Simple Trees in Complex Forests: Growing Heuristics through Approximate Bayesian Computation

Eric Schulz

Department of Experimental Psychology

University College London

Maarten Speekenbrink

Department of Experimental Psychology

University College London

Björn Meder

Center for Adaptive Behavior and Cognition

Max Planck Institute for Human Development, Berlin

Abstract

Heuristic models of decision making have historically been unable to explain how elaborate heuristic strategies can emerge from smaller building blocks. We propose Approximately Bayesian Computed Take-The-Best (TTB-ABC) as a solution to this problem. Using this algorithm, we show how non-compensatory strategies such as Take-The-Best can emerge from smaller probabilistically updated building blocks without relying on heavy computation. Based on a self-reinforcing sampling scheme, different non-compensatory building blocks are proposed and, over time, tree-like heuristics emerge. We demonstrate that TTB-ABC is able to recover a data set that was generated by standard Take-The-Best, leads to sensible inferences about cue validities, can outperform Take-The-Best, and trades-off performance and computational complexity explicitly. This constitutes as a first proof of concept of how a tool from the heuristic tool box can emerge from smaller building blocks.

Simple Trees in Complex Forests: Growing Heuristics through Approximate Bayesian Computation

Introduction

A main problem within the literature of heuristic decision making can be called the *strategy emergence problem* and it reads as follows: How can a heuristic strategy evolve from smaller building blocks over time? Take the heuristic *Take-The-Best* (henceforth TTB) as an example (Gigerenzer & Goldstein, 1999): TTB is a non-compensatory strategy to choose between two options (Hoffrage, 1999). Non-compensatory means that, if one orders the used variables in terms of their importance, the first variable has a stronger effect than all the other variables combined, the second variable has a stronger effect than all the remaining variables combined, and so forth. In fact, it is postulated that once a variable is able to decide between the options, the other variables are not even looked up any further. Consider the decision about which of two cities has a larger population size and let us assume that the two compared cities can be described by answering the following yes-no-questions:

$\mathcal{Q}1$**:** Is it a capital city?

$\mathcal{Q}2$**:** Does it have an airport?

$\mathcal{Q}3$**:** Does it have a major league football team?

$\mathcal{Q}4$**:** Is it a trade fair town?

In order to compare two cities, one only has to look up the questions Q1-Q4 in the above order. As soon as one criterion weighs favorably for one city over the other, the comparison stops and the response is that the city with the favorable cue is bigger. For example, comparing London and Manchester, one could ask “Is one of them a capital city?”, which is true for London, but false for Manchester and at which point the comparison stops and the final response is to say that London is bigger. Comparing Manchester and Oxford, one would ask Q1, which is false for both; then Q2, true for both; and then Q3, which is only true for Manchester. Thus, the final response is to say

that Manchester is bigger than Oxford. If none of the questions can distinguish between the options, for example, comparing London and Paris, then the final remedy would be to guess. Although relatively simple, TTB has been shown to perform surprisingly well in many different classification tasks such as predicting city sizes, professors' salaries, high school drop out rates, and homelessness of US-American cities (Gigerenzer & Todd, 1999), to name but a few. Yet, TTB also provides an almost paradigmatic example of the *strategy emergence problem*: this problem arises when taking the adaptive toolbox metaphor literally (Gigerenzer & Gaissmaier, 2011). The adaptive toolbox is normally assumed to contain different building blocks that loosely fall within 3 categories: *Search rules* dictate how a decision maker looks up information, for example sequentially from Q1 to Q4. *Stopping rules* tell us when to stop the search, for example as soon as one of the 4 question is answered with a yes for one but not for the other city. Lastly, *decision rules* make the actual decision after the search has been stopped, for example, indicating that the city with a major football league team is bigger than the city without one. If we now look at these three rules, we see that all of the building blocks, that is the search, the stopping, and the decision rule are already given. The only thing a decision maker has to learn over time is the question order, also sometimes referred to as cue validity. Commonly, TTB's cue validity is determine by how well the different questions can predict the observed outcome (Bröder, 2000). This definition, however, completely ignores the question of how a decision maker knows that she can apply exactly these three rules in a given scenario. Contrarily, in keeping with the original definition of the heuristic toolbox, one would assume that a decision maker approaches a problem with a set of search, stopping, and decision rules in mind and –over time– learns how to combine these rules successfully. This combination, in the end, might result in a non-compensatory model that looks like TTB. Even more, one would assume that there exist smaller rules that together compose increasingly bigger heuristics such as how to choose a particular strategy. Therefore, we think that a viable theory of heuristic decision making has to explain the dynamics of the heuristic toolbox; it has to explain how different building blocks are combined and heuristic strategies emerge.

However, none of the current approaches towards heuristic decision making addresses this problem explicitly . It is this explanatory dearth that motivates this paper.

In what follows, we will introduce and define TTB-ABC (Approximately Bayesian Computed Take-The-Best), a novel algorithm that can learn non-compensatory decision strategies on the fly by using approximate Bayesian computation of sampled building blocks. This approach does not need complicated likelihood manipulations, but rather depends on the acceptance and rejection of simple mental simulations. When heuristic subunits are fed into this algorithm, a powerful yet elegant machinery is created that learns non-compensatory strategies by mimicking a Bayesian reinforcement learning algorithm. It will be shown that this approach can recover structures that are solely based on traditional TTB, leads to sensible inferences about cue validities, outperforms traditional TTB-models, and can be utilized to trade-off performance and computational complexity explicitly. To the best of our knowledge, this provides a first ever proof of concept of how a heuristic can emerge from smaller building blocks.

Approximate Bayesian Computation

The goal of Bayesian inference is to calculate the posterior of a parameter θ, given some prior $\pi(\theta)$ and having observed some data $\mathcal{D}$ (Lindley, Lindley, & Lindley, 1972).

