Computational Complexity of K_n

15-453 Project Presentation

Project Advisor: Leonid Kontorovich

Jeremiah Blocki

What is K_n anyway and why do we care?

Definitions:

Given a fixed alphabet Σ and $x,y \in \Sigma^*$

- $DFA(n,\Sigma) = \{M: M \text{ is a DFA with n states and alphabet } \Sigma\}$
- $K_n(x,y) = |\{M: M \in DFA(n,\Sigma) \text{ and } \{x,y\} \subset L(M)\}|$

Why is K_n interesting?

- It is useful for learning regular languages, a topic Vinay will discuss later today
- K_n is a universal regular kernel which
  "can be easily adapted to render all the regular languages linearly separable"
- Our focus, however, will be the computational complexity of computing K_n

Counting…

$|DFA(n,\Sigma)| = ?$

- There are n states, and $|\Sigma|$ transitions for each state.
- There are n possibilities for each of the $n|\Sigma|$ transitions.
- There are 2^n ways to label of final states
  (by default q_0 is the start state)

Hence, $|DFA(n,\Sigma)| = 2^n n^{n|\Sigma|}$

Note: $DFS(n,\Sigma) := \{M$: M is a semi-automaton
(no labeled accept\reject states)
with n states and alphabet $\Sigma\}$

$| DFS(n,\Sigma)| = n^{n|\Sigma|}$

Examples

Let $\Sigma = \{0,1\}$, $x \in \Sigma$

1. $K_n(x,x) = ?$

   $= |DFA(n,\Sigma)|/2$

2. $K_n(1,\varepsilon) = ?$

   $= (1/n)*|DFA(n,\Sigma)|/2 + (n-1)/n*|DFA(n,\Sigma)|/4$

   $= (n+1)|DFA(n,\Sigma)|/(2n)$

The probability that a semi-automaton ends on the same state given strings "1", ε is

- $Pr[\delta(q_0,1) = q_0] = 1/n$

The probability that a semi-automaton ends on different states given strings "1", ε is

- $Pr[\delta(q_0,1) \neq q_0] = (n-1)/n$

If a semi-automaton ends on a different state on x,y then exactly a fourth of the labelings of final states will result in a DFA accepting both x and y

Otherwise exactly half of the labelings of final states will result in a DFA accepting both x and y

Results

1. C++ Code for Brute Force Computation of K_n
2. Closed form expressions for $K_n(x,y)$, $(x,y \in \Sigma^*$ s.t $|x| \leq 2$, $|y| \leq 2)$
3. An Algorithm to "Efficiently" Compute $K_n(x,y)$ given sufficiently short strings x,y
4. Proof that computing $K_n(x,y)$ requires completely reading both strings x and y
5. A modified version of the problem is at least as hard as factoring
6. The corresponding universal kernel for Turing Machines is incomputable.

Brute Force

Let $\Sigma = \{0,1\}$

Idea: Represent a Semi-Automaton as a 2xn array of integers $\{0,\ldots,n-1\}$, and compress/decompress the semi-automaton to an integer

Brute Force computation is feasible for $n \leq 6$, after this time is a huge constraint (not to mention that there are only 32 bits used to store an integer…)

Recall that:

$|DFS(n,\Sigma)| = n^{n\Sigma} = n^{2n}$

Brute Force (Code)

```
// Assuming the DFA is "small enough" (<= 6 states)
// At 7 states already 7^(2*7) > 2^32 - 1, takes in an input
// array of dimensions: ALPHABET_SIZE x MAX_STATES, but only
// considers the first n states to compress the DFA
inline unsigned int compressDFS(int Delta[ALPHABET_SIZE][MAX_STATES], int n) {
    unsigned int total = 0;
    for (int i = 0; i < ALPHABET_SIZE;i++) {
        for (int j = 0; j < n; j++) {
          total*=n;
          total += Delta[i][j];
            }
     }
     return total;
}
// Only works for n <= 6, see note above
//
// Only fills the first n states, decompresses based on n
inline void decompressDFS(unsigned int compressedDFA, int n,
int DFS[ALPHABET_SIZE][MAX_STATES]) {
    unsigned int comp = compressedDFA;
    for (int i = ALPHABET_SIZE-1; i >= 0; i--){
        for (int j = n-1; j >=0; j--) {
            DFS[i][j] = comp % n;
            comp/=n;
          }
     }
}
```

