Frontier Search

Sun Yi, Tobias Glasmachers, Tom Schaul, and Jürgen Schmidhuber
IDSIA, University of Lugano
Galleria 2, Manno, CH6928, Switzerland

Abstract

How to search the space of programs for a code that solves a given problem? Standard asymptotically optimal Universal Search orders programs by Levin complexity, implementing an exponential trade-off between program length and runtime. Depending on the problem, however, sometimes we may have a good reason to greatly favor short programs over fast ones, or vice versa. *Frontier Search* is a novel framework applicable to a wide class of such trade-offs between program size and runtime, and in many ways more general than previous work. We analyze it in depth and derive exact conditions for its applicability.

Introduction

In an inversion problem, the aim is to find a program p that produces a desired output x. Algorithms that search the space of programs for p are guided (implicitly or explicitly) by an optimality criterion, which is generally based on program length and runtime. Levin complexity, a criterion where the trade-off between program length and runtime is exponential, can readily be optimized using Levin Search (Lev73). The framework of 'speed priors' (Sch02) results in a more flexible search scheme. The aim of this paper is to develop a search scheme applicable to an even wider class of user-defined optimality criteria.

More formally, consider a programming language L and a (countable) set $\mathscr{P}$ of programs. Let $p : \mathbb{N} \to \mathscr{P}$ be an enumeration of $\mathscr{P}$. We refer to the i-th program as p_i. Then, Levin Search finds $p \in \mathscr{P}$ such that $L(p) = x$. It works by executing in parallel all programs in $\mathscr{P}$ such that the fraction of time allocated to the i-th program is $2^{-l(p_i)}/S$, where $l(p_i)$ is the length of a *prefix-free* binary encoding of p_i, and $0 < S \leq 1$ is a normalization constant. Alternatively, a growing number of programs can be executed for a fixed exponentially growing time one after the other, which involves restarting the programs several times. This simpler algorithm performs worse only by a constant factor.

Levin Search, though simple in its form, enjoys two important theoretical properties. The first property concerns *the time required to find a solution*. It is guaranteed that Levin Search solves the inversion problem within time $2^{l(p^\star)+1} \cdot S \cdot \tau(p^\star)$, where $p^\star \in \mathscr{P}$ is the fastest program that solves the problem, and $\tau(p^\star)$ is the number of time steps after which $p^\star$ halts. Since $p^\star$ depends solely on the problem itself, one can claim that Levin Search solves the problem in time linear to the runtime of the fastest program available, despite the prohibitively large multiplicative constant.

The second property, on the other hand, characterizes *the quality of the solution*. It has been shown that the program found by Levin Search (asymptotically) optimizes the Levin complexity K_t defined as

$$K_t(x) = \min_{p \in \mathscr{P}} \left\{ l(p) + \log \tau(p) \;\middle|\; L(p) = x \right\} \;,$$

which is a computable, time-bounded version of the Kolmogorov complexity (LV93). Note that in this paper, all logarithms are to base 2.

Whereas the linear time bound property of Levin Search receives considerable appreciation, less attention is paid to the quality of the solution. In general, solution quality is measured by the complexity function. Thus, a particular search scheme such as Levin Search implies a complexity function it (asymptotically) minimizes. In this paper we approach the problem from the other end, assuming that a complexity function is given, but not a search scheme. The central question asked in this paper is:

Given a certain optimiality criterion,
how do we search the space of programs?

The remainder of the paper is structured as follows. First we discuss the space of possible complexity criteria, then we introduce our algorithm, Frontier Search, and give exact conditions on its applicability. We find that this approach allows for optimality criteria that are more flexible than the speed prior. Finally we present an approximatation to Frontier Search that achieves asymtotically constant overhead complexity.

Generalized Complexity Criteria

Let us first focus on the form of K_t. Assume both p_1 and p_2 solve the problem $L(p) = x$ and achieve the same value of $l(p) + \log(\tau(p))$. If p_1 is m bits shorter than p_2, the execution time of p_1 would be 2^m times larger than for p_2. This encodes an inherent trade-off between the program execution time and its length, namely, *how much more time we are willing to invest for finding a solution which is 1 bit shorter*. In the remainder of this paper we replace the concept of program length with program order in the sense

of the enumeration $p : \mathbb{N} \to \mathscr{P}$. The familiar length encoding can be recovered by enumerating programs by increasing length.

