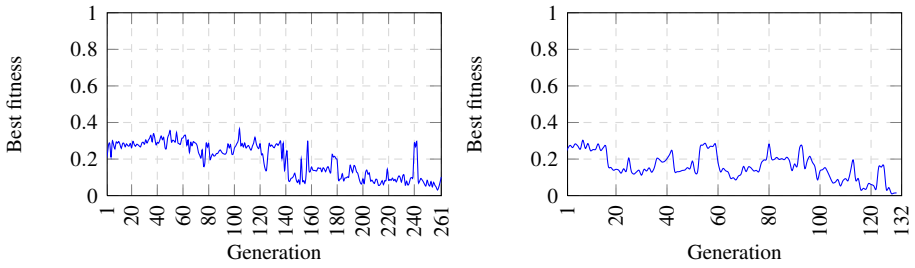
(b) Running for 83 generations

(c) Running for 132 generations

(d) Running for 133 generations

(e) Running for 904 generations

Figure 4.3: Fitness per generation in simulations with variable population size that survived. The blue line shows the fitness of the fittest individual within each generation. Unlike in the fixed size population, there is no visible growth in fitness. The fitness consistently fluctuated wildly, but mostly staying between 20% and 40%.



(a) Running for 261 generations

(b) Running for 132 generations

Figure 4.4: Fitness per generation in simulations with variable population size that died out. The blue line shows the fitness of the fittest individual within each generation. Like in the surviving populations, the best fitness in these simulations shows no sign of growth, but instead fluctuates wildly. Unlike the surviving population, the best fitness is actually *declining* over time.

Chapter 5

Discussion

This thesis started out by discussing human decision making, and the adaptive toolbox as put forward by Gigerenzer et al. (1999). To investigate the question "which conditions would allow an adaptive toolbox to evolve?", we used evolutionary algorithms with two conditions, one where the population has a fixed size and one where the population has a variable size.

Looking at the variable size populations, the toolboxes perform poorly. The results show that even the best toolboxes rarely achieve a fitness higher than 40%, most of the time even hovering below 30%. Furthermore, the difference between a population dying out and surviving is small. That means the way of selecting survivors used here does not make it plausible to evolve an adaptive toolbox.

Fixed size populations do better, and regularly result in toolboxes with fitness levels of over 80%, far better than the 50% watershed point where they are more often right than wrong. On the other hand, they never get close to 100%, while the world has been deliberately kept simple to make a perfect toolbox possible.

The results of the simulations under both of these conditions show that it is implausible that evolution alone would produce adaptive toolboxes. Our simulations have shown that surviving variable size populations can exist without crossing the watershed 50% fitness. These results align well with the formal argument from Otworowska et al. (2015), where the expected number of generations needed to produce a toolbox grows exponentially. Even for 10 different cues and 50 different actions, an expected number of two million generations would be needed to evolve an individual with a fitness of at least 50%. For a higher number of cues and actions, the expected number of generations would be much larger. Additionally, the number of generations needed to evolve a whole population of well-performing individuals is much larger, since individuals still survive with a much lower fitness, and the high odds of mutation and crossover leading to offspring with a low fitness. Mutation and crossover are applied every generation, so even if there is an individual with 50% fitness or higher, it is still very likely that individual's offspring has a much lower fitness, but still manages to survive and procreate.

In addition, the differences between the results of the conditions show the need for a stable population size in order to come close to evolving adaptive toolboxes. If we take the existence of the adaptive toolbox as a given, that would restrict the conditions in which an adaptive toolbox can arise to a population that can remain stable while the toolbox evolves. If this condition is not met, then that would provide evidence against the plausibility of the adaptive toolbox modeling human decision making processes

arising from purely phylogenetic processes.

This does not mean the adaptive toolbox cannot be a plausible model for human decision making. Even with our findings, Otworowska et al. (2015) makes the argument that adaptive toolboxes could still arise due to ontogenetic processes (i.e. learning and development), or even a combination of both ontogenetic and phylogenetic processes, where evolution could have produced learning mechanisms that can produce adaptive toolboxes within a single lifetime. Ontogenetic processes are able to actively search for possible solutions, unlike phylogenetic processes, e.g. by building a model of the environment, and using that model to construct a toolbox in a way that makes it perform better than evolution alone could have. However, this requires non-frugal learning mechanisms to explain how adaptive toolboxes arise. As such, resolving this would be an important matter for future research into resource-bounded decision making.

