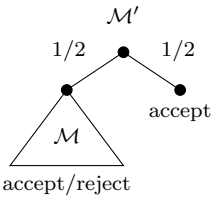
Using our probabilistic definition of the class NP, we can prove the following result.

Theorem 4. NP $\subseteq$ PP $\subseteq$ PSPACE.

Proof. First, we show that NP $\subseteq$ PP. Consider the computation tree of any nondeterministic Turing machine $\mathcal{M}$ recognizing a decision problem L. We can construct a PP-machine $\mathcal{M}'$ recognizing the same decision problem L by taking $\mathcal{M}$, adding a new initial state, and adding a nondeterministic transition out of the new initial state to one of two subtrees: the first subtree is rooted at the original computation tree of $\mathcal{M}$, while the second subtree is simply a leaf corresponding to an accepting computation.



Since $L \in$ NP, if $w \in L$, then there exists at least one accepting computation in the computation tree of $\mathcal{M}$, so $\mathbb{P}[\mathcal{M} \text{ accepts } w] > 0$ following the probabilistic definition of NP. As a consequence, in the computation tree of $\mathcal{M}'$, we accept with probability $1/2$ if we enter the "accepting computation" subtree from the initial configuration, and we accept with probability greater than 0 if we enter the "computation tree of $\mathcal{M}$" subtree from the initial configuration. Therefore, if $w \in L$, then $\mathbb{P}[\mathcal{M}' \text{ accepts } w] > 1/2$. We can make a similar argument when $w \notin L$. Thus, $\mathcal{M}'$ is a PP-machine.

To see that PP $\subseteq$ PSPACE, observe that the height of the computation tree of a PP-machine is bounded by some polynomial, since every computation branch runs in polynomial time. Thus, we can check every possible computation branch using a polynomial amount of space and count the number of accepting branches using a logarithmic-space binary counter. Overall, this process requires a polynomial amount of space. □

Remark. You may have noticed in our proof showing NP $\subseteq$ PP that, if $w \notin L$, then our computation tree for $\mathcal{M}'$ accepts w with probability exactly $1/2$, while our definition of PP states that we must accept w with probability strictly less than $1/2$! Fortunately, this doesn't present a problem due to our earlier observation that we can select any value $0 < \alpha < 1$ as our threshold and obtain a reasonable definition of the class PP.

In light of the fact that PP contains NP, it is clear that working with PP directly may be more trouble than it's worth, given that its definition is so general. What would be nice, then, is to come up with a more refined complexity class that "acts like" PP, but avoids the pitfall of arbitrarily small probability gaps.

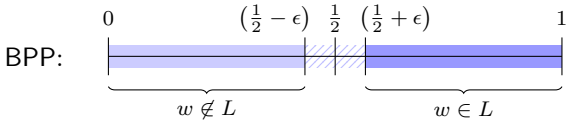
We can achieve this by enforcing the property that the probability gap maintains a minimum size; in other words, that our probabilistic Turing machine accepts inputs $w \in L$ with some probability $1/2$ plus a constant. This means that the probability gap cannot become arbitrarily small by, say, relying on the size of the input. By bounding the minimum size of the probability gap, we obtain the class BPP of *bounded probabilistic polynomial time* decision problems.

Definition 5 (The class BPP)**.** A decision problem L belongs to the complexity class BPP if there exists a probabilistic Turing machine $\mathcal{M}$ and a constant $0 < \epsilon \leq 1/2$ such that, given an input word w,

- if $w \in L$, then $\mathbb{P}[\mathcal{M} \text{ accepts } w] \geq 1/2 + \epsilon$; and
- if $w \notin L$, then $\mathbb{P}[\mathcal{M} \text{ accepts } w] \leq 1/2 - \epsilon$.

In the literature, you may see authors taking specific values such as $2/3$ to be the probability of acceptance—as before, this is arbitrary, and in that case such authors simply took $\epsilon = 1/6$ to be their constant. Any positive value can be used in the definition while still retaining the spirit of the class BPP.

