Project 3: Mutation

Juan Correa, John Daciuk and Patrick Kwok

August 4, 2020

Contents

1 Introduction

1.1 Problem Statement

Our task is to develop computational methods to experimentally discover the nature of a mystery mutagen. Mutations are composed of 1 or more rules, each defined by a pattern and action pair. The pattern indicates where in the genome the rule can take effect and the action defines the change to the original genome. To understand the syntax of rules, consider a concrete example: ac;t $\rightarrow$ tg. This particular rule could act anywhere in the genome where there is an a or c followed by t; the action is to replace that strand of two bases with t followed by g. Adding expression to the language, this rule could also be written as ac;t $\rightarrow$ 1g, indicating that the first component of the action is index 1 of the pattern. [1] We uncover hidden mutagens by interacting with a simulator: given an experimental genome of length 1,000 that we design, the simulator will attempt to randomly apply m mutations and return the mutated genome. We then may guess a set of rules, which will be rejected if not perfectly correct[2]. The process repeats for some predetermined number of experiments or time. The simulator *does not* encourage us if our guesses are close, and this lack of guidance is only the beginning of the challenges we face.

1.2 Challenges

The core challenge is the extent of ambiguity embedded into the problem. For instance, consider even a single strand of genome and the result after a single mutation rule applied:

```
     aactg
ctggtcgtatgttgg
```

In the absence of more evidence, it is impossible to definitively recover the rule that applied above for 3 reasons, which we now take some time to explore:

1. **Ambiguity of context location**: constrained to rules with pattern and action windows of length 5, we would know precisely where this rule applied; rules cannot apply to windows smaller than those of the action by definition[3]. However, rules *can* apply to windows bigger than those of the action. Above a rule of length 6 could have applied, starting one to the left of the action effect. Then the true pattern trigger would not have just been cgtat but *t*cgtat. Alternatively the true pattern could have been cgtat*g*. In general, a rule of size w could have applied at $w - 5 + 1$ different locations. Simply put, seeing the change is not enough to know the full pattern that was the trigger because rules don't necessarly change each pattern base.

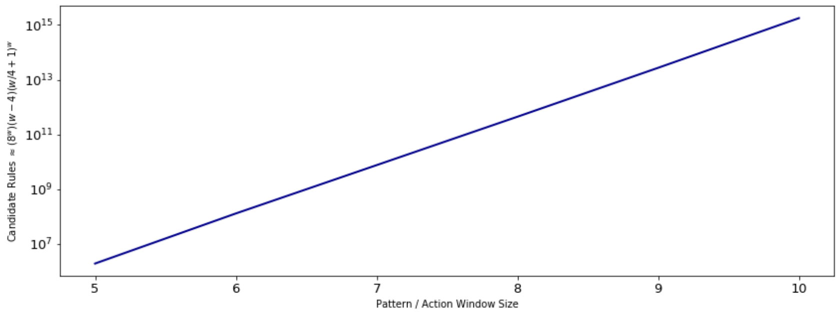
2. **Ambiguity of pattern disjunction**: Knowing where the rule was applied tells us *a* pattern that matched the rule's left-hand side; however, rules with disjunctions will match more than 1 pattern instance. For example, ignore ambiguity of location by optimistically assuming that the rule above was only of length 5: there are still 8 possible sets that could match each index of the rule's pattern. To enumerate, {c}, {ac}, {cg}, {ct}, {acg}, {act}, {cgt} and {acgt} are all valid sets for pattern index 0 because they all match "c". Thus, we are still left with $8^5 = 32,768$ possible patterns! For rules of length 10, we have $8^{10} = 1,073,741,824$ possible patterns, even if we *could* observe with certainty one

[1]For more about rule technicalities, see: http://www.cs.columbia.edu/∼kar/4444f19/node19.html
[2]This changed midway project to allow an equivalent set of rules, even if they yield a different distribution of changes seen.
[3]The window of an action is the larger of either the number of indices or one more than the highest number used.

complete pattern to genome match. Since our uncertainty is proportional to both ambiguity of location and ambiguity of disjunction, we are saddled with on the order of 10 billion candidate patterns that can explain the evidence above.

3. **Ambiguity of action**: suppose we have an oracle that reveals the location and pattern of the single rule above. Because the action can consist of letters or numbers, we are still unable to confidently guess the rule. For example, we see above that "`a`" changed to "`t`", so we may be tempted to claim that the action index that dictated that change was simply "`t`". Unfortunately, we must also consider that there would be at least two possible numbers in that action index that would have had the same effect, since we observe at least two "`t`'s" in the pattern. For a randomly generated experiment, an action window size of w would have about $\frac{w}{4}$ possible symbols for each index. Thus, for a single rule with a window size of 10 applied to the above, we must take the 10 billion possible patterns and multiply by the roughly $3^{10} = 59,049$ possible actions, resulting in about $(10^{10})(10^{4}) = 100,000,000,000,000$ possible single rule mutations.

Figure 1: Explosion of candidate rules as a function of window size, assuming we have the single piece of evidence as above and are searching to explain it with a single rule. Number of possible rules blows up exponentially as a function of window size.



This daunting level of ambiguity is only compounded by the fact that the number of rules is also unknown; furthermore, mutations can overlap to muddy the evidence. Perhaps this mutation could have resulted from in fact two separate rules, applied adjacently. Ultimately, the challenge of this project is to develop strategies for collecting and integrating many such pieces of evidence to successively update and refine our beliefs, effectively winnowing down the set of mystery rules that could be at play.