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)\pi(\theta)}{\int_\theta p(\mathcal{D}|\theta)\pi(\theta)\mathrm{d}\theta} \tag{1}$$

$$\propto p(\mathcal{D}|\theta)\pi(\theta) \tag{2}$$

This means that the posterior depends on the prior belief $\pi(\theta)$ about θ via the likelihood of the observed data $p(\mathcal{D}|\theta)$ under that particular prior. Bayesian inference can provide a strong tool to cope with uncertainty as it models a person's subjective beliefs of the world as a mixture of prior assumptions and incoming data. It is therefore not surprising that it has been used to model cognition in many different domains (Oaksford & Chater, 2007). However, Bayesian inference also suffers from potentially

high computational costs and, occasionally, from analytical intractability (Dagum & Luby, 1993). For many of those problems, sophisticated sampling schemes have been developed that approximate the posterior distribution by the means of Monte Carlo Markov chains (Gilks, 2005). This and an increase in computational power has lead to what has been referred to as the "Bayesian Revolution", the increased application of of Bayesian models, over the last decades (Brooks, 2003). It has even be claimed that some classic psychological effects such as anchoring or the availability bias can be explained by sampling-based models similar to MCMC (Lieder, Griffiths, & Goodman, 2012). Here, we want to focus on a strategy that sounds simple at first, but turns out to be an elegant approach to approximate posterior inference. This strategy is called *Approximate Bayesian Computation* (Turner & Van Zandt, 2012). Approximate Bayesian Computation (ABC) is an approximative method that, instead of computing the required likelihood directly, substitutes it with surrogate simulations of a given model (Csilléry, Blum, Gaggiotti, & François, 2010).

Imagine that you are trying to figure out how much water you should use to water a plant. You have some expectations about how different amounts of water might lead to different growth rates. Additionally, you might have some priors about what a sensible amount of water might be; 0.01 litre probably will not help your plant to grow a lot. What Approximate Bayesian Computation now means is that you observe different growth rates in the world and that you internally simulate realizations of your priors when plugged into your model (your explanation of the world) and only keep the ones that come out reasonably close to what you have actually observed. This simple mechanism of internalized mental simulations and external reality checks provides a useful tool to approximate posteriors. A more formal description of ABC is shown in Algorithm 1.

Algorithm 1 Approximate Bayesian Computation

Require: Prior $\pi(\theta)$; model $\mathcal{M}$; data $\mathcal{D} = (\mathbf{X}, \mathbf{y})$; tolerance ϵ
  for $j = 1, 2, \ldots$ **do**
    Sample $\theta^* \sim \pi(\theta)$
    Simulate $\mathbf{y}^* = \mathcal{M}_{\theta^*}(\mathbf{X})$
    Calculate $\delta = \rho\Big(\mathcal{S}(\mathbf{y}), \mathcal{S}(\mathbf{y}^*)\Big)$
    if $\delta \leq \epsilon$ **then**
      accept θ^*
    else
      reject θ^*
  end for

This algorithm samples a proposed parameter from a prior $\pi(\theta)$ and plugs the proposal into a model $\mathcal{M}$ in order to simulate an output $\mathbf{y}^*$ from the given data $\mathcal{D}$ at hand. It then calculates a summary statistic $\mathcal{S}$ of both the simulated data and the real data and accepts proposals that have produced a summary that is reasonably close to the summary of the real data, allowing for some error ϵ. The proximity of the two summary statistics is normally estimated by a measure of distance δ, for example the euclidean distance. A more psychological explanation of this algorithm is that an agent has some prior assumptions about how the world works and repeatedly checks whether or not these prior assumptions can, on average, produce similar patterns to the ones seen by applying constant mental simulations; that is sanity-checking if the assumption are good enough to produce the seen outcomes. If they are, then those assumptions are accepted. If they are not, then they are rejected. Even though this algorithm sounds simple, it has been shown to approximate posteriors well for many complex scenarios and is currently the topic of vivid ongoing research debates. In particular, ABC has been described to be "embarrassingly parallelizable" as the aforementioned simulation-based approach can easily be set up on many different computing units (Marjoram, Molitor, Plagnol, & Tavaré, 2003) and can be used in scenarios where the likelihood is intractable.

For a simple example, consider the case of a linear regression with only one parameter as shown in Equation 3.

$$y = \beta x + \epsilon \tag{3}$$

$$\epsilon \sim \mathcal{N}(0, \sigma^2) \tag{4}$$

Of course, one could simply use standard Bayesian calculations to estimate the posterior of β. However, here we want to demonstrate how ABC can be used to approximate a posterior distribution when we know the underlying truth. For this simple example, 1000 observations of a single variable $\mathbf{x}$ were created to be distributed normally $\mathbf{x} \sim \mathcal{N}(0, 0.1)$, β was fixed to be 1, and the dependent variable $\mathbf{y}$ was simulated as in Equation **??**. ABC then was implemented based on a uniform prior $\pi(\beta) \sim \mathcal{U}[-5, 5]$ and 100,000 simulations were created from that prior. These simulations were then used to create a proposed output vector $\mathbf{y}_{\text{prop}}$. Afterwards, the Euclidean distance (see Equation 5) between the generated and the observed data, based on 100 sampled data points, was calculated and tracked for every proposed sample.

$$\rho(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} \sqrt{(x_i - y_i)^2} \tag{5}$$

This situation can be easily described as before: if we want to find out how much water leads to different rates of plant growth, then we could check different data pairs of water and growths from the past, mentally simulate outcomes, and check if the simulated growth is somewhat similar to the observed growth. Figure 1 shows the true underlying data, the simulation results of the distances as well as the posterior estimate of β depending on different values of ϵ. The true underlying data is just based on the model defined above. The produced distances based on 100,000 simulation from the prior demonstrated that many proposal ended up close to a distance of 0. This is due to the fact that we had a good prior assumption about β. The last graph shows different posterior estimates for β depending on how low ϵ was set. We can also see that the posterior estimates become more exact, the lower we set ϵ. Indeed, if ϵ was set to 0, we

would get pure rejection sampling. If we had set it to infinity, then ABC would have returned the prior as posterior as all proposals would have been accepted. Finally, the standard error for our posterior estimates gets slightly bigger, the lower we set ϵ. This is due to the fact that more samples will be deemed useless and therefore thrown away for low ϵs: we notice that ABC can be wasteful, especially when the prior is far away from the true posterior.

