An effective Procedure for Speeding up Algorithms

Marcus Hutter

Istituto Dalle Molle di Studi sull'Intelligenza Artificiale
IDSIA, Galleria 2, CH-6928 Manno-Lugano, Switzerland
marcus@idsia.ch, http://www.idsia.ch/~marcus

2001

Table of Contents

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 $f:X\to Y$, 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 $g:Y \to X$, if g can be evaluated quickly.

[Levin 1973]

SIMPLE is as fast as SEARCH

- SIMPLE: run all programs $p_1 p_2 p_3 \ldots$ 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, ... $time_{\text{SIMPLE}}(x) \leq 2^k time^+_{p_k}(x) + 2^{k-1}$.
- SEARCH: run all p of length less than i for $2^i 2^{-l(p)}$ steps in phase $i = 1, 2, 3, \ldots$. $time_{\text{SEARCH}}(x) \leq 2^{K(k)+O(1)} time^+_{p_k}(x), \quad K(k) \ll k$.
- Refined analysis: SEARCH itself is an algorithm with some index $k_{\text{SEARCH}} = O(1)$
  $\Longrightarrow$ SIMPLE executes SEARCH every $2^{k_{\text{SEARCH}}}$-th step
  $\Longrightarrow$ $time_{\text{SIMPLE}}(x) \leq 2^{k_{\text{SEARCH}}} time^+_{\text{SEARCH}}(x)$
  $\Longrightarrow$ SIMPLE and SEARCH have the same asymptotics also in k.
- Practice: SEARCH should be favored because the constant $2^{k_{\text{SEARCH}}}$ is rather large.

Main New Result (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 $t_p(x)$.
- $time_{t_p}(x)$ is the time needed to compute the time bound $t_p(x)$.
- Then the algorithm M_{p^*} computes $p^*(x)$ in time

$$time_{M_{p^*}}(x) \;\leq\; 5\cdot t_p(x) + d_p\cdot time_{t_p}(x) + c_p$$

- 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 $M_{p^*}(x)$.

[Hutter, 2000]

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^*}

$\boxed{M_{p^*}(x)}$

Initialize the shared variables

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

Start algorithms A, B, and C

in parallel with 10%, 10% and 80%

computational resources, respectively.

$\boxed{A}$

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.

