Probabilistic Algorithms

Klaus Sutner

Carnegie Mellon University

45-rand-classes 2017/12/15 23:17

1. Some Probabilistic Algorithms

- Probabilistic Primality Testing

- RP and BPP

We have an apparently correct abstract definition of randomness, regrettably but necessarily one that excludes computability.

On the other hand, there are methods to get reasonably random bits by iteration (Mersenne twister) or via quantum physics (Quantis card). We can use these pseudo-random bits to speed up algorithms.

It remains to identify complexity classes that correspond to these randomized algorithms, and check how they relate to traditional classes.

We are given a list $A = a_1, \ldots, a_n$ and some index k, $1 \leq k \leq n$.

The problem is to find the k-smallest element in the list:

$$\text{ord}(k, A) = a'_k$$

where $a'_1, a'_2, \ldots, a'_n$ is the sorted version of the list.

For simplicity, assume all elements are distinct, so there is a unique solution.

Special cases: $k = 1$: min, $k = n$: max, $k = n/2$: median.

The obvious answer is: sort the damn sequence. Alas, this is overkill and does more than the problem specifies. For small values of k we could also use a buffer.

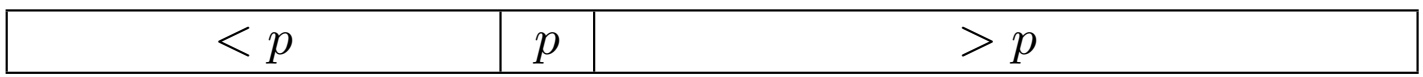
Here is a different idea: if someone gave us an alleged solution b, we could easily verify that b is the right answer: count the elements less than b and check that there are exactly $k-1$ of them.

$$b = \mathsf{ord}(k, A) \iff |\{\, i \mid a_i < b \,\}| = k - 1.$$

Of course, verification is not enough, we need get our hands on b.

So we need to pick the element b that is larger than exactly $k-1$ others.

Insight: This looks vaguely (OK, very vaguely) like the partitioning technique from quick sort.



Suppose we pick some pivot p at random and partition. If in fact $p = \mathsf{ord}(k, A)$ then the left block will have size $k-1$.

Of course, in general $p = \text{ord}(m, A)$ for some $m \neq k$ and the left block will have size $m - 1$.

Let's write (B, p, C) for the result of partitioning with pivot p and let m be the position of p after partitioning.

- If $m = k$: return p.
- If $m > k$ repeat with (B, k).
- If $m < k$ repeat with $(C, k - m)$.

If we are out of luck, this will lead to bad splits and linear recursion depth. But if we pick the pivot at random this method is very fast on average.

Lemma

Partitioning solves the order statistics problem in expected linear time.

For quite some time it was believed that order statistics was as difficult as sorting and could not be done in less than log-lin time for the worst case (assuming that that only full comparisons between one item and another are possible).

In fact, people tried to show that sorting could be reduced to order statistics.

It was a huge surprise when finally divide-and-conquer algorithm was found that runs in a linear time.

Unfortunately, the constants are so bad that the algorithm is of no practical importance.

Theorem (Blum, Floyd, Pratt, Rivest, Tarjan 1973)

Order statistics can be handled in linear time.

Problem:	**Polynomial Zero Testing (PZT)**
Instance:	A polynomial $P(x_1, \ldots, x_n)$ with integer coefficients.
Question:	Is P identically 0?

Wait, this is totally trivial ...

Yes, but there is a glitch: P need not be given in explicit form as a cofficient vector.

For example, we could have $P = P_1 \cdot P_2 \cdot \ldots \cdot P_r$ where the P_i are polynomials. Of course, we can obtain the explicit form by multiplying out, but that is a potentially exponential operation.

In fact, we should think of the polynomial as given in form of a straight-line program or an arithmetic circuit using operations $\{+, -, \times\}$.

Let $\mathbb{F}$ be a field (such as the rationals, reals, complexes).

Lemma (Schwartz-Zippel 1980)

Let $P \in \mathbb{F}[x_1, \ldots, x_n]$ be of degree d and $S \subseteq \mathbb{F}$ a set of cardinality s. If P is not identically zero, then P has at most $d\, s^{n-1}$ roots in S.

Proof.

The proof is by induction on n, the number of variables. The case $n = 1$ is clear.

