# Jürgen Schmidhuber - The New AI: General & Sound & Relevant for Physics (2007)

Tom Rochette <tom.rochette@coreteks.org>

November 2, 2024 — 36c8eb68

### 0.1 Context

### 0.2 Learned in this study

### 0.3 Things to explore

## 1 Overview

### 1.1 1 Introduction

- Once there is an optimal, formally describable way of predicting the future, we should be able to construct a machine that continually computes and executes action sequences that maximize expected or predicted reward

### 1.2 2 More Formally

- Most previous work has focused on very limited settings, such as Markovian environments, where the optimal next action, given past inputs, depends on the current input only
- We make a much weaker and therefore much more general assumption, namely, that the environment's responses are sampled from a computable probability distribution
- $B^*$: the set of finite sequences over the binary alphabet $B = \{0, 1\}$
- $B^\infty$: the set of infinite sequences over $B$
- $\lambda$: the empty string
- $B^\sharp = B^* \cup B^\infty$
- $x, y, z, z^1, z^2$: strings in $B^\sharp$
- $P(xy|x) = \frac{P(x|xy)P(xy)}{P(x)} \propto P(xy)$

### 1.3 4 Super Omegas and Generalizations of Kolmogorov Complexity & Algorithmic Probability

- An object $X$ is formally describable if a finite amount of information completely describes $X$ and only $X$
- $X$ should be representable by a possibly infinite bitstring $x$ such that there is a finite, possibly never halting program $p$ that computes $x$ and nothing but $x$ in a way that modifies each output bit at most finitely many times
- The "true" information content of some (possibly infinite) bitstring $x$ actually is the size of the shortest nonhalting program that converges to $x$, and nothing but $x$, on a Turing machine that can edit its previous outputs

## 1.4 5 Computable Predictions Through the Speed Prior Based on the Fastest Way of Describing Objects

- Postulate 1: The cumulative prior probability measure of all x incomputable within time t by any method is at most inversely proportional to t

## 1.5 6 Speed Prior-Based Predictions for Our Universe

- We make a stronger assumption by adopting Zuse's thesis, namely, that the very universe is actually being computed deterministically
- To compute our universe's precise algorithm:
  - Systematically create and execute all programs for a universal computer, such as a Turing machine or a CA (cellular automaton); the first program is run for one instruction every second step on average, the next for one instruction every second of the remaining steps on average, and so on
- All computable universes, including ours and ourselves as observers, are computed by the very short program that generates and executes all possible programs (in other words, any longer program is built recursively from this shorter program)

## 1.6 8 Optimal Universal Search Algorithms

- Define a probability distribution P on a finite or infinite set of program for a given computer
- P represents the searcher's initial bias
- Method Lsearch (Levin Search):
  - Set current time limit T = 1
  - While problem not solved
    - Test all programs q such that t(q), the maximal time spent on creating and running and testing q, satisfies t(q) < P(q) x T
    - Set T = 2T
- Adaptive Lsearch: Whenever Lsearch finds a program q that computes a solution for the current problem, q's probability P(q) is substantially increased using a "learning rate", while probabilities of alternative programs decrease appropriately