This thesis has explored which conditions are required for adaptive toolboxes to evolve, the model put forth by Gigerenzer et al. (1999) to explain the mechanics of decision making strategies. Our results indicate that it is not plausible for adaptive toolboxes to evolve under conditions solely depending on phylogenetic processes. Which conditions apply to real life is an important question to resolve for future research.

References

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

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

Gigerenzer, G., Todd, P. M., & the ABC research group. (1999). *Simple heuristics that make us smart*. 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*. Wiley.

Otworowska, M., Sweers, M., Wellner, R., Uhlmann, M., Wareham, T., & Rooij, I. van. (2015). How did *Homo Heuristicus* become ecologically rational?

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

Sweers, M. (2015). *Adapting the adaptive toolbox*. Unpublished master's thesis, Radboud Universiteit.

Appendices

Appendix A

Code listing

A.1 Fixed population size

```
from random import choice, random, randrange, sample
from copy import deepcopy
from sys import argv

# Constant parameters for simulation
K_size = 10
nactions = 50
NSELECTED = 200
NPOPULATION = 1000
KTRIES = 1000
WBT_SIZE = 5
WBT_HEURISTIC_SIZE = 5
NTOURNAMENT = 5
POINT_MUTATION = 0.11
MAX_SIZE = 30

# Construct cues
cues = ([lambda K, i=i: K[i] for i in range(K_size)] +
        [lambda K, i=i, neg=True: not K[i] for i in range(K_size)])

# Construct helper structure for the world generation process
pieces_of_knowledge = {}
for i in range(K_size):
    s = {cues[i], cues[i + K_size]}
    pieces_of_knowledge[cues[i]] = s
    pieces_of_knowledge[cues[i + K_size]] = s

# Construct the actions
actions = range(nactions)

# Evaluating a toolbox in a certain situation returns the action that
# toolbox associates with that situation
```

```
def eval_toolbox(t, K):
    for c, h in t:
        if c(K):
            return eval_heuristic(h, K)
    return eval_heuristic(t[0][1], K)

def eval_heuristic(h, K):
    for c, a in h:
        if c(K):
            return a
    return h[-1][1]

# Given two toolboxes and a situation, do both toolboxes in that
# situation evaluate to the same action?
def same_action(t1, t2, K):
    return eval_toolbox(t1, K) == eval_toolbox(t2, K)

# Different types of mutations for selector
_t_mutations = ['insert', 'delete', 'change_cue', 'swap']

# Different types of mutations for heuristic
_h_mutations = ['insert', 'delete', 'change_cue', 'change_action',
                'swap']

# Perform mutations on a selector
def _mutation_selector(t):
    # make a copy of the original toolbox, to prevent
    # double mutation when a single toolbox is selected twice
    t = deepcopy(t)
    for i, (c, h) in enumerate(t):
        if chance(POINT_MUTATION):
            mut = choice(_t_mutations)
            if mut == 'insert':
                if toolbox_size(t) < MAX_SIZE:
                    t.insert(i, (choice(cues),
                                 [(choice(cues), choice(actions))]))
            elif mut == 'delete':
                if len(t) > 1:
                    t.pop(i)
            elif mut == 'change_cue':
                t[i] = (choice(cues), h)
            elif mut == 'swap':
                j = randrange(len(t))
                t[i], t[j] = t[j], t[i]
    return t

# Perform mutations on a heuristic
def _mutation_heuristic(t, h):
    for i, (c, a) in enumerate(h):
        if chance(POINT_MUTATION):
```

```
            mut = choice(_h_mutations)
            if mut == 'insert':
                if toolbox_size(t) < MAX_SIZE:
                    h.insert(i, (choice(cues), choice(actions)))
            elif mut == 'delete':
                if len(h) > 1:
                    h.pop(i)
            elif mut == 'change_cue':
                h[i] = (choice(cues), a)
            elif mut == 'change_action':
                h[i] = (c, choice(actions))
            elif mut == 'swap':
                j = randrange(len(h))
                h[i], h[j] = h[j], h[i]

# Perform mutations on a toolbox
def mutate(t):
    t = _mutation_selector(t)
    for c, h in t:
        _mutation_heuristic(t, h)
    return t

def _crossover(t1, t2):
    i1 = randrange(len(t1))
    i2 = randrange(len(t2))
    return t1[:i1] + t2[i2:], t2[:i2] + t1[i1:]