Now consider the following three scenarios:

1. We are trying to find a relatively tight upper bound on the Kolmogorov complexity of a string x. This amounts to finding a concise representation for a given x, and the length of the program found matters much more than its execution time. In this case, we might choose a different complexity criterion instead of K_t which emphasizes the program length more, for example,
$$K_1(x) = \min\left\{[l(p)]^s + \log(\tau(p)) \;\middle|\; L(p) = x\right\}$$
with $s > 1$. (In the limit $s \to \infty$ we get Kolmogorov complexity (LV93). Unfortunately, it is incomputable.)
2. We are searching for a representation of x which is assumed to be used a lot in the future, amounting to executing the resulting program p regularly. We may argue that quicker programs are prefered despite their slightly longer length since they will be executed often. In this case, the complexity criterion
$$K_2(x) = \min\left\{[l(p)]^{1/s} + \log(\tau(p)) \;\middle|\; L(p) = x\right\}$$
with $s > 1$, which favours quicker programs, makes more sense.
3. We have prior knowledge telling us that programs with a certain structure (in the simplest case, programs of a certain length) should be prefered, and we would like to encode such knowledge into the complexity criterion. An extreme example is that we do not want to run programs of trivial length (e.g., $l(p) = 1$) for half of the total running time as suggested in Levin Search. (Certainly, such prior knowledge can be incorporated into the programming language itself, but that necessitates re-designing the language every time we vary the requirement (SS10).)

All these scenarios call for a more general approach: We want our search to respect a complexity criterion suitable for the problem at hand. *Starting from a complexity criterion which encodes the desired trade-off between execution time and program order, we build up a search algorithm that finds the optimal solution in the sense of the given complexity criterion.* The search algorithm should be invariant w.r.t. any monotonically increasing (i.e., order preserving) transformation of the complexity criterion, since the program minimizing $l(p) + \log(\tau(p))$ would also minimize $2^{l(p)} \cdot \tau(p)$, or in general, $f(l(p) + \log(\tau(p)))$ for any monotonically increasing function $f : \mathbb{R} \to \mathbb{R}$.

Our answer to the problem above is a simple search algorithm called *Frontier Search*. It maintains a 'frontier' of the possible execution steps and at each iteration selects the one minimizing the given complexity criterion. We prove that under reasonable technical constraints on the complexity criterion Frontier Search indeed finds the optimal program. Also, we show the connection between Frontier Search and Levin Search, as well as universal search with 'speed prior' (Sch02), and demonstrate that Frontier Search is more general since it allows the encoding of speed preferences which cannot be represented using the speed prior approach.

Frontier Search

We consider the general complexity criterion

$$K_\psi(x) = \min_{i \in \mathbb{N}} \left\{ \psi(i, \tau_i) \;\middle|\; L(p_i) = x \right\} \;,$$

where τ_i is the execution time of p_i, and $\psi : \mathbb{N} \times \mathbb{N} \to \mathbb{R}$ is a complexity function encoding the trade-off between program length and execution time. For example, for the choice $\psi(i, \tau) = 2^i \cdot \tau$, we recover Levin Search under the trivial encoding $p_i = \texttt{1}\cdots\texttt{10}$ (i ones, one zero). Futhermore, $\psi(i, \tau) = \tau/\pi_i$, with $\pi_i > 0$ and $\sum_{i \in \mathbb{N}} \pi_i = 1$, encodes universal search based on the speed prior π (Sch02).

Algorithm 1 presents the pseudocode for Frontier Search, and Figure 1 illustrates its operation. The set $\left\{(i, \tau_i) \middle| i \in \{1, \ldots, n\}\right\} \subset \mathbb{N} \times \mathbb{N}$ with $\tau_n = 1$ forms the current 'frontier', i.e., available executions in the next time step. If program p_n gets executed, then the frontier automatically expands to include a new program p_{n+1}. We assume that for multiple j minimizing $\psi(j, \tau_j + 1)$ the smallest j is chosen.

Algorithm 1: Frontier Search.

Input: ψ, L, x, $\mathscr{P}$
Output: $p \in \mathscr{P}$ such that $L(p) = x$
$n \leftarrow 1$;
$\tau_n \leftarrow 0$;
while *true* **do**
  $i \leftarrow \arg\min \left\{ \psi(j, \tau_j + 1) \;\middle|\; j \in \{1, \ldots, n\} \right\}$;
  execute p_i for 1 step;
  if p_i halts and $L(p_i) = x$ **then return** p_i;
  $\tau_i \leftarrow \tau_i + 1$;
  if $i = n$ **then**
    $n \leftarrow n + 1$;
    $\tau_n \leftarrow 0$;
  end
end

The following definitions will prove handy for the analysis of Frontier Search.

Definition 1. *Formally, the set of all possible frontiers is given by*

$$\mathscr{F} = \Big\{ \{(1, \tau_1), \ldots, (n-1, \tau_{n-1}), (n, 1)\} \;\Big|\; n \in \mathbb{N} \text{ and } \tau_i \in \mathbb{N} \;\; \forall i \in \{1, \ldots, n-1\} \Big\} \quad \cong \bigcup_{n \in \mathbb{N}} \mathbb{N}^{n-1} \;.$$

For a given frontier $F = \{(1, \tau_1), \ldots, (n, 1)\} \in \mathscr{F}$ we say that the grid points $(i, \tau) \in F$ are *on* the frontier, points (i, τ) with $i < n$ and $\tau < \tau_i$ are *inside* the frontier, and all other grid points are *outside* the frontier, see also Figure 1. The points inside the frontier correspond to the program steps already executed by Frontier Search in order to reach the current frontier.

