6.541/18.405 Problem Set 2

due on **Tuesday, April 2, 11:59pm**

Rules: You may discuss homework problems with other students and you may work in groups, but we require that you *try to solve the problems by yourself before discussing them with others*. Think about all the problems on your own and do your best to solve them, before starting a collaboration. If you work in a group, include the names of the other people in the group in your written solution. **Write up your *own* solution to every problem**; don't copy answers from another student or any other source. Cite **all** references that you use in a solution (books, papers, people, websites, etc) at the end of each solution.

We encourage you to use LaTeX, to compose your solutions. The source of this file is also available on Piazza, to get you started!

How to submit: Use Gradescope entry code **2P3PEN**.

Please use a separate page for each problem.

Problem 1: NP, BPP, RP (2 points)

Question

Prove that if $\mathbf{NP} \subseteq \mathbf{BPP}$, then $\mathbf{NP} = \mathbf{RP}$.

Hint: Try using the self-reducibility of SAT.

Answer

Say $\mathbf{NP} \subseteq \mathbf{BPP}$. Then SAT $\in \mathbf{BPP}$ so there is a polynomial time probabilistic Turing machine M which decides SAT with probability at least 2/3. From this we can derive a TM $\bar{M}$ which decides SAT on any formula ϕ with probability at least $1 - 2^{-(n+m)}$ where n is the number of variables and m is the number of clauses in ϕ.

We can use $\bar{M}$ to produce an RP algorithm for SAT, which will try to produce a satisfying assignment $x_1, \ldots, x_n$, then verify that it is correct. First, construct formula ϕ_1 by setting $X_1 = 1$ in ϕ. Run $\bar{M}$ on ϕ_1. If this returns that ϕ_1 is satisfiable, set $x_1 = 1$; else set $x_1 = 0$. Now recurse this procedure on ϕ_1 if $x_1 = 1$, or on $\phi_{\neg 1}$ generated by fixing $X_1 = 0$ in ϕ if $x_1 = 0$. Now let ϕ be replaced with the restriction of ϕ to $X_1 = x_1$. Shift down the index of each variable (so what was called X_2 is now called X_1, etc.). Now recurse this procedure. This will ultimately produce a formula ϕ with no variables. If the resulting variable-free formula reduces to false, return that there is no satisfying assignment. If the formula reduces to true, consider the assignment $x_1, \ldots, x_n$ produced by this process. Deterministically check that it satisfies ϕ. If it does, return true, else, return false.

Observe that if the formula is not satisfiable, this algorithm will never return 1, since the assignment $x_1, \ldots, x_n$ which is generated will not satisfy the formula. Thus, to show that this algorithm puts SAT in $\mathbf{RP}$, we need to show that if the formula is satisfiable, the algorithm will return 1 with probability at least 2/3.

This algorithm has n recursive calls, and the probability level i assigns a variable incorrectly, or incorrectly states that the formula is unsatisfiable, is at most $2^{-(i+m)}$. Thus the overall probability something goes wrong is less than or equal to

$$\sum_{i=1}^{n} 2^{-(i+m)} < 2^{-m} < 2/3$$

This shows that SAT $\in \mathbf{RP}$, and since $\mathbf{RP} \subseteq \mathbf{NP}$, we have $\mathbf{NP} = \mathbf{RP}$.

Problem 2: A Tighter Circuit Lower Bound (2 points)

Question

We showed in class that for every k, there is a function $f_k \in \Sigma_3\mathbf{P}$ that does not have kn^k-size circuits. Use the ideas in the proof of this fact to prove that there is an $\varepsilon > 0$ and a function $f \in \mathbf{EXP}^{(\mathbf{NP}^{\mathbf{NP}})}$ that does not have $\varepsilon 2^n/n$ size circuits.