def crossover(t1, t2):
    if chance(.1):
        return _crossover(t1, t2)
    else:
        return t1, t2

# Helper function that returns true with a chance of p
def chance(p):
    return random() < p

# The fitness of a toolbox is the fraction of situations where it
# evaluates to the same action as the world
def fitness(t, world, situations):
    return (sum(1.0 for K in situations if same_action(t, world, K)) /
            len(situations))

def toolbox_size(t):
    return sum(len(h) for c, h in t)

# Tournament selection in which the best of NTOURNAMENT is chosen
def select_parent(generation):
    return generation[max(sample(range(NPOPULATION), NTOURNAMENT))]
```

```
# Prepares the selection by choosing a list of situations
# and sorting the generation, with the best at the end
def selection_phase(generation, world, recording, i, fdr):
    situations = [[chance(.5) for _ in range(K_size)]
                  for _ in range(KTRIES)]
    generation.sort(key=lambda t: fitness(t, world, situations))
    recording.append((fitness(generation[-1], world, situations),
                      fitness(generation[0], world, situations),
                      sum(fitness(toolbox, world, situations)
                          for toolbox in generation) / len(generation),
                      fitness(generation[len(generation) // 2],
                              world, situations)))
    return generation

# Select parents from the old generation, performs cross-over and
# mutation and returns the new generation
def variation_phase(generation):
    for i in range(NPOPULATION//2):
        t1, t2 = crossover(select_parent(generation),
                           select_parent(generation))
        yield mutate(t1)
        yield mutate(t2)

def next_generation(generation, world, recording, i, ftb, fdr):
    generation = selection_phase(generation, world, recording, i, fdr)
    return list(variation_phase(generation))

# Return a random toolbox with fixed dimensions to be used as a world
# That world has these properties:
# * the selector chooses between WBT_SIZE heuristics
# * each heuristic consists of WBT_HEURISTIC_SIZE - 1 cues and
#   WBT_HEURISTIC_SIZE actions
# * cues and actions are chosen at random with replacements
def world_building_toolbox():
    cues_left = set(cues)
    return [generate_world_heuristic(cues_left) for _ in range(WBT_SIZE)]

def generate_world_heuristic_item(cues_left):
    c = choice(list(cues_left))
    cues_left -= pieces_of_knowledge[c]
    return c, choice(actions)

def generate_world_heuristic(cues_left):
    c = choice(list(cues_left))
    cues_left -= pieces_of_knowledge[c]
    cues_left = cues_left.copy()
    return c, [generate_world_heuristic_item(cues_left)
               for _ in range(WBT_HEURISTIC_SIZE)]

# Generate a simple toolbox, used in the first generation, to
```

```
# provide a start for the toolbox to evolve
def primitive_toolbox():
    return [(choice(cues),
             [(choice(cues), choice(actions))
              for _ in range(randrange(1, 4))])
             for _ in range(3)]

def evolve():
    world = world_building_toolbox()
    generation = [primitive_toolbox() for _ in range(NPOPULATION)]
    recording = []
    name = argv[1] if len(argv) > 1 else 'results'
    ngen = int(argv[2]) if len(argv) > 2 else 1000
    try:
        ftb = open('results/toolbox_{}.txt'.format(name), 'w')
        fr = open('results/ga_{}.csv'.format(name), 'w')
        fdr = open('results/ga_distr_{}.csv'.format(name), 'w')
        print('world:', file=ftb)
        pretty_print_toolbox(world, ftb)
        print('generation,best_fitness,worst_fitness,mean_fitness,'
              'median_fitness', file=fr)
        for i in range(ngen):
            generation = next_generation(generation, world, recording,
                                         i, ftb, fdr)
            best, worst, mean, median = zip(*recording)
            print('{},{},{},{},{}'.format(i, sum(best)/len(best),
                                          sum(worst)/len(worst),
                                          sum(mean)/len(mean),
                                          sum(median)/len(median)),
                  file=fr)
            recording = []
    finally:
        ftb.close()
        fr.close()
        fdr.close()

if __name__ == '__main__':
    evolve()
```

