Short lists with short programs in short time

Bruno Bauwens
Anton Makhlin
Nikolay Vereshchagin
Marius Zimand

April 2013

- U - universal TM, $U(p) = x$, we say p is a program for x.
- $C(x) = \min\{|p| \mid p \text{ program for } x\}$.
- $C(x)$ - canonical example of an **uncomputable** function.
- Finding a shortest program for x: also uncomputable.

- **Our question:** Is it possible to compute a short list containing a short program for x?

- DEFINITION. p is a c-short program for x if $U(p) = x$ and $|p| = C(x) + c$.

- DEFINITION. A function f is a list approximator for c-short programs if
  $\forall x$, $f(x)$ is a finite list containing a c-short program for x.

Our results

- There exists a computable list approximator f for $O(1)$-short programs, with size $O(n^2)$.
- For any computable list approximator for c-short programs, size is $\Omega(n^2/c^2)$.
- There exists a **poly.-time computable** list approximator for $O(\log n)$-short programs, with size $\mathrm{poly}(n)$.

Our results

What about lists containing a shortest program?
Answer: It depends on the universal machine.

- For some U, any computable list containing a shortest program for x has size $2^{n-O(1)}$.
- For some U, there is a computable list containing a shortest program of size $O(n^2)$.

Proof of the upper bounds

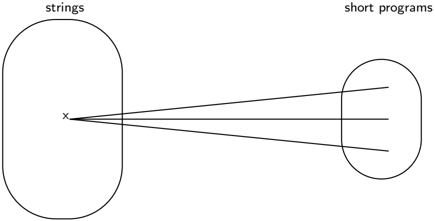
Upper bounds
Th. 1: There exists a computable list approximator f for $O(1)$-short programs with size $O(n^2)$.
Th. 2 There exists a **poly time computable** list approximator for $O(\log n)$-short programs, with size $\mathrm{poly}(n)$.

Bipartite graphs with online matching with overhead c.

Proof of the upper bounds

Upper bounds
Th. 1: There exists a computable list approximator f for $O(1)$-short programs with size $O(n^2)$.
Th. 2 There exists a **poly time computable** list approximator for $O(\log n)$-short programs, with size $\mathrm{poly}(n)$.

Bipartite graphs with online matching with overhead c.



Proof of the upper bounds

Upper bounds
Th. 1: There exists a computable list approximator f for $O(1)$-short programs with size $O(n^2)$.
Th. 2 There exists a **poly time computable** list approximator for $O(\log n)$-short programs, with size $\mathrm{poly}(n)$.

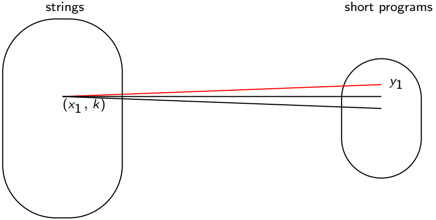
Bipartite graphs with online matching with overhead c.

- matching requests arrive one by one
- request (x, k): match $x \in \mathit{LEFT}$ with a free node $y \in N(x)$ s.t. $|y| \leq k + c$.
- Promise: $k \leq |x|$ and $\forall k$ there are $\leq 2^k$ requests $(*, k)$.
- Requirement: all requests should be satisfied online (before seeing the next request).

Proof of the upper bounds

Upper bounds
Th. 1: There exists a computable list approximator f for $O(1)$-short programs with size $O(n^2)$.
Th. 2 There exists a **poly time computable** list approximator for $O(\log n)$-short programs, with size $\mathrm{poly}(n)$.

Bipartite graphs with online matching with overhead c.

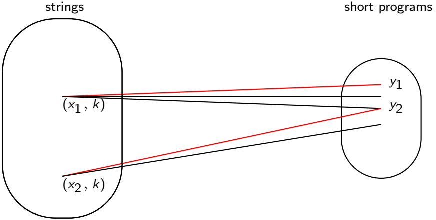


- matching requests arrive one by one
- request (x, k): match $x \in \mathit{LEFT}$ with a free node $y \in N(x)$ s.t. $|y| \leq k + c$.
- Promise: $k \leq |x|$ and $\forall k$ there are $\leq 2^k$ requests $(*, k)$.
- Requirement: all requests should be satisfied online (before seeing the next request).