1.3 Solution Summary

Our strategy is to break the problem down into sub-problems, and integrate the sub-problems with sound mathematical reasoning and intuition. To start at the highest level, it is natural to first develop a strong single rule guesser and later generalize to multiple rule mutations.

1.3.1 Single Rule Guesser

For single rule guessing, there are a number of useful mathematical tools that we use to quickly narrow down the rules that best explain the evidence collected:

- We can make a number of independence assumptions about the elements of the pattern and action that form a rule. For example, just knowing that some index in the pattern is "c" tells us nothing about the rest of the rule. This allows us to break down the probability of a candidate rule as the product of probabilities of its components. More important, we can build rules to guess from the most likely pattern and action components.
- As illustrated above, the reality of disjunctions in rule patterns greatly confounds the search for the true pattern. However, we will later demonstrate that simpler rules (with less disjunction) that can explain the evidence are far more likely. Furthermore, given *multiple* mutation examples from a single rule *with* disjunctions, incorrect patterns lacking the disjunctions can be definitively ruled out very quickly.
- We have many experiments to observe, each of which may have multiple mutations that provide valuable evidence. Because in the single rule case the true mutagen must fully explain each piece of evidence, the probability of incorrect mutagens can be made to exponentially fall off or else drop to zero. This is because the probability of a rule effectively becomes the product of probabilities derived from each piece of evidence.

In addition to narrowing our initial search to a single rule, we can further break down the problem by assuming we know the window size of the rule and incrementing that assumption when we have good reason to do so. Following the logic of our program for the smallest window size will be instructive to see the last principle we use to break down the problem. For a window size of 1, we do the following:

1. `Initialize our distribution of belief about` P_0 `and` A_0 `with reasonable priors.`
2. `Process an incoming experiment by separating it into before and after pairs for single letter changes, referred to here as evidence.`
3. `Update our belief about the distributions for` P_0 `and` A_0 `by multiplying current candidate` p_0 `and` a_0 `probabilities by the probability that they explain each new piece of evidence.`
4. `Guess a rule composed of the highest product probability` P_0 `and` A_0 `that we have not already guessed.`
5. `Iterate with more experiments or guesses.`

When the window size expands beyond 1, to take advantage of our independence assumptions about each P_i and A_i, we maintain a distribution that stores our current belief state about each such rule element. It's this way of breaking down the problem that is really the core to our whole strategy. We avoid enumerating all possible rules that can explain the evidence because there are far too many rules to consider for large window sizes; instead, we track probabilities for candidate pattern and action **elements** *for each index*.

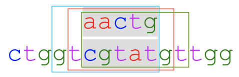


Figure 2: Each window, outlined in blue, red and green, is a possible context for a length 7 rule given this piece of mutated DNA. Although only 1 of these windows reflects reality, we don't know which, so we average over them.

For concreteness, let's assume we are currently searching for a single rule of length 7 and we see the change figured above. We process this change by retrieving 3 pairs of original and mutated strings corresponding to the rectangular length 7 windows above. Since we don't know which of these windows is the right context, we update our state of belief regarding each P_i and A_i element by the average of how well candidates explain these windows. Again, we continue by iterating through many such observations of changes, effectively increasing the probability of the elements that make up the true rule. We continue guessing by building current most probable rules from current most probable pattern and action elements.

1.3.2 Multiple Rule Guesser

Finally, to leverage our single rule guesser to build a player capable of guessing multiple rules, we seek to split the observed mutations into as many sets as we assume there are rules. We then assume that a single rule can explain all the mutations in set 1, another rule can explain set 2 and so on. Our problem now reduces to finding a reasonable way to choose the splits, which is not trivial since with m mutations and r rules we have m^r possible splits. Although we will later highlight more sophisticated ideas, we settled on a simple approach. We first take all evidence to be in set 1 until we build high confidence in rule 1. If rejected by the simulator, we engage the multi-rule hypothesis and keep rule 1 as a permanent member of our guessing set while moving onto building confidence in rule 2. Critically, while narrowing the possibilities for rule 2, we reject any evidence that is explained particularly well by rule 1. Iterating the process, we can in principle attempt to guess any number of rules.

2 Problem Formalization

Let r be one particular rule of a mutagen, then r can be represented as

$$r = p_0; p_1; p_2; p_3; p_4; p_5; p_6; p_7; p_8; p_9 @ a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9, \tag{1}$$

where each $p_i \in \{\texttt{a}, \texttt{c}, \texttt{g}, \texttt{t}\}^*$ and $a_i \in \{\texttt{a}, \texttt{c}, \texttt{g}, \texttt{t}, \texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}\}$.

Let R represent a random variable that takes rules as values. Similarly, let P_i and A_i be variables representing, respectively, individual elements of the pattern and action part of the rule. For simplicity let $P = \{P_i\}$, $A = \{A_i\}$ for $i = 0, \ldots, 9$. Lowercase letters p_i, a_i, p, a represent a particular pattern element, action element, a whole pattern and a whole action, respectively.

Consider a string $S = \{s_i\}_{i=0,\ldots,9}$, $s_i \in \{\texttt{a}, \texttt{c}, \texttt{g}, \texttt{t}\}$. If every s_i matches the corresponding pattern position p_i, the rule's action can be applied. Let M_i, $i = 0, \ldots, 9$ be indicator variables of these individual matches. Moreover, let M be an indicator for the event $\forall_i M_i = 1$, so that $M = 1$ if the string matches the pattern P and 0 otherwise.

It follow that, if $M = 1$, the original string S can be mutated into a string $T = \{t_i\}_{i=0,\ldots,9}$, $t_i \in \{\texttt{a}, \texttt{c}, \texttt{g}, \texttt{t}\}$ based on the rule's action A. If $M = 0$, then $T = S$.