A model to grow heuristic strategies

Let us now define a model to learn non-compensatory decision strategies from smaller building blocks directly. Given that every heuristic is made up of a search, stopping, and a decision rule, we want to find a model that can result into a similar tree structure as TTB, but which learns the structure of the tree automatically over time. The nodes of such a tree can be seen as small building blocks of the heuristic. The way these nodes are assembled at the end, then corresponds to the search rule. The implied depth of the tree can be seen as representing the stopping rule which is reached as soon as a node makes a decision. Finally, the decision at a decisive node represents the overall decision rule. The way TTB-ABC learns and makes predictions is sketched out in Algorithm 2.

Algorithm 2 TTB-ABC

function TTBABCLEARN($X_{learn},\ y_{learn},\ \epsilon,\ \rho,\ \eta$)
- counter $\leftarrow 0$ ▷ initialize counter
- **while** counter$< \eta$ **do** ▷ η simulations
  - sample ρ of $\mathcal{D}_\rho = \{X_\rho, y_\rho\}$ ▷ test set
  - sample $p_i \sim \mathcal{B}(\alpha + k_i, \beta - k_i + n_{trials})$ ▷ beta-binomial posterior
  - proposal-tree: $\mathcal{T} : p(C_i = pos_i) \propto p_i$ ▷ sampling without replacement
  - predict $y_{prop} = \mathcal{T}(X_\rho)$ ▷ y proposal
  - calculate $\delta = \sum_{i=1}^{\rho} |y_{\rho i} - y_{predi}|$ ▷ calculate distance
  - **if** $\delta < \epsilon$ **then** ▷ accept if smaller than ϵ
    - $k_i \leftarrow k_i + 1$ ▷ update count if successful
    - $n_{trials} \leftarrow n_{trials} + 1$ ▷ update trials
    - counter $\leftarrow$ counter+1 ▷ increment counter
- **return** k, n_{trials} ▷ return counts

function TTBABCPREDICT($k,\ n_{trials},\ X_{predict},\ \eta_{aggregate}$)
- **for** i in 1:$\eta_{aggregate}$ **do** ▷ generate $\eta_{aggregate}$ predictions
  - sample $p_i \sim \mathcal{B}(\alpha + k_i, \beta - k_i + n_{trials})$ ▷ sample from posterior
  - proposal-tree: $\mathcal{T} : p(C_i = pos_i) \propto p_i$ ▷ create proposal tree
  - $y_{pred} \leftarrow \mathcal{T}(X_{predict})$ ▷ generate prediction
- **return** $y_{predict}$ ▷ return aggregated prediction

The function `ttbabclearn` defines the model's learning process. It starts out with a Beta-prior that induces a likelihood for each cue to be successful. It then samples probabilities from the prior for each cue. These probabilities are then used to generate a proposal tree. The cues for the proposal tree are sampled without replacement and in proportion to the sampled p-values from the prior before. As these betas will be updated based on a cue's success, this means that cues which are more likely to be successful are sampled at the top of the proposal tree with a higher probability than cues that have not been very successful. The built proposal tree is then used and applied to a randomly sampled subset of the whole data. The size of the randomly sampled subset depends on the parameter ρ, for which $0 < \rho \leq 1$, which means that the subset can be anything from one observation to the full data set. The proposed tree is then used to make predictions within the subset and these predictions are compared with the actual outcomes. If more than ϵ of the predictions turn out to be correct, then the tree is accepted and the involved cues of that tree get updated by adding one

success to their posterior beta-binomial-distribution. If too many classifications turn out to be wrong, then the proposal tree is rejected and no update happens. This whole process repeats until η successful proposals have been generated and the function returns the posterior beta-binomial-distribution of all cues.

The way this function learns is similar to the linear example described before. The generating model is the implied distribution over cue orders, based on the beta-binomial distribution. The used statistic is the performance in the sub-sample, it is a statistic as it does not use the whole sample, but rather approximates the performance in the whole set based on the sampled set. The way this algorithm updates cues can also be described as a Polya urn sampling scheme, where balls are put into an urn and successful (chosen) balls get reinforced over time. On a more descriptive level, TTB-ABC starts out with a prior over all possible cue orders, which is defined by the current probability of the cues to be chosen. It then generates proposal trees, and checks how well these trees perform within a subset of the data, a current stream of mental simulations, and sanity checks these proposals. It then reinforces successful combinations by a Bayesian reinforcement learning algorithms (Poupart, 2010) built on Approximate Bayesian Computation. In this way, TTB-ABC starts out with small building blocks (the cues) and puts them together and –over time– heuristic structures emerge. Notice that TTB is only one possible sub-structure of TTB-ABC. Many other possibly even more greedy strategies can be built. One example of such a strategy is one where one cue takes up more and more of the overall importance and thus almost always gets used on its own.

The function `ttbabcpredict` takes in as an argument the posterior of the beta-binomial distribution constructed by `ttbabclearn` and a parameter $n_{aggregate}$. The posterior is then used again to generate a proposal tree in dependency of how well the cues performed in the training period, and this tree is used to generate predictions for the test set. This procedure is repeated $n_{aggregate}$ times and the mode of all predictions is used as the final prediction.

