Probabilistic algorithm on Turing Machine

Zipeng (Leo) Lin

December 11, 2022

In this paper, we illustrate how some probabilistic algorithms might outdo original algorithms in the scope of theory of computation. Specifically, this paper introduces probabilistic Turing Machine (PTM), how it works, and some examples, including how PTM could recognize recursively enumerable sets, and how it might recognize some languages faster than deterministic Turing Machine.

1 Introduction

Probabilistic methods used in computation have been used for problems that have probabilistic concerns. For instance, Monto Carlo methods are used to obtain information about the behavior of large systems. Monto Carlo methods could also be used to mimic integration and convolution.

2 Probabilistic Turing machine

The main tool to explore in this paper is probabilistic Turing machine, a computer that has ability to make random decisions. That is, the same input might decide different outputs from the model since the output is randomized.

We are motivated to explore whether adding randomness to the Turing Machine adds power to its computation. Specifically, this paper tries to display some problems that can be solved by a probabilistic Turing Machine better than the deterministic Turing Machine. The criteria in doing "better" includes solving problems in polynomial time by the probabilistic Turing Machine but those problems are not possible to solve in polynomial time by the deterministic Turing machine in polynomial time, since there have been literature that claims that adding randomness to Turing Machine does not add computation power to the Turing Machine.

Definition 1 (Probabilistic Turing Machine)**.** A Probabilistic Turing Machine is a tuple of nine elements,

$$M_p = \{Q, \Sigma, \Gamma, \delta_1, \delta_2, q_0, \sqcup, q_f, q_r\}$$

The meaning of those symbols are

1. Q is the set of states, Σ is the input alphabet, Γ is tape alphabet, $\sqcup$ is the blank symbol.
2. q_0, q_f, q_r are initial state, accepting state and rejecting state.
3. $\delta_1, \delta_2 : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ are two probabilistic transition function whose outputs include whether moving one cell to the left or right on the tape.
4. We choose either δ_1 or δ_2 for transition function at each step. The probability to choose either of them is $1/2$.

We notice that the only difference a probabilistic Turing machine have from a deterministic one is in the transition functions. PTM has two transition functions.

3 Error induced by PTM

Since the PTM is not deterministic, it is bound to make errors. Therefore, instead of the usual definition of accepting a language by a Turing machine, a language N is said to be recognized

with error probability ϵ by a Turing machine M if:

Definition 2 (Error probability)**.** A language N is said to be recognized **with error probability** ϵ by a Turing machine M if: If a string s is in N, then $Pr[M \text{ accepts } w] \geq 1 - \epsilon$, and the reject probability is no smaller than $1 - \epsilon$ otherwise.

4 PTM accepting a recursively enumerable set

Recall the definitions induced by deterministic Turing Machines, a set S is **recursively enumerable** if there is a Turing machine M such that $S = \mathbb{L}(M)$. We can construct a probabilistic Turing machine to accept S at a finite average run time.

Proof. We suppose a recursively enumerable set S is accepted by a Turing machine M, then we construct a probabilistic machine M' by follows:

We repeat to flip a coin as long as it is not landing heads, and after each tail coin flip we run one step of $M(x)$ (x is the input), and if x is accepted here then we accept it. Besides, after the coin toss is heads, then we do another coin flip, and if it is heads then we accept and reject otherwise.

We have for x in S, we accept it with an probability bigger than $1/2$ (so it is in the domain of M'), and it not, we reject it with probability of $1/2$ and is not in the domain of M'. The average run time would be $1/2 * 2 + 1/4 * 3 + \ldots \approx 3$. □

5 Example of PTM's speedup

In this section, the paper gives a example of language that can be recognized more quickly by one-tape PTM and one-tape DTM.

Here is the setup before giving the example: denote the ith symbol of a string w by $w[i]$. Let $r(n)$ to be the fraction of symbols in the first half of w, a string of length $2n$, which equal the corresponding symbols in the latter half of w. Also, define P_λ to be the palindrome-like language

of binary strings w of even length such that $r(w) \geq \lambda$. If $\lambda = 1$, we can see that P_1 would be a palindrome, since the ratio of $r(n)$ is one, which means all the characters in the first half is equal to characters in the second half.

The example is the following

Example 3 (Language where PTM recognizes faster)**.** Suppose that λ is a rational number such that $0 < \lambda < 1$. There is a one-tape PTM , M', that recognizes P_λ with bounded error probability and runs faster for infinitely many inputs than every one-tape deterministic Turing machine that recognizes P_λ. That is to say: if M is any one-tape deterministic Turing machine that recognizes P_λ, then the maximum runtime of M' is less than the run time of M for infinitely many inputs [2].

The conclusion of the example above means that if M' is any one-tape deterministic Turing machine that recognizes P_λ, then the maximum run time of M is less than the rum time of M' for infinitely many inputs.

Lemma 4 (A PTM that has a good runtime)**.** For every rational number $r \in (0, 1)$ there is one-tape probabilistic Turing machine M that recognizes P_r with bounded error probability and maximum run time of $O(n \log n)$ for inputs in P_1.