For any given frontier there exists a complexity function ψ that makes Frontier Search indeed reach this frontier. A simple choice is to set ψ to $1/2$ for all points inside the frontier, and to $i + \tau$ for all other points.

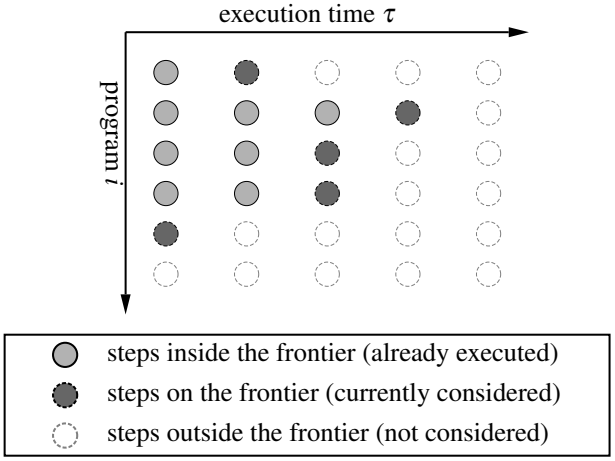

Figure 1: Illustration of Frontier Search. In each iteration, a frontier of possible execution steps $(i, \tau_i + 1)$, i.e., executing the $(\tau_i + 1)$-th command of program p_i, is maintained. The step which minimizes $\psi(i, \tau_i + 1)$ is executed.

Definition 2. *We define the partial order relation*

$$\{(1,\tau_1),\dots,(n,1)\} \leq \{(1,\tau_1'),\dots,(n',1)\}$$
$$\Leftrightarrow\ n \leq n' \text{ and } \tau_i \leq \tau_i' \text{ for all } i \in \{1,\dots,n\}$$

on the set $\mathscr{F}$ of frontiers.

In this canonical order relation it holds $F \leq F'$ if and only if the points inside F are a subset of the points inside F'. Thus, for each complexity function ψ Frontier Search generates a strictly growing sequence $(F_t^{\psi})_{t\in\mathbb{N}}$ of frontiers.

Definition 3. *For a frontier $F = \{(1,\tau_1),\dots,(n,1)\} \in \mathscr{F}$ we define the time $T(F) = \sum_{i=1}^{n-1}(\tau_i - 1)$ necessary for frontier search to reach this frontier.*

The identity $T(F_t^{\psi}) = t$ is obvious.

Definition 4. *Assume step $\tau+1$ of program p_i is executed by Frontier Search with complexity function ψ in finite time. Then we associate the frontier*

$$F_{(i,\tau)}^{\psi} = \max\left\{F_t^{\psi} \;\middle|\; t \in \mathbb{N} \text{ and } (i,\tau) \in F_t^{\psi}\right\}$$

with the tuple (i,τ).

Let us introduce a useful auxiliary property of complexity functions:

Definition 5. *We say that a complexity function $\psi : \mathbb{N}\times\mathbb{N} \to \mathbb{R}$ is* frontier-bounded *if for any $(i,\tau) \in \mathbb{N}\times\mathbb{N}$ there exist $n > i$ and $(\tau_1,\dots,\tau_{i-1},\tau_{i+1},\dots,\tau_{n-1}) \in \mathbb{N}^{n-2}$, such that*

$$\begin{aligned}
\psi(j,\tau_j) &> \psi(i,\tau) \quad \forall\, j \in \{1,\dots,i-1\}\\
\psi(j,\tau_j) &\geq \psi(i,\tau) \quad \forall\, j \in \{i+1,\dots,n-1\}\\
\psi(n,1) &\geq \psi(i,\tau)\ .
\end{aligned}$$

Note that only the first of the three inequalities is strict. Intuitively, the definition states that for each (i,τ) there exists a frontier $\left\{(i,\tau_i)\,\middle|\, i \in \{1,\dots,n\}\right\} \ni (i,\tau)$ containing this tuple, leading to the execution of step τ of program p_i in the next iteration.

The following statement provides us with two simple criteria implying frontier-boundedness.

Proposition 6. *A complexity function $\psi : \mathbb{N}\times\mathbb{N} \to \mathbb{R}$ fulfilling one of the properties*

$$\left|\left\{(i',\tau') \in \mathbb{N}\times\mathbb{N} \;\middle|\; \psi(i',\tau') \leq \psi(i,\tau)\right\}\right| < \infty \quad \forall (i,\tau) \in \mathbb{N}\times\mathbb{N} \tag{1}$$

or

$$\lim_{i\to\infty} \psi(i,1) = \infty \quad \text{and} \quad \lim_{\tau\to\infty} \psi(i,\tau) = \infty\ \ \forall i \in \mathbb{N} \tag{2}$$

is frontier-bounded.