Proof of the upper bounds

Upper bounds
Th. 1: There exists a computable list approximator f for $O(1)$-short programs with size $O(n^2)$.
Th. 2 There exists a **poly time computable** list approximator for $O(\log n)$-short programs, with size $\mathrm{poly}(n)$.

Bipartite graphs with online matching with overhead c.



- matching requests arrive one by one
- request (x, k): match $x \in LEFT$ with a free node $y \in N(x)$ s.t. $|y| \leq k + c$.
- Promise: $k \leq |x|$ and $\forall k$ there are $\leq 2^k$ requests $(*, k)$.
- Requirement: all requests should be satisfied online (before seeing the next request).

Short lists - combinatorial characterization

Theorem

$\exists$ (poly time) computable G with on-line matching with overhead c and the matching strategy is computable

IFF

$\exists$ (poly time) computable list of size $deg(x)$ containing a $c + O(1)$-short program for x

Proof. ($\Rightarrow$ only)

- Enumerate strings in LEFT(G) as they are produced by U.
- Say, at step s, $U(q) = x$, with $|q| = k$.
- Make request (x, k).
- x is matched with y of length $k + c$.
- y is a program for x.
- Why: on input y, re-play the matching process till some left string is matched to it; output this left string, which will be x.

So when q is the shortest program for x, we get a program for x of length $C(x) + c$.

The list consists of x's neighbors in G.

How to build a graph with online matching

Focus first on requests of type $(*, k)$ with fixed k.

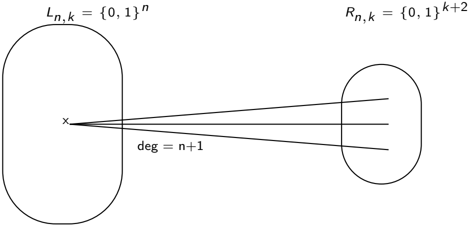
DEFINITION. A bipartite graph G is a (K, K')-expander if every K left nodes have $\geq K'$ neighbors.

Lemma

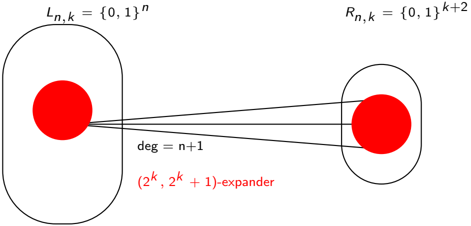
Let G be a $(K, K+1)$-expander. If $2K$ matching requests are made,then less than K are rejected.

Proof.
If $|REJECTED| \geq K$, then $|N(REJECTED)| \geq K+1$.
But each node in N(REJECTED) has satisfied a request.
So number of requests would be at least $(K+1)$(satisfied) $+$ K (rejected) $= 2K+1$. Contradiction.

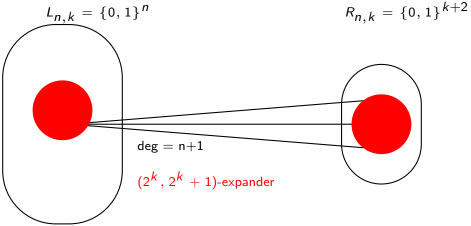
$G_{n,k}$ - basic building brick.



$G_{n,k}$ - basic building brick.



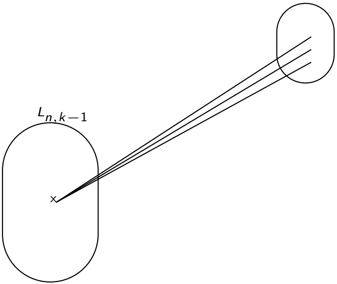
$G_{n,k}$ - basic building brick.



Can be built with the probabilistic method + exhaustive search.