Finally, let N be an indicator variable such that $N = 1$ if R is minimal and $N = 0$ otherwise.

Based on the process described above, the causal process by which S is transformed into T, can be represented with the graph depicted in Figure 3.

This is not only a curious picture, but it encodes important constraints of the process that are greatly helpful in solving the problem at hand, as we will see in the next section. Formally speaking, this is causal diagram can be though of as a Bayesian Network, and conditional independencies among variables can be read using the *d-separation* criterion [Koller and Friedman, 2009, Pearl, 2000].

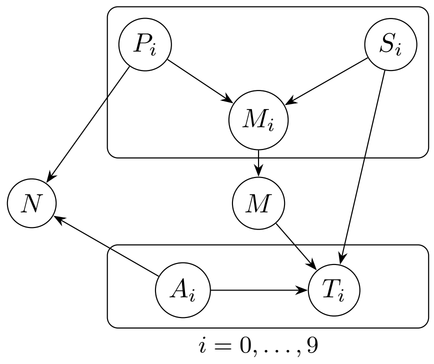


Figure 3: Graphical model representing the causal process by which a mutagen rule transforms a string S into a string T. Each box in the graph is repeated for $i = 0, \dots, 9$, and arrows crossing boxes' boundaries should be considered as being from/to each variable with subscript i.

Our goal is to guess what the unknown rule R is based on what is known, that is, a string S that has been mutated by the rule into a string T, provided that S matched ($M_{0,\dots,9}$, $M = 1$) the pattern of the same rule for the mutation to take place.

As framed in the previous section a rule R is composed of a pattern P and an action description A. Next, we assume that the best guess for a rule is one with the maximum likelihood based on the evidence, that is, the rule $R = \{P, A\}$ that maximizes $P(P, A \mid S, T, M_{0,\dots,9} = 1, M = 1)$, which the probability of R being the rule in play given that we observe the strings S, T knowing that S was matched to P.

Formally, our guess $\hat{r}$ is defined as

$$\hat{r} = \underset{r=\{p,a\}}{\arg\max}\, P(P = p, A = a \mid S = s, T = t, M_{0,\dots,9}, M = 1). \tag{2}$$

In order to find $\hat{r}$, we will use some of the independencies among variables entailed by the mutation process:

(I_1) $(P_i \perp\!\!\!\perp A_j \mid S, T, M_{0,\dots,9}, M)$, $i, j \in \{0, \dots, 9\}$: The pattern and action elements are independent when the original string S and the mutated string T are known together with the fact that there was a match $(M_{0,\dots,9} = 1, M = 1)$.

(I_2) $(P_i \perp\!\!\!\perp P_j \mid S, T, M_{0,\dots,9}, M)$, $i, j \in \{0, \dots, 9\}, i \neq j$: each element of the pattern is independent of one another when we know they have been matched (individually) and globally, and S, T are fixed.

(I_3) $(A_i \perp\!\!\!\perp A_j \mid S, T, M_{0,\dots,9}, M)$, $i, j \in \{0, \dots, 9\}, i \neq j$: action elements are independent of one another when S and T are fixed and we know the rule was matched to S.

(I_4) $(P_i \perp\!\!\!\perp T, S_j, M, M_j \mid S_i, M_i)$, $i, j \in \{0, \dots, 9\}, i \neq j$: each P_i is independent of the mutated string T, any element of the original string S that is not S_i and the fact that the pattern was globally matched $(M_j = 1, M = 1)$ if it is known what S_i is and that this particular P_i was matched with S_i ($M_i = 1$).

(I_5) $(A_i \perp\!\!\!\perp T_j, M_{0,\dots,9} \mid S, T_i, M)$, $i, j \in \{0, \dots, 9\}, i \neq j$: action element A_i is independent of any part of the mutated string T that is not T_i and the fact that individual parts of S matched P, once it is known that the whole string was matched ($M = 1$), and the original string and T_i are known.

Those constraints allows to rewrite the probability distribution on the right-hand-side of (2)

$$
\begin{aligned}
&P(P,A|S,T,M_{0,\ldots,9}{=}1,M{=}1) && (3)\\
&= P(P|S,T,M_{0,\ldots,9}{=}1,M{=}1)P(A|S,T,M_{0,\ldots,9}{=}1,M{=}1) && (4)\\
&= \left(\prod_i P(P_i|S,T,M_{0,\ldots,9}{=}1,M{=}1)\right)\left(\prod_i P(A_i|S,T,M_{0,\ldots,9}{=}1,M{=}1)\right) && (5)\\
&= \left(\prod_i P(P_i|S_i,M_i{=}1)\right)\left(\prod_i P(A_i|S,T_i,M{=}1)\right). && (6)
\end{aligned}
$$

Equation (6) implies that $\hat{r}$ as defined in (2) is equal to

$$
\begin{aligned}
\hat{r} &= \{\hat{p},\hat{a}\}, && (7)\\
\hat{p} &= \{\hat{p}_i\}_{i=0,\ldots,9}, && (8)\\
\hat{a} &= \{\hat{a}_i\}_{i=0,\ldots,9}, && (9)\\
\hat{p}_i &= \arg\max_{p_i} P(P_i|S_i,M_i{=}1), \text{ and} && (10)\\
\hat{a}_i &= \arg\max_{a_i} P(A_i|S,T_i,M{=}1); && (11)
\end{aligned}
$$

which follows from the fact that a product can be optimized by optimizing each one of its independent factors.