Taken together, these two function constitute a first approach to how a cue order can be learned from the very beginning and how a heuristic can emerge by combining

smaller building blocks, nodes of trees, and then mentally simulating the success of proposals which generates a process, that –over time– is able to find and utilize simple trees within the complex forest of all possible trees.

Recovery performance

A first sanity check is to see if TTB-ABC can recover the true underlying model from a data set that has been solely produced by TTB. Given the assumption that TTB is just one subset of all possible models captured by TTB-ABC, it should be obvious that TTB-ABC can recover TTB from a simulated data set. Thus, 5 independent variables $\mathbf{X} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_4})^\top$ were created by using a trinomial distribution, where each vector represents the results of pairwise comparisons of two hypothetical objects (for example, cities). The probability for the first object to win (1) was fixed to be the same as for the second object to win (-1), which was half of that of a draw (0). Therefore, $p(x_i = D) = p(x_i = W \vee L) = \frac{1}{2}$ which means that a draw was set up to be as likely as a decision at each node and $p(x_i = W) = 0.25$. The final outcome, which was binary with $y = (0, 1)$, was produced by the underlying TTB-model shown in Figure 2 below. The decision strategy generating this data set is exactly the same as normal TTB, i.e. it could be seen as a comparison between two different cities just like in the example above, where a win means that city A has something that city B has not (for example, an airport) and an outcome of $y = 1$ means that city A is bigger than city B. However, one of the provided cues is uninformative as it was not included in the underlying model. Therefore, we expect our TTB-ABC model to recover a tree structure that looks like the one shown in Figure 2.

A data set of size $n = 1000$ was created and the algorithm was run until 100 proposed trees were accepted. Figure 3 shows the trace plots of the cue importance and the cue weights over time. The traces were calculated by dividing the p-value of each beta posterior mean by the total sum of values over all cues. This then equals the probability of a cue to be chosen. It can be seen that –even after a few accepted proposals– the mode tree (that is the most likely tree) is the same as the one generating

the data. Shortly after the begin of the simulation, the cue order grows to the correct ranking, and the unimportant cue is chosen less frequently, exactly as expected given the underlying structure.

A plot of the density over all possible trees is shown in Figure reffig:treedense. It can be seen that the density for the right tree (C1,C2,C3,C4) becomes more and more likely over time.

Inference for real data

Another check is whether or not TTB-ABC generates sensible and interpretable inferences in a real data set. Thus, we applied our algorithm to the classic city size data set in order to see how the trace plot looks like for this data. The city size data set has been frequently used in the past and contains 9 different binary variables that all can be used to decide which of two cities is bigger (Chater, Oaksford, Nakisa, & Redington, 2003; Gigerenzer & Brighton, 2009). Figure 5 shows the trace plots for the city size simulation over 100 accepted proposal trees. Again, the output looks sensible, finding that having a university, an exposition site, and a major league soccer team are the three most important cues.

Notice that contrary to the classic estimation of TTB, the capital city cue turned out to decrease in probability over time instead of being the most important cue. This is due to the fact that Berlin is the only city in the whole data set that is a capital city and therefore this cue could not be applied very often, a finding that has been coped with in the literature before (Bröder, 2012).

Predictive comparison

Another important benchmark is how well TTB-ABC can predict different outcomes. Therefore, we sought out to compare TTB-ABC's predictive performance within three different data sets: the city size data, professors' salary, and homelessness in the US. We let three different TTB models learn within a learning set and classify new items within a test set. TTB-ABC was fitted by using the algorithm described

above. Standard TTB was fitted by calculating the cue validity in the way originally proposed:

$$v = \frac{\mathbb{1}_{x_i=y_i}}{\sum_i^n \mathbb{1}_{x_i \neq 0}} \tag{6}$$

which basically just sums up how often a cue can be used to make a successful predictions, given that it can be used at all. TTB-Updated is trying to adjust for this problem Newell, Rakow, Weston, and Shanks (2004) and corrects the cue validity by the discrimination rate, that is how often a cue can actually be applied.

$$d = \frac{\sum_{i=1}^n \mathbb{1}_{x_i \neq y_i}}{n} \tag{7}$$

$$v_{\text{updated}} = v \times d + 0.5 \times (1 - d) \tag{8}$$

In order to see which of the two models performs best for different sizes, test sets were created to be of the size $s = \{10\%, 20\%, \ldots, 90\%\}$ of the total data set (before the comparisons were generated). The learning set always contained the remaining data points that were not used in the test set. For every size, 50 randomly generated repetitions were run and the mean performance was tracked.

Additionally, for each data set, we calculated the best possible cue order for the whole data set by a brute force algorithm directly. Brute force means that all possible cue combinations were generated and the one that overall lead to the lowest classification error was taken to be the best order. This order could then be compared to the order determined by each model and also tracked for different set sizes over the 50 trials. The distance to the best underlying cue order was calculated by using the discounted cumulative gain as shown in Equation 9 (Neth, Schooler, Buikstra, Teije, & van Harmelen, n.d.).

$$DCG_l = \sum_{i=1}^{l} \frac{Q_i}{\log_2(i+1)} \tag{9}$$

The resulting sum is then divided by the best possible order, which was determined by the brute force algorithm and again tracked overall. Results are shown in Figure 6. It can be seen that TTB-ABC outperforms standard TTB across the whole range. This finding is even more surprising given that TTB has consistently been found to outperform many other models within this data set.

It can be seen that TTB-ABC performs better than the other two approaches across all different data sets. However, on average, TTB-ABC tends to decrease in its ability to find the best possible cue order. This is due to the fact that only the mode cue order, that is the most likely a posteriori order, has been used for this calculation. Moreover, the number of samples that TTB-ABC samples naturally get smaller as the learning set gets bigger. In general though, TTB-ABC performs better than the other two approaches while generating simple distributions over cue orders at the same time.