Proof. Consider a error probability bound $\epsilon > 0$. Let m be an big integer such that $((1+\lambda)/2)^m < \epsilon$. This is possible because $(1 + \lambda)/2 < 1$ Suppose M_0 is any standard one-tape deterministic Turing machine that recognizes P_λ. Let M be a one-tape probabilistic machine that with input w operates as follows.

Firstly, check the input length is even, and then get the binary representation of n called n' (this takes $O(n \log n)$ steps for the Turing machine to convert and write.)

Secondly, select m numbers $i_1, i_2, \ldots, i_m$ such that $1 \leq i_j \leq n$. Those m numbers are decided by choosing random bit from α_j and add one to it after taking mod n. Then, we compare the i_jth entry of the string w and $n + i_j$th entry of w for each j. If they are the same for each j then we accept. Otherwise, we stop comparing and simulate the deterministic M_0 to determine

whether the sting w belongs to P_r. We have the inputs in P_1 are accepted by M by getting all the $n + i_j$th and i_jth element the same. Therefore, it takes $O(n \log n)$ runtime because it takes $O(n \log n)$ runtime to select and write down the bit.

After proving that we can accept using the PTM in $O(n \log n)$ runtime, we show that M is has bounded error probability. That is to say, M recognizes P_r with error probability less than ϵ. We have there are more than $(1 - \lambda)n$ of the numbers between 1 and n for i such that ith entry for w is not equal to $n + i$th entry at w. We have $1 - (1 - \lambda)n/2n = (1 + \lambda)/2$ is the probability that the entry are the same. Therefore, the probability that they are the same for every j is at most $((1 + \lambda)/2)^m < \epsilon$ so we are done.

Another lemma used to prove the statement is that the deterministic Turing machine has runtime $O(n^2)$. The proof of the lemma uses crossing sequence heavily, so it would be out of scope of this paper. □

6 Classes of languages for PTM

This section devotes to explore classes of languages computable probabilistically in polynomial time and investigate relationships between those classes.

Definition 5 (Polynomial bounded)**.** A probabilistic Turing machine is *polynomial bounded* if there is a polynomial $p(n)$ such that every possible computation of the machine on inputs of length n halts in at most $p(n)$ steps.

Definition 6 (Recognize)**.** A probabilistic Turing machine *recognizes* a language if the machine computes the characteristic function of the language.

Definition 7 (Several classes of languages)**.** *PP* is the class of languages recognized by polynomial bounded PTM's. (**Polynomial** Probabilistic (TM)).

BPP is the class of languages recognized by polynomial bounded PTM's with bounded error probability (Bounded Polynomial Probabilistic (TM)).

ZPP is the class of languages recognized by PTMs with polynomial bounded average run time and zero error probability.

Proposition 8 (Relationship between classes of langguages)**.** We have

$$\text{ZPP} \subset \text{BPP} \subset \text{PP}$$

also, PP, BPP, and ZPP are closed under complementation. BPP and ZPP are closed under union and intersection.

Proof. By definition, we have BPP $\subset$ PP because of the definition. To prove that ZPP $\subset$ BPP, suppose a language L is recognized by a PTM M with no error and has a average polynomial runtime $O(p(n))$. Consider another PTM, M', that recognizes the language L by simulating M for up to $cp(n)$ steps on inputs of length n. If the simulated computation of M does not halt, then M' halts with an arbitrary answer since any number could work here. M requires more than $cp(n)$ steps with probability less than $1/c$, the error probability of the polynomial bounded machine M' is at most $1/c$. Therefore, if we pick c to be bigger than 2, $1/c < 1/2$ so M would not require more than $cp(n)$ because the probability is less than one half here.

To prove that PP, BPP, and ZPP are closed under complementation, we can just change the reject state to accept state and vice versa.

ZPP is closed under intersection because for two Turing machine M_1 and M_2 and their languages L_1 and L_2, then we have $L_1 \cap L_2$ is also in ZPP since its runtime is less than the maximum of one of it. For union, we have the runtime would be at most the sum of them, which is still in ZPP.

For BPP, we just need to make sure the error ϵ is still bounded for machines that recognize both languages. Suppose L_1 and L_2 belong to BPP. For error $\epsilon > 0$ we can find polynomial bounded probabilistic Turing machines M_1 and M_2 that belong to BPP with error probability at most $\epsilon/2$. The machine that recognizes $L_1 \cup L_2$ would have error probability at most $\epsilon/2 + \epsilon/2 = \epsilon$, so BPP is closed under union. For intersection, we can also pick machines with error probability $\epsilon/2$. $\square$

7 Some conjectures and further work

There is an conjecture about $P = BPP$ [1], and it is analogically to $NP = P$. The interpretation behind $P = BPP$ is that does randomness truly add power (can it recognize more strings?).

There is still progress on showing $P = BPP$, such as

- $P \subset BPP$
- $P \subset P/\text{poly}$, where P/poly is a class such that if a language L is in the class of P/poly, the there is a function $h(n)$ whose length $|h(n)|$ is bounded by a polynomial of n such that there is a deterministic polynomial-time Turing machine M_d that takes $\langle x, h(|x|)\rangle$ and decides whether it is in L.

In the future, the author will further explore the content of P/poly.

References

[1] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[2] John T Gill III. "Computational complexity of probabilistic Turing machines". In: *Proceedings of the sixth annual ACM symposium on Theory of computing*. 1974, pp. 91–95.