Having the ability to find $\hat{r}$ by means of finding individual optimal $\hat{P}_i$ and $\hat{A}_i$ overcomes a great deal of combinatorial considerations.

2.1 Taking a Bayesian Approach

Consider the definition of $\hat{p}_i$, we have that

$$
\begin{aligned}
P(P_i|S_i,M_i{=}1) &= \frac{P(P_i,S_i,M_i{=}1)}{P(S_i,M_i{=}1)} && (12)\\
&= \frac{P(M_i{=}1 \mid P_i,S_i)P(S_i \mid P_i)P(P_i)}{\sum_{p_i} P(M_i{=}1 \mid P_i,S_i)P(S_i \mid P_i)P(P_i)} && (13)\\
&= \frac{P(M_i{=}1 \mid P_i,S_i)P(S_i)P(P_i)}{\sum_{p_i} P(M_i{=}1 \mid P_i,S_i)P(S_i)P(P_i)} && (14)\\
&= \frac{P(M_i{=}1|S_i,P_i)P(P_i)}{\sum_{p_i} P(M_i{=}\,1|S_i,P_i)P(P_i)}. && (15)
\end{aligned}
$$

Now, for $\hat{a}_i$, it follows

$$
\begin{aligned}
P(A_i|S,T_i,M{=}1) &= \frac{P(A_i,S,T_i,M{=}1)}{P(S,T_i,M{=}1)} && (16)\\
&= \frac{P(T_i \mid S,M{=}1,A_i)P(S,M{=}1 \mid A_i)P(A_i)}{\sum_{a_i} P(T_i \mid S,M{=}1,A_i)P(S,M{=}1 \mid A_i)P(A_i)} && (17)\\
&= \frac{P(T_i \mid S,M{=}1,A_i)P(S,M{=}1)P(A_i)}{\sum_{a_i} P(T_i \mid S,M{=}1,A_i)P(S,M{=}1)P(A_i)} && (18)\\
&= \frac{P(T_i \mid S,M{=}1,A_i)P(A_i)}{\sum_{a_i} P(T_i \mid S,M{=}1,A_i)P(A_i)}. && (19)
\end{aligned}
$$

2.2 Application of Theory to Inferring the Components of a Single Rule

Let's now turn to further simplifying our equations for $P(P_i|S_i, M_i{=}1)$ and $P(A_i|S, T_i, M{=}1)$.

2.2.1 $P(P_i|S_i, M_i{=}1)$

Note that the term $P(M_i{=}1|S_i, P_i)$ in equation (15) is deterministic according to the problem statement: the probability that P_i will match S_i is 1 so long as $S_i \in P_i$ and 0 otherwise. Now let's also temporarily assume that any particular pattern element p_i that matches S_i is equally likely and there are n_{pi} possible matching elements. In the case that P_i matches S_i, equation (15) becomes:

$$P(P_i|S_i, M_i{=}1) = \frac{P(M_i{=}1|S_i, P_i)P(P_i)}{\sum_{p_i} P(M_i{=}\ 1|S_i, P_i)P(P_i)} = \frac{1}{n_{pi}} = \frac{1}{8}$$

And if P_i does not match S_i the above probability is 0. For example, say $S_i = \texttt{a}$ and $P_i = \{\texttt{c}, \texttt{g}, \texttt{t}\}$; we do not have a match and $P(\{\texttt{c}, \texttt{g}, \texttt{t}\} \mid S_i{=}\texttt{a}, M_i{=}1) = 0$. On the other hand, if $S_i = \texttt{a}$ and $P_i = \{\texttt{a}, \texttt{c}, \texttt{g}, \texttt{t}\}$ we do have a match and $P(\{\texttt{a}, \texttt{c}, \texttt{g}, \texttt{t}\} \mid S_i{=}\texttt{a}, M_i{=}1) = \frac{1}{8}$ since 8 sets in the power set $\{\texttt{a}, \texttt{c}, \texttt{g}, \texttt{t}\}$ match any particular S_i.

Now, while it may seem reasonable to make the probability of P_i uniform over all the sets that match S_i, it leads to practical problems and counter intuitions. The practical concern is that our algorithm would have strong motivation to guess acgt for each P_i: every piece of evidence is consistent with the full disjunction set. To acknowledge the counter intuition, consider the simple case of a single rule a@t. Clearly after observing this change many times, we should be able to safely rule out acgt@t. To complete the intuition with math, assume that we know there is a single rule of window size 1 and we make a single observation $a \to t$ from a random genome. We will compare $p(P_0 = a|observe\ a)$ with $p(P_0 = acgt|observe\ a)$:

$$p(P_0 = a|observe\ a) \propto P(observe\ a|P_0 = a)P(P_0 = a) = (1)\left(\frac{1}{4}\right) = \frac{1}{4}$$

$$p(P_0 = acgt|observe\ a) \propto P(observe\ a|P_0 = acgt)P(P_0 = a) = \left(\frac{1}{4}\right)\left(\frac{1}{4}\right) = \frac{1}{16}$$

Calculating the same for 2 and 3 letter disjunctions that match the observation a and normalizing we develop the following prior distribution for the length of the set P_i after a single observation: $P(len = 1) = \frac{4}{15}$, $P(len = 2) = \frac{6}{15}$, $P(len = 3) = \frac{4}{15}$, $P(len = 1) = \frac{1}{15}$. In implementation, we tweaked $P(P_i|S_i, M_i = 1)$ to not be $\frac{1}{8}$ for all P_i that match S_i, but rather to prefer shorter P_i's and hence our player prefers to guess rules with fewer disjunctions.