A.2 Variable population size

```
from random import choice, random, randrange, sample
from copy import deepcopy
from sys import argv

# Constant parameters for simulation
K_size = 10
nactions = 50
NSELECTED = 200
```

```
NPOPULATION = 1000
ENDGAME = 1200
KTRIES = 1000
WBT_SIZE = 5
WBT_HEURISTIC_SIZE = 5
NTOURNAMENT = 5
POINT_MUTATION = 0.11
MAX_SIZE = 30

P_DEATH = 0.00045
P_SURVIVAL = 1 - P_DEATH

# Construct cues
cues = ([lambda K, i=i: K[i] for i in range(K_size)] +
        [lambda K, i=i, neg=True: not K[i] for i in range(K_size)])

# Construct helper structure for the world generation process
pieces_of_knowledge = {}
for i in range(K_size):
    s = {cues[i], cues[i + K_size]}
    pieces_of_knowledge[cues[i]] = s
    pieces_of_knowledge[cues[i + K_size]] = s

# Construct the actions
actions = range(nactions)

# Evaluating a toolbox in a certain situation returns the action that
# toolbox associates with that situation
def eval_toolbox(t, K):
    for c, h in t:
        if c(K):
            return eval_heuristic(h, K)
    return eval_heuristic(t[0][1], K)

def eval_heuristic(h, K):
    for c, a in h:
        if c(K):
            return a
    return h[-1][1]

# Given two toolboxes and a situation, do both toolboxes in that
# situation evaluate to the same action?
def same_action(t1, t2, K):
    return eval_toolbox(t1, K) == eval_toolbox(t2, K)

# Different types of mutations for selector
_t_mutations = ['insert', 'delete', 'change_cue', 'swap']

# Different types of mutations for heuristic
_h_mutations = ['insert', 'delete', 'change_cue', 'change_action',
```

```
                'swap']

# Perform mutations on a selector
def _mutation_selector(t):
    # make a copy of the original toolbox, to prevent
    # double mutation when a single toolbox is selected twice
    t = deepcopy(t)
    for i, (c, h) in enumerate(t):
        if chance(POINT_MUTATION):
            mut = choice(_t_mutations)
            if mut == 'insert':
                if toolbox_size(t) < MAX_SIZE:
                    t.insert(i, (choice(cues),
                                 [(choice(cues), choice(actions))]))
            elif mut == 'delete':
                if len(t) > 1:
                    t.pop(i)
            elif mut == 'change_cue':
                t[i] = (choice(cues), h)
            elif mut == 'swap':
                j = randrange(len(t))
                t[i], t[j] = t[j], t[i]
    return t


# Perform mutations on a heuristic
def _mutation_heuristic(t, h):
    for i, (c, a) in enumerate(h):
        if chance(POINT_MUTATION):
            mut = choice(_h_mutations)
            if mut == 'insert':
                if toolbox_size(t) < MAX_SIZE:
                    h.insert(i, (choice(cues), choice(actions)))
            elif mut == 'delete':
                if len(h) > 1:
                    h.pop(i)
            elif mut == 'change_cue':
                h[i] = (choice(cues), a)
            elif mut == 'change_action':
                h[i] = (c, choice(actions))
            elif mut == 'swap':
                j = randrange(len(h))
                h[i], h[j] = h[j], h[i]

# Perform mutations on a toolbox
def mutate(t):
    t = _mutation_selector(t)
    for c, h in t:
        _mutation_heuristic(t, h)
    return t
```

```
def _crossover(t1, t2):
    i1 = randrange(len(t1))
    i2 = randrange(len(t2))
    return t1[:i1] + t2[i2:]

def _crossover2(t1, t2):
    for (c1, h1), (c2, h2) in zip(t1, t2):
        yield choice([c1, c2]), _crossover(h1, h2)
    if chance(.5):
        for item in t1[len(t2):]:
            yield item
        for item in t2[len(t1):]:
            yield item

def crossover(t1, t2):
    if chance(.1):
        if chance(.5):
            return _crossover(t1, t2)
        else:
            return list(_crossover2(t1, t2))
    else:
        return t1

# Helper function that returns true with a chance of p
def chance(p):
    return random() < p

# The fitness of a toolbox is the fraction of situations where it
# evaluates to the same action as the world
def fitness(t, world, situations):
