Intelligence as Inference or Forcing Occam on the World

Peter Sunehag and Marcus Hutter
{Peter.Sunehag, Marcus.Hutter}@anu.edu.au

Research School of Computer Science
Australian National University
Canberra Australia

Abstract. We propose to perform the optimization task of Universal Artificial Intelligence (UAI) through learning a reference machine on which good programs are short. Further, we also acknowledge that the choice of reference machine that the UAI objective is based on is arbitrary and, therefore, we learn a suitable machine for the environment we are in. This is based on viewing Occam's razor as an imperative instead of as a proposition about the world. Since this principle cannot be true for all reference machines, we need to find a machine that makes the principle true. We both want good policies and the environment to have short implementations on the machine. Such a machine is learnt iteratively through a procedure that generalizes the principle underlying the Expectation-Maximization algorithm.

1 Introduction

Universal Artificial Intelligence (UAI) [Hut05,LH07] formalizes intelligence as an incomputable optimization problem. We will here recast optimization problems as inference problems and define an iterative procedure that generalize the expectation maximization algorithm [DLR77] beyond its original form and context. The idea has been used previously in more narrow settings [DH97,WGRT11]. Our approach here brings "planning as inference" [Bot12] to a universal setting and we will discuss how to completely reduce the design of intelligent agents to building an inference mechanism. However, we also note the importance of the training environment. We note that it has been argued [HB04] that the human neo-cortex relies on only one mechanism and that, therefore, it should be possible to understand higher level cognitive abilities as one mechanism interacting with the environment with lower level processing in between.

Since we want a practical approach, we are only interested in computable policies. Hence, we are considering a search for programs achieving the highest possible return. The return function is not explicit and is evaluated for a program by running it and observing what return was achieved. This means that the impact of issues like running speed and memory use are subsumed into the return function. The setting is essentially that of achieving bounded optimality (rationality) as defined in [Rus97,RN10].

Our approach is to iteratively develop a reference machine on which good programs are short and, thereby, representing what we have learnt so far. The reference machine defines a distribution over programs by letting the probability of p be proportional to $2^{-\ell(p)}$ where $\ell(p)$ is the length of p. For every iteration we want the machine to represent probabilities which are such that the probability of the Turing machine T implemented by program p on the previous machine U, becomes proportional to the product of its (expected) return and its previous probability. If this is achieved we are guaranteed an improvement in expected return which means we got a better reference machine for finding rewarding programs.

In a practical approach the update is based on a population of sample programs. These can be sampled for the machine by coin-flipping the bits of the program, but they can include any evaluated programs from any sources. This is an important point of the approach since all programs written by human programmers can be used and their intelligence can be mined for good ideas this way. In fact, it is already used this way in the development of programming languages and their libraries that aim to ease the development of good programs and in some cases make bad things harder to do. The goal can be viewed as making Occam's razor true as a proposition about the world, while having to make it so is in line with viewing Occam as an imperative [Web10]. This procedure can be seen far beyond programming languages, e.g. in science where concepts and terminology are created such that complex phenomena can be described in compact form. The perspective of this article makes much of human progress into a quest towards artificial general intelligence. In fact, we can go a bit further by first looking at the background of our approach.

Evolution. In [DH97], the authors identified the Relative Payoff Procedure (RPP) as an Expectation-Maximization (EM) procedure. RPP makes binary choices independently and with probabilities proportional to the payoff relative to the total payoff and then iterates. It has some similarities with evolutionary algorithms (and natural evolution) where the most fit part of the population grows at the expense of the less fit, but like the probabilistic model-building genetic algorithms [Pel12], an explicit distribution representing current preferences is iteratively estimated resulting in more powerful optimization/learning in an optimization-as-inference paradigm. Unlike most evolutionary approaches, the expectation-maximization approach is learning also from the unfit and the degrees of fitness of all samples.