Answer

$\mathsf{HardFunction}(x : \{0,1\}^n)$

{ This is the lexographically first hard function from $\{0,1\}^n$ to $\{0,1\}$. }
Initialize a list of bits L with 2^n slots. Fill them all with 0.
{L will store a truth table for the hard function.}
for $j = 0, 1, \ldots, 2^n - 1$ **do**
    {This loop writes a truth table for the hard function to list L.}
    $b \leftarrow \mathsf{CheckIfHardCompletionExists}(L, j, 1^{2^n})$
    $L[j] \leftarrow \neg b$
end for
return $L[x]$

Algorithm HardFunction implements the lexographically first function g from $\{0,1\}^n$ to $\{0,1\}$ such that no circuit of size $\epsilon 2^n/n$ computes g. Such a function exists if we take $\epsilon = 1/10$, for instance.

HardFunction works by explicitly constructing a truth table for g, by writing down a truth table L of 2^n bits. It then returns $L[x]$. To construct the lexographically first string, when writing position j into the truth table, HardFunction checks if it is possible to turn the current list L into a full hard function if $L[j]$ is set to 0. If this is possible, it does this; otherwise it resorts to setting $L[j] \leftarrow 1$.

The procedure requires running 2^n checks, each of which requires writing 1^{2^n} onto the tape to call CheckIfHardCompletionExists. Therefore the algorithm takes $O(2^n \times 2^n) = O(2^{2n}) = 2^{O(n)}$ time, if we have an oracle for CheckIfHardCompletionExists.

Finally, we argue that indeed given an $\mathbf{NP}^{\mathbf{NP}} = \Sigma_2^P$ oracle, we can implement procedure CheckIfHardCompletionExists. The procedure existentially guesses a function g consistent with the partial truth table L and the choice $g(j) = 0$, and then verifies that for all circuits C, C does not implement g. (This last check requires looping over 2^n values $z \in \{0,1\}^n$.) This is in Σ_2^P with respect to the input which has size $\geq 2^n$.

Acknowledgements. I asked ChatGPT if $\mathbf{EXP}^{\Sigma_2^P}$ and $\Sigma_2^{\mathbf{EXP}}$ were the same (and it said no), which helped me discover the thread of thought I needed

CheckIfHardCompletionExists$(L, j, 1^n)$

{ Given a list L with $x - 1$ bits, return 1 iff there exists a function $g : \{0,1\}^n \to \{0,1\}$ such that $g(i) = L(i)$ for all $i < x$, and where $g(x) = 0$. }
Existentially guess a sequence B of $2^n - x$ bits.
Universally choose a circuit C of size $\epsilon 2^n/n$.
for $z \in \{0,1\}^n$ **do**
    {Compute $b \leftarrow g(z)$.}
    if $z = x$ **then**
        $b \leftarrow 0$
    else if $z < x$ **then**
        $b \leftarrow L[z]$
    else
        $b \leftarrow B[z - x]$
    end if
    {Verify $\exists z. C(z) \neq g(z)$.}
    if $C(z) \neq b$ **then**
        return 1
    end if
end for
return 0

to follow for this problem.

Problem 3: P=NP Implies Better Circuit Lower Bounds (3 Points)

Question

Prove that if $\mathbf{P} = \mathbf{NP}$ then there is an $\varepsilon > 0$ and $f \in \mathsf{TIME}[2^{O(n)}]$ such that $f \notin \mathbf{SIZE}[\varepsilon 2^n/n]$.

Answer

In the previous problem we showed that $\mathsf{HardFunction}$ can be implemented in $O(2^n(2^n+K(2^n)))$ time, where $K(m)$ is the runtime of $\mathsf{CheckIfHardCompletionExists}$. We also showed that $\mathsf{CheckIfHardCompletionExists} \in \mathbf{NP}^{\mathbf{NP}}$. Thus if $\mathbf{P} = \mathbf{NP}$, $\mathsf{CheckIfHardCompletionExists} \in \mathbf{P}$, so $K(m)$ is a polynomial, say km^k. Then $\mathsf{HardFunction}$ can be run in $O(2^n(2^n + k(2^n)^k))$. This simplifies to $O(2^{2n} + 2^{n+kn})$, which is $2^{O(n)}$. Since no circuits of size $\epsilon 2^n/n$ can decide $\mathsf{HardFunction}$, this yields the desired result.

