10 APPROXIMATIONS & APPLICATIONS

- Universal Search
- The Fastest Algorithm (FastPrg)
- Time-Bounded AIXI Model (AIXItl)
- Brute-Force Approximation of AIXI (AIXIξ)
- A Monte-Carlo AIXI Approximation (MC-AIXI-CTW)
- Feature Reinforcement Learning (ΦMDP)
Approximations & Applications: Abstract

Many fundamental theories have to be approximated for practical use. Since the core quantities of universal induction and universal intelligence are incomputable, it is often hard, but not impossible, to approximate them. In any case, having these “gold standards” to approximate (top→down) or to aim at (bottom→up) is extremely helpful in building truly intelligent systems. A couple of universal search algorithms ((adaptive) Levin search, FastPrg, OOPS, Gödel-machine, ...) that find short programs have been developed and applied to a variety of toy problem. The AIXI model itself has been approximated in a couple of ways (AIXItl, Brute Force, Monte Carlo, Feature RL). Some recent applications will be presented.
# Towards Practical Universal AI

Goal: Develop efficient general-purpose intelligent agent

- **Additional Ingredients:**
  - Universal search: Schmidhuber (200X) & al.
  - Information: Wallace, Rissanen
  - Complexity/Similarity: Li & Vitanyi (2008)
  - Optimization: Aarts & Lenstra (1997)
10.1 **Universal Search: Contents**

- Blum’s Speed-up Theorem and Levin’s Theorem.
- The Fastest Algorithm $M_p^*$. 
- Applicability of Levin Search and $M_p^*$.
- Time Analysis of $M_p^*$.
- Extension of Kolmogorov Complexity to Functions.
- The Fastest *and* Shortest Algorithm.
- Generalizations.
- Summary & Outlook.
Introduction

- Searching for fast algorithms to solve certain problems is a central and difficult task in computer science.

- Positive results usually come from explicit constructions of efficient algorithms for specific problem classes.

- A wide class of problems can be phrased in the following way:
  - Find a fast algorithm computing $f : X \rightarrow Y$, where $f$ is a formal specification of the problem depending on some parameter $x$.

- The specification can be formal (logical, mathematical), it need not necessarily be algorithmic.

- Ideally, we would like to have the fastest algorithm, maybe apart from some small constant factor in computation time.
Blum’s Speed-up Theorem (Negative Result)

There are problems for which an (incomputable) sequence of speed-improving algorithms (of increasing size) exists, but no fastest algorithm.

[Blum, 1967, 1971]

Levin’s Theorem (Positive Result)

Within a (large) constant factor, Levin search is the fastest algorithm to invert a function \( g: Y \rightarrow X \), if \( g \) can be evaluated quickly.

[Levin 1973]
**Simple is as fast as Search**

- **Simple**: run all programs \( p_1 p_2 p_3 \ldots \) on \( x \) one step at a time according to the following scheme: \( p_1 \) is run every second step, \( p_2 \) every second step in the remaining unused steps, ... if \( g(p_k(x)) = x \), then output \( p_k(x) \) and halt \( \Rightarrow time_{Simple}(x) \leq 2^k time_{p_k}^+(x) + 2^{k-1}. \)

- **Search**: run all \( p \) of length less than \( i \) for \( \lfloor 2^i 2^{-l(p)} \rfloor \) steps in phase \( i = 1, 2, 3, \ldots \). \( time_{Search}(x) \leq 2^{K(k) + O(1)} time_{p_k}^+(x), \ K(k) \ll k. \)

- **Refined analysis**: Search itself is an algorithm with some index \( k_{Search} = O(1) \)
  \[ \Rightarrow \] SIMPLE executes Search every \( 2^{k_{Search}} \)-th step
  \[ \Rightarrow \] \( time_{Simple}(x) \leq 2^{k_{Search}} time_{Search}^+(x) \)
  \[ \Rightarrow \] SIMPLE and Search have the same asymptotics also in \( k. \)

- **Practice**: Search should be favored because the constant \( 2^{k_{Search}} \) is rather large.
Bound for the Fast Algorithm $M_{p^*}$