Again, drawing a "picture" of the class BPP on a probability line gives us the following, where $0 < \epsilon \leq 1/2$ is our constant:



Note that, now, the probability gap is far more pronounced, and the difference between a correct output and an incorrect output is easier for us to discern depending on our value of ϵ. Indeed, a very nice property of the class BPP is that we can transform an arbitrary BPP-machine into one that has a probability of acceptance extremely close to 1 for all inputs $w \in L$ and extremely close to 0 for all inputs $w \notin L$, while still retaining the overall polynomial runtime of the machine. We will later see the technique that allows us to perform such a transformation.

For now, let us prove a couple more relationships between BPP and our other complexity classes. As before, when we showed that a decision problem in NP is a special case of a decision problem in PP, we can show that a decision problem in P is a special case of a decision problem in BPP.

Theorem 6. $\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{PP}$.

Proof. We begin by showing that $\mathsf{P} \subseteq \mathsf{BPP}$. Observe that, for any deterministic Turing machine $\mathcal{M}$ recognizing a language $L \in \mathsf{P}$, $\mathcal{M}$ accepts words $w \in L$ with probability $1 = 1/2 + 1/2$ and accepts words $w \notin L$ with probability $0 = 1/2 - 1/2$. Thus, $\mathcal{M}$ is a BPP-machine where $\epsilon = 1/2$.

Showing that $\mathsf{BPP} \subseteq \mathsf{PP}$ is rather straightforward. From the definition of the class BPP, if $w \in L$, then $\mathbb{P}[\mathcal{M} \text{ accepts } w] \geq 1/2 + \epsilon$ for some $\epsilon > 0$. Therefore, $\mathbb{P}[\mathcal{M} \text{ accepts } w] > 1/2$, which corresponds to the definition of the class PP. We can make a similar argument when $w \notin L$. Thus, every BPP-machine is also a PP-machine. □

Interestingly, while some researchers conjecture that $\mathsf{P} = \mathsf{BPP}$, nothing is known about the relationship between NP and BPP! However, it seems unlikely that $\mathsf{NP} \subseteq \mathsf{BPP}$, since this would imply that we can solve NP-complete decision problems reasonably efficiently.

3.2 One-Sided Error: RP

We now consider what happens when we slightly restrict the behaviour of our probabilistic Turing machine so that its computation still runs in polynomial time, but may err in only one of the two outcomes. In this case, our decision problem belongs to the class RP, representing *randomized polynomial time* decision problems.

Definition 7 (The class RP)**.** A decision problem L belongs to the complexity class RP if there exists a probabilistic Turing machine $\mathcal{M}$ and a constant $0 < \epsilon \leq 1/2$ such that, given an input word w,

- if $w \in L$, then $\mathbb{P}[\mathcal{M} \text{ accepts } w] \geq 1/2 + \epsilon$; and
- if $w \notin L$, then $\mathbb{P}[\mathcal{M} \text{ accepts } w] = 0$.

Just as the definition of the class PP corresponded to decision problems solvable by a Monte Carlo algorithm with *two*-sided error, the definition of the class RP exactly matches the behaviour of a Monte Carlo algorithm with *one*-sided error: in the case where the answer is "yes", it is incorrect with some bounded probability, and in the case where the answer is "no", it is always correct.

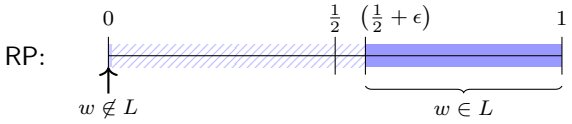
This means that RP-machines are effectively a strengthening of BPP-machines: they behave identically in the case where $w \in L$, but when $w \notin L$ an RP-machine will never accept. As a consequence, we have the following immediate result:

Theorem 8. RP $\subseteq$ BPP.