    return (sum(1.0 for K in situations if same_action(t, world, K)) /
            len(situations))

def toolbox_size(t):
    return sum(len(h) for c, h in t)

def mistakes(situations, t, world):
    return sum(1.0 for K in situations
               if not same_action(t, world, K))

# Prepares the selection by choosing a list of situations
# and sorting the generation, with the best at the end
def selection_phase(generation, world, recording, i, fdr, name):
    situations = [[chance(.5) for _ in range(K_size)]
                  for _ in range(KTRIES)]
    generation.sort(key=lambda t: fitness(t, world, situations))
    recording.append(','.join([str(fitness(generation[-1],
                                           world, situations)),
                               str(fitness(generation[0],
```

```
                                    world, situations)),
                                str(sum(fitness(toolbox, world, situations)
                                        for toolbox in generation)
                                 / len(generation)),
                                str(fitness(generation[len(generation) // 2],
                                            world, situations))]))
    return [t for t in generation
            if chance(P_SURVIVAL ** mistakes(situations, t, world))]

# Select parents from the old generation, performs cross-over and mutation
# and returns the new generation
def variation_phase(generation):
    for p in generation:
        for _ in range(choice((1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                               2, 2, 2, 2, 2, 2, 2, 2, 2))):
            yield mutate(crossover(p, choice(generation)))

def next_generation(generation, world, recording, i, ftb, fdr, name):
    generation = selection_phase(generation, world, recording, i, fdr, name)
    recording[-1] = recording[-1] + ',' + str(len(generation))
    return list(variation_phase(generation))

# Return a random toolbox with fixed dimensions to be used as a world
# That world has these properties:
# * the selector chooses between WBT_SIZE heuristics
# * each heuristic consists of WBT_HEURISTIC_SIZE - 1 cues and
#   WBT_HEURISTIC_SIZE actions
# * cues and actions are chosen at random with replacements
def world_building_toolbox():
    cues_left = set(cues)
    return [generate_world_heuristic(cues_left)
            for _ in range(WBT_SIZE)]

def generate_world_heuristic_item(cues_left):
    c = choice(list(cues_left))
    cues_left -= pieces_of_knowledge[c]
    return c, choice(actions)

def generate_world_heuristic(cues_left):
    c = choice(list(cues_left))
    cues_left -= pieces_of_knowledge[c]
    cues_left = cues_left.copy()
    return c, [generate_world_heuristic_item(cues_left)
               for _ in range(WBT_HEURISTIC_SIZE)]

# Generate a simple toolbox, used in the first generation, to
# provide a start for the toolbox to evolve
def primitive_toolbox():
    return [(choice(cues),
            [(choice(cues), choice(actions))
```

```
            for _ in range(randrange(1, 4))])
        for _ in range(3)]

def evolve():
    world = world_building_toolbox()
    generation = [primitive_toolbox() for _ in range(NPOPULATION)]
    recording = []
    name = argv[1] if len(argv) > 1 else 'results'
    ngen = int(argv[2]) if len(argv) > 2 else 1000
    try:
        ftb = open('results/toolbox_varpop_{}.txt'.format(name), 'w')
        fr = open('results/ga_varpop_{}.csv'.format(name), 'w')
        fdr = open('results/ga_distr_varpop_{}.csv'.format(name), 'w')
        print('world:', file=ftb)
        pretty_print_toolbox(world, ftb)
        for i in range(ngen):
            generation = next_generation(generation, world, recording,
                                         i, ftb, fdr, name)
            recording[-1] = recording[-1] + ',' + str(len(generation))
            if not generation or len(generation) >= ENDGAME:
                break
    finally:
        print('\n'.join(recording), file=fr)
        ftb.close()
        fr.close()
        fdr.close()

if __name__ == '__main__':
    evolve()
```