Trading off performance and computation time

Another benefit of our TTB-ABC model is that performance and computational time can be traded-off against each other explicitly. As the ϵ-parameter defines the proportions of predictions that have to be correct in order for a proposal tree to be accepted, increasing this parameter will lead to more proposal trees being rejected. This in turn will lead to a longer computation time as more proposals have to be created overall. However, better proposals also mean better performance as the proposals have a better quality. The η-parameter in turn determines the number of accepted proposed trees. Thus, more accepted trees means more proposals needed, means longer computation time, which in again will lead to better performance. The same holds true for the proportion of sampled data points ρ. The higher the proportion of the sampled data set, the better the proposed trees have to be, and the longer it takes to find good trees.

Using `ttbabclearn`, we can model these parameters explicitly to find out how they influence performance and computation time (here used as a proxy variable for complexity). Therefore, we created all 81 possible combinations of $\epsilon = \{0.1, 0.2, \ldots, 0.9\}$ and $\rho = \{0.1, 0.2, \ldots, 0.9\}$ and applied the resulting model to the scenario in which TTB had generated the data as in the first recovery task above. The resulting models were applied to this data set with $\eta = \{1, 10, 50, 100\}$ simulations. We used the R-package `microbenchmark` to track the the time it took to learn within the different sets, averaged over 100 trials. The results of the overall computation time are shown in Figure 7 below. We can see that for low ηs, the computational time does not differ much for high and low ϵs and ρs. However, for many number of simulations, the computation time seems to increase exponentially, if the proportion of samples as well as the quality of the to be accepted models increases.

For the performance of the resulting models, we averaged the models' predictions over the whole set over 30 trials and calculated the mean proportion of correct predictions over the whole set. Results are shown in Figure 8.

We can see that, especially for high number of samples η, tuning the ϵ and ρ-parameters can increase performance by up to 20%, whereas for lower sample numbers the configuration matters less.

Taking these two results together, we see that there seems to be a clear trade-off between time and performance as time increases exponentially, but performance increases not as much as time. The performance gained per time spent therefore increases with the number of samples. Using TTB-ABC we can model this trade-off explicitly by changing the different parameters and actually test the performance-accuracy trade-off, a crucial assumption of heuristic theories of decision making, directly.

Discussion & Conclusion

We have introduced TTB-ABC, a model that can grow heuristic decision trees via Approximate Bayesian Computation. Using an approximately Bayesian learning

scheme, this algorithm can learn heuristic decision strategies over time and constitutes as a first proposal of how a tool from the heuristic toolbox can emerge from smaller building blocks.

We have shown that TTB-ABC can recover original TTB in a simple simulated data set and leads to sensitive estimates of expected cue depth, cue orders, and cue weights within the frequently used city size data set. In a predictive performance task, TTB-ABC outperformed TTB across the board, even though it can lead to more simple inferences at the same time. Lastly, TTB-ABC can be used to model the time-accuracy trade-off directly by adjusting its input parameters. Doing so, we have found that there are scenarios where increased computation time diminishes in returns of predictive accuracy.

TTB-ABC constitutes as a first proof of concept that a heuristic can emerge from smaller building blocks over time and we expect it to be further developed as well as applied to human experimental data soon.

Other ways of letting strategies emerge over time such as inductive programming (Muggleton, 1994), evolutionary (symbolic) regression (Dechter, Malmaud, Adams, & Tenenbaum, 2013), as well as the composition of smaller sub-units via greedy search algorithms (Duvenaud, Lloyd, Grosse, Tenenbaum, & Ghahramani, 2013) could also be defined and tested in future studies.

In the long term, a valid goal for heuristic theories of decision making seems to be defining how many different heuristic models, that is not only tree-like structures, can emerge from smaller compositional building blocks, thereby defining a grammar of rules within a heuristic toolbox.

References

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.

Bröder, A. (2012). The quest for take the best—insights and outlooks from experimental research. *Ecological rationality: Intelligence in the world*, 216–240.

Brooks, S. P. (2003). Bayesian computation: a statistical revolution. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, *361*(1813), 2681–2697.

Chater, N., Oaksford, M., Nakisa, R., & Redington, M. (2003). Fast, frugal, and rational: How rational norms explain behavior. *Organizational behavior and human decision processes*, *90*(1), 63–86.

Csilléry, K., Blum, M. G., Gaggiotti, O. E., & François, O. (2010). Approximate bayesian computation (abc) in practice. *Trends in ecology & evolution*, *25*(7), 410–418.

Dagum, P., & Luby, M. (1993). Approximating probabilistic inference in bayesian belief networks is np-hard. *Artificial intelligence*, *60*(1), 141–153.

Dechter, E., Malmaud, J., Adams, R. P., & Tenenbaum, J. B. (2013). Bootstrap learning via modular concept discovery. In *Proceedings of the twenty-third international joint conference on artificial intelligence* (pp. 1302–1309).

Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J. B., & Ghahramani, Z. (2013). Structure discovery in nonparametric regression through compositional kernel search. *arXiv preprint arXiv:1302.4922*.

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: take the best and its relatives. *Simple Heuristics that Make Us Smart. Oxford University Press,*

New York, 75–95.

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

Gilks, W. R. (2005). *Markov chain monte carlo*. Wiley Online Library.

Hoffrage, U. (1999). *When do people use simple heuristics, and how can we tell?* New York: Oxford University Press.

Lieder, F., Griffiths, T., & Goodman, N. (2012). Burn-in, bias, and the rationality of anchoring. In *Advances in neural information processing systems* (pp. 2690–2798).

Lindley, D. V., Lindley, D. V., & Lindley, D. V. (1972). *Bayesian statistics: A review*. SIAM.

Marjoram, P., Molitor, J., Plagnol, V., & Tavaré, S. (2003). Markov chain monte carlo without likelihoods. *Proceedings of the National Academy of Sciences*, *100*(26), 15324–15328.