It is interesting to note that evolution theory itself has begun to place importance on more than genes [WE03], namely on gene-expression that changes during life based on the environment. An adaptation is learnt through (or causing) changed gene-expression (which can even be inherited) and then, according to this theory, a gene can be discovered that *accommodates* this adaptation and locks it in. The accommodation theory of evolution introduce a purpose-driven aspect into the process. This layer is represented in the mentioned algorithms by the distribution which allows for learning and exploration of complex relationships, if the distribution is more sophisticated than the independence in RPP

allows for. In the accommodation theory of evolution, behavior is explored before it is accommodated by DNA. The suitable gene that accommodated the adaptation could have been useless without the adaptation already being to some extent in place. Gene expression (and also we here speculate brains and culture when available) is what makes it possible for some biological adaptation before a change in DNA takes place. A startling speculative possibility is that species that can learn better during life also have an accelerated genetic evolution making high levels of intelligence a much more probable outcome than it would be with a simpler entirely gene-centered understanding of evolution. That the more simplistic understandings of evolution cannot learn something complex in feasible time has been understood for a long time, e.g. by the authors of [Len80]. As one might conclude that biological evolution was geared towards intelligence, one can argue that our current technological and scientific evolution is moving towards Artificial General Intelligence even without someone making it the goal.

Logical reasoning. We believe that logical reasoning is not essential in the building of an intelligent agent and is not a natural part of the human mind. It has been repeatedly shown that plausibility-based reasoning is the innate human thinking. People systematically place more belief in an event than a superset containing it, if the event creates a plausible narrative [Kah11] (not advocated here). Logical reasoning likely evolved from rhetoric in ancient Greece and does not seem to have been accommodated genetically and might not confer advantage except for some very particular constructed environments like computer programming.

We believe that logic is learnt from the human created environment. Moreover, practical logical reasoning is seldom fully formally logical, e.g. mathematical proofs are not written out in full but just aims at plausibility of each step and hence originally accepted proofs are sometimes found to be flawed. Searching through proof space suffers from combinatorial explosion and looking for plausible chains guided by one's belief about what is likely true (will hold up to scrutiny) or not is more practical. Our approach is based on plausible improvement by improving a sample-based approximation of what would guarantee improvement, and not on search for improvements with formal proof as the Gödel machine [Sch07]. The Gödel machine starts with a set of axioms defining an agent and then aims to update to a provably better set. The shifting inductive bias approach [SZW97] is closer to what is considered here though a direct comparison is difficult.

With probabilistic architectures with parameterized programs like e.g. Deep Belief Networks [HOT06], it is easier to generate programs that run and do something. The involved pattern recognition based reasoning is far more efficient than dealing with logical programs and architectures.

Outline. We first provide background and notation for Universal Artificial Intelligence (UAI) and general reinforcement learning. Then we introduce the objective whose optimization defines how one should update a reference machine to have a guaranteed improvement of expected utility when sampling programs from the machine. We go on to discuss a setting where generated observations in

the optimization process are taken into account. We then we change from optimizing the given UAI objective to actually learning about the world through developing the reference machine for the UAI objective. Finally we discuss merging the notions of actions and observations as well as agent and environment before we conclude in the last chapter.

2 Learning a Reference Machine for UAI

We will consider an agent [RN10,Hut05] that interacts with an environment through performing actions a_t from a finite set $\mathcal{A}$ and receives observations o_t from a finite set $\mathcal{O}$ and rewards r_t from a finite set $\mathcal{R} \subset [0,1]$ resulting in a history $h_t := o_1 r_1 a_1, ..., o_t r_t$. Let $\mathcal{H} := \cup_n (\mathcal{O} \times R \times \mathcal{A})^n \times (\mathcal{O} \times \mathcal{R})$ be the set of histories and let ϵ be the empty history. A function $\nu : \mathcal{H} \times \mathcal{A} \to \mathcal{O} \times \mathcal{R}$ is called a deterministic environment. A function $\pi : \mathcal{H} \to \mathcal{A}$ is called a (deterministic) policy or an agent. We define the value function V based on geometric discounting by $V_\nu^\pi(h_{t-1}) = \sum_{i=t}^\infty \gamma^{i-t} r_i$ where the sequence r_i are the rewards achieved by following π from time step t onwards in the environment ν after having seen h_{t-1}.

Instead of viewing the environment as a function $\mathcal{H} \times \mathcal{A} \to \mathcal{O} \times \mathcal{R}$ we can equivalently write it as a function $\nu : \mathcal{H} \times \mathcal{A} \times \mathcal{O} \times \mathcal{R} \to \{0,1\}$ where we also write $\nu(o, r|h, a)$ for the function value $\nu(h, a, o, r)$ (which is not the probability of the four-tuple). It equals zero if in the first formulation (h, a) is not sent to (o, r) and 1 if it is. In the case of stochastic environments we instead have a function $\nu : \mathcal{H} \times \mathcal{A} \times \mathcal{O} \times \mathcal{R} \to [0,1]$ such that $\sum_{o,r} \nu(o, r|h, a) = 1 \; \forall h, a$. Furthermore, we define $\nu(h_t|\pi) := \Pi_{i=1}^t \nu(o_i r_i | a_i, h_{i-1})$ where $a_i = \pi(h_{i-1})$. $\nu(\cdot|\pi)$ is a probability measure over strings or sequences and we can define $\nu(\cdot|\pi, h_{t-1})$ by conditioning $\nu(\cdot|\pi)$ on h_{t-1}. We define $V_\nu^\pi(h_{t-1}) := \mathbb{E}_{\nu(\cdot|\pi,h_{t-1})} \sum_{i=t}^\infty \gamma^{i-t} r_i$.