Problem 4: BPL (3 points)

Question

A probabilistic TM is said to work in space $s(n)$ if every branch requires $O(s(n))$ space for inputs of size n and terminates in $2^{O(s(n))}$ time. The machine has *one-way access* to a read-only random tape, i.e. each random bit can only be read once.

Define the class **BPL** (for bounded-error probabilistic log-space) as follows: a language L is in **BPL** if there exists an $O(\log n)$-space probabilistic TM M such that

1. If $x \in L$, then $\Pr[M(x) \text{ accepts}] \geq 2/3$.
2. If $x \notin L$, then $\Pr[M(x) \text{ accepts}] \leq 1/3$.

Prove the following:

(a) $\mathbf{BPL} \subseteq \mathsf{SPACE}[(\log n)^2]$.

(b) $\mathbf{BPL} \subseteq \mathbf{P}$.

Answer to (a)

Let L be in **BPL**, witnessed by a log-space Turing Machine T with a 2/3 promise. Given x of length n, let C be the set of confirugations for T on x. W.l.o.g. let c_{init} be the unique initial state and let c_{acc} be the unique accepting state. Let kn^k be an upper bound on the runtime of T. Let $T_0, T_1 : C \to C$ be the functions describing the configuration T transitions to from a given state $c \in C$, where $T_0(c)$ is the new state if a 0 is read from the random tape, and $T_1(c)$ is the new state if a 1 is read from the random tape. Let M be the $|C| \times |C|$ matrix where $M_{i,j}$ is the probability that T eventually passes through state j, given that it began in state i. Our goal is to find a $\log(n)^2$ space Turing machine which can determine whether $M_{c_{\text{init}},c_{\text{acc}}} \geq 2/3$.

Let M^m be the $|C| \times |C|$ matrix where $M^m_{i,j}$ is the probability of going from configuration i to configuration j in $\leq m$ steps. Then $M = M^{kn^k}$. Observe that any element $M^m_{i,j}$ can be computed by enumerating over all sequences of m random bits and counting the number of transitions from i to j. Also observe that

$$M^{2^m}_{i,j} = \sum_{c \in C} M^{2^{m-1}}_{i,c} M^{2^{m-1}}_{c,j}$$

The algorithm $\mathsf{ApproximateTransitionProbability}(T, i, j, m, b)$ below approximately computes $M^{2^m}_{i,j}$, and represents the result as a b-bit binary fraction. (Approximation is forced by needing to store the result in b bits.)

Say $\hat{M}^{kn^k}_{i,j}$ is the value returned by

$$\mathsf{ApproximateTransitionProbability}(T, c_{\text{init}}, c_{\text{acc}}, \log(kn^k), a\log(n))$$

where a is a constant independent of the input x. (Note that for simplicity, I will henceforth assume kn^k is a power of 2, to avoid complicating the math by rounding kn^k up to the next power of 2 in all the following expressions.)

Say we can show that the following two properties hold:

1. $|M_{i,j}^{kn^k} - \hat{M}_{i,j}^{kn^k}| < \frac{1}{10}$
2. $\mathsf{ApproximateTransitionProbability}(T, c_{\text{init}}, c_{\text{acc}}, \log(kn^k), a\log(n))$ uses $O((\log n)^2)$ space

Then $L \in \mathsf{SPACE}[(\log n)^2]$, because we can decide $x \in L$ by computing $\hat{M}_{i,j}^{kn^k}$ and returning true iff $M_{i,j}^{kn^k} \geq \frac{2}{3} - \frac{1}{10}$.