Muggleton, S. (1994). Bayesian inductive logic programming. In *Proceedings of the seventh annual conference on computational learning theory* (pp. 3–11).

Neth, H., Schooler, L., Buikstra, A., Teije, A., & van Harmelen, F. (n.d.). Recognizing the truth: Automatically ranking lod query results with a cluster heuristic.

Newell, B. R., Rakow, T., Weston, N. J., & Shanks, D. R. (2004). Search strategies in decision making: The success of "success". *Journal of Behavioral Decision Making*, *17*(2), 117–137.

Oaksford, M., & Chater, N. (2007). *Bayesian rationality the probabilistic approach to human reasoning.* Oxford University Press.

Poupart, P. (2010). Bayesian reinforcement learning. In *Encyclopedia of machine learning* (pp. 90–93). Springer.

Turner, B. M., & Van Zandt, T. (2012). A tutorial on approximate bayesian computation. *Journal of Mathematical Psychology*, *56*(2), 69–85.

Figure 1. Plots of underlying data, the resulting distances, and the posterior mean dependent on ϵ.

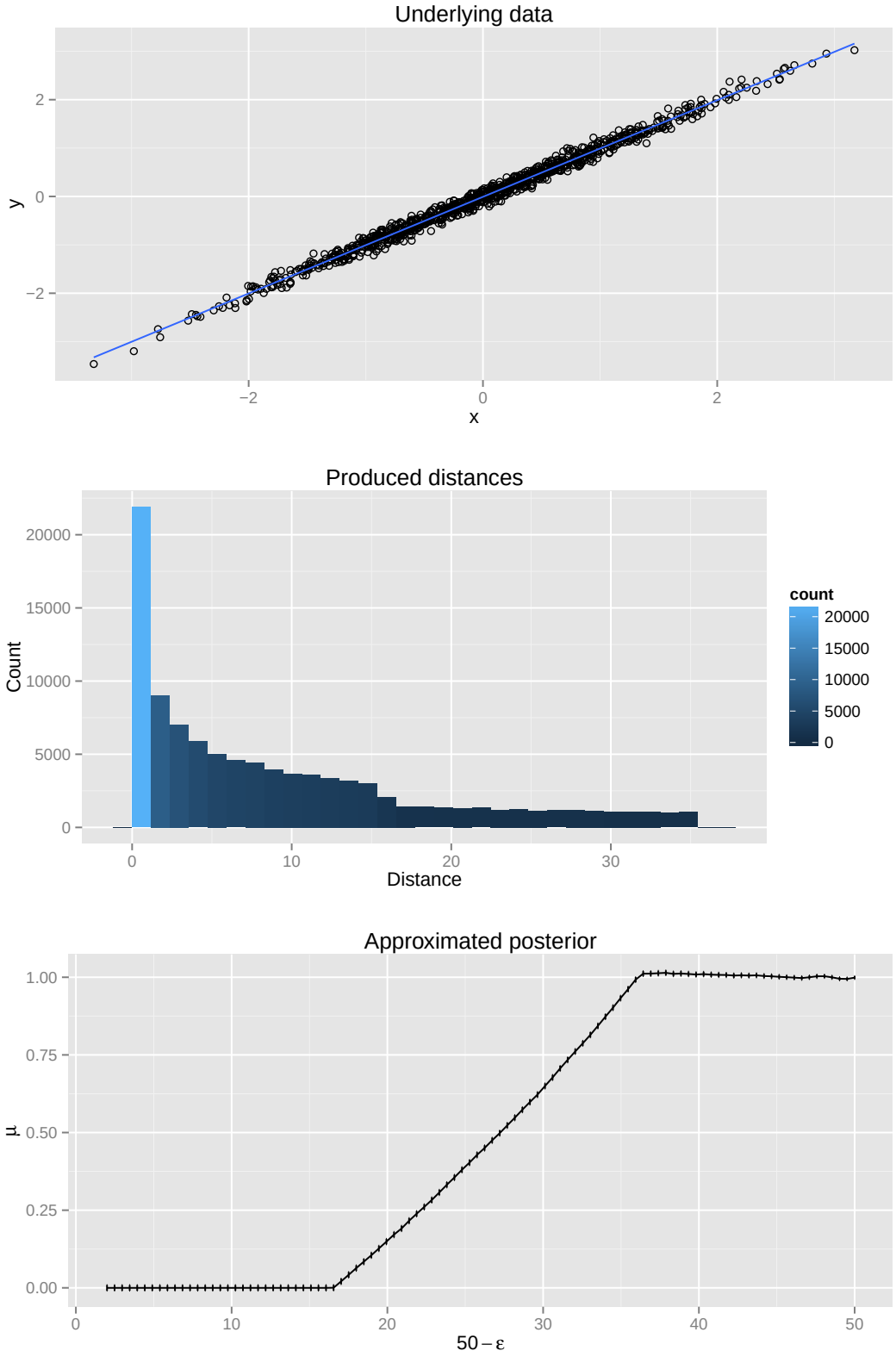

Figure 2. Underlying TTB-model. Only cues C_1, C_2, C_3 are important.