Given a countable class of environments and strictly positive prior weights w_ν for all ν in the class, we define the a-priori environment ξ by letting $\xi(\cdot) = \sum w_\nu \nu(\cdot)$ and the AIXI agent [Hut05] is defined by following the policy

$$\pi^* := \arg\max_\pi V_\xi^\pi(\epsilon).$$

The optimistic-AIXI agent [SH12b], which is an extension of AIXI, takes the decision

$$\pi^\circ := \arg\max_\pi \max_{\xi \in \Xi} V_\xi^\pi(\epsilon)$$

for a finite set of a priori environments Ξ. Other variations include the space-time embedded agents of [OR12] who are part of (computed by) the environment.

This article is dealing with the Universal Artificial Intelligence (UAI) setting where AIXI is a mixture of all computable, or lower semi-computable, environments. The mixture weights are defined from a choice of reference Universal Turing Machine (UTM) as $2^{-K(\nu)}$ for environment ν, and K is Kolmogorov complexity (length of the shortest program that implements the argument) with respect to U. The resulting a priori environment ξ can, equivalently, be defined

as sampling a program for U and run it. The probability of a program p is proportional to $2^{-\ell(p)}$ where $\ell(p)$ is the length of p. The expected reward of a policy/agent is viewed as a measure of its intelligence [LH07]. However, this is dependent on the reference machine and so is the AIXI agent defined from it by maximizing that intelligence measure. Any agent is super-intelligent for some reference machine. The optimistic extension is both meant to be a more explorative version with more uniformly good performance across environments but also to diminish the dependence on the reference machine by being able to choose a large finite set of machines instead of just one. We will later in this article address the problem of choosing a good reference machine to base AIXI on, but we will first deal with the involved optimization problem by iteratively learning a reference machine that represents a preference between policies such that good policies have shorter programs on this machine.

Learning reference machines. Let $\mathcal{T}$ be the set of all Turing machines and suppose we are given a (possibly unknown) return function $R : \mathcal{T} \to \mathbb{R}$ that for each $T \in \mathcal{T}$ says how rewarding this machine is. We ideally want a machine from

$$\arg\max_{T \in \mathcal{T}} R(T).$$

A prominent example is when we have a general reinforcement learning environment μ as defined in [Hut05] (and above) and T computes a policy. $R(T)$ is then the expected discounted reward sum when acting according to T in μ. Given a Universal Turing Machine U, any Turing machine can be represented as a program p for U. Hence, given U, the search for a Turing machine T becomes a search for a program p for U. Though the choice of reference machine U does not affect which Turing machines are good, it affects how long/complex its implementation is. We suppose that a practical search would favor shorter programs over longer ones and, therefore, the choice of reference becomes critical for success. In fact, we will replace the search for programs with the task of incrementally inferring what a good choice of machine is. The choice of machine will encapsulate everything we have learned up to that time about how good various programs are by making the good ones shorter than worse alternatives. We want a machine U with high expected return defined by

$$\sum_p 2^{-\ell_U(p)} R(p).$$

One approach is to propose a different machine and estimate the expected return by running sampled programs on this machine. We will here instead consider a generalized Expectation-Maximization procedure similar to what [DH97] discussed in their narrow setting, which allows you to evaluate the new machine without running any programs on it. One only requires the ability to measure the length of a program translated for it. However, all of what follows can be expressed in terms of the expected utility objective above.