- Let $p^*: X \to Y$ be a given algorithm or specification.
- Let $p$ be any algorithm, computing provably the same function as $p^*$ with computation time provably bounded by the function $t_p(x)$.
- $\text{time}_{t_p}(x)$ is the time needed to compute the time bound $t_p(x)$.
- Then the algorithm $M_{p^*}$ computes $p^*(x)$ in time

$$\text{time}_{M_{p^*}}(x) \leq 5 \cdot t_p(x) + d_p \cdot \text{time}_{t_p}(x) + c_p$$

- with constants $c_p$ and $d_p$ depending on $p$ but not on $x$.
- Neither $p$, $t_p$, nor the proofs need to be known in advance for the construction of $M_{p^*}(x)$. 
Applicability

- Prime factorization, graph coloring, truth assignments, ... are Problems suitable for Levin search, if we want to find a solution, since verification is quick.
- Levin search cannot decide the corresponding decision problems.
- Levin search cannot speedup matrix multiplication, since there is no faster method to verify a product than to calculate it.
- Strassen’s algorithm $p'$ for $n \times n$ matrix multiplication has time complexity $time_{p'}(x) \leq t_{p'}(x) := c \cdot n^{2.81}$.
- The time-bound function (cast to an integer) can, as in many cases, be computed very fast, $time_{t_{p'}}(x) = O(\log^2 n)$.
- Hence, also $M_{p^*}$ is fast, $time_{M_{p^*}}(x) \leq 5c \cdot n^{2.81} + O(\log^2 n)$, even without known Strassen’s algorithm.
- If there exists an algorithm $p''$ with $time_{p''}(x) \leq d \cdot n^2 \log n$, for instance, then we would have $time_{M_{p^*}}(x) \leq 5d \cdot n^2 \log n + O(1)$.
- Problems: Large constants $c$, $c_p$, $d_p$. 
**The Fast Algorithm** $M_{p^*}$

$M_{p^*}(x)$

Initialize the shared variables

$L := \{\}, \quad t_{fast} := \infty, \quad p_{fast} := p^*$.

Start algorithms $A$, $B$, and $C$

in parallel with 10%, 10% and 80%

computational resources, respectively.

$A$

Run through all proofs.

if a proof proves for some $(p, t)$ that

$p(\cdot)$ is equivalent to (computes) $p^*(\cdot)$

and has time-bound $t(\cdot)$

then add $(p, t)$ to $L$.

$B$

Compute all $t(x)$ in parallel

for all $(p, t) \in L$ with

relative computation time $2^{-\ell(p) - \ell(t)}$.

if for some $t$, $t(x) < t_{fast}$,

then $t_{fast} := t(x)$ and $p_{fast} := p$.

continue

$C$

for $k := 1, 2, 4, 8, 16, 32, \ldots$ do

run current $p_{fast}$ for $k$ steps

(without switching).

if $p_{fast}$ halts in less than $k$ steps,

then print result and abort $A$, $B$ and $C$.

else continue with next $k$.  

Fictitious Sample Execution of $M_{p^*}$

- content of shared variable $t_{fa}$
- time-bound for $p$ executed by $C$
- number of executed steps of $p$ in $C$
- guaranteed stopping point
Time Analysis

\[ T_A \leq \frac{1}{10\%} \cdot 2^{\ell(\text{proof}(p'))} + 1 \cdot O(\ell(\text{proof}(p'))^2) \]

\[ T_B \leq T_A + \frac{1}{10\%} \cdot 2^{\ell(p')} + \ell(t_{p'}) \cdot \text{time}_{t_{p'}}(x) \]

\[ T_C \leq \begin{cases} 
4T_B & \text{if } C \text{ stops not using } p' \text{ but on some earlier program} \\
\frac{1}{80\%} 4t_{p'} & \text{if } C \text{ computes } p'. 
\end{cases} \]

\[
\text{time}_{M_{p^*}}(x) = T_C \leq 5 \cdot t_p(x) + d_p \cdot \text{time}_{t_p}(x) + c_p
\]

\[ d_p = 40 \cdot 2^{\ell(p) + \ell(t_p)}, \quad c_p = 40 \cdot 2^{\ell(\text{proof}(p))} + 1 \cdot O(\ell(\text{proof}(p))^2) \]
Kolmogorov Complexity

Kolmogorov Complexity is a universal notion of the information content of a string. It is defined as the length of the shortest program computing string $x$.

$$K(x) := \min_p \{\ell(p) : U(p) = x\}$$

[Kolmogorov 1965 and others]

Universal Complexity of a Function

The length of the shortest program provably equivalent to $p^*$

$$K''(p^*) := \min_p \{\ell(p) : \text{a proof of } [\forall y : u(p, y) = u(p^*, y)] \text{ exists}\}$$

[H’00]

$K$ and $K''$ can be approximated from above (are co-enumerable), but not finitely computable. The provability constraint is important.
The Fastest and Shortest Algorithm for $p^*$

Let $p^*$ be a given algorithm or formal specification of a function.