If $2K$ online matching requests are made, $< K$ are rejected (where $K = 2^k$)

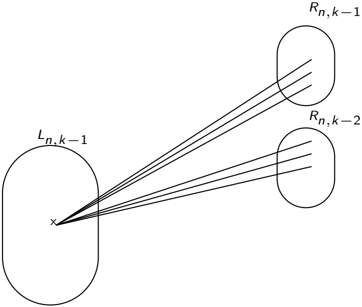
$H_{n,k}$



K requests

$< K/2$ rejected

$H_{n,k}$

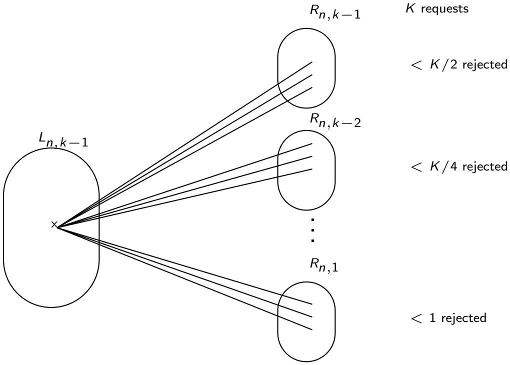


K requests

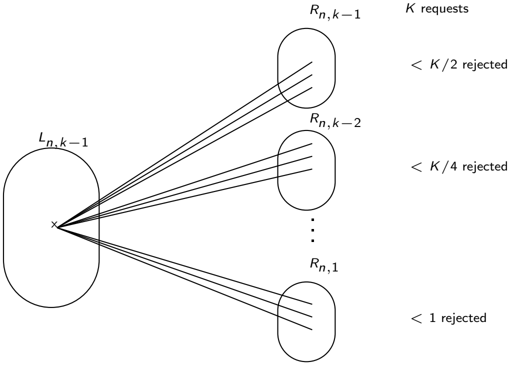
$< K/2$ rejected

$< K/4$ rejected

$H_{n,k}$


K requests
$R_{n,k-1}$
$< K/2$ rejected
$R_{n,k-2}$
$L_{n,k-1}$
$< K/4$ rejected
x
$R_{n,1}$
< 1 rejected

$H_{n,k}$



Satisfies all requests (x, k) with $|x| = n$, fixed k.

left degree $= (k-1)(n+1) = O(n^2)$.

overhead $= 1$

$H_n = H_{n,n} \cup H_{n,n-1} \cup \ldots \cup H_{n,1}$

Satisfies all requests (x, k), $|x| = n$, $k \leq n$.

left degree $= O(n^3)$.

overhead $= 1$

$H_n = H_{n,n} \cup H_{n,n-1} \cup \ldots \cup H_{n,1}$

Satisfies all requests (x, k), $|x| = n$, $k \leq n$.

left degree $= O(n^3)$.

overhead $= 1$

H=$H_1 \cup H_2 \cup \ldots \cup H_n \cup \ldots$, but append to each right node in H_n a code for n

Satisfies all requests

left degree$(x) = O(|x|^3)$.

overhead $O(\log n)$ (due to codes).

$H_n = H_{n,n} \cup H_{n,n-1} \cup \ldots \cup H_{n,1}$

Satisfies all requests (x, k), $|x| = n$, $k \leq n$.

left degree $= O(n^3)$.

overhead $= 1$

$H = H_1 \cup H_2 \cup \ldots \cup H_n \cup \ldots$, but append to each right node in H_n a code for n

Satisfies all requests

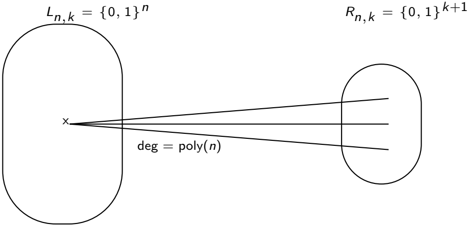
left degree$(x) = O(|x|^3)$.

overhead $O(\log n)$ (due to codes).