Given reference machine U, we want to change to U' (including a mapping of programs on U to programs on U') if

$$\sum_p 2^{-\ell_U(p)} R(p) \ell_U(p) > \sum_p 2^{-\ell_U(p)} R(p) \ell_{U'}(p)$$

where $\ell_U(p)$ is the length of the program p for U while $\ell_{U'}(p)$ is the length of the translation of p for U'. We suppress the search over translation programs to simplify notation. The larger the expectation of $R(p)\ell_U(p)$, the stronger the correlation between return and length. Hence, the smaller the expression $\sum_p 2^{-\ell_U(p)} R(p)\ell_U(p)$ is, the stronger the correlation between high return and short length. This update says that we want programs with high reward to have low complexity and, furthermore, guarantees that

$$\sum_p 2^{-\ell_{U'}(p)} R(p) > \sum_p 2^{-\ell_U(p)} R(p).$$

This result is simply proven using Jensen's inequality similar to [DH97].

$$\log \sum_p 2^{-\ell_{U'}(p)} R(p) - \log \sum_p 2^{-\ell_U(p)} R(p)$$

$$= \log \sum_p \frac{2^{-\ell_U(p)} R(p)}{\sum_q 2^{-\ell_U(q)} R(q)} \frac{2^{-\ell_{U'}(p)}}{2^{-\ell_U(p)}}$$

$$\geq \sum_p \frac{2^{-\ell_U(p)} R(p)}{\sum_q 2^{-\ell_U(q)} R(q)} \log \frac{2^{-\ell_{U'}(p)}}{2^{-\ell_U(p)}}$$

$$= \frac{1}{\sum_q 2^{-\ell_U(q)} R(q)} \big(\sum_p 2^{-\ell_U(p)} R(p) \ell_U(p) - \sum_p 2^{-\ell_U(p)} R(p) \ell_{U'}(p) \big) \geq 0.$$

The desired result follows since log is monotone increasing.

Approximations with sample programs. $\sum_p 2^{-\ell_U(p)} R(p) \ell_{U'}(p)$ is not computable. Hence, we cannot exactly evaluate if U' is better than U. Instead we will in practice need to rely on a set of sample programs p and their evaluations $R(p)$. If R is of the form of an expectation of a stochastic return one needs to run the evaluation repeatedly and average. To sample with a reference machine U is simple since it is done by coin flipping on the input tape. In other words, one randomly picks characters for the programming code which is typically a practical disaster if one does not have a reference machine that makes a reasonable program out of many short random strings. This problem disappears if as program we simply set parameters for an algorithm that can be run for any choice of such. An important point is that the set of sample programs do not all have to be sampled from the current reference machine but interesting programs developed elsewhere can also be introduced to be learnt from. Further, one needs a translation to the new machine. This is not necessarily difficult if the proposed change is to make some routines from the best programs part of

the machine (language), reducing a block of code like a matrix multiplication to a macro. This way, the process becomes one of discovering and collecting useful routines while developing a high-level language. In the case when parameterized probabilistic functions are used, one can instead learn to imitate successful programs by finding parameters that lead to close reproduction of the programs' behavior.

Including observations/data and environment. The setting discussed above has a return function that goes straight from programs (Turing machines) to a real valued return. However, an alternative is to include more structure where running the program generates data $d \in \mathcal{D}$ ($\mathcal{D}$ can be finite or countably infinite), which might consist of a sequence of observations each of which can contain a reward, and then there is a return function $R : \mathcal{D} \to \mathbb{R}$. R can e.g. be defined as a discounted reward sum. Though this can, as was mentioned before, be viewed as just an example of the optimization task above if we have been provided with a general reinforcement learning environment (which could be a Bayesian mixture of environments as for AIXI), we can also try to take advantage of the extra information and structure. This is done by replacing p with a pair (p, d) but we must let the coding of d be done by the machine $\hat{U}$ that represents the environment since we cannot change the objective optimized as long as we are in the optimization setting. Hence we end up with pairs (p, q) where q is a program on $\hat{U}$ that is run conditioned on p, i.e. q becomes an environment and p a policy. In other words, we want to change from U to U' such that

$$\sum_{p,q} 2^{-(\ell_U(p)+\ell_{\hat{U}}(q))} R(p,q)\ell_U(p,q) > \sum_{p,q} 2^{-(\ell_U(p)+\ell_{\hat{U}}(q))} R(p,q)\ell_{U'}(p,q)$$

where $\ell_U(p, q)$ is the length of a program on U (given by a translation) that produce the outcome of running policy p in environment q. This is simultaneously aiming for a machine where good policies have short program and where the likely data to be generated when such a policy is run, can also be coded compactly. This strategy is a sound method for representing plausibility and desirability in the same way and simultaneously modeling the world and the value of different choices in it. In the optimization discussed so far, we know the objective we optimize a policy for and the learning of this has so far been a separate matter. We will below take the next step of our investigation.