There exists a program $\tilde{p}$, equivalent to $p^*$, for which the following holds

i) $\ell(\tilde{p}) \leq K''(p^*) + O(1)$

ii) $\text{time}_{\tilde{p}}(x) \leq 5 \cdot t_p(x) + d_p \cdot \text{time}_{t_p}(x) + c_p$

where $p$ is any program provably equivalent to $p^*$ with computation time provably less than $t_p(x)$. The constants $c_p$ and $d_p$ depend on $p$ but not on $x$. [H'00]

Proof

Insert the shortest algorithm $p'$ provably equivalent to $p^*$ into $M$, that is $\tilde{p} := M_{p'} \quad \Rightarrow \quad \ell(\tilde{p}) = \ell(p') + O(1) = K''(p^*) + O(1)$. 
Generalizations

- If $p^*$ has to be evaluated repeatedly, algorithm $A$ can be modified to remember its current state and continue operation for the next input ($A$ is independent of $x$!). The large offset time $c_p$ is only needed on the first call.

- $M_{p^*}$ can be modified to handle i/o streams, definable by a Turing machine with monotone input and output tapes (and bidirectional working tapes) receiving an input stream and producing an output stream.

- The construction above also works if time is measured in terms of the current output rather than the current input $x$ (e.g. for computing $\pi$).
Summary

- Under certain provability constraints, $M_{p^*}$ is the asymptotically fastest algorithm for computing $p^*$ apart from a factor 5 in computation time.

- The fastest program computing a certain function is also among the shortest programs provably computing this function.

- To quantify this statement we defined a novel natural measure for the complexity of a function, related to Kolmogorov complexity.

- The large constants $c_p$ and $d_p$ seem to spoil a direct implementation of $M_{p^*}$.

- On the other hand, Levin search has been successfully extended and applied even though it suffers from a large multiplicative factor [Schmidhuber 1996-2004].
Outlook

• More elaborate theorem-provers could lead to smaller constants.

• Transparent or holographic proofs allow under certain circumstances an exponential speed up for checking proofs [Babai et al. 1991].

• Will the ultimate search for asymptotically fastest programs typically lead to fast or slow programs for arguments of practical size?
10.2 Approximations & Applications of AIXI: Contents

- Time-Bounded AIXI Model (AIXItl)
  (theoretical guarantee)

- Brute-Force Approximation of AIXI (AIξ)
  (application to 2×2 matrix games)

- A Monte-Carlo AIXI Approximation (MC-AIXI-CTW)
  (application to mazes, tic-tac-toe, pacman, poker)
Computational Issues

- If $\mathcal{X}$, $\mathcal{Y}$, $m$, $\mathcal{M}$ finite, then $\xi$ and $p^\xi$ (theoretically) computable.

- $\xi$ and hence $p^\xi$ incomputable for infinite $\mathcal{M}$, as for Solomonoff’s prior $\xi_U$.

- Computable approximations to $\xi_U$:
  - Time bounded Kolmogorov complexity $K^t$ or $K^t$.
  - Time bounded universal prior like speed prior $S$ [Schmidhuber:02].

- Even for efficient approximation of $\xi_U$, exponential (in $m$) time is needed for evaluating the expectimax tree in $V^*_\xi$.

- Additionally perform Levin search through policy space, similarly to OOPS+AIXI [Schmidhuber:02].

- Approximate $V^*_\xi$ directly: AIXItl [Hutter:00].
**Computability and Monkeys**

SP$\xi$ and AI$\xi$ are not really uncomputable (as often stated) but ...

$\dot{y}_k^{AI}\xi$ is only asymptotically computable/approximable with slowest possible convergence.