Proof. Follows from the definitions of the classes RP and BPP. □

Additionally, as we noted for the class BPP, our choice of value for the probability of acceptance in the definition of RP is arbitrary, and any positive value can serve as the threshold while again retaining the spirit of the class RP.

A "picture" of the class RP on a probability line looks like the following, where $\epsilon > 0$ is some constant:



You may have observed from the definition of the class RP that its acceptance condition for words $w \in L$ is effectively a strengthening of the acceptance condition in our probabilistic definition of the class NP. Therefore, we can draw the following connections between RP and our more familiar complexity classes.

Theorem 9. P $\subseteq$ RP $\subseteq$ NP.

Proof. The first relationship showing that P $\subseteq$ RP is established in exactly the same way as we established the relationship P $\subseteq$ BPP.

To see that RP $\subseteq$ NP, suppose we have a probabilistic Turing machine $\mathcal{M}$ recognizing a language $L \in$ RP. Using the specification of $\mathcal{M}$, we can construct a nondeterministic Turing machine $\mathcal{M}'$ that simply guesses which computation branch to follow instead of using randomization. Since $\mathcal{M}$ accepts words $w \in L$ with probability at least $1/2 + \epsilon$ for some $0 < \epsilon \leq 1/2$, over half of all computation branches lead to an accepting configuration. This means that at least one accepting computation branch exists, and so $\mathcal{M}'$ will accept words $w \in L$ with probability greater than 0. Thus, $L \in$ NP. □

Complementing the Class RP

From our earlier illustration of the class RP, we can clearly see that the acceptance conditions of an RP-machine are not symmetric for words $w \in L$ compared to words $w \notin L$. Ultimately, this means that unlike the classes PP and BPP, we don't know in general how to construct an RP-machine for a given complement language $\overline{L}$.

We may therefore analogously define a complementary class coRP to handle Monte Carlo algorithms with "other-sided" error in the usual way.

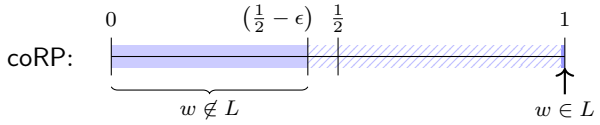
Definition 10 (The class coRP)**.** A decision problem L belongs to the complexity class coRP if there exists a probabilistic Turing machine $\mathcal{M}$ and a constant $0 < \epsilon \leq 1/2$ such that, given an input word w,

- if $w \in L$, then $\mathbb{P}[\mathcal{M} \text{ rejects } w] = 0$; and
- if $w \notin L$, then $\mathbb{P}[\mathcal{M} \text{ rejects } w] \geq 1/2 + \epsilon$.

Observe that, in our definition of coRP, we have shifted our perspective from probabilities of *acceptance* to probabilities of *rejection*. If some word belongs to the language of a coRP-machine, then it is guaranteed to accept that word, but it may err in rejecting if the word does *not* belong to its language.

Equivalently, we can simply say that a decision problem L belongs to the class coRP if and only if its complementary decision problem $\overline{L}$ belongs to the class RP. Since the acceptance conditions for PP and BPP are symmetric, we have that both PP = coPP and BPP = coBPP. On the other hand, the question of whether RP and coRP are equal remains open.

If we reinterpret the statement "$\mathbb{P}[\mathcal{M}$ rejects $w] \geq 1/2 + \epsilon$" to read "$\mathbb{P}[\mathcal{M}$ accepts $w] \leq 1/2 - \epsilon$", then we can illustrate a "picture" of the class coRP to go with that which we had for the class RP:



Take special note here that this illustration, like all of the others, depicts probabilities of *acceptance*. A coRP-machine will always accept words $w \in L$, while it may (with at most $1/2 - \epsilon$ probability) mistakenly accept words $w \notin L$.