$\mathsf{ApproximateTransitionProbability}(T, i, j, m, b)$

{ Approximate $\mathbb{P}$[T transitions from i to j within m steps]. Output the answer using b bits (ie. output $0.x_1x_2 \ldots x_b$). }
$p \leftarrow 0$ {Transition probability.}
if m $= 0$ **then**
  for $v \in \{0, 1\}$ **do**
    if $T_v(i) = j$ **then**
      $p \leftarrow p + \frac{1}{2}$
    end if
  end for
else
  for $c \in C$ **do**
    $p_1^c \leftarrow \mathsf{ApproximateTransitionProbability}(T, i, c, m - 1, b)$
    $p_2^c \leftarrow \mathsf{ApproximateTransitionProbability}(T, c, j, m - 1, b)$
    $p \leftarrow p + (p_1^c \times p_2^c)$
  end for
  $\hat{p} \leftarrow p$ rounded to b bits
end if
return $\hat{p}$

I will now prove that properties (1) and (2) hold.

Proof of property 1 (error analysis). Observe that the final rounding step, which converts p to $\hat{p}$ by rounding to b bits, introduces no more than $\frac{1}{2^b}$ error, ie.

$$|p - \hat{p}| \leq \frac{1}{2^b}$$

Say that for all m, i, and j,

$$|M_{i,j}^{2^m} - \hat{M}_{i,j}^{2^m}| < \delta_m$$

Observe that we can take $\delta_0 = 0$ so long as $b \geq 1$. I will now upper bound δ_m in terms of δ_{m-1}.

Consider the value of p computed for a particlar m, i, and j. We have

$$
\begin{aligned}
|p - M_{i,j}^{2^m}| &= |\sum_{c\in C} (\hat{M}_{i,c}^{2^{m-1}} \times \hat{M}_{c,j}^{2^{m-1}}) - \sum_{c\in C} (M_{i,c}^{2^{m-1}} \times M_{c,j}^{2^{m-1}})| \\
&= |\sum_{c\in C} (\hat{M}_{i,c}^{2^{m-1}} \times \hat{M}_{c,j}^{2^{m-1}} - M_{i,c}^{2^{m-1}} \times M_{c,j}^{2^{m-1}})| \\
&= |\sum_{c} M_{i,c}^{2^{m-1}} (\hat{M}_{c,j}^{2^{m-1}} - M_{c,j}^{2^{m-1}}) - \sum_{c} \hat{M}_{c,j}^{2^{m-1}} (M_{i,c} - \hat{M}_{i,c}^{2^{m-1}})| \\
&\leq |\sum_{c} M_{i,c}^{2^{m-1}} (\hat{M}_{c,j}^{2^{m-1}} - M_{c,j}^{2^{m-1}})| + |\sum_{c} \hat{M}_{c,j}^{2^{m-1}} (M_{i,c}^{2^{m-1}} - \hat{M}_{i,c}^{2^{m-1}})| \\
&\leq \sum_{c} M_{i,c}^{2^{m-1}} \delta_{m-1} + \sum_{c} \hat{M}_{c,j}^{2^{m-1}} \delta_{m-1} \\
&\leq \delta_{m-1} + \sum_{c} \hat{M}_{c,j}^{2^{m-1}} \delta_{m-1} \\
&\leq \delta_{m-1} + \delta_{m-1}(1 + |C|\delta_{m-1}) \\
&= 2\delta_{m-1} + |C|\delta_{m-1}^2
\end{aligned}
$$

The second to last inequality uses $\sum_c M_{i,c}^{2^{m-1}} = 1$ (which is true since $M_{i,c}^{2^{m-1}}$ is a probability distribution on c). The last inequality uses the fact that $\sum_c \hat{M}^{2^{m-1}} \leq 1 + |C|\delta_{m-1}$, which uses the fact that each $\hat{M}^{2^{m-1}}$ is within δ_{m-1} of $M^{2^{m-1}}$, and these values form a probability distribution.