Proof. Case (1): For fixed $(i,\tau) \in \mathbb{N}\times\mathbb{N}$ consider the set $S = \{(i',\tau') \in \mathbb{N}\times\mathbb{N} \,|\, \psi(i',\tau') \leq \psi(i,\tau)\}$. We define $n = 1 + \max\{i \in \mathbb{N} \,|\, \exists \tau \in \mathbb{N} \text{ such that } (i,\tau) \in S\}$ as well as $\tau_i = 1 + \max\{\tau \in \mathbb{N} \text{ such that } (i,\tau) \in S\}$ for all $i \in \{1,\dots,i-1\} \cup \{i+1,\dots,n-1\}$. All maxima exist because S is finite per assumption, and using the convention $\max(\emptyset) = 0$.
Case (2): Again we fix $(i,\tau) \in \mathbb{N}\times\mathbb{N}$. From $\lim_{n\to\infty}\psi(n,1) = \infty$ we conclude that there exists $n \in \mathbb{N}$ such that $\psi(n,1) \geq \psi(i,\tau)$. Now for all $j \in \{1,\dots,i-1\} \cup \{i+1,\dots,n-1\}$ we have $\lim_{\tau_j\to\infty}\psi(j,\tau_j) = \infty$, from which we conclude the existence of τ_j such that $\psi(j,\tau_j) \geq \psi(i,\tau)$.
By construction in both cases the frontier size $n \in \mathbb{N}$ and the tuples $(\tau_1,\dots,\tau_{i-1},\tau_{i+1},\dots,\tau_{n-1})$ fulfill the conditions of Definition 5. □

The next proposition clarifies the significance of frontier-boundedness, namely that this property guarantees that Fontier Search executes every program for sufficiently many steps.

Proposition 7. *Frontier Search applied to a complexity function ψ executes program p_i for τ steps in finite time for all $(i,\tau) \in \mathbb{N}\times\mathbb{N}$ iff ψ is frontier-bounded.*

Proof. ($\Leftarrow$) Assume ψ is frontier-bounded and fix $(i,\tau) \in \mathbb{N}\times\mathbb{N}$. Then we define $n = \min\{n' \in \mathbb{N} \,|\, n' > i \text{ and } \psi(n',1) \geq \psi(i,\tau)\}$, which is well-defined due to the frontier-boundedness of ψ. Accordingly we define $\tau_j = \min\{\tau' \in \mathbb{N} \,|\, \psi(j,\tau') > \psi(i,\tau)\}$ for each $j \in \{1,\dots,i-1\}$ and $\tau_j = \min\{\tau' \in \mathbb{N} \,|\, \psi(j,\tau') \geq \psi(i,\tau)\}$ for all $j \in \{i+1,\dots,n-1\}$, which are all well-defined (none of the arguments of the min-operator is empty) due to ψ being frontier-bounded. When starting from the corresponding frontier $F = \{(1,\tau_1),\dots,(n-1,\tau_{n-1}),(n,1)\}$, Frontier Search executes program p_i in the next step. Obviously it is impossible for Frontier Search to pass any point of this frontier without executing (i,τ_i). The search can spend only $T(F) < \infty$ steps before reaching this frontier. Thus, step τ_i of program p_i is executed in step $T(F)+1 < \infty$. As an aside, this argument shows $F = F_{(i,\tau)}^{\psi}$. In other words, the frontier $F_{(i,\tau)}^{\psi}$ associated with (i,τ) can be constructed as described above.
($\Rightarrow$) For $(i,\tau) \in \mathbb{N}\times\mathbb{N}$ let $F_{(i,\tau)}^{\psi} = \{(1,\tau_1),\dots,(n,1)\}$ denote the associated frontier. Then n and $(\tau_1,\dots,\tau_{i-1},\tau_{i+1},\dots,\tau_{n-1})$ per construction fulfill the requirements of Definition 5. □

Corollary 8. *Assume there exists $p \in \mathscr{P}$ with $L(p) = x$. Then Frontier Search applied to a frontier-bounded complexity function ψ halts.*

If ψ is not frontier-bounded some steps never get executed. Let us have a look at two illustrative counter examples: First consider $\psi(i,\tau) = \tau - 1/i$. This complexity function is not frontier-bounded since for all $n \in \mathbb{N}$ we have $\psi(n,1) = 1 - 1/n < 1 = \psi(1,2)$. In this case, Frontier Search executes every program only for a single step. Second, consider $\psi(i,\tau) = i - 1/\tau$, which is not frontier-bounded since $\psi(1,\tau) < \psi(2,1)$ for all $\tau \in \mathbb{N}$. With this ψ, Frontier Search executes the first program forever (provided that it doesn't halt). The same behavior results for constant $\psi(i,\tau)$, or for $\psi(i,\tau) = l(p_i)$ corresponding to Kolmogorov complexity.