For $n > 1$, define d_1 and $P_1(x_2, \ldots, x_n)$ to be the degree of x_1 in P and the coefficient, respectively. Hence $P(x_1, \ldots, x_n) = x_1^{d'} P_1(x_2, \ldots, x_n) + \text{stuff}$.

Suppose $a_2, \ldots, a_n \in S$ is a root of P_1, by induction we know there are at most $s^{n-1}(d - d')$ such roots. Then $a, a_2, \ldots, a_n$ could be a root for P for all $a \in S$. Otherwise, there are at most d' such roots. Adding, we get the claim.

□

The set $S \subseteq \mathbb{F}$ here could be anything. For example, over $\mathbb{Q}$ we might choose $S = \{0, 1, 2, \ldots, s-1\}$.

The main application of the lemma is to give a probabilistic algorithm to check whether a polynomial is identically zero.

Suppose P is not identically zero and has degree d. Choose a point $\boldsymbol{a} \in S^n$ uniformly at random and evaluate $P(\boldsymbol{a})$. Then

$$\Pr[P(\boldsymbol{a}) = 0] \leq \frac{d}{s}$$

So by selecting S of cardinality $2d$ the error probability is $1/2$.

Note that the number of variables plays no role in the error bound. To lower the error bound, we can repeat the basic step (relying on independence), or we can increase s.

Some combinatorial problems can be translated relatively easily into PZT.

For example, suppose $G = \langle \{1, 2, \ldots, n\}, E \rangle$ is a undirected graph. Define its Tutte matrix by

$$A(i,j) = \begin{cases} x_{ij} & \text{if } (i,j) \in E \text{ and } i < j, \\ -x_{ij} & \text{if } (i,j) \in E \text{ and } i > j, \\ 0 & \text{otherwise.} \end{cases}$$

The determinant of this matrix is a polynomial in up to n^2 variables x_{ij}.

Theorem (Tutte 1947)

G has a perfect matching iff its Tutte matrix has non-zero determinant.

The idea behind the proof is that in the standard representation of the determinant

$$|M| = \sum_{\pi} \mathsf{sign}(\pi) \prod_{i} M_{i,\pi(i)}.$$

perfect matchings π contribute a term, but nothing else does.

Hence, in explicit form, $|M|$ can be exponentially large.

But we can use Schwartz's lemma to determine whether it is zero in polynomial time, with small error probability.

This result holds for general graphs, not just bipartite ones. The proof is significantly harder in the general case. In the end, one obtains an $O(n^2e)$ algorithm, but it's tricky.

J. Edmonds

Paths, Trees, and Flowers

Canad. J. Math. 17(1965)449–467

- Some Probabilistic Algorithms

2. Probabilistic Primality Testing

- RP and BPP

The multiplicative subgroup of $\mathbb{Z}_m$ is

$$Z_m^\star = \{\, x \in \mathbb{Z}_m \mid \gcd(x, m) = 1 \,\}$$

Definition (Quadratic Residues)

$a \in Z_m^\star$ is a quadratic residue modulo m if $x^2 = a \pmod m$ has a solution over $Z_m^\star$, and a quadratic non-residue otherwise.

Thus $Z_m^\star$ can be partitioned into QR_m and QNR_m. For example

$$\mathsf{QR}_7 = \{1, 2, 4\}$$
$$\mathsf{QNR}_7 = \{3, 5, 6\}$$

Here are the 2 quadratic residues a for $m = 20$, a composite modulus, together with the solutions to $x^2 = a \pmod{m}$.

$$\begin{aligned} a &: x \\ 1 &: 1, 9, 11, 19 \\ 9 &: 3, 7, 13, 17 \end{aligned}$$

The non-residues are 3, 7, 11, 13, 17, 19.

This was done by brute-force computation (compute squares mod m).

Is there a way to check efficiently whether a number is a quadratic residue?

Suppose p is an odd prime and $z \in Z_p^*$. The Legendre symbol $\mathrm{LS}(z,p)$ is defined by:

$$\mathrm{LS}(z,p) = \begin{cases} +1 & \text{if } z \text{ is a quadratic residue,} \\ -1 & \text{otherwise.} \end{cases}$$

Proposition

$\mathrm{LS}(z,p) = z^{(p-1)/2} \pmod{p}$.

Write $\mathrm{M}(k)$ for the number of steps needed in multiplying two k-bit numbers.