Going along with the results we established in Theorems 8 and 9, it is possible for us to prove both that coRP $\subseteq$ BPP and that P $\subseteq$ coRP $\subseteq$ coNP. Moreover, if our earlier noted conjecture that P = BPP is indeed true, then we would ultimately have a collapse of complexity classes, where P = RP = coRP.

3.3 "Zero"-Sided Error: ZPP

For most decision problems solvable by a Monte Carlo algorithm, we need only concern ourselves with one side of the error (that is, erring either on "yes" answers or on "no" answers) because there exists only the one randomized algorithm for the problem. Thus, if we get a "no" answer, for example, we can be confident in its correctness, while we may need to take extra steps to verify a "yes" answer.

For a certain subset of decision problems, however, we can devise *two* Monte Carlo algorithms to correctly handle "yes" answers and "no" answers, respectively. In this case, since each algorithm errs in a different side of the output, combining these algorithms and ignoring the erroneous side of the output gives us a way of always obtaining the correct answer.

If a decision problem has two randomized algorithms handling either side of the output in this way, then we say it belongs to the class ZPP of *zero-error probabilistic polynomial time* decision problems.

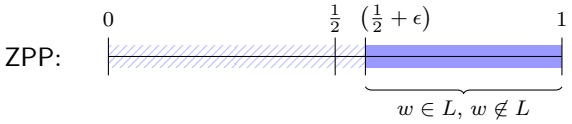
Definition 11 (The class ZPP)**.** A decision problem L belongs to the complexity class ZPP if there exists a probabilistic Turing machine $\mathcal{M}$ and a constant $0 < \epsilon \leq 1/2$ such that, given an input word w,

- if $w \in L$, then $\mathbb{P}[\mathcal{M}$ accepts $w] \geq 1/2 + \epsilon$, and otherwise halts in a "don't know" state; and
- if $w \notin L$, then $\mathbb{P}[\mathcal{M}$ rejects $w] \geq 1/2 + \epsilon$, and otherwise halts in a "don't know" state.

In our definition, the purpose of the "don't know" answer is to allow us to ignore the erroneous side of each algorithm's output, as we noted earlier.

Since a ZPP-machine always produces either the correct answer or a "don't know" answer, we can be confident that the ZPP-machine is correct if it either accepts or rejects its input word. However, since it is possible to receive "don't know" as the output, we can't say with confidence how long it will take for a ZPP-machine to produce its accept-or-reject answer. The class ZPP therefore corresponds to the set of decision problems solvable by a Las Vegas algorithm.

Once more, we can draw a "picture" of the class ZPP on a probability line, where $\epsilon > 0$ is some constant. However, here we simply interpret the probability line as illustrating the probability of getting the correct answer (i.e., if $w \in L$, we consider the probability of accepting, while if $w \notin L$, we consider the probability of rejecting):



Since the probabilities of a ZPP-machine accepting and rejecting its input word are symmetric, much like we had with the classes PP and BPP, it is the case that ZPP = coZPP.

Directly from the definition, we see that the class ZPP is effectively the set of all decision problems that can be solved by both an RP-machine and a coRP-machine. Indeed, this is what we mean by the decision problem having two randomized algorithms handling either side of the output. We can prove this fact formally as follows.

Theorem 12. $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$.

Proof. To show that $\mathsf{ZPP} \subseteq \mathsf{RP} \cap \mathsf{coRP}$, let $\mathcal{M}_{\mathrm{Z}}$ be a ZPP-machine recognizing a language L. We can construct an RP-machine $\mathcal{M}_{\mathrm{R}}$ recognizing the same language L by changing all "don't know" outputs of $\mathcal{M}_{\mathrm{Z}}$ to "reject" outputs. Now, $\mathcal{M}_{\mathrm{R}}$ accepts with probability $1/2 + \epsilon$ if $w \in L$ and rejects with probability 1 otherwise, so $\mathsf{ZPP} \subseteq \mathsf{RP}$. Similarly, we can construct a coRP-machine $\mathcal{M}_{\overline{\mathrm{R}}}$ recognizing L by changing all "reject" outputs of $\mathcal{M}_{\mathrm{Z}}$ to "accept" outputs and changing all "accept" and "don't know" outputs of $\mathcal{M}_{\mathrm{Z}}$ to "reject" outputs. Thus, $\mathsf{ZPP} \subseteq \mathsf{coRP}$ as well.