Assume $\psi(i,\tau)$ is non-decreasing in τ. Intuitively, the proceeding of frontier search can be understood by a mechanical picture. Consider a landscape on top of the positive quadrant of the plane, with grid altitude profile given by ψ. The execution of Frontier Search amounts to flooding water into the landscape at the origin, such that exactly one integer square is flooded in each iteration, corresponding to the next program step executed. See Figure 2 for an illustration.

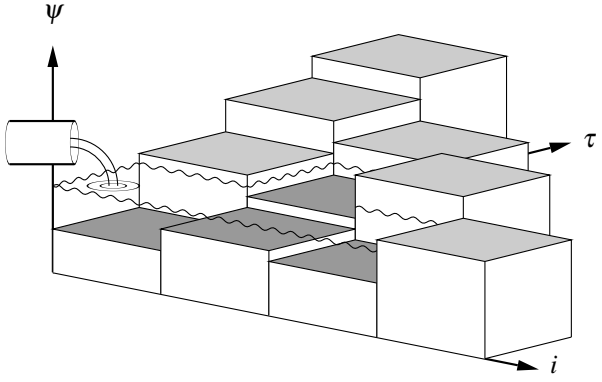

Figure 2: The operation of Frontier Search can be thought of as filling water into the landscape given by $\psi(i,\tau)$, which in this case must be monotonic in τ.

Since ψ serves as a complexity criterion, it is reasonable to assume that quicker programs are always preferred. If possible, we will further assume that programs are preordered by complexity. This leads us to the definition of two handy conditions on complexity functions ψ:

Definition 9. *We say that a complexity function ψ is* proper *if it is frontier-bounded and fulfills the monotonicity conditions*

$$\begin{aligned} \psi(i,\tau) &\leq \psi(i,\tau+1) \;\; \forall (i,\tau) \in \mathbb{N}\times\mathbb{N} \\ \text{and } \psi(i,1) &\leq \psi(i+1,1) \;\; \forall i \in \mathbb{N} \;\; . \end{aligned}$$

We call a complexity function separable *if it is frontier-bounded and is of the form $\psi(i,\tau) = \eta_i \cdot \tau$ with $\eta_i > 0$ and $\eta_i \leq \eta_{i+1}$ for all $i \in \mathbb{N}$.*

A few notes are in order. First, it is easy to see that separability implies properness. Second, a separable complexity function is equivalent, for example, to one of the form $\psi(i,\tau) = \log(\eta_i) + \log(\tau)$ (or any other monotonic transformation). However, in the following we will stick to the multiplicative form, which has a straight forward interpretation: We fix the same cost η_i for all execution steps of program p_i. This turns $\pi_i = 1/\eta_i$ into a (non-normalized) prior over the space $\mathscr{P}$ of programs. For example, Levin Search induces the prior $\pi_i = 2^{-l(p_i)}$. In contrast to speed prior-based search, the prior $\pi_i = 1/\eta_i$ may be improper[1] without distorting Frontier Search in any way. This make Frontier Search widely applicable, even in the restricted case of separable complexity functions.

While we already know that frontier-boundedness makes sure that Frontier Search finds a solution to the problem (if one exists), this property is not sufficient to guarantee optimality. Here, properness comes into play:

Proposition 10. *We consider Frontier Search with proper complexity function ψ. Then Frontier Search finds the solution which minimizes ψ. Let $i^\star$ be the index of the minimizing program solving the problem in $\tau^\star$ steps, then the total number of steps $T(F^{\psi}_{(i^\star,\tau^\star)}) + 1$ executed by Frontier Search is given by*

$$\begin{aligned} &\left|\left\{(i,\tau) \in \mathbb{N}\times\mathbb{N} \;\middle|\; \psi(i,\tau) < \psi(i^\star,\tau^\star)\right\}\right| \\ +&\left|\left\{(i,\tau) \in \mathbb{N}\times\mathbb{N} \;\middle|\; \psi(i,\tau) = \psi(i^\star,\tau^\star) \text{ and } i \leq i^\star\right\}\right| \;\; . \end{aligned}$$

Proof. Consider the last frontier $F^{\psi}_{(i^\star,\tau^\star)} = \{(1,\tau_1),\ldots,(n-1,\tau_{n-1}),(n,1)\}$ before executing the final statement of $p_{i^\star}$. In this moment we have $\psi(i,\tau_i) \geq \psi(i^\star,\tau^\star)$ for all points on the frontier. Now the monotonicity ensures $\psi(i',\tau') \geq \psi(i',\tau_{i'}) \geq \psi(i,\tau)$ for all $i' \in \{1,\ldots,n\}$ and $\tau' > \tau_{i'}$, and $\psi(i',\tau') \geq \psi(i',1) \geq \psi(n,1) \geq \psi(i,\tau)$ for all $i' > n$ and $\tau \in \mathbb{N}$. Thus, all points outside the frontier have complexity larger or equal to $\psi(i,\tau)$, independently of whether they solve the problem or not. On the other hand, all points inside the frontier have complexity values of at most $\psi(i^\star,\tau^\star)$. But all steps corresponding to these points have been executed without solving the problem and halting.