Then the proposition provides an $O(\lg p\, \mathrm{M}(\lg p))$ algorithm to compute $\mathrm{LS}(z,p)$: use a fast exponentiation algorithm modulo p.

Unfortunately, we need p to be an odd prime here.

Suppose $q = p_1 \cdot \ldots \cdot p_r$ where the p_i are odd primes, not necessarily distinct, z relatively prime to q. The Jacobi symbol $\left(\frac{z}{q}\right)$ is defined by:

$$\left(\frac{z}{q}\right) = \mathsf{LS}(z, p_1) \cdot \mathsf{LS}(z, p_2) \cdot \ldots \cdot \mathsf{LS}(z, p_r) \in \{-1, +1\}.$$

If we knew the p_i then we could easily compute the Jacobi symbol according to its definition.

But what if all we know is q?

The following properties of the Jacobi symbol will help in finding an algorithm. The proof is quite elementary.

$$\left(\frac{z}{q}\right) = \left(\frac{z \bmod q}{q}\right)$$

$$\left(\frac{z_1 \cdot z_2}{q}\right) = \left(\frac{z_1}{q}\right) \cdot \left(\frac{z_2}{q}\right)$$

$$\left(\frac{1}{q}\right) = 1$$

$$\left(\frac{-1}{q}\right) = \begin{cases} -1 & \text{if } q = 3 \pmod 4, \\ +1 & \text{otherwise.} \end{cases}$$

$$\left(\frac{2}{q}\right) = \begin{cases} -1 & \text{if } q = 3, 5 \pmod 8, \\ +1 & \text{otherwise.} \end{cases}$$

We need one more ingredient for the algorithm. This is one of Gauss's favorite results, he gave several proofs for it.

Theorem (C. F. Gauss 1798)

Let p, q be two odd primes. Then we have

$$\left(\frac{p}{q}\right) = \begin{cases} -\left(\frac{q}{p}\right) & \text{if } p = q = 3 \pmod 4, \\ +\left(\frac{q}{p}\right) & \text{otherwise.} \end{cases}$$

The theorem together with the proposition makes it possible to apply mods to cut down the size of the numbers (much like in the computation of the GCD).

We may safely assume $0 < p < q$.

```
sign = 1;
while( p > 0 ) {
    while( p even ) {          // eliminating even part
        p /= 2;
        if( q mod 8 == 3, 5 )
            sign = -sign;
    }
    swap p and q;              // quadratic reciprocity
    if( p, q == 3  mod 4 )
        sign = -sign;
    p = p mod q;
}
if( q == 1 )
    return  sign
else
    return 0;                  // coprimality broken
```

First note that the algorithm maintains the invariant: $0 < p < q$, both odd.

If both numbers are coprime, they will stay so during the execution of the loop (we return 0 if it turns out that coprimality is violated). Correctness now follows from the mentioned properties of the Jacobi symbol.

For running time, note that all the numbers have at most k bits where k is the length of p and q. Hence, all the arithmetic operations are polynomial time. The same argument that shows that Euclidean algorithm is polynomial time can be used to show that the loop executes no more than $2k$ times.

With a little more effort one can show that the algorithm runs in time $O(\log p \ \log q)$.

The following table shows the computation for $p = 117$ and $q = 271$.

Result: -1.

p	q	sign
117	271	$+$
271	117	$+$
117	37	$+$
6	37	$+$
3	37	$-$
37	3	$-$
1	3	$-$

The Jacobi symbol is an extension of the Legendre symbol in the sense that whenever q is prime we have $\left(\frac{p}{q}\right) = \mathsf{LS}\,(p, q)$.

Note, however, that $\left(\frac{p}{q}\right) = +1$ no longer implies that p is a QR modulo q. For example, $\left(\frac{2}{9}\right) = \left(\frac{2}{3}\right)^2 = 1$ but $2 \in \mathsf{QNR}_9$.

Hence we now have two different ways to compute the Legendre symbol for q prime. This is the basic idea behind the Solovay-Strassen algorithm: pick z at random and compute $\left(\frac{z}{p}\right)$ in two ways, say, with results LS_1 and LS_2.

If $LS_1 \neq LS_2$ then p is not prime; if $LS_1 = LS_2$ then p is prime with a certain high probability.