$$
\begin{aligned}
|p - M_{i,j}^{2^m}| = |\sum_{c\in C}(\hat{M}_{i,c}^{2^{m-1}} \times \hat{M}_{c,j}^{2^{m-1}}) - \sum_{c\in C}(M_{i,c}^{2^{m-1}} \times M_{c,j}^{2^{m-1}})| \\
\leq \sum_{c\in C}\hat{M}_{i,c}^{2^{m-1}} \times \hat{M}_{c,j}^{2^{m-1}} - \sum_{c\in C}(M_{i,c}^{2^{m-1}} \times M_{c,j}^{2^{m-1}}) \\
\leq \sum_{c\in C}(|\hat{M}_{i,c}^{2^{m-1}} - M_{i,c}^{2^{m-1}}| + |\hat{M}_{c,j}^{2^{m-1}} - M_{c,j}^{2^{m-1}}|) \\
\leq \sum_{c\in C} 2\delta_{m-1} = 2|C|\delta_{m-1}
\end{aligned}
$$

The second inequality follows because the values of M are no greater than 1.

Combining this bound with our bound on $|p - \hat{p}|$ above, and noting that $\hat{M}_{i,j}^{2^m} = \hat{p}$ by definition, we derive that we can have δ_m satisfy

$$\delta_m \leq 2|C|\delta_{m-1} + \frac{1}{2^b}$$

A simple induction proof verifies that this means we can have

$$\delta_m \leq \frac{1}{2^b}\sum_{j=0}^{m-1}(2|C|)^j$$

Because $2|C| \geq 2$, this bound implies

$$\delta_m \leq \frac{(2|C|)^m}{2^b} = 2^{\log(2|C|)m-b}$$

Observe that if

$$b > \log(2|C|)m - \log(\epsilon)$$

then $\delta_m \leq \epsilon$.

At the top level of the recursion, $m = \log(kn^k) = \theta(\log(n))$. Also, $|C| \leq kn^k$. Thus, to achieve an ϵ error bound for arbitrary ϵ it suffices to have

$$b > \theta(\log(2kn^k)\log(n) - \log(\epsilon)) = \theta(k\log(n)^2 - \log(\epsilon)) = \theta(\log(n)^2)$$

I have not figured out how to reduce this to $\theta(\log(n))$ rather than $\theta(\log(n)^2)$, as required to achieve the space bound this problem asks for.

Proof of property 2 (space complexity analysis). We now need to analyze the space utilization of

$$\mathsf{ApproximateTransitionProbability}(T, i, j, kn^k, a\log(n))$$

This algorithm recurses $m = \theta(\log(n))$ times and at each level of the recursion it needs to store $\theta(\log(n))$ bits. Therefore in total it uses $\theta(\log(n)^2)$ bits, as required.

Acknowledgements. I read parts of Noam Nisan 1992, "RL $\subseteq$ SC" while working on this problem. I also want to thank Zixuan for helpful answers to a couple of questions on Piazza.

Answer to (b)

The above procedure can be used to decide L in polynomial time. Thus $\mathbf{BPL} \subseteq \mathbf{P}$.

Runtime analysis. Consider the runtime of this procedure. Let $m^* = \log(kn^k)$. The procedure produces a tree of calls with 1 call with value $m = m^*$, 2 calls with value $m = m^* - 1$, 4 with value $m = m^* - 2$, and so on up to 2^{m^*} calls with value 0. This in total is $\leq 2^{m^*+1}$ calls. Each recursive call requires multiplying two b-bit values and summing $|C|$ such products, and so takes $O(|C|b)$ time. So the full runtime is bounded above by $|C|b2^{m^*+1}$. Since $m^* = \theta(\log(kn^k))$ and $b = \theta(\log(n))$, the runtime is bounded by $2|C|bkn^k = O(\log(n)n^k) = O(n^{k+1})$.