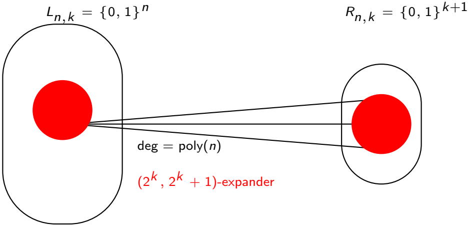
With a more elaborate construction, left degree (x) $= O(|x|^2)$, overhead $= O(1)$.
So :

Th. 1: There exists a computable list approximator f for $O(1)$-short programs, with size $O(n^2)$

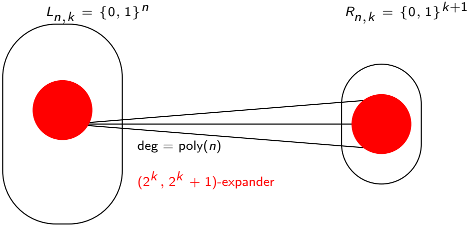
$G_{n,k}$ Basic building brick in poly time



$G_{n,k}$ Basic building brick in poly time

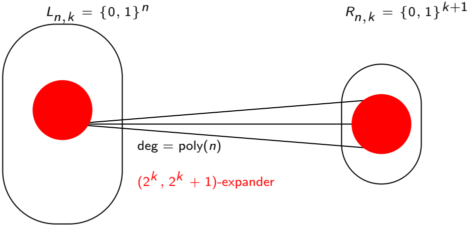


$G_{n,k}$ Basic building brick in poly time



- Use [Ta-Shma, Umans, Zuckerman'07] disperser:
- $L = \{0,1\}^n, R = \{0,1\}^{k-O(\log n)}$, deg $= \mathrm{poly}(n)$
- every $A \subseteq L, |A| = K \Rightarrow |N(A)| \geq \frac{1}{2}|R| = \frac{K}{2\mathrm{poly}(n)}$
- We take $\mathrm{poly}(n)$ copies of R to get $(K, K+1)$-expander.

$G_{n,k}$ Basic building brick in poly time



- Use [Ta-Shma, Umans, Zuckerman'07] disperser:
- $L = \{0,1\}^n, R = \{0,1\}^{k-O(\log n)}$, deg $= \mathrm{poly}(n)$
- every $A \subseteq L, |A| = K \Rightarrow |N(A)| \geq \frac{1}{2}|R| = \frac{K}{2\mathrm{poly}(n)}$
- We take $\mathrm{poly}(n)$ copies of R to get $(K, K+1)$-expander.

So:

Th. 2 There exists a **poly time computable** list approximator for $O(\log n)$-short programs with size $\mathrm{poly}(n)$.

Further developments

[Teutsch 2012] overhead $O(1)$, so list has $O(1)$-short programs - arxiv 1212.0682

[Zimand 2013] simpler, shorter proof, list size $= n^{6+\epsilon}$

Short lists with short programs in short time - a short proof - arxiv 1302.1109

Lower bounds - 1

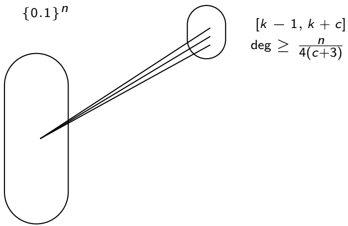
Th. If a computable list contains a c-short program for x, then its size is $\Omega(n^2/(c+O(1))^2)$.

- $x \to$ list $f(x) = \{y_1, \ldots, y_t\}$ contains a c-short program for x.
- bipartite G: LEFT $= \{0,1\}^n$, $\forall x \in LEFT$, $(x, y_i) \in E$ for all $y_i \in f(x)$.
- G has on-line matching with overhead $c \Rightarrow$ also has off-line matching with overhead c.
- $G[\ell, k]$ is G from which we cut right nodes y with $|y| < \ell$ or $|y| > k$.
- $\forall k$, $G[0, k+c]$ is $(2^k, 2^k)$-expander
- So, $\forall k$, $G[k-1, k+c]$ is $(2^k, 2^{k-1}+1)$-expander.

LEMMA

Graph G has $|LEFT| = 2^\ell$, $|RIGHT| = 2^{k+c}$, and is a $(2^k, 2^{k-1}+1)$-expander.
Then $\exists x \in LEFT$ with $deg(x) \geq \min(2^{k-2}, \frac{\ell-k}{c+2})$.