Solovay-Strassen Primality Testing Algorithm
Input: n

pick z at random, $1 < z < n$

if $\gcd(z, n) > 1$
then return NO

$LS_1 = z^{(n-1)/2} \pmod{n}$
$LS_2 = \left(\frac{z}{n}\right)$

if $LS_1 \neq LS_2$
then return NO
else return YES?

Theorem (Solovay-Strassen)

If n is prime, the Solovay-Strassen test returns YES?; otherwise, the test returns NO with probability at least $1/2$.

The running time is $O(\log n \, \mathsf{M}(\log n))$.

Proof.

Suppose n is composite and consider the set S of bad choices for our algorithm where we get a false positive:

$$S = \left\{ z \in Z_n^\star \,\middle|\, z^{(n-1)/2} = \left(\frac{z}{n}\right) \pmod{n} \right\}.$$

Claim: $|S| \leq |Z_n^\star|/2$.

First note the S is a subgroup because of the multiplicativity property of the Jacobi symbol.

It remains to show that S is proper. Assume otherwise, so that in particular $z^{n-1} = 1 \pmod{n}$ for all $z \in Z_n^\star$.

Consider the prime factor p^e of n with maximal exponent e and set $m = n/p^e$.

Pick a generator g for the multiplicative subgroup of $\mathbb{Z}_{p^e}$ (which is known to exist).

By the CRT, there is an element $a \in Z_n^\star$ such that $a = g \pmod{p^e}$ and $a = 1 \pmod{m}$. From our assumptions, $a^{n-1} = 1 \pmod{n}$, and therefore $g^{n-1} = 1 \pmod{p^e}$, so that $\mathsf{ord}(g; Z_{p^e}^\star) \mid n-1$.

If $e > 1$ then we have $\text{ord}(g; Z^\star_{p^e}) = p^{e-1}(p-1) \mid n-1$, a contradiction.

If $e = 1$ then n must be square-free and contain at least two prime factors. Clearly $\left(\frac{a}{p}\right) = -1$ since g is a generator, and $\left(\frac{a}{q}\right) = 1$ for all the other prime factors of n. Hence $a^{(n-1)/2} = \left(\frac{a}{n}\right) = -1 \pmod{n}$. But that contradicts $a = 1 \pmod{m}$.

It follows that n is composite then the probability of the Solovay-Strassen algorithm returning NO is at last $1/2$.

□

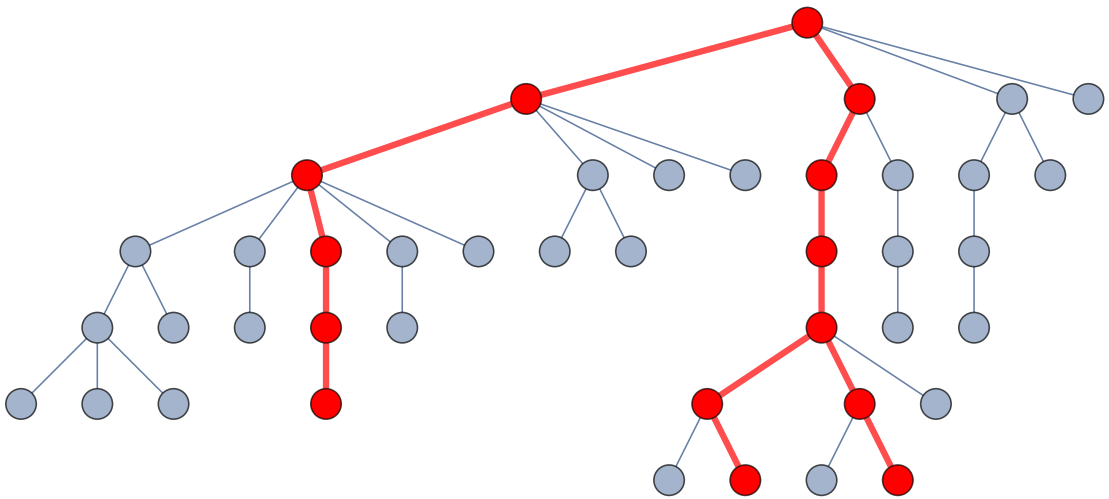
Note that Solovay-Strassen shows that Composite is in $\mathbb{NP}$ (also shown by V. Pratt in 1975 using a different method).

- Some Probabilistic Algorithms
- Probabilistic Primality Testing

3. RP and BPP

These examples and many more make it tempting to expand our complexity zoo a bit: there should be complexity classes corresponding to probabilistic algorithms.