Closed Form

Closed form expressions for $K_n(x,y)$, ($x,y \in \Sigma^*$ s.t $|x| \leq 2$, $|y| \leq 2$)

Example: ($\Sigma = \{0,1\}$)

$K_n(11,01) = ?$

- Case 1:
  - $\delta(q_0,1) = \delta(q_0,0) = q_i$ (probability: 1/n)
  - Then semi-automata must end at the same state on "11" and "01"
- Case 2:
  - $\delta(q_0,1) \neq \delta(q_0,0)$ (probability: (n-1)/n)
  - The semi-automata ends on the same state with probability:

    $\Pr[\delta(\delta(q_0,0),1) = \delta(\delta(q_0,1),1)] = 1/n$

    (Here we invoke the fact that one of the final transitions is as of yet undetermined)

Closed Form Results

- $K_n(x,x) = 2^{n-1}n^{|\Sigma|n}$
- $\sigma_1, \sigma_2 \in \Sigma\ (\sigma \neq \sigma_2) \rightarrow K_n(\sigma_1,\sigma_2) = 2^{n-2}n^{|\Sigma|n}(n+1)/n$
- $\sigma \in \Sigma \rightarrow K_n(\varepsilon,\sigma) = 2^{n-2}n^{|\Sigma|n}(n+1)/n$
- $\sigma \in \Sigma \rightarrow K_n(\varepsilon, \sigma\sigma) = 2^{n-2}n^{|\Sigma|n}(2/n+2/n*((n-1)/n) + ((n-1)/n)^2)$
- $\sigma_1, \sigma_2 \in \Sigma, (\sigma_1 \neq \sigma_2) \rightarrow K_n(\varepsilon,\sigma_1\sigma_2) = K_n(\varepsilon,\sigma_2\sigma_1) =$
  $= 2^{n-2}n^{|\Sigma|n}(n+1)/n$
- $K_n(\sigma,\sigma\sigma) = 2^{n-2}n^{|\Sigma|n}(1+(2n-1)/n^2)$

(More) Closed Form Results

- $K_n(\sigma_1,\sigma_2\sigma_1) = 2^{n-2}n^{|\Sigma|n}(2/(n^2) + 2(n-1)/n^2 + 2*(n-1)/n^2 + ((n-1)/n)^2)$
- $K_n(\sigma_1,\sigma_2\sigma_2) = 2^{n-2}n^{|\Sigma|n}(2/n^2 + (n-1)/n^2 + 2(n-1)/n^2 + ((n-1)/n)^2)$
- $K_n(\sigma_1,\sigma_1\sigma_2) = 2^{n-2}n^{|\Sigma|n}((n+1)/n)$
- $K_n(\sigma_1\sigma_2,\sigma_2\sigma_1) = 2^{n-2}n^{|\Sigma|n}(1-1/n^2- 1/n^3 + 3/n)$
- $K_n(\sigma_1\sigma_1,\sigma_2\sigma_2) = 2^{n-2}n^{|\Sigma|n}(1-1/n^2-1/n^3 + 3/n)$
- $K_n(\sigma_1\sigma_2, \sigma_2\sigma_2) = 2^{n-2}n^{|\Sigma|n}(2/n^2 + 2(n-1)/n^2 + ((n-1)(n+1))/n^2)$
- $K_n(\sigma_1\sigma_1, \sigma_1\sigma_2) = 2^{n-2}n^{|\Sigma|n}((n+1)/n)$
- $K_n(\sigma_2^k\sigma_1,\sigma_2^j) = 2^{n-2}n^{|\Sigma|n}((n+1)/n)$
- Also Note: $K_n(x,y) \leq K_n(xz,yz)$

Algorithm to Compute K_n Efficiently Given Short String x and y

Idea: Start with a completely un initialized semi-automata M (only distinguished state is q_0). Run M on x and y until we finish or until we reach an undefined transition on both x and y. For each possible transition fix that transition, mark any new states and recursively run M on the remainder of the strings.

Note this algorithm takes exponential time on |x| and |y|, but shows that $K_n(x,y)$ can be easily computed if $|x|,|y| << n$

Computing $K_n(x,y)$ requires reading in x,y completely

Let $\Sigma = \{0,1\}$

Pf:

Consider $x = 1^{n+n!}$, $y = 1^n$

Given a semi-automaton M with n states, reading in 1^n places on some state q_i, on some cycle of 1 transitions (by the pigeon hole principle). This cycle has length $L \leq n$. Reading in $1^{n!}$ starting at q_i takes us around the cycle n!/L times and ends at q_i again.