The number of steps follows from the first statement, just notice that we assume that the program with smaller index is selected whenever two steps achieve the same complexity. □

Corollary 11. *We consider Frontier Search with proper complexity function ψ. Let $i^\star$ be the index of the minimizing program solving the problem in $\tau^\star$ steps. Then the total number of steps is bounded by*

$$\begin{aligned} &\left|\left\{(i,\tau) \in \mathbb{N}\times\mathbb{N} \;\middle|\; \psi(i,\tau) < \psi(i^\star,\tau^\star)\right\}\right| \\ &\leq\; T(F^{\psi}_{(i^\star,\tau^\star)}) < T(F^{\psi}_{(i^\star,\tau^\star)}) + 1 \;\; \leq \\ &\left|\left\{(i,\tau) \in \mathbb{N}\times\mathbb{N} \;\middle|\; \psi(i,\tau) \leq \psi(i^\star,\tau^\star)\right\}\right| \end{aligned}$$

Now we can effectively bound the total number of steps executed by Frontier Search for any given proper complexity criterion. We demonstrate three important cases:

[1] The prior π is proper if it can be normalized to sum to one, i.e., if $\sum_{i\in\mathbb{N}} \pi_i < \infty$.

Example 12. *Consider the criterion in the Levin complexity $\psi(i,\tau) = 2^i \cdot \tau$. This function is separable with prior $\pi_i = 2^{-i}$. If program i halts after running τ steps, the total execution time is upper bounded by*

$$\begin{aligned} T(F^{\psi}_{(i,\tau)}) &\leq \left|\left\{(i',\tau') \in \mathbb{N}\times\mathbb{N} \;\middle|\; 2^{i'}\cdot\tau' \leq 2^i\cdot\tau\right\}\right| \\ &= \left|\left\{(1,\tau') \;\middle|\; \tau' \leq 2^{i-1}\tau\right\}\right| \\ &\quad + \left|\left\{(2,\tau') \;\middle|\; \tau' \leq 2^{i-2}\tau\right\}\right| \\ &\quad + \cdots + \left|\left\{(i,\tau') \;\middle|\; \tau' \leq \tau\right\}\right| \\ &\quad + \left|\left\{(i+1,\tau') \;\middle|\; \tau' \leq \frac{\tau}{2}\right\}\right| + \cdots + 1 \\ &\leq 2^{i-1}\tau + 2^{i-2}\tau + \cdots + \tau + \left\lfloor\frac{\tau}{2}\right\rfloor + \left\lfloor\frac{\tau}{4}\right\rfloor + \cdots + 1 \\ &\leq 2^i\tau \in O(\tau) \;. \end{aligned}$$

So the total execution time is linear in τ. The same calculation works for arbitrary proper speed priors π_i.

Example 13. *As a second example we consider the separable criterion $\psi(i,\tau) = i\cdot\tau$, which corresponds to the improper prior $\pi_i = 1/i$. Again, let program p_i halt after τ steps. The number of steps for Frontier Search to execute is bounded by*

$$\begin{aligned} T(F^{\psi}_{(i,\tau)}) &\leq \left|\left\{(i',\tau') \in \mathbb{N}\times\mathbb{N} \;\middle|\; i'\cdot\tau' \leq i\cdot\tau\right\}\right| \\ &\leq i\tau\cdot\log(i\tau) \in O(\tau\cdot\log(\tau)) \;, \end{aligned}$$

which may still be considered affordable.

Example 14. *Last but not least we consider the complexity function $\psi(i,\tau) = \tau\cdot(i+\tau)$, which puts a strong emphasis on short execution time. It is proper, but not separable, because it increases the penalty per step the longer a program runs. Let program p_i halt after τ steps, and let $c = \psi(i,\tau) = \tau\cdot(i+\tau)$ be its complexity. Then the total number of steps executed is* lower *bounded by*

$$\begin{aligned} T(F^{\psi}_{(i,\tau)}) &\geq \left|\left\{(i',\tau') \in \mathbb{N}\times\mathbb{N} \;\middle|\; \tau'\cdot(i'+\tau') < c\right\}\right| \\ &= \left|\left\{(1,\tau') \;\middle|\; \tau' < \frac{-1+\sqrt{1+4c}}{2}\right\}\right| \\ &\quad + \left|\left\{(2,\tau') \;\middle|\; \tau' < \frac{-2+\sqrt{4+4c}}{2}\right\}\right| \\ &\quad + \cdots + \left|\left\{(l,\tau') \;\middle|\; \tau' < \frac{-l+\sqrt{l^2+4c}}{2}\right\}\right| \;, \end{aligned}$$

where

$$l = \arg\min_k \left\{\frac{-k+\sqrt{k^2+4c}}{2} \leq 2\right\} \Rightarrow l \geq \frac{c}{2} - 2 \;.$$