In the other direction, to show that $\mathsf{RP} \cap \mathsf{coRP} \subseteq \mathsf{ZPP}$, let $\mathcal{M}_{\mathrm{R}}$ be an RP-machine for some language L and let $\mathcal{M}_{\overline{\mathrm{R}}}$ be an RP-machine for the complement language $\overline{L}$. We can construct a ZPP-machine $\mathcal{M}_{\mathrm{Z}}$ for L in the following way by simulating the computations of both $\mathcal{M}_{\mathrm{R}}$ and $\mathcal{M}_{\overline{\mathrm{R}}}$ and taking the result to be (z_1, z_2), where $z_i \in \{\text{accept}, \text{reject}\}$ for $i \in \{1, 2\}$. The machine then makes its decision based on which result is produced:

- If the result is (accept, reject), then $\mathcal{M}_{\mathrm{Z}}$ accepts.
- If the result is (reject, accept), then $\mathcal{M}_{\mathrm{Z}}$ rejects.
- If the result is (reject, reject), then $\mathcal{M}_{\mathrm{Z}}$ outputs "don't know".
- The result (accept, accept) is impossible.

In the first case, $\mathcal{M}_{\mathrm{Z}}$ accepts with probability at least $1/2 + \epsilon$, and in the second case, $\mathcal{M}_{\mathrm{Z}}$ rejects with probability at least $1/2 + \epsilon$. Thus, $\mathsf{RP} \cap \mathsf{coRP} \subseteq \mathsf{ZPP}$.

Since both directions of the subset inclusion hold, we have that $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$ as desired. □

As a fun consequence, the proof of Theorem 12 gives us a method of converting two Monte Carlo algorithms to one Las Vegas algorithm, which nicely complements our Las-Vegas-to-Monte-Carlo conversion procedure given by Theorem 2.

We can of course establish a number of straightforward relationships between the class ZPP and other complexity classes. For instance, as a consequence of Theorem 9, we have that $\mathsf{ZPP} \subseteq \mathsf{NP} \cap \mathsf{coNP}$. In addition to this, since we know that both $\mathsf{P} \subseteq \mathsf{RP}$ and $\mathsf{P} \subseteq \mathsf{coRP}$, we have that $\mathsf{P} \subseteq \mathsf{RP} \cap \mathsf{coRP}$ and so, with Theorem 12, we can conclude that $\mathsf{P} \subseteq \mathsf{ZPP}$.

Finally, with ZPP-machines, we lose our guarantee that running the two algorithms together can always be done in polynomial time. However, it is possible for us to at least obtain an expectation on the runtime, and we can show this in the following way.

Theorem 13. *For any decision problem L, $L \in \mathsf{ZPP}$ if and only if L is recognized by a probabilistic Turing machine $\mathcal{M}$ that always correctly accepts or rejects its input word in expected polynomial time.*

Proof. ($\Rightarrow$): Suppose that $L \in \mathsf{ZPP}$; that is, $L \in \mathsf{RP} \cap \mathsf{coRP}$ by Theorem 12. Then there exists an RP-machine $\mathcal{M}_{\mathrm{R}}$ recognizing L as well as an RP-machine $\mathcal{M}_{\overline{\mathrm{R}}}$ recognizing $\overline{L}$. For any input w of length n, $\mathcal{M}_{\mathrm{R}}$ runs in some polynomial time $p_1(n)$ while $\mathcal{M}_{\overline{\mathrm{R}}}$ runs in some polynomial time $p_2(n)$. Moreover, we have the following cases depending on the input word w being read:

- If $w \in L$, then $\mathcal{M}_{\mathrm{R}}$ accepts with probability at least $1/2 + \epsilon$, while $\mathcal{M}_{\overline{\mathrm{R}}}$ rejects.
- If $w \notin L$, then $\mathcal{M}_{\mathrm{R}}$ rejects, while $\mathcal{M}_{\overline{\mathrm{R}}}$ accepts with probability at least $1/2 + \epsilon$.