- Take $k \in (n/4, n/2]$. By LEMMA (with $\ell = 3n/4$), in $G[k-1, k+c]$, all $x \in LEFT$, except $2^{3n/4}$, have $deg(x) \geq \frac{n}{4(c+3)}$.



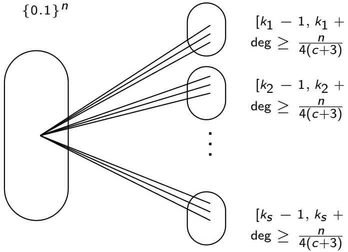
• Take $k \in (n/4, n/2]$. By LEMMA (with $\ell = 3n/4$), in $G[k-1, k+c]$, all $x \in LEFT$, except $2^{3n/4}$, have $deg(x) \geq \frac{n}{4(c+3)}$.

• Pick $n/4 < k_1 < k_2 < \ldots < k_s < n/2$, and $(c+2)$ apart from each other; $s \approx \frac{n}{4(c+2)}$.

• In each $G[k_i - 1, k_i + c]$, all left nodes, except $2^{3n/4}$, have deg $\geq \frac{n}{4(c+3)}$.

• The RIGHT sets are disjoint.

• So, $\exists x \in LEFT$, with $deg(x) \geq s \cdot \frac{n}{4(c+3)} = \Omega(\frac{n^2}{(c+3)^2})$.



Lower Bounds - 2

Th. For some U, any list containing a shortest program for x has size $2^{\Omega(n)}$.

$f(x) =$ the list $= \{y_1, \ldots, y_{|f(x)|}\}$ containing x^* -shortest program for x.

Clearly $C(x^* \mid x) \leq \log |f(x)| + O(1)$.

Would like $C(x^* \mid x)$ to be big, but $C(x^* \mid x) \leq \log n + O(1)$.

Use CT - total conditional complexity.

$CT(u \mid v) = \min\{|p| \mid U(p, v) = u$ and $U(p, w) \downarrow$ for all w.

We still have $CT(x^* \mid x) \leq \log |f(x)| + O(1)$.

We show: $\exists U_0$, for infinitely many x, every shortest program p for x w.r.t. U_0, $CT(p \mid x) \geq n - O(1)$.

So, $|f(x)| \geq 2^{n-O(1)}$.

Start with U standard machine. Build V s.t. for every n, $\exists p, x, |p| = |x| = n$:

(a) p is the unique shortest program for x w.r.t V,

(b) $C_U(x) \geq n - 3$

(c) $CT_U(0p \mid x) \geq n - 3$.

Start with U standard machine. Build V s.t. for every n, $\exists p, x, |p| = |x| = n$:

(a) p is the unique shortest program for x w.r.t V,

(b) $C_U(x) \geq n - 3$

(c) $CT_U(0p \mid x) \geq n - 3$.

Next define U_0:

(1) $U_0(0q) = V(q)$,

(2) $U_0(1^5 q) = U(q)$ (so U_0 is a standard universal machine).

Start with U standard machine. Build V s.t. for every n, $\exists p, x, |p| = |x| = n$:

(a) p is the unique shortest program for x w.r.t V,

(b) $C_U(x) \geq n - 3$

(c) $CT_U(0p \mid x) \geq n - 3$.

Next define U_0:

(1) $U_0(0q) = V(q)$,

(2) $U_0(1^5 q) = U(q)$ (so U_0 is a standard universal machine).

Then,

$U_0(0p) = V(p) = x$

$0p$ is the unique shortest program for x w.r.t. U_0

$CT_U(0p \mid x) \geq n - 3$.

Start with U standard machine. Build V s.t. for every n, $\exists p, x, |p| = |x| = n$:

(a) p is the unique shortest program for x w.r.t V,

(b) $C_U(x) \geq n - 3$

(c) $CT_U(0p \mid x) \geq n - 3$.