Problem 5: Advice Removal (2 points)

Question

Assume $s : \mathbb{N} \to \mathbb{N}$ is a time constructible function. Recall $\mathsf{P}/s(n)$ is the class of languages decidable by a polynomial time algorithm with $s(n)$ advice.

Prove that if $\mathsf{SAT} \in \mathsf{P}/s(n)$ then SAT can be solved in $2^{O(s(n))} \cdot \text{poly}(n)$ time. That is, there is an algorithm running in $2^{O(s(n))} \cdot \text{poly}(n)$ time which, given any formula ϕ of size n, outputs a satisfying assignment to ϕ when one exists.

(This result is interesting, in part because it is a major open problem whether $\mathsf{SAT} \in \mathsf{P}/\mathsf{poly}$ implies $\mathbf{P} = \mathbf{NP}$ or not!)

Hint: Try using the self-reducibility of SAT.

Answer

Say $\mathsf{SAT} \in \mathbf{P}/s(n)$, witnessed by an algorithm $A(\phi, y)$ which decides whether ϕ is satisfiable using advice y, and which runs in poly(n) time.

Here is a $2^{O(s(n))}\text{poly}(n)$ time algorithm for deciding if a given formula ϕ of description length n is satisfiable. The idea is to use a loop that runs up to $2^{s(n)}$ iterations to try solving SAT with each of the $2^{s(n)}$ pieces of advice $y \in \{0,1\}^n$. Within each iteration of the loop, an assignment $\vec{x}$ is constructed using the regular polynomial-time SEARCH-SAT procedure, assuming $A(\phi, y)$ decides SAT. Since some y won't make A correctly decide SAT, the algorithm then checks that $\vec{x}$ is actually a valid assignment before returning. Since there is at least one piece of advice y that makes A decide SAT, it is guaranteed to find a satisfying assignment on one iteration if one exists.

$\mathsf{SolveSAT}(\phi)$

for $y \in \{0,1\}^{s(n)}$ **do**
 $\vec{x} \leftarrow []$
 for $i = 1, \ldots, n$ **do**
  $\phi_i \leftarrow \phi$ substituted with $X_j = \vec{x}[j]$ for each $j = 1, \ldots, i-1$.
  $\phi_i \leftarrow \phi_i$ substituted with $X_i = 0$
  $b \leftarrow A(\phi_i, y)$
  If b then $x_i \leftarrow 0$ else $x_i \leftarrow 1$.
  $\vec{x} \leftarrow \text{append}(\vec{x}, x_i)$
 end for
 if $\phi(\vec{x})$ **then**
  return 1 {If $\vec{x}$ is a satisfying assignment, return 1.}
 end if
end for
return 0

Problem 6: Pairwise Independence (6 Points, 2 for each sub-problem)

Question

In this problem, we will show how to generate "pseudo-random bits" with a nice property. These are sometimes useful for derandomizing algorithms. We will use them to prove other theorems in the class.

A set of discrete random variables $X_1, X_2, X_3, \ldots, X_n$ is *pairwise independent* if for every $i \neq j$,

$$\Pr[X_i = a \mid X_j = b] = \Pr[X_i = a]$$

for every a, b in the range of the variables. X_i is *unbiased* if each element of the range of X_i is equally likely.

(a) **Generating Pairwise Independent Bits.** Let V_n be the set of all n-bit vectors excluding the all-0 vector. For every $v \in V_n$, define $X_v(r) = \langle v, r \rangle$ where r is any n-bit vector and $\langle , \rangle$ is inner product modulo 2. Picking r uniformly at random, the X_v's are well-defined random variables.
Show that the X_v's are unbiased and pairwise independent.

Note we use n random bits to select r, and obtain $2^n - 1$ pairwise independent bits X_v! (One can think of the above process as a "pairwise independent" pseudorandom generator that takes an n-bit seed and produces $2^n - 1$ bits.) Also note if we include the all-0 vector in V_n, we would lose the unbiased property, but the X_i's would still be pairwise independent.