Combined learning of agent and environment. The above described strategy starts with a reference machine $\hat{U}$ that AIXI's initial model ξ is defined from. The choice of $\hat{U}$ is arbitrary and any policy/agent can be considered superintelligent dependent on the reference machine. The reference machine defines what is considered a simple hypothesis and defines belief based on viewing simpler hypothesis as more plausible than more complex ones, i.e. Occam's razor is relied on. In this article, we learn a reference machine that implicitly incorporates an understanding of the world that makes Occam true. We replace q by d and $2^{-\ell_{\hat{U}}(q)}$ by the unknown true environment probability $\mu(d|p)$ and aim for higher $\sum_{p,d} 2^{-\ell_U(p)}\mu(d|p)R(d)$ by choosing U' with

$$\sum_{p,d} 2^{-\ell_U(p)}\mu(d|p)R(d)\ell_U(p) > \sum_{p,d} 2^{-\ell_U(p)}\mu(d|p)R(d)\ell_{U'}(p).$$

In the sampling setting we simply sample from the world instead of using $\hat{U}$. This is making the simplifying assumption that we have a reset after a program has been run and evaluated, i.e. we can start every program in the same situation. This is feasible if one wants to train an agent in a rich world like a general game playing situation but where one can start over.

Exploration is important, both for the individual programs who act sequentially for a long time and whose total return depends on exploration, as well as for the sampling of programs. The learnt bias towards programs believed to have high reward ensures that we aim to explore promising possibilities and not waste time on finding out exactly how bad something is. Optimistic hypotheses have the beneficial property that if the outcome does not clearly contradict the hypothesis, the agent has received high return [SH12a].

When we above wrote that we want $\sum_{p,d} 2^{-\ell_U(p)}\mu(d|p)R(d)$ to be high, there are two possible interpretations depending on what we expect the programs to be doing. We can have p that is simply implementing a policy, i.e. it delivers actions and need an observation in between each to deliver the next. However, we can also allow programs that produce output that can be understood as pairs of actions and observations or whole sequences of such pairs (i.e. all of the data d), but where the observation part can be changed by the environment resulting in a different return. This stays within the usual agent-environment framework while we can go further and still use the same expression above, by letting the environment rewrite the whole output. In this setting, a program is sampled and then data is generated. The data might not entirely coincide with the data coded/generated by the program on the reference machine, because the program acts in the (unknown and uncertain) world. In this setting there is no distinction between actions and observations. The program aims to write a certain (say) 1000 bits and there are another, possibly different 1000 bits resulting which the return is based on.

Reward-modulated inference. If we want to implement our strategy using a parameterized probabilistic architecture, the objective becomes one of optimizing a reward weighted likelihood/loss. One can e.g. envision the action being part of the observation and one tries to improve the observation to achieve higher reward.

In the field of neuro-science where finding the update equation for synaptic weights used by biological brains is sought, reward-modulation of spike-timing-dependent synaptic plasticity has been the recent focus of much research. Herrstein's law [Her70] states that an animal, including humans, tend to select actions with a frequency proportional to the reward accumulated while taking that action. [LS06] shows that such operant matching is the result if one has correlation between reward and neural activity in a spiking neural network. [LPM07] began analyzing such methods from a learning-theoretic perspective

and [FSG10] interpreted a broad class of candidate rules as having a reward-driven part and an unsupervised part. They considered the latter to be undesirable for reinforcement learning and introduced a reward predicting critic to cancel it out. However, Deep Learning [HOT06,BLPL07] often starts with an unsupervised phase and then tunes for the actual objective. [SL14] recently argued that model-free reinforcement learning cannot account for human behavior but argues that some modeling of observations is taking place in the human brain.

[DH97] points out the difference between their iterative procedure, which is the basic idea used in this article, and the simpler matching procedure which stay with the same frequencies. The difference is that the current probability of taking an action is taken into account in the expectation and then, if the most rewarding actions stay the same, its frequency will for each iteration keep increasing towards one. If it changes, the procedure is still able to change and move in a new direction. That the simpler matching is so prevalent in nature might imply that the natural environment is so uncertain, changing and even dramatically adversarial that this modest amount of optimization for current conditions is suitable and can also still be challenging in a complex environment.