Hence,

$K_n(1^{n+n!},1^n) = |DFA(n,\Sigma)| /2$.

Now build a semi-automaton M' with n states having a cycle of length 2. $1^{n+n!}$, $1^{n+n!+1}$ must on different states hence,

$K_n(1^{n+n!+1}, 1^n) \neq |DFA(n,\Sigma)| /2$

But it is impossible to tell if $x = 1^{n+n!}$ or $x = 1^{n+n!+1}$ without reading in the entire string. □

Computing $K_n(x,y)$ requires reading in x,y completely

Interesting Note: The proof implies that it is impossible for any DFA with less than n+1 states to distinguish $1^{n+n!}$ and 1^n

So if we let $S = \{1^{n+n!}\}$ and $S' = \{1^n\}$, then the minimum consistent DFA must have more than n states

Modified Problem That is At Least As Hard as Factoring

Let $A(n,x) = K_n(1^{n+x},1^n)$

IMPORTANT NOTE: the input length is $O(\log n + \log x)$, while the input word have length $O(x)$, so this does not necessarily imply that computing K_n is hard

Let $M \in \mathbb{N}$ be given, and let $n = \lceil M^{1/2} \rceil$

Note that: $A(n,M,0) = A(n,1,0) \Leftrightarrow$

$K_n(1^{n+M},1^n) = K_n(1^{n+1},1^n) \Leftrightarrow$

$\forall j \in [2,\ldots,n]$, j does not divide M

Pf: Note that if a semi-automaton has a cycle of length 1 then it does not distinguish 1^n from 1^{n+b} for any b. Otherwise, if the semi-automaton has a 1-cycle of length greater than 1 then it must distinguish 1^n from 1^{n+1}

If there is $j \in [2,\ldots,b]$, st $j \mid M$ then consider a semi-automaton with n states and a cycle of length j. Then the semi-automaton does not distinguish between 1^{n+M} and 1^n. Hence, $K_n(1^{n+M},1^n) > K_n(1^{n+1},1^n)$.

Otherwise, if there is no $j \in [2,\ldots,b]$, st $j \mid M$, then consider any semi-automaton with n states having a cycle of length (L) greater than 1. Because L does not divide M the semi-automaton must distinguish 1^{n+M} from 1^n. Hence, $K_n(1^{n+M},1^n) = K_n(1^{n+1},1^n)$

Modified Problem That is At Least As Hard as Factoring

Conclusion: An efficient algorithm to compute A(n,x) would yield an efficient algorithm to factor any integer M. (Could simply perform binary search for the smallest factor)

In the original problem the length of the input would be x and not log x, however this proof seems to indicate that the “hardness” in computing K_n does not just come from reading in the strings.

Turing Machine Kernel?

Given a fixed alphabet Σ and $x,y \in \Sigma^*$,

Define:

$T^{(n)}(x,y) := \{<H>:H$ is a Turing Machine with n states st. H accepts x and y $\}$

$K_n^{(TM)}(x,y) := |T^{(n)}(x,y)|$

Claim: $K_n^{(TM)}(x,y)$ is NOT computable.

Turing Machine Kernel is Incomputable

Suppose that there was some Turing Machine T_K to compute $K_n^{(TM)}(x,y)$. Then we could decide A_{TM} as follows:

H = "On input (<M>,w)

- Enumerate all Turing Machines with n = |M| states (there are finitely many such Turing machines, and M is one of them)
- Use T_k to compute $K_n^{(TM)}(w,w)$
- Set finishedCounter = 0
- Run each enumerated machine on w in parallel (each one step at a time)
- If M ever Accepts/Rejects then Accept/Reject
- If some machine finishes (accepts or rejects) increment finishedCounter
- If finishedCounter == $K_n^{(TM)}(x,y)$ and M has not halted then Reject (There are only $K_n^{(TM)}(w,w)$ Turing machines with n states that accept w and hence M is not one of them). Note that $K_n^{(TM)}(x,y)$ of the Turing Machines MUST accept w after finitely many steps.

This is a contradiction: A_{TM} is not decidable. Hence, $K_n^{(TM)}(x,y)$ is not computable.