(b) **Pairwise Independent Hashing.** Let p be prime and $\mathbb{F}_p$ be the field of integers modulo p. For a, b independently and uniformly chosen at random from $\mathbb{F}_p$, let $Y_i = a \cdot i + b$.
Show that $\{Y_i \mid i \in \mathbb{F}_p\}$ are unbiased and pairwise independent.

Hint: For any fixed values Y_i and Y_j, the equations $Y_i = ai + b$ and $Y_j = aj + b$ can be uniquely solved for a and b.

(c) We will work over $\mathbb{F}_2$. Let V_n be the set of all n-bit vectors excluding the all-0 vector. Letting M be a uniformly randomly chosen Boolean $m \times n$ matrix, define the random variable $X_v = Mv$.
Show that the X_v's are unbiased and pairwise independent.

Define the function family $f_M(v) = Mv$, as we vary over all matrices M. **Show for all $v \neq v'$, when we pick a function f uniformly at random from this family, $\Pr_f[f(v) = a \wedge f(v') = b]$ is the same for all a and b.**

Such a family of functions is called a *pairwise independent family of hash functions*. They are very useful!

Answer to (a)

Verifying unbiasedness. Fix $v \in V_n$. Let $I := \{i \in \mathbb{Z}_n : v_i = 1\}$. Observe

$$X_v(r) = \sum_{i \in I} r_i \mod 2$$

I proceed by induction on $|I|$.

If $|I| = 1$, let i be the unique member of I. Then $X_v(r) = r_i$, which is an unbiased random bit.

Now say we know that for any set J so that $|J| = m$, $\sum_{j \in J} r_j \mod 2$ is unbiased. Consider a set I with $|I| = m+1$. Set $i^* := \sup I$ and set $J = I \setminus \{i^*\}$. Then $Y(r) := \sum_{j \in J} r_j \mod 2$ is an unbiased bit. Observe

$$X_v(r) = \sum_{i \in I} r_i \mod 2 = r_{i^*} + Y(r) \mod 2$$

Here, r_{i^*} and $Y(r)$ are independent unbiased bits. It is straightforward to verify from this that $X_v(r)$ is itself an unbiased random bit, by enumerating the probability table of the 4 joint assignments to $X_v(r)$ and $Y(r)$.

Verifying pairwise independence. Let $v^1, v^2 \in V_n$, with $v^1 \neq v^2$. Let $I^1 := \{i \in \mathbb{Z}_n : v^1_i = 1\}$ and let $I^2 := \{i \in \mathbb{Z}_n : v^2_i = 1\}$.

Let $I^* := I^1 \Delta I^2$ $(= I^1 \cup I^2 \setminus (I^1 \cap I^2))$. I proceed by induction on $|I^*|$.

If $|I^*| = 1$, WLOG say $v^1_i = 1$ but $v^2_i = 0$ (otherwise swap v^1 and v^2). Then $X_{v^1}(r) = X_{v^2}(r) + r_i \mod 2$, with r_i independent from $X_{v^2}(r)$ (which is a sum of r_j for $j \neq i$). Then $X_{v^1}(r)$ is uniform on $\{0, 1\}$, conditional either on $X_{v^2}(r) = 1$ or $X_{v^2}(r) = 0$, so $X_{v^1}(r)$ and $X_{v^2}(r)$ are independent.

Now say we know that if $|I^*| = m$, $X_{v^1}(r)$ and $X_{v^2}(r)$ are independent. Say we have $|I^*| = m + 1$. WLOG say i is some index so $v^1_i = 1$ but $v^2_i = 0$ (if this doesn't exist, swap v^1 and v^2). Let $\bar{v}$ be v^1 but with index i set to 0. Then

$$\bar{x}(r) := \langle r, \bar{v} \rangle \mod 2 = X_{v^1}(r) - r_i \mod 2$$

By the inductive hypothesis, $\bar{x}(r)$ is independent from $X_{v^2}(r)$. And it is also certainly independent from r_i since it is a sum of r_j all with $j \neq i$. Thus $X_{v^1}(r) = \bar{x}(r) + r_i \mod 2$ is a function of two values independent of $X_{v^2}(r)$, and hence is independent of $X_{v^2}(r)$.