2.2.2 $P(A_i|S, T_i, M{=}1)$

Note that $P(T_i \mid S, M{=}1, A_i)$ from equation (19) is deterministic; i.e. assuming the action is activated and we know the original string S there is no question about the fate of T_i. As we did for patterns, let's also assume that all actions are equally likely; moreover, let's assume that the action may be completely contained to a window of size $w \leq 10$. This will help us to apply what we derive to our player, which guesses conditioned on the current window size its considering. Equation (19) then becomes:

$$P(A_i \mid S, T_i, M{=}1, w) = \frac{P(T_i \mid S, M{=}1, A_i, w)P(A_i)}{\sum_{a_i} P(T_i \mid S, M{=}1, A_i, w)P(A_i)} = \frac{1}{1 + \sum_{j=0}^{w-1} 1(S_j = T_i)}$$

In other words, the probability of any particular action given the evidence is the reciprocal of the total number of possible actions. An action that is consistent with T_i must either be the letter T_i or use a number to sub in a matching letter from S_i within the window. For a concrete example, suppose the window size is 3, $S = \texttt{aat}$ and $T = \texttt{aaa}$. Let's calculate $P(A_2 = \texttt{a})$ given the evidence. The possible actions to mutate $T_2 \rightarrow \texttt{a}$ would be $\in \{\texttt{0}, \texttt{1}, \texttt{a}\}$, thus $P(A_2 = \texttt{a}) = \frac{1}{3}$.

To summarize, let's recall the context and importance of this discussion: now that we can compute the probabilities of candidate P_i and A_i rule components from single mutation blocks in an experiment, we can start to *build most probable rules from most probable components*. Furthermore, equation (6) tells us that the probability of a rule is just the product of probabilities of each of its components.

2.3 Bridge to Single Rule Guessing Implementation

So far in discussing how to apply theory to a robust single rule guesser, we have swept aside 2 crucial implementation details:

1. We have assumed that we have a subsection of experiment genome within some window size ≤ 10 in which we know a mutation of that same length occurred. In practice we rarely have this guarantee; instead we must collect before and after sub-strings that we believe were the result of a single mutation and deal with the ambiguity of location discussed in the intro.
2. Although this project is really about integrating many pieces of evidence to inform our player, we also have not discussed exactly how to update our P_i and A_i distributions.

In the implementation section, we start by addressing these details before moving onto the broader mechanics of our player.

3 Implementation

3.1 Extracting Evidence from Experiments

We are given how many mutations the simulator was able to successfully perform, but as discussed, there is still ambiguity of context, i.e. we do not know precisely where these mutations happened. In this section we will discuss how we dealt with ambiguity of context while gathering evidence. Again, assume we are searching for the effects of a rule with a known window size [4] (our player iterates through possible window size assumptions).

One of our concerns when collecting evidence when m, the number of mutations, is large is the possibility of composition of mutations, referred to as overlapping mutations. If a rule with window size w operates on a single 1,000 length genome twice, the probability of *non*-overlapping mutations is roughly $\frac{1000-2w}{1000}$. This

[4] Recall that we define the window size of a rule to be the minimum length genome string the rule needs to fully express itself, which will be the minimum of either the pattern length, action length or number used in the action + 1.

is a consequence of the fact that there are roughly $2w$ places in the genome where the second mutation could start and overlap the first. In general, for m mutations the probability of no overlap is on the order of $\prod_{n=1}^{n=m} \frac{1000-2w(n-1)}{1000}$.

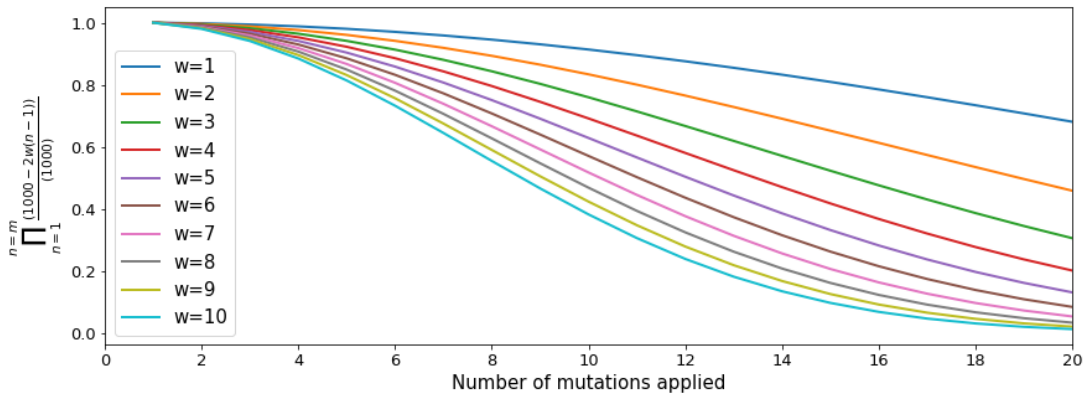


Figure 4: Overlap of mutations in a single experiment becomes likely as m approaches 20.

As we can see, unless the window size is very small, mutation overlap becomes plausible when $m \geq 5$ in a single experiment. For dozens of experiments, mutation overlap becomes almost certain. Considering that the probability we assign to any rule is proportional to the probability that it explains *each* block of evidence, deceiving evidence may suddenly misguide us to infer the true rule is impossible!

Let's now assume that we are considering rules with a window size of 7 and contrast deceiving evidence with clear and constructive evidence:

```
                gtcaa gt aaga
        ggggaagcaactgggtccggtctttgtactagtc
```

Figure 5: This is evidence we would reject because there is risk of overlapping mutations.