## 1.7 9 Optimal Ordered Problem Solver (OOPS)

- **Bias-optimal searchers:** Given is a problem class $\mathcal{R}$, a search space $\mathcal{C}$ of solution candidates (where any problem $r \in \mathcal{R}$ should have a solution in $\mathcal{C}$), a task dependent bias in form of conditional probability distribution $P(q|r)$ on the candidates $q \in \mathcal{C}$, and a predefined procedure that creates and tests any given $q$ on any $r \in \mathcal{R}$ within time $t(q, r)$ (typically unknown in advance).
  - A searcher is n-bias-optimal ($n \geq 1$) if for any maximal total search $T_{max} > 0$ it is guaranteed to solve any problem $r \in \mathcal{R}$ if it has a solution $p \in \mathcal{C}$ satisfying $t(p, r) \leq \frac{P(p|r)T_{max}}{n}$. It is bias-optimal if n = 1.
- Primitives: User-defined primitive behaviors represented by a token. Must be interruptible at any time
  - Ex: theorem provers, matrix operators for neural network-like parallel architectures, trajectory generators for robot simulations, state update procedures for multiagent systems
- Task-specific prefix codes: Complex behaviors are represented by token sequences or programs. OOPS tries to sequentially compose an appropriate complex behavior from primitive ones. Programs are grown incrementally, token by token; their beginnings/prefixes are immediately executed while being created; this may modify some task-specific internal state or memory, and may transfer control back to previously selected tokens
  - This procedure yields task-specific prefix codes on program space: with any given task, programs that halt because they have found a solution or encountered some error cannot request any more tokens
- Access to previous solutions: Let $p^n$ denote a found prefix solving the first $n$ tasks. The search for $p^{n+1}$ may greatly profit from the information conveyed by (or the knowledge embodied by) $p^1, p^2, ..., p^n$

which are stored or frozen in special nonmodifiable memory shared by all tasks, such that they are accessible to $p^{n+1}$.
- Bias: The searcher's initial bias is embodied by initial, used-defined, task dependent probability distributions on the finite or infinite search space of posible program prefixes. In the simplest case we start with a maximum entropy distribution on the tokens, and define prefix probabilities as the products of the probabilities of their tokens.
- Two searches: Search exhaustively through all possible prefixes on all tasks up to $n + 1$. The second search is much more focused; it only searches for prefixes that start with $p^n$, and only tests them on task $n + 1$, which is safe, because we already know that such prefixes solve all tasks up to $n$.

## 1.8 10 OOPS-Based Reinforcement Learning

- A reinforcement learner will try to find a policy (a strategy for future decision making) that maximizes its expected future reward
- In many applications, the policy that works best in a given set of training trials will also be optimal in future test trials
- AIXI is far more general than traditional RL approaches
- However, AIXI is not practically feasible
- Two OOPS modules are needed
  - The first is called the predictor or world model
  - The second is an action searcher using the world model
  - The life of the entire system consists of a sequence of cycles
  - At each cycle, a limited amount of computation time will be available to each module
  - For simplicity we assume that during each cycle the system may take exactly one action
- At any given cycle, the system executes the following procedure:
  - For a time interval fixed in advance, the predictor is first trained in bias-optimal fashion to find a better world model, that is, a program that predicts the inputs from the environment (including rewards, if there are any), given a history of previous observations and actions. The n-th task of the first OOPS module is to find (if possible) a better predictor than the best found so far.
  - Using the current world model/prediction program found by the first OOPS module, the second module will search for a future action sequence that maximizes the predicted cumulative reward (up to some time limit). The n-th task of the second OOPS module will be to find a control program that computes a control sequence of actions, to be fed into the program representing the current world model (whose input predictions are successively fed back to itself), such that this control sequence leads to higher predicted reward than the one generated by the best control program found so far.
  - After the current cycle's time for control program search is finished, we will execute the current action of the best control program found in the previous step.
  - A new cycle begins.

## 1.9 11 The Gödel Machine

- Designed to solve arbitrary computational problems beyond those solvable by plain OOPS
- While executing some arbitrary initial problem solving strategy, the Gödel machine simultaneously runs a proof searcher which systematically and repeatedly tests proof techniques
- Proof techniques are programs that may read any part of the Gödel machine's state, and write on a reserved part which may be reset for each new proof technique test
- This writable storage includes the variables proof and switchprog
- switchprog holds a potentially unrestricted program whose execution could completely rewrite any part of the Gödel machine's current software
- check(): Tests whether proof currently holds a proof showing that the utility of stopping the systematic proof searcher and transferring control to the current switchprog at a particular point in the near future exceeds the utility of continuing the search until some alternative switchprog is found

## 2 See also

## 3 References


- arXiv:cs/0302012 [cs.AI]
- DOI: 10.1007/978-3-540-68677-4_6