Suppose we run both machines for time $p_1(n) + p_2(n)$. Then, if either machine accepts, we are sure of the output. However, since acceptance isn't guaranteed, we simply rerun the computations if neither machine accepts.

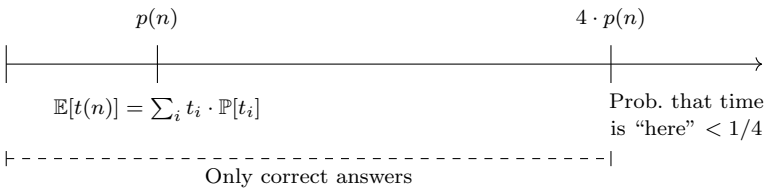
Because of this, there is a chance that the computation may (unfortunately) run forever. However, we can bound the expected running time in the following way:

$$\begin{aligned}\mathbb{E}[t(n)] &\leq (p_1(n) + p_2(n)) \cdot \left(1 \ + \ \frac{1}{2} \cdot 2 \ + \ \frac{1}{4} \cdot 3 \ + \ \frac{1}{8} \cdot 4 \ + \ \ldots\right) \\ &\leq (p_1(n) + p_2(n)) \cdot \left(\sum_{i=1}^{\infty} \frac{i}{2^{i-1}}\right) \\ &\leq (p_1(n) + p_2(n)) \cdot 4,\end{aligned}$$

where the sum term comes from the probability that a definite accepting output is not produced by either machine multiplied by the number of runs of both machines. Therefore, the expected running time is polynomial.

($\Leftarrow$): Suppose that the expected running time of some probabilistic Turing machine $\mathcal{M}$ recognizing a language L is bounded by a polynomial $p(n)$. We can use Theorem 12 to construct both an RP-machine recognizing L and an RP-machine recognizing the complement language $\overline{L}$.

- For the RP-machine recognizing L, given an input of length n, run $\mathcal{M}$ for $4 \cdot p(n)$ computation steps. If $\mathcal{M}$ accepts, then we accept. Otherwise, we reject. The probability of error in this case is at most $1/4$, since errors only occur if $\mathcal{M}$ accepts after $4 \cdot p(n)$ computation steps.



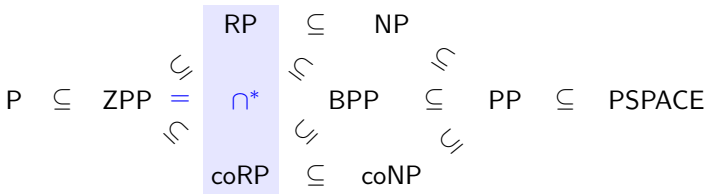
- For the RP-machine recognizing $\overline{L}$, we perform the same simulation as we did for L, but if $\mathcal{M}$ rejects, then we accept. Otherwise, we reject. The error probability in this case is again at most $1/4$.

Since, for either machine, the error probability is bounded by at most $1/4$, we have that $L \in \mathsf{ZPP}$. □

3.4 The Randomized Complexity Hierarchy

Combining all of our relationships between each of the randomized complexity classes we defined here, and including relationships between our fundamental complexity classes such as P, NP, and PSPACE, we can obtain a finer-grained randomized complexity hierarchy that refines our fundamental complexity hierarchy by "filling in" the relationships between P and PSPACE.

This randomized complexity hierarchy looks like the following:



Remark. The starred symbol between RP and coRP denotes intersection; that is, $\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$. It should not be interpreted as a strict inclusion relationship between RP and coRP.