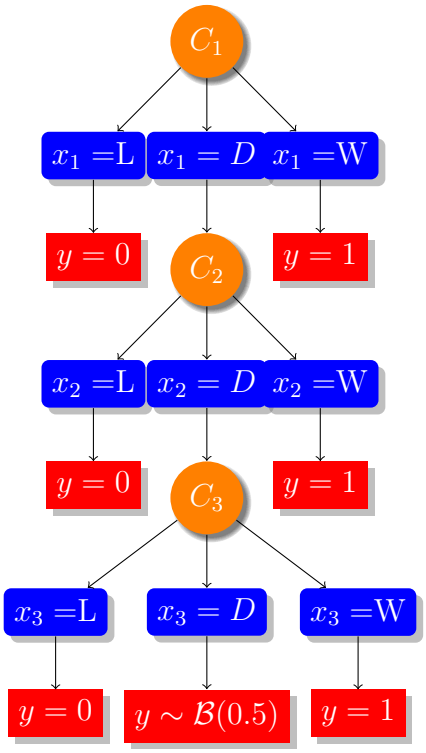

Figure 3. Trace plots of recovery simulation

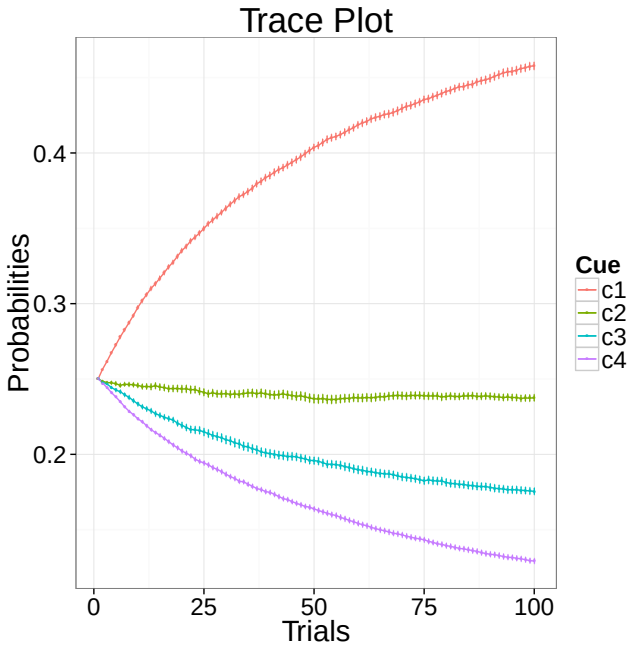

Figure 4. Density over trees per sampling step

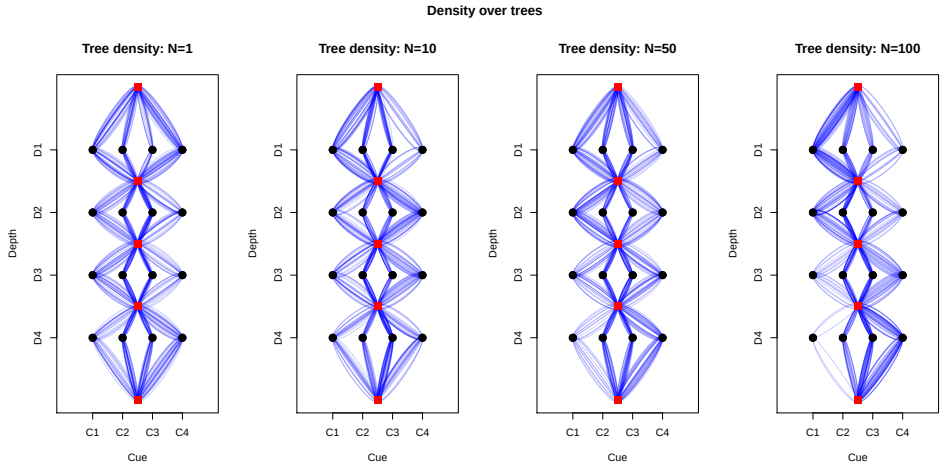

Figure 5. Trace plots of city size data estimation.