Because the changes in this block span 13 characters of genome, we can be certain that no single length 7 mutation was the cause. Although we can not be sure that there was in fact overlap, we have reason to fear it. The ambiguity of context for the left most mutation *includes* that of the right most. By contrast, if the span length of change had been 14 characters, the location of a supposed two mutations crystallizes with certainty. In practice we prefer to accept evidence from blocks of changes that are separated by at least 1 window length of non-mutated string.

```
     gtc a
cgtaacgagaaaa
```

Figure 6: This is evidence we would take into account because overlapping mutations are unlikely.

While even here lurks the possibility of overlapping mutations, it is far less likely. Thus we would process this block of string into before and after pairs as follows:

Since we have no way of knowing in which of these windows the proposed length 7 rule actually applied, we accept the before and after strings from each window or possible "world" as evidence. To process the P_i and A_i distributions for these worlds, we use the equations described in the theory section and average over the

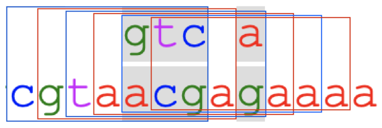


Figure 7: Processing a block of evidence into separate "worlds"

worlds. The idea is that probable P_i and A_i components will at least explain 1 of the worlds well, whereas improbable P_i and A_i components may not explain even a single world. Over many such pieces of evidence our player homes in on the truth.

3.2 Accounting for Plurality of Evidence: The Running Distribution

As discussed, the crux of this entire problem is to *effectively leverage a plurality of evidence to conquer the ambiguity posed by any single piece of evidence.* Fortunately we have a final independence assumption that clarifies how to update a running distribution of P_i and A_i candidates: *mutations are independent events.* Albeit there is still a small possibility of overlapping mutations and the history of mutations will affect our experiment design, this assumption is roughly correct and highly effective. Consequently to update our running distribution with each new block of valid evidence, we multiply the probabilities of our P_i and A_i candidates by the probability that they were components of the rule [5] that caused the change observed. Importantly, the P_i and A_i running distributions also get re-normalized to enforce that our distributions sum to 1.0. Because of re-normalization, even though P_i and A_i elements successively get multiplied by numbers $\in \{0, 1\}$, their probabilities can increase as other candidates dwindle in probability or die out completely. In many cases candidate P_i and A_i's can even have probabilities of 1.0 if no viable alternatives exist. Because the probabilities of candidate rules are just the product of their component probabilities, our belief about rules is implicitly updated just like the P_i and A_i probabilities: the probability of rules also turns out to be proportional to how well they explain each block of evidence we accept. However, recall that we do not need to explicitly keep a list of all possible rules and their probabilities; rather, we can build rules and assign them probabilities whenever we need to. As a minor implementation note, we work with all probability calculations in log space to avoid numerical imprecision issues.

3.3 Building Rules from the Running Distribution

Building the current most likely rule reduces to picking the max for each P_i and A_i in our running distribution. If we want to enumerate many current possible rules, we take a recursive approach.

Let's consider a concrete case to get a feel for what our distributions might look like, how we will select most likely rules and how recursive rule building will work. Each tuple in the following is a pair that attaches a probability to a possible rule index symbol:

$$P_0 = \Big[(a, .7), (ac, .2), (ag, .1)\Big]$$

$$P_1 = \Big[(c, .5), (g, .3), (cgt, .2)\Big]$$

$$P_2 = \Big[(g, .4), (ct, .3), (t, .3)\Big]$$

[5] At their respective indices

$$A_0 = \Big[(0, .6), (t, .4)\Big]$$

$$A_1 = \Big[(c, .5), (2, .5)\Big]$$

$$A_2 = \Big[(1, 1.0)\Big]$$

For the most likely rules, we pick the peak or peaks [6] of each distribution. a;c;g@0c1 and a;c;g@021 here both have a max probability of (.7)(.5)(.4)(.6)(.5)(1) = .042. We can build 3 x 3 x 3 x 2 x 2 x 1 = 108 possible rules from these distributions. Building these possibilities from left to right, we can start by enumerating the possible rules up to the 0 index, our base case:

1. a
2. ac
3. ag

To extend this list to all rules up to the 1st index, we append all the possibilities for P_1 to what we already have:

1. a;c
2. a;g
3. a;cgt
4. ac;c
5. ac;g
6. ac;cgt
7. ag;c
8. ag;g
9. ag;cgt

In this way we can always enumerate all possible rules up to index i + 1, given all the rules up to index i. Since the base case is also always well defined, a recursive function is an effective approach for turning many possible P_i and A_i candidates into many actual rules we might eventually guess or use to design experiments.

3.4 Extension to Multiple Rule Guessing

Up to now we've focused on the sub-problem of single rule mutagens, assuming that we could generalize later. However, in theory and practice the existence of multiple rule mutagens significantly complicates matters. While we did successfully guess some multiple rule mutagens in the tournament, we fail for many cases; furthermore the implementation of multiple rule generalization even interfered with the strength of our initial single rule guesser. We'll start by illustrating the challenges, and conclude with two solutions.

[6] In fact there will often be many rules tied for the most likely.