$\boxed{B}$

Compute all $t(x)$ in parallel

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

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

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

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

`continue`

$\boxed{C}$

`for` k:=1,2,4,8,16,32,... `do`

run current p_{fast} for k steps

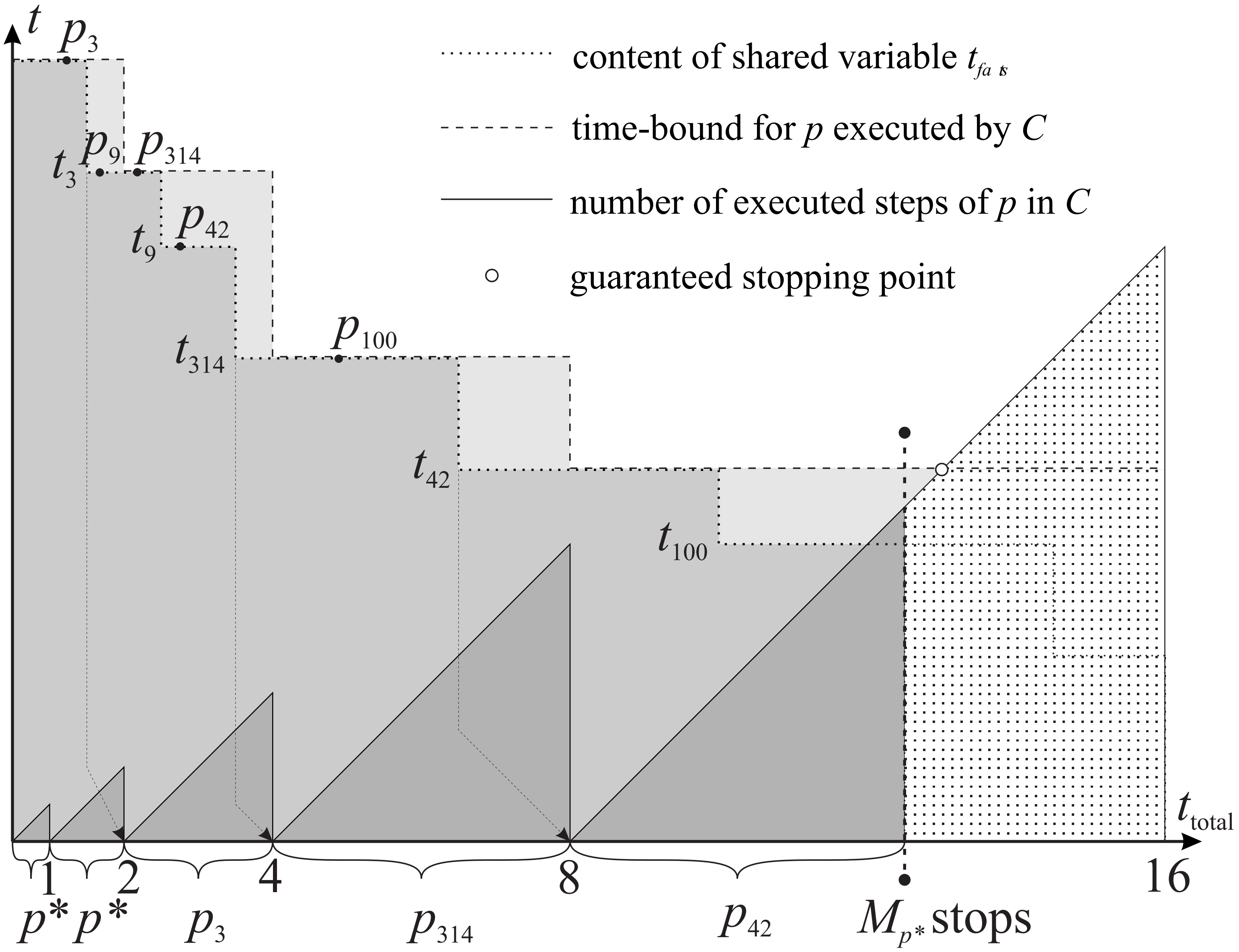
(without switching).

`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^*}



Time Analysis

$$T_A \leq \frac{1}{10\%}\cdot 2^{l(proof(p'))+1}\cdot O(l(proof(p'))^2)$$

$$T_B \leq T_A + \frac{1}{10\%}\cdot 2^{l(p')+l(t_{p'})}\cdot time_{t_{p'}}(x)$$

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

$$time_{M_{p^*}}(x) = T_C \;\leq\; 5\cdot t_p(x) + d_p\cdot time_{t_p}(x) + c_p$$

$$d_p = 40\cdot 2^{l(p)+l(t_p)}, \quad c_p = 40\cdot 2^{l(proof(p))+1}\cdot O(l(proof(p)^2)$$

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.

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

[Kolmogorov 1965 and others]

Universal Complexity of a Function

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

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

[Hutter, 2000]

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

$$\begin{array}{lll} i) & l(\tilde{p}) & \leq K''(p^*) + O(1) \\ ii) & time_{\tilde{p}}(x) & \leq 5\cdot t_p(x) + d_p\cdot time_{t_p}(x) + c_p \end{array}$$

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.

[Hutter, 2000]

Proof

Insert the shortest algorithm p' provably equivalent to p^* into M, that is $\tilde{p} := M_{p'}$.

$$l(\tilde{p}) = l(p') + O(1) = K''(p^*) + O(1)$$

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 π).

Summary & Outlook

- 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 measures 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 applied even though it suffers from a large multiplicative factor [Schmidhuber 1997]
- 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?