So when c is sufficiently large,

$$\begin{aligned} &\left|\left\{(i',\tau') \in \mathbb{N}\times\mathbb{N} \;\middle|\; \tau'\cdot(i'+\tau') < c\right\}\right| \\ &\geq \sum_{k=1}^{\frac{c}{2}-2} \frac{-k+\sqrt{k^2+4c}}{2} \\ &\geq \left(\frac{c}{2}-2\right)\frac{2-\frac{c}{2}+\sqrt{\frac{c^2}{4}+4+3c}}{2} \in \Omega(c) = \Omega(\tau^2) \;. \end{aligned}$$

Thus, the search requires $\Omega(\tau^2)$ steps.[2]

Reduction of Overhead Complexity

Algorithm 1 has one serious drawback compared to plain Levin Search: The 'arg min'-operation used to decide which program to execute next takes at least $\log(n)$ operations (using efficient data structures), where n is the size of the current frontier. This growing overhead, compared to the constant time spent on executing the underlying programs, is unsatisfactory, because asymptotically the fraction of time spent on program execution tends to zero. In this section we provide an algorithm that, under reduced requirements, achieves an amortized constant overhead.

Instead of strictly minimizing the complexity function ψ in each iteration we weaken the requirements as follows:

- We consider separable complexity functions. Furthermore, we assume that η_i is available in a binary encoding.
- The minimization of the complexity function may be only approximate. Let τ_i denote the position of the current frontier for program p_i, and let $\tilde{\tau}_i$ be the number of steps actually executed. Then we require $\lim_{\tau_i\to\infty} \tilde{\tau}_i/\tau_i = 1$ for all $i \in \mathbb{N}$.
- The complexity function does not need to be minimized in each single iteration. Instead, we ask for a growing sequence $(t_n)_{n\to\infty}$ of iterations in which the current frontier approximately minimizes the complexity function.

Approximate Frontier Search is introduced in Algorithm 2. It approximates Frontier Search in the above sense. The algorithm runs in epochs, maintaining a growing target complexity C. In each epoch it executes all programs with single-step complexity $\eta_i \leq C/\lceil\log(C)\rceil^2$ until they reach the target complexity, or in other words the frontier $\psi \approx C$. The frontier is approximated by delaying the execution of programs with relatively high single-step complexity $\eta_i > C/\lceil\log(C)\rceil^2$. It is easy to see that Approximate Frontier Search indeed fulfills the conditions listed above. As soon as $C/\lceil\log(C)\rceil^2$ (which tends to infinity) exceeds η_i the condition $\tilde{\tau}_i = \tau_i$ is fulfilled for the sequence $(t_e)_{e\in\mathbb{N}}$ of iterations finishing epochs.

In the following we analyze the complexity of the overhead created by Algorithm 2.

Proposition 15. *The number of operations of Algorithm 2 in between executing two program steps is upper bounded by a constant in an amortized analysis.*

Proof. We need a few basic facts about operations on binary encoded numbers. Recall that adding a constant value to a variable takes amortized constant time. Therefore counting in a loop from 1 to m takes $O(m)$ time. Computing $a+b$ takes $O(\min\{\log(a),\log(b)\})$ operations, and so does the comparison $a < b$. The multiplication $a\cdot b$ requires $O(\log(a)\cdot\log(b))$ operations, and an integer division $\lfloor a/b\rfloor$ costs $O(\log(a/b)\cdot\log(b)) \leq O((\log(a))^2)$ computational time. The computation of $\lceil\log(a)\rceil$ can be performed in at most $O(\log(a))$ operations.

[2] A function fulfills $f(x) \in \Omega(g(x))$ if there exist $N \in \mathbb{N}$ and $c \in \mathbb{R}$ such that $|f(n)| > c\cdot g(n)$ for all $n > N$.

Algorithm 2: Approximate Frontier Search.