3.4.1 Challenges

A single rule with disjunction can behave like multiple rules; therefor, single rules can be difficult to distinguish from multiple rules. One clarifying approach is to claim that rules with disjunction are equivalent to multiple rules without disjunction. For example, it would be convenient to state an equivalence between the rule sets $\{ac \rightarrow g\}$ and $\{a \rightarrow g,\, c \rightarrow g\}$. Then we are relieved from the burden of ambiguity of disjunction outlined as a principle challenge in the intro! However, such a pretense of equivalence is, unfortunately, false. Consider a genome made entirely of the base "a" to which we apply m mutations. Using the disjunctive single rule form the above example, we would observe m successful mutations, whereas the two rule form would only produce on average $\frac{m}{2}$ mutations. This is a consequence of the fact that the simulator randomly picks a rule to apply. Now, we cannot claim that a mutagen that strikes only half as often as another behaves equivalently. Thus it seems wrong to relax the constraints of the project to allow for completely eliminating disjunction. Rather we thought it more interesting to face the ambiguity of disjunction instead of further breaking down the entire problem into a matter of building evidence for any number of far more conspicuous non-disjunctive rules. [7]

All of this is to develop the point that it's hard to know how many rules there even are. Moreover, a manifest change from rule #2 may befuddle our algorithm if this change is fed as evidence for our rule #1 candidates. Ultimately, we strive to find a way to split blocks of evidence into as many sets as we have rules so that we can run our single rule algorithm on the sets separately. Then the challenges reduce to finding the right split out of r^M possibilities, where r is the number of rules and M is the total number of mutations we have seen. As with rule window size, we can also treat the number of rules as known and increment with time or good reason. Although r is likely no larger than 3, M could be 15 for each experiment and we may have hundreds of experiments, so the exponential nature of the possible set assignments defies brute force search.

3.4.2 Greedy Approach

Recall that each time we incorporate a new piece of evidence into the search for a single rule, we update the distributions that represent our belief about each candidate pattern and action index that could make up candidate rules. Although in the beginning our distributions will be less stable, over time we should expect some level of convergence such that more evidence will do little to shake our distributions– unless, of course, that evidence is contrary to what we have seen because it was produced by a different rule! In such a manner, we can build intuition for the greedy approach, which is as follows: keep r separate P_i and A_i distributions for the r rules we assume to exist. To digest each new block of evidence, update copies of of the r P_i and A_i distributions. Compute which of the r distributions peaks were the least disturbed by the new evidence and finally commit the evidence to updating that distribution. The approach is greedy because we decide how to put the M pieces of evidence into r buckets by making step by step locally optimal decisions. In practice we did not have time to debug this approach, and so we now turn to what we actually did.

3.4.3 Develop Confidence, One Rule at a Time

A simpler approach is to let our single rule guesser run until it develops a threshold level of confidence in a single rule being a part of the final rule set. If the single rule is still rebuffed by the simulator, we keep that rule as a member of the final set and move onto guessing another rule. This process can then iterate to make an attempt at any number of rules. Most importantly, when processing evidence for rules after the first, we

[7] Of course if we guess a disjunctive rule that was really two rules, the simulator gives us credit, but the point is that we are willing to consider disjunctive rules as guesses.

reject any evidence that explains a rule already in our final set. This was the approach that we actually took to some level of success. One of the drawbacks is that we don't have a way of neglecting evidence that will be explained by future rules we may accept into the final set. For example, when considering the first rule, evidence that was caused by the second rule will still murky the waters.

3.5 Player

3.5.1 Main Loop

At the heart of our player is the following loop:

1. `Generate experiment`
2. `Get mutated genome from simulator`
3. `Get q, the number of performed mutations from simulator`
4. `Record mutations`
5. `Generate best guess`
6. `Learn from incorrect guesses`

Almost all of the complexity of our player is confined to step 4 in the loop, recording mutations. Within recording mutations the pivotal step is to preprocess the evidence and update our belief state, to which we now turn.

3.5.2 Preprocessing Evidence and Updating Beliefs

The first question we ask while digesting incoming evidence– and immediately before updating our beliefs –is whether or not our player has a large enough assumed window size. As discussed, our player keeps a global window size that dictates one of the key assumptions about the mutation. As we break the evidence into valid blocks, as discussed in section 3.1, we are in a position to determine whether our current window size is adequate. We do this by comparing the number of evidence blocks extracted with the number of known mutations the simulator applied. Since each block should correspond to 1 actual mutation application, too many blocks indicates that the current window size is too narrow. In those cases, we must have broke a single applied mutation into 2 or more separate blocks of evidence; thus, it is here that we increment our considered window size to match the current full genome evidence.

As a final processing step, we apply two filters to the full evidence genome: one to filter out changed blocks that may have overlapping mutations and another to eliminate blocks that we're already fairly certain are explained by other rules we intend to guess (if we will guess multiple rules).

Updating our beliefs is then a matter of multiplying the new P_i and A_i distributions with their prior distributions, which is delegated to classes outside that of the main player since it's an involved calculation. [8]

[8] See section 2.

3.6 Additional Details

Note that the domain of each element of the patter P_i is the power set of $\{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$ minus the empty set. We can efficiently represent those 15 elements using a 4-bit string. For instance, '`at`' would be `1001` (making the bit significance order follow the alphabetical order of the bases).

4 Designing Better Experiments

One of the challenges for our approach to guessing the rules (mutagen) is to distinguish between multiple possible rules with equal or close probabilities. Some reasons may be that the rules with equal probability may be very similar to each other, or we may not have enough evidence from the random test strings to make their probabilities diverge, or out of pure coincidence. In order to better distinguish between such rules, we can design experiments, i.e. design the test strings.

4.1 Design Experiments For the Single-Rule Case

For the multiple candidate rules with equal (or similar) probabilities, at most one of them can be the actual rule. Thus, we can construct a string that specifically contains the sub-strings that match the patterns in the candidate rules. In the following analysis, for simplicity let's assume there are only two candidate rules.