**Idea of the typing monkeys:**

- Let enough monkeys type on typewriters or computers, eventually one of them will write Shakespeare or an AI program.

- To pick the right monkey by hand is cheating, as then the intelligence of the selector is added.

- **Problem:** How to (algorithmically) select the right monkey.
The Time-bounded AIXI Model

- Let p be any (extended chronological self-evaluating) policy
- with length \( \ell(p) \leq l \) and computation time per cycle \( t(p) \leq t \)
- for which there exists a proof of length \( \leq l_P \) that p is a valid approximation of (not overestimating) its true value \( V^* \).
- AIXI_{tl} selects such p with highest self-evaluation.

Optimality of AIXI_{tl}

- AIXI_{tl} depends on \( l, t \) and \( l_P \) but not on knowing p.
- It is effectively more or equally intelligent w.r.t. intelligence order relation \( \succeq^c \) than any such p.
- Its size is \( \ell(p^{best}) = O(\log(l \cdot t \cdot l_P)) \).
- Its setup-time is \( t_{setup}(p^{best}) = O(l_P^2 \cdot 2^{l_P}) \).
- Its computation time per cycle is \( t_{cycle}(p^{best}) = O(2^l \cdot t) \).
Outlook

• Adaptive Levin-Search (Schmidhuber 1997)

• The Optimal Ordered Problem Solver (Schmidhuber 2004) (has been successfully applied to Mazes, towers of hanoi, robotics, ...)

• The Gödel Machine (Schmidhuber 2007)

• Related fields: Inductive Programming
Brute-Force Approximation of AIXI

• **Truncate expectimax tree** depth to a small fixed lookahead $h$. Optimal action computable in time $|\mathcal{Y} \times \mathcal{X}|^h \times$ time to evaluate $\xi$.

• Consider mixture over **Markov Decision Processes** (MDP) only, i.e.
  $$\xi(x_1:m | y_1:m) = \sum_{\nu \in \mathcal{M}} w_\nu \prod_{t=1}^m \nu(x_t | x_{t-1} y_t).$$  
  Note: $\xi$ is *not* MDP

• Choose **uniform prior** over $w_\mu$.
  Then $\xi(x_1:m | y_1:m)$ can be computed in linear time.

• Consider (approximately) Markov problems
  with very small action and perception space.

• **Example application**: $2 \times 2$ Matrix Games like Prisoner’s Dilemma,
  Stag Hunt, Chicken, Battle of Sexes, and Matching Pennies. [PH06]
AIXI Learns to Play $2 \times 2$ Matrix Games

- Repeated prisoners dilemma.
- Game unknown to AIXI. Must be learned as well.
- AIXI behaves appropriately.

<table>
<thead>
<tr>
<th>Loss matrix</th>
<th>cooperates</th>
<th>defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>cooperates</td>
<td>=0.3 years</td>
<td>=1 year</td>
</tr>
<tr>
<td>defects</td>
<td>free</td>
<td>=0.7 years</td>
</tr>
</tbody>
</table>

![Graph showing average probability of cooperation ratio over time](chart.png)
A Monte-Carlo AIXI Approximation

Consider class of Variable-Order Markov Decision Processes.

The Context Tree Weighting (CTW) algorithm can efficiently mix (exactly in essentially linear time) all prediction suffix trees.

Monte-Carlo approximation of expectimax tree: Upper Confidence Tree (UCT) algorithm:

- **Sample** observations from CTW distribution.
- **Select** actions with highest upper confidence bound.
- **Expand** tree by one leaf node (per trajectory).
- **Simulate** from leaf node further down using (fixed) playout policy.
- **Propagate back** the value estimates for each node.

Repeat until timeout. [VNH+11]

Guaranteed to converge to exact value.

**Extension:** Predicate CTW not based on raw obs. but features thereof.
Monte-Carlo AIXI Applications

![Graph showing normalized average reward per cycle for different games]

[Joel Veness et al. 2009]
Extensions of MC-AIXI-CTW [VSH12]

• **Smarter** than random playout policy, e.g. learnt CTW policy.

• **Extend the model class** to improve general prediction ability. However, not so easy to do this in a comput. efficient manner.

• **Predicate CTW**: Context is vector of (general or problem-specific) predicate=feature=attribute values.