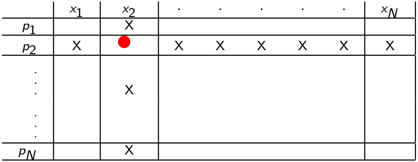
Defining V via a game

V on inputs of length n; $N = 2^n$; use $N \times N$ board.

	x_1	x_2	$\cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot$	x_N
p_1				
p_2				
$\vdots$				
$\vdots$				
p_N				

Defining V via a game

V on inputs of length n; $N = 2^n$; use $N \times N$ board.



WHITE can pass or put a pawn, but only one in a row or a column.
WHITE moves define V: pawn placed on $(p, x) \Rightarrow V(p) = x$.

Defining V via a game

V on inputs of length n; $N = 2^n$; use $N \times N$ board.

	x_1	x_2	.	.	.	.	.	x_N
p_1		X					X	
p_2	X	●	X	X	X	X	X	X
$\vdots$		X					X	
$\vdots$							X	
p_N		X					X	

WHITE can pass or put a pawn, but only one in a row or a column.
WHITE moves define V: pawn placed on $(p, x) \Rightarrow V(p) = x$.

BLACK can pass, or disable all cells in a column

Defining V via a game

V on inputs of length n; $N = 2^n$; use $N \times N$ board.

	x_1	x_2	.	.	.	.	.	x_N
p_1		X	X					
p_2	X	X	X	X	X	X	X	X
$\vdots$	X	X						
$\vdots$				X				
p_N		X					X	

WHITE can pass or put a pawn, but only one in a row or a column.
WHITE moves define V: pawn placed on $(p, x) \Rightarrow V(p) = x$.

BLACK can pass, or disable all cells in a column, or disable a cell in each column.

Defining V via a game

V on inputs of length n; $N = 2^n$; use $N \times N$ board.

	x_1	x_2	.	.	.	.	.	x_N
p_1		X	X					
p_2	X	X	X	X	X	X	X	X
⋮	X	X						
⋮				X				
p_N		X					X	

WHITE can pass or put a pawn, but only one in a row or a column.
WHITE moves define V: pawn placed on $(p, x) \Rightarrow V(p) = x$.

BLACK can pass, or disable all cells in a column, or disable a cell in each column.

BLACK can do $< N/4$ disabling moves.

WHITE loses if at some point, after her turn, all pawns are in disabled cells.

WHITE has a winning strategy (later).

Defining V via a game (2)

Build V s.t. for every n, $\exists p, x, |p| = |x| = n$:

(a) p is the unique shortest program for x w.r.t V,

(b) $C_U(x) \geq n - 3$

(c) $CT_U(0p \mid x) \geq n - 3$.

Consider the following BLACK's blind strategy:

(a) Enumerate all strings x of length n with $C_U(x) < n - 3$

(b) and all q, $|q| < n - 3$ s.t. $U(q, x) \downarrow$ for all strings x of length n.

At step s: if (a) happens, disable column x.

if (b) happens, disable all cells (p, x) s.t. $U(q, x) = 0p$.

BLACK does $< 2^{n-3} + 2^{n-3} = N/4$ moves.
WHITE moves define V: pawn placed on $(p, x) \Rightarrow V(p) = x$.
WHITE wins: some cell (p, x) has pawn and is not disabled. Then (p, x) satisfies (a), (b), (c).

WHITE's winning strategy

(1) Initially, put a pawn on any cell
(2) When that cell gets disabled, put another pawn in an available cell.

BLACK makes $< N/4$ moves, so WHITE makes $\leq N/4$ moves

At each move, BLACK disables N cells

At each WHITE's move, $(2N-1)$ cells become unavailable (row + col)

So, total number of unavailable cells is:

$$\leq (N/4)N + N/4(2N-1) < N^2/4 + N^2/2 < N^2$$

So WHITE can always place a pawn.

SUMMARY

- One can compute a $O(n^2)$-sized list containing a $O(1)$-short program.
- Any computable list containing a $O(1)$-short program has size $\Omega(n^2)$.
- One can compute in poly time a $\mathrm{poly}(n)$-sized list containing a $O(\log n)$-short program.
- For some universal TM, any computable list containing a shortest program has size $\Omega(2^n)$.
- There exists a universal TM, with a computable $O(n^2)$-sized list containing a shortest program.

Thank you.