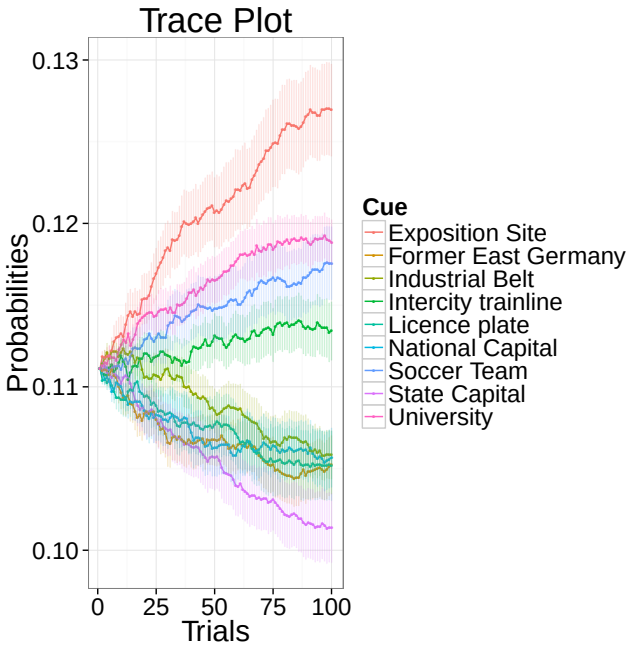

Figure 6. Performance of different models in dependency of training size

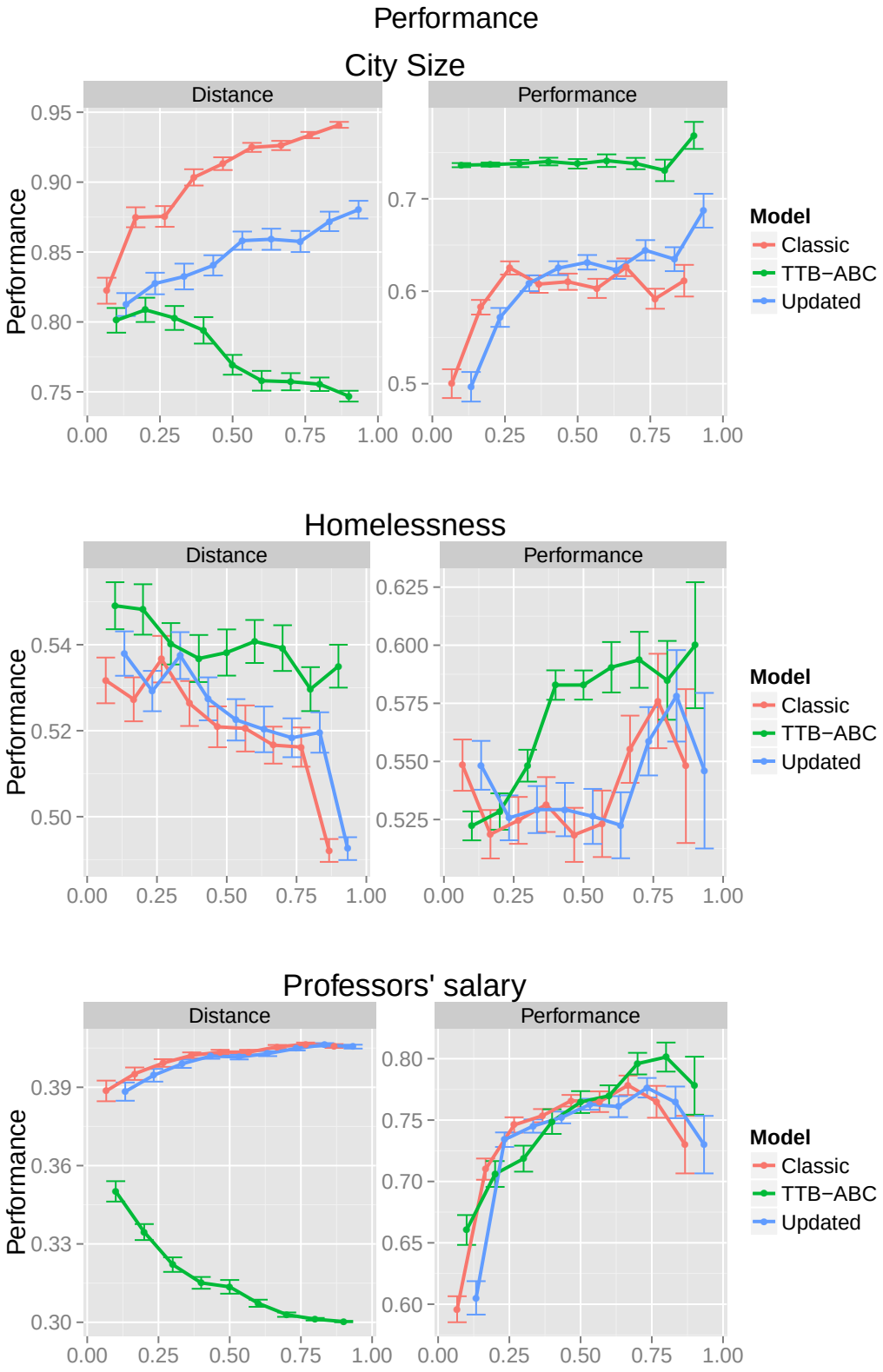

Figure 7. Assessed computational time of different ϵ-ρ-combinations for $\eta = \{1, 10, 50, 100\}$ samples.

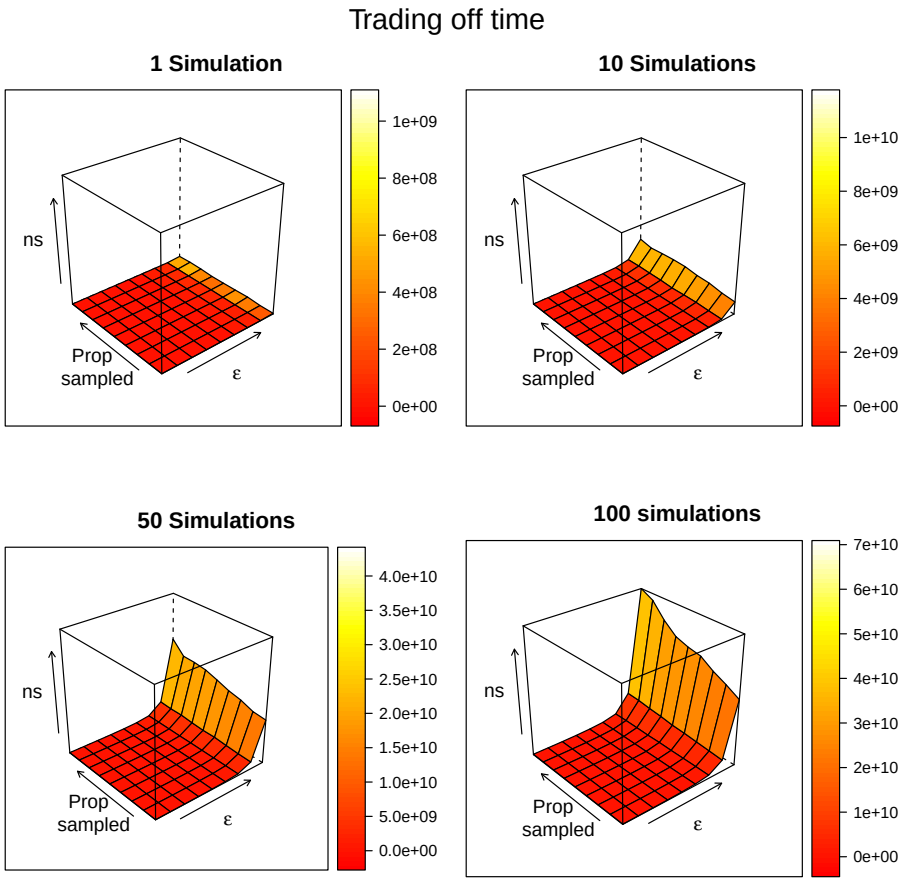

Figure 8. Assessed performance of different ϵ-ρ-combinations for $\eta = \{1, 10, 50, 100\}$ samples.