• **Convex Mixing** of predictive distributions. Competitive guarantee with respect to the best fixed set of weights.

• **Switching**: Enlarge base class by allowing switching between distr. Can compete with best rarely changing sequence of models.

• **Improve underlying KT Est.**: Adaptive KT, Window KT, KT0, SAD

• **Partition Tree Weighting** technique for piecewise stationary sources with breaks at/from a binary tree hierarchy.

• **Mixtures of factored models such as quad-trees for images**. [BVB13]
10.3 Feature Reinforcement Learning: Contents

- Markov Decision Processes (MDPs)
- The Main Idea: Map Real Problem to MDP
- Criterion to Evaluate/Find/Learn Map the Automatically
- Algorithm & Results
Feature Reinforcement Learning (FRL)

Goal: Develop efficient general purpose intelligent agent. [Hut09b]

State-of-the-art: (a) AIXI: Incomputable theoretical solution.
(b) MDP: Efficient limited problem class.
(c) POMDP: Notoriously difficult. (d) PSRs: Underdeveloped.

Idea: $\Phi$MDP reduces real problem to MDP automatically by learning.

Accomplishments so far: (i) Criterion for evaluating quality of reduction. [Hut09c]
(ii) Integration of the various parts into one learning algorithm. [Hut09c]
(iii) Generalization to structured MDPs (DBNs). [Hut09a]
(iv) Theoretical and experimental investigation. [SH10, DSH12, Ngu13]

$\Phi$MDP is promising path towards the grand goal & alternative to (a)-(d)

Problem: Find reduction $\Phi$ efficiently (generic optimization problem?)
Markov Decision Processes (MDPs)

a computationally tractable class of problems

- MDP Assumption: State $s_t := o_t$ and $r_t$ are probabilistic functions of $o_{t-1}$ and $a_{t-1}$ only.

- Further Assumption:
  State = observation space $S$ is finite and small.

- Goal: Maximize long-term expected reward.

- Learning: Probability distribution is unknown but can be learned.

- Exploration: Optimal exploration is intractable but there are polynomial approximations.

- Problem: Real problems are not of this simple form.
Map Real Problem to MDP

Map history $h_t := o_1a_1r_1...o_{t-1}$ to state $s_t := \Phi(h_t)$, for example:

Games: Full-information with static opponent: $\Phi(h_t) = o_t$.

Classical physics: Position+velocity of objects = position at two time-slices: $s_t = \Phi(h_t) = o_to_{t-1}$ is (2nd order) Markov.

I.i.d. processes of unknown probability (e.g. clinical trials ≍ Bandits), Frequency of obs. $\Phi(h_n) = (\sum_{t=1}^n \delta_{o_to})_{o\in\mathcal{O}}$ is sufficient statistic.

Identity: $\Phi(h) = h$ is always sufficient, but not learnable.

Find/Learn Map Automatically

$\Phi^{best} := \arg\min_\Phi \text{Cost}(\Phi|h_t)$

- What is the best map/MDP? (i.e. what is the right Cost criterion?)
- Is the best MDP good enough? (i.e. is reduction always possible?)
- How to find the map $\Phi$ (i.e. minimize Cost) efficiently?
ΦMDP: Computational Flow

Transition Pr. $\hat{T}$
Reward est. $\hat{R}$

exploration bonus

frequency estimate

Bellman

Feature Vec. $\Phi$

Cost($\Phi|h$) minimization

History $h$

$(\hat{Q})$ Value

implicit

Best Policy $\hat{p}$

reward $r$ observation $o$

Environment

action $a$
**ΦMDP Results**

- **Theoretical guarantees:** Asymptotic consistency. [SH10]

- **Example Φ-class:** As Φ choose class of suffix trees as in CTW.

- **How to find/approximate Φ_{best}:**
  - Exhaustive search for toy problems [Ngu13]
  - Monte-Carlo (Metropolis-Hastings / Simulated Annealing) for approximate solution [NSH11]
  - Exact “closed-form” by CTM similar to CTW [NSH12]

- **Experimental results:** Comparable to MC-AIXI-CTW [NSH12]

- **Extensions:**
  - Looping suffix trees for long-term memory [DSH12]
  - Structured/Factored MDPs (Dynamic Bayesian Networks) [Hut09a]
Literature