Input: η, L, x, $\mathscr{P}$, C_0
Output: $p \in \mathscr{P}$ such that $L(p) = x$
$e \leftarrow 1$; $t \leftarrow 1$; $n \leftarrow 1$; $\tau_1 \leftarrow 0$;
$C \leftarrow \max\{\eta_1^2, C_0\}$; $M \leftarrow \lceil \log(C) \rceil^2$;
while *true* **do**
  $C \leftarrow 4 \cdot C$; $M \leftarrow M + 4$;
  for $i = 1, \ldots, n$ **do**
    $m \leftarrow \lfloor C/\eta_i \rfloor - \tau_i$;
    run program p_i for m steps;
    if p_i halts and $L(p_i) = x$ **then return** p_i;
    $\tau_i \leftarrow \tau_i + m$; $t \leftarrow t + m$;
  end
  while *true* **do**
    $m \leftarrow \lfloor C/\eta_{n+1} \rfloor$;
    if $m < M$ **then** break;
    $n \leftarrow n + 1$;
    run program p_n for m steps;
    if p_n halts and $L(p_n) = x$ **then return** p_n;
    $\tau_n \leftarrow m$; $t \leftarrow t + m$;
  end
  $t_e \leftarrow t$; $e \leftarrow e + 1$;
end

Note that adding four to M and quadrupling C corresponds to maintaining the relation $M = \lceil \log(C) \rceil^2$.

Consider the number m computed in the for-loop. We show by induction that this number exceeds M: In the first iteration we have $n = 1$, such that the loop only runs over a single program, and $C = 4 \cdot \max\{\eta_1^2, C_0\}$ makes sure that $\lfloor C/\eta_1 \rfloor \geq M = \lceil \log(C) \rceil^2$ for suitable C_0. In later iterations we know that $\lfloor C/\eta_i \rfloor - \tau_i \geq M$ was fulfilled in the previous iteration for all $i \in \{1, \ldots, n\}$ (this is trivially fulfilled for the programs added in the inner while loop), implying $\lfloor C/\eta_i \rfloor \geq M$, which reads $\lfloor C/(4 \cdot \eta_i) \rfloor \geq M - 4$ in the notation of the current iteration. Together with $\tau_i \leq \lfloor C/(4 \cdot \eta_i) \rfloor$ and $\lfloor C/2 \rfloor > 4$ (for $C_0 \geq 1/2$) this implies $\lfloor C/\eta_i \rfloor \geq M$. Thus, all programs executed in an epoch are executed for at least M steps. Instead of choosing C_0 unnecessarily large, we may let the target complexity C start small and wait for C to exceed C_0 after finitely many epochs.

The budget available per epoch is linear in the number of program steps executed, which is $O(n \cdot M)$. It is easy to see that all additions, subtractions, and loop counters/comparisons easily fit into this time budget. The potentially most costly operation in the program is the division $\lfloor C/\eta_i \rfloor$. Its complexity is upper bounded by $O(\lceil \log(C) \rceil^2)$, which by construction coincides with $O(M)$. □

Discussion

Frontier Search provides a very flexible alternative to Levin Search. This increased flexibility enables us to respect complicated complexity functions in the search.

The algorithm is known to work with a quite general set of complexity functions (see Proposition 7), while the still very flexible space of *proper* complexity functions is minimized by the algorithm exactly (see Proposition 10). For the more restricted case of separable complexity functions we provide the algorithm Approximate Frontier Search which achieves constant overhead, while preserving the asymptotic properties of Frontier Search.

Even the relatively restricted case of separable complexity functions provides interesting search schemes. The only restriction on the growing sequence η_i of step costs is that it takes infinitely many different values. This excludes nearly everywhere constant priors, but it does not require the prior to be proper.

Non-separable cases may be of even greater interest. There are different reasons why we may wish to vary the cost of executing a command over time. On the one hand one may search for a program with runtime in the 'right' order of magnitude by only penalizing steps that exceed the next power of, e.g., ten. Or one may, like in example 14, increase the penality over time, strongly favoring short execution time.

All these different search schemes can be realized with Frontier Search. The specification of a particular search scheme is implicitly done by providing a complexity function, which does not require any changes to the Frontier Search algorithm itself, and is intuitive and therefore easy to specify by the user.

Conclusion

We demonstrate the theoretically powerful search algorithm Frontier Search, which automatically finds programs optimal w.r.t. a given complexity criterion. It is provably more general than Levin Search and speed prior-based search in several respects: We can handle improper priors, and even time-varying execution costs under weak and intuitively meaningful technical conditions. For the case of separable complexity functions we propose Approximate Frontier Search, which achieves constant computational overhead.

Like Levin Search, the current approach is limited to programs computing a fixed output x. We leave generalizations to more relevant cases such as minimizing a loss function given data to future work.

Acknowledgments

This work was funded in part by SNF grants 200021-111968 and 200021-113364.

References

L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9:265–266, 1973.

M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. 1993.

J. Schmidhuber. The Speed Prior: a new simplicity measure yielding near-optimal computable predictions. In *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, pages 216–228. Springer, Sydney, Australia, 2002.

T. Schaul and J. Schmidhuber. Towards Practical Universal Search. 2010. *Submitted to the Conference on Artificial General Intelligence*.