Answer to (b)

Unbiasedness. Fix i and a. Observe that since b is uniform over $\mathbb{Z}_p$, $ai + b$ is also uniform over $\mathbb{Z}_p$. Thus $\mathbb{P}[Y_i = y|a] = 1/p$ for all y. Thus $\mathbb{P}[Y_i = 1] = \sum_{a=0}^{p-1} \frac{1}{p} \mathbb{P}[Y_i = y|a] = \frac{1}{p}$.

Pairwise independence. Fix i and j. We want to show that $\mathbb{P}[Y_i = y_i | Y_j = y_j] = \frac{1}{p}$. Because $\mathbb{Z}_p$ is a field, for any value a and any value y_j, there is exactly one value b such that $y_j = a \cdot j + b$. Thus, the event $E_j := \{Y_j = y_j\}$

equals the event $\{(a, b) : a \in \mathbb{Z}_p \wedge b = y_j - a \cdot j\}$, an event with probability $p/p^2 = 1/p$. Since $\mathbb{Z}_p$ is a field, the system of linearly independent equations

$$y_i = a \cdot i + b; \qquad y_j = a \cdot j + b$$

has a unique solution (a^*, b^*). Thus $\{Y_j = y_j \wedge Y_i = y_i\} = \{a^*, b^*\}$ which has probability $1/p^2$. Thus $\mathbb{P}[Y_i = y_i | Y_j = y_j] = \frac{\mathbb{P}[Y_i = y_i, Y_j = y_j]}{\mathbb{P}[Y_j = y_j]} = \frac{1/p^2}{1/p} = 1/p$.

Answer to (c)

Unbiasedness. From part (a) we know that $(Mv)_i$ is a uniform bit for each i, where $(Mv)_i$ is the ith element of vector Mv. Furthermore, since $(Mv)_i$ depends on a different row of M for each i, the collection of values $\{(Mv)_i\}_i$ is an independent collection of random variables. Thus Mv is uniformly distributed on the space of binary n-dimensional vectors.

Pairwise independence. Fix v^1, v^2. Observe that

$$\begin{aligned}
&\mathbb{P}[Mv^1 = v | Mv^2] = \\
&\qquad \mathbb{P}[(Mv^1)_0 = v_0 | Mv^2] \times \mathbb{P}[(Mv^1)_1 = v_1 | Mv^2, (Mv^1)_0] \times \cdots \times \\
&\qquad\qquad \mathbb{P}[(Mv^1)_{n-1} = v_{n-1} | Mv^2, (Mv^1)_{0:n-2}]
\end{aligned}$$

For any i,

$$\mathbb{P}[(Mv^1)_i = v_i | Mv^2, (Mv^1)_{0:i-1}] = \mathbb{P}[(Mv^1)_i = v_i | Mv^2_i]$$

because all the values being conditioned on other than Mv^2_i depend on entirely independent rows of M from those that affect the value $(Mv^1)_i$. And from part (a), we know $(Mv^1)_i$ and $(Mv^2)_i$ are independent for all i, so $\mathbb{P}[(Mv^1)_i = v_i | Mv^2_i] = 1/2$. Thus $\mathbb{P}[Mv^1 = v | Mv^2] = 1/2^n = \mathbb{P}[Mv^1 = v]$.

Uniformity on joint assignments.

$$\mathbb{P}[f(v) = a \wedge f(v') = b] = \mathbb{P}[f(v) = a] \times \mathbb{P}[f(v') = b | f(v) = a] = \frac{1}{2^n} \times \frac{1}{2^n}$$

where the last equality follows due to the pairwise independence and uniformity proven above.