For anyone familiar with nondeterministic classes like $\mathbb{NP}$ this is not much of a stretch: we already have computation trees



All we need to do is impose additional conditions on accepting branches.

Technically, it is convenient to define a probabilistic Turing machine (PTM) M to be a Turing machine acceptor with two transition functions δ_0 and δ_1.

At each step in the computation, M chooses $\delta_{0/1}$ with probability $1/2$ (and, of course, independently of all other choices).

Write $M(x) \in \mathbf{2}$ for the random variable that indicates acceptance/rejection.

We say that M runs in time $t(n)$ if it halts in at most $t(|x|)$ steps regardless of the random choices made.

So there may be as many as $2^{t(n)}$ branches in the computation tree of M on an input x of length n.

The critical part of any argument here will be a bound on errors. There are two types of errors that are a priori independent. We would like $M(x) = L(x)$ for some language $L \subseteq \mathbf{2}^\star$. Alas ...

- False Positives
  We may have $x \notin L$ but $M(x) = 1$.
  Remedy: $\mathsf{Pr}[\, M(x) = 1 \mid x \notin L \,]$ small.

- False Negatives
  We may have $x \in L$ but $M(x) = 0$.
  Remedy: $\mathsf{Pr}[\, M(x) = 0 \mid x \in L \,]$ small.

We get different probabilistic classes by imposing different constraints on false positives/negatives.

Let $t : \mathbb{N} \to \mathbb{N}$ be a reasonable running time and $L \subseteq \mathbf{2}^\star$ a language.

Definition (Bounded Error Probabilistic Polynomial Time)

A PTM M decides L in time $t(n)$ if for all $x \in \mathbf{2}^\star$, M on x halts in at most $t(|x|)$ steps (regardless of random choices) and

$$\Pr[M(x) = L(x)] \geq 2/3$$

BPP is the class of all languages decided by a PTM in time $O(\mathrm{poly})$.

BPP seems to capture the intuitive notion of a problem efficiently solvable by a feasible (possibly probabilistic) algorithm very well.

To be clear: by $\Pr[M(x) = L(x)] \geq 2/3$ we mean

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq 2/3$$

$$x \notin L \Rightarrow \Pr[M(x) = 0] \geq 2/3$$

So the machine can make errors in both directions, but the probability of false positives/negatives is at most $1/3$.

This is quite different from classes like $\mathbb{NP}$ where there are no "errors" when $x \notin L$.

Prime, the problem of determining whether a number is prime, is obviously in co-$\mathbb{NP}$.

V. Pratt showed that Prime is in $\mathbb{NP}$ by constructing short witnesses to primality, a clever use of basic number theory. Alas, his method does not yield a BPP algorithm.

Probabilistic primality testing algorithm showed Prime to be in BPP, but it was not known to be in $\mathbb{P}$.

Then in 2002 Agrawal, Kayena and Saxena ruined everything by showing that Prime is in $\mathbb{P}$ (using no more than high school arithmetic in the process).

As with $\mathbb{NP}$, one can avoid the funky probabilistic Turing machines by invoking witnesses.

Theorem

L is in BPP *iff there is a polynomial time deterministic Turing machine M and a polynomial p such that for all $x \in \mathbf{2}^\star$:*

$$\Pr[M(x,w) = L(x) : w \in \mathbf{2}^{p(|x|)}] \geq 2/3$$

Again, this is different from $\mathbb{NP}$ in that we require not just one witness but lots of them.

This shows that BPP $\subseteq \mathrm{EXP}$: we can simply enumerate the potential witnesses and count the good ones.

The magic constant $2/3$ in these definitions is by no means sacred.

Define a new class BPP$'$ to contain all languages L such that there exists a poly time PTM M such that for some constant c:

$$\Pr[M(x) = L(x)] \geq 1/2 + |x|^{-c}$$

Theorem

BPP $=$ BPP$'$

Sketch of proof.

For any constant d, we can design a new PTM M' such that

$$\Pr[M'(x) = L(x)] \geq 1 - 2^{-|x|^d}$$

M' simply runs M $8|x|^{2c+d}$ times and takes a "majority vote." The correctness proof uses a Chernoff bound.

□

Lemma

BPP *is closed under union, intersection and complement.*

Proof.

Closure under complementation follows directly from the definitions.