Optimizing the world=agent+environment. In our final setting we do not hold a true environment nor an a priori reference machine fixed but evolve the machine that defines the environment. This means that we no longer have a clear separation between agent and environment. The only thing fixed in this setting is the return function R. For this to be useful we must either be in a setting where the return is not just internal in the sense that it sums up rewards produced by a program on the reference machine, but that it is external to the environment we discuss. For example, if we want an agent to produce a music song in a music school that is then sold outside, we have a meaningful setting where it is useful to optimize the whole operation and not a subset that is interpreted as the agent. Alternatively, we can view the environment as being everything and the return as being internal, but where it is a hard task to change the total world to increase the reward. The update objective $\sum_p 2^{-\ell_U(p)} R(p) \ell_{\tilde{U}}(p)$ is telling us that we want to change U to U' such that the expression becomes larger with $\tilde{U} = U'$ than with $\tilde{U} = U$, resulting in $\sum_p 2^{-\ell_{U'}(p)} R(p) > \sum_p 2^{-\ell_U(p)} R(p)$.

3 Conclusions

We discussed reducing all of intelligence to an inference mechanism and properties of the environment. We introduced a formal approach that iteratively develops a reference machine suitable for both, implementing a good model of the environment as well as good policies for it.

Acknowledgement This work was supported by ARC grant DP120100950.

References

[BLPL07] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *NIPS'2007*. MIT Press, 2007.

[Bot12] M. Botvinick, M.; Toussaint. Planning as inference. *Trends in cognitive sciences*, 16(10):485 – 488, 2012.

[DH97] P. Dayan and G. Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

[DLR77] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Stat. Soc.: B*, 39:1–38, 1977.

[FSG10] N. Fremaux, H. Sprekeler, and W. Gerstner. Functional requirements for reward-modulated spike timing-dependent plasticity. *Journal of Neuroscience*, 30(40):13326–13337, 2010.

[HB04] J. Hawkins and S. Blakeslee. *On Intelligence*. Times Books, 2004.

[Her70] R. J. Herrnstein. On the law of effect. *Journal of the Experimental Analysis of Behavior*, 13:243–266, 1970.

[HOT06] G. Hinton, S. Osindero, and Y.W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[Hut05] M. Hutter. *Universal Articial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2005.

[Kah11] D. Kahneman. *Thinking, fast and slow*. 2011.

[Len80] D. Lenat. The plausible mutation of DNA. Technical report, Standford University, 1980.

[LH07] S. Legg and M. Hutter. Universal Intelligence: A defintion of machine intelligence. *Mind and Machine*, 17:391–444, 2007.

[LPM07] R. Legenstein, D. Pecevski, and W. Maass. Theoretical analysis of learning with reward-modulated spike-timing-dependent plasticity. In *NIPS*, 2007.

[LS06] Y. Loewenstein and S. Seung. Operant matching is a generic outcome of synaptic plasticity based on the covariance between reward and neural activity. *PNAS, 15224–15229*, 103(41), 2006.

[OR12] L. Orseau and M. Ring. Space-time embedded intelligence. In *Artificial General Intelligence*, pages 209–218. Springer Berlin Heidelberg, 2012.

[Pel12] M. Pelikan. Probabilistic model-building genetic algorithms. In *GECCO, 777-804*. ACM, 2012.

[RN10] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 3^{nd} edition, 2010.

[Rus97] Stuart Russell. Rationality and intelligence. *Artificial Intelligence*, 1997.

[Sch07] J. Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial General Intelligence, 199-226*. 2007.

[SH12a] P. Sunehag and M. Hutter. Optimistic agents are asymptotically optimal. In *Proceedings of the 25:th Australasian AI conference*, pages 15–26, 2012.

[SH12b] P. Sunehag and M. Hutter. Optimistic AIXI. In *Proceedings of the 4:th conference on Artifical General Intelligence (AGI'2012)*, pages 312–321, 2012.

[SL14] H. Shteingart and Y. Loewenstein. Reinforcement learning and human behavior. *Current Opinion in Neurobiology*, 25(0):93 – 98, 2014.

[SZW97] J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive Levin search, and incremental self-improvement. *Machine Learning*, 28:105–130, 1997.

[WE03] M.J. West-Eberhard. *Developmental Plasticity and Evolution*. Oxford University Press, USA, 2003.

[Web10] G. Webb. Occam's razor. In *Encl. of Machine Learning*. Springer, 2010.

[WGRT11] D. Wingate, N. Goodman, Kaelbling L. Roy, D., and J. Tenenbaum. Bayesian policy search with policy priors. In *IJCAI, 1565-1570*, 2011.