To show closure under intersection, suppose M_i decides L_i in BPP, $i = 1, 2$. Build a new machine M that does the following:

- Compute $b_i = M_i(x)$.
- Return $\min(b_1, b_2)$.

We need to show that M is a BPP machine.

Case 1: $x \in L_1 \cap L_2$

Then $\mathsf{Pr}[M(x) = 1] \geq 2/3 \cdot 2/3 = 4/9$, which can be fixed by amplifying M_1 and M_2.

Case 2: $x \notin L_1 \cap L_2$

$$\begin{aligned}\mathsf{Pr}[M(x) = 0] &= \mathsf{Pr}[M_1(x) = 0 \vee M_2(x) = 0] \\ &= \mathsf{Pr}[M_1(x) = 0] + \mathsf{Pr}[M_2(x) = 0] - \mathsf{Pr}[M_1(x) = 0 \wedge M_2(x) = 0] \\ &\geq 2/3\end{aligned}$$

For the last step, consider the three cases $x \notin L_1 \cup L_2$, $x \in L_1 - L_2$ and $x \in L_2 - L_1$.

A BPP machine can make errors in both directions. Here is a more restricted version where in the case of a NO-instance the answer is always NO.

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq 2/3$$

$$x \notin L \Rightarrow \Pr[M(x) = 0] = 1$$

Definition

RP is the class of all languages decided by a one-sided error PTM in polynomial time.

So RP produces no false positives, but may produce false negatives, with low probability.

It follows immediately that RP $\subseteq \mathbb{NP}$; alas, closure under complementation vanishes.

Similarly one defines $\mathrm{co}-\mathrm{RP}$, the complements of all languages in RP.

So this means no false negatives, but potentially false positives.

PZT via Schwartz-Zippel and Solovay-Strassen are both in $\mathrm{co}-\mathrm{RP}$ (or in RP if you flip the question).

There is a similar truth amplification result as for BPP.

Lemma

RP *is closed under union and intersection.*

Here is a wild idea: how about a PTM that never makes a mistake?

$$x \in L \Rightarrow M(x) = 1$$

$$x \notin L \Rightarrow M(x) = 0$$

Here is the glitch: for some computations the running time may not be polynomial; the machine is fast only on average.

ZPP is the class of all such PTM with expected polynomial running time.

These are also called Las Vegas algorithms, as opposed to the more civilized Monte Carlo algorithms.

The randomized version of quicksort can be construed as a Las Vegas type algorithm (though withing poly time): with small probability it will have quadratic running time, on average the running time is log-linear.

In reality, we would use a clock to halt the algorithm if it has not returned any (necessarily correct) answer after a polynomial amount of time.

In this case we can think of the output as "don't know."

Lemma

A problem is in ZPP *iff it has an always-correct algorithm that has polynomial average-case running time.*

If we get a "don't know" we just run the the algorithm again.

The most permissive class of randomized polynomial time computation is obtained by considering PTM with poly running time and error bounds

$$x \in L \Rightarrow \Pr[M(x) = 1] > 1/2$$

$$x \notin L \Rightarrow \Pr[M(x) = 1] \leq 1/2$$

So in this case the error might get arbitrarily close to $1/2$ depending on the input (say, for input of size n it's $1/2 + 2^{-n}$). This wreaks havoc with arguments based on repeating the algorithm to amplify truth.

PP is closed under complement, union and intersection, but the proofs require quite a bit of effort.

One can show that PP $\subseteq \mathrm{PSPACE}$ by counting accepting computations.

Lemma

$\mathbb{P} \subseteq \mathsf{ZPP} = \mathsf{RP} \cap \mathrm{co}{-}\mathsf{RP}$

$\mathsf{RP}, \mathrm{co}{-}\mathsf{RP} \subseteq \mathsf{BPP} \subseteq \mathsf{PP}$

Some would argue that BPP is a better formalization of the elusive notion of "efficiently solvable problem" than $\mathbb{P}$.

However, it is not unreasonable to conjecture that $\mathbb{P} = \mathsf{BPP}$, so there may be no class between the two descriptions.

Alas, the relationship between BPP and $\mathbb{NP}$ is currently open.

It is known, though, that

$$\text{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$$

near the bottom of the polynomial time hierarchy.

It follows that $\mathbb{P} = \mathbb{NP}$ implies $\mathbb{P} = \text{BPP}$.