Case 1: One of the candidate rules is the actual rule

In this case, after we construct the test string, it should contain both the sub-strings that match the pattern in the correct candidate rule and the sub-strings that match the pattern in the incorrect candidate rule. After the mutation, we should be able to see evidence supporting the correct candidate rule, as the sub-strings matching the pattern in the correct candidate rules get mutated as expected. In addition the probability for that rule should *increase.* On the other hand, the sub-strings matching the pattern in the incorrect candidate do not get mutated as expected, thus the probability for the incorrect candidate rule should *decrease.*

Case 2: Both candidate rules are incorrect

In this case, after we construct the test string, it should contain the sub-strings that match the pattern in either of the two incorrect candidate rules. After the mutation, we would not see evidence supporting these two candidate rules, and the probabilities for them should decrease. On the other hand, we may see evidence supporting other (potentially correct) candidate rules, and their probabilities would increase as a result.

It should be noted that in case 1, some sub-strings that match the pattern in one candidate rule may also match the pattern in the other candidate rule. If the actions in the two rules are different, then we are still able to distinguish between the two rules since these sub-strings are expected to mutate according to the correct candidate rule. On the other hand, if the actions in the two rules are the same, then we may be unable to distinguish between the two rules based only on these sub-strings. However, as the matching patterns in the two rules are not identical (otherwise the two rules would be equivalent to each other), we should expect to see some sub-strings elsewhere in the test string that match only the pattern in one of the two rules. We can distinguish between the two rules based on those sub-strings instead.

4.2 Design Experiments For the Multiple-Rule Case

The general methods for designing experiments under the single-rule case can also be extended to the multiple-rule case. Assuming we know there are n rules, we can divide the test string into n parts of equal length, each part testing the candidates for one rule. The way to construct each part of the test string is the same as the way to construct the whole test string under the single-rule case. In other words, part i of the test string should contain the sub-strings that match the patterns in the candidate rules for rule i.

4.3 Implementation

For the single-rule case, we design the test string using the following method (the test string begins as an empty string):

1. `Pick a random candidate rule from the set of rules with the highest probability.`
2. `Add a random string that matches the pattern in this candidate rule, with the length equal to the window size of the rule.`
3. `Add a random string with the length equal to 10 or two times the window size of the previous candidate rule, whichever is larger. This random string does not have to match any specific pattern.`
4. `Pick a random candidate rule from the set of rules with the second highest probability.`
5. `Repeat step 2 for this new candidate rule.`
6. `Repeat step 3 for this new candidate rule.`
7. `Repeat all previous steps until the test string has length at least 1000, or another pre-determined value. At this point, if the length of the test string is larger than 1000, then the first 1000 characters of the test string is returned. Otherwise, add random characters to the end of the test string until the length reaches 1000.`

As an example, suppose the set of rules with the highest probability contains rules “a;a;ac@cat” and “a;c;c@cat” while the set of rules with the second highest probability contains rule “c;a;t@0cc”. Then the test string may begin as “aaa cactgaatgc cat ttagcgtgga acc gagctacaat cat ctcatagcga aac ...” (spaces added for clarity).

Ideally the test string should contain many (depending on the window sizes) sub-strings that match the patterns in different candidate rules, and the mutation results from this test string should return more discriminating evidence. The reason for adding random characters between two sub-strings matching the patterns of candidate rules is to lower the probability of overlapping mutations or a situation where two mutations occur too close to one another. We strive to avoid such occurrences that create misleading evidence, as discussed earlier.

The proportion of the test string that is ”designed” according to the aforementioned method can also be adjusted, as a string consisting of random characters can be useful to discover new mutations and new candidate rules. Suppose we want only 80% of the test string to be ”designed”, then we can follow the same method until the length of the test string reaches 800, finally appending 200 random characters. During actual implementation, however, we set that proportion to be 100%, meaning the entire test string is "designed", as this leads to a better performance. Nonetheless, we only start designing experiments after 5 initial purely random string experiments. These initial random strings are used to discover mutations and set the initial probabilities for candidate rules.

Unfortunately, due to limited time, we did not have the opportunity to implement the method for designing experiments under the multiple rule case. This will be a potential area for future improvement.

5 Analysis of Performance

5.1 Summary of Tournament Results

The results for tournament mutagens with $m = 10$ and 5 runs are as follows[9]:

Mutagen	Our Result	Average Result	Groups' Results
c;acg;t@11g	2	2.875	4-2-2
ac;gt;g@tcc	5	4.875	7-1-0
a;acgt;c;acgt;g@987atc	1	1.375	2-1-5
c;a;t@tag + g@3	1	1.375	2-1-5
c;ct;a;t@0a23g + ac;t;actg;ag;ct@c12t4	0	1.875	3-0-5
c;a;g;t@atata + ac;a;ac;a@0gg	0	3.125	5-0-3
a;g;c@cgt + a@5	0	1.250	2-0-6
ct;ac;t@agt0 + ga;at;t@gattaca	3	1.500	1-3-4
c;a;t;g@tatc + t;a;g@gat	3	4.125	6-1-1
a@t + g@c	0	3.125	5-0-3
c;c@tt + c@t	0	3.750	6-0-2
acg;cgt;agt;act@tacgg + t;a;c;g;g@gtacg	0	0.125	0-1-7
a;ca;g;tc@ct1a + a;c;t;g@gattaca + c;a;t@tac	0	1.625	2-1-5
t;c;a;t;g@atccg + a;t;c;c;g@tcatg + a;t;c@g98	0	0.000	0-0-8
t;c;a;t;g;c@atccgt + a;t;c;c;g;t@tcatgc + a;t;c@gat	0	0.000	0-0-8
Average (by run)	1	2.067	
Overall (by mutagen)	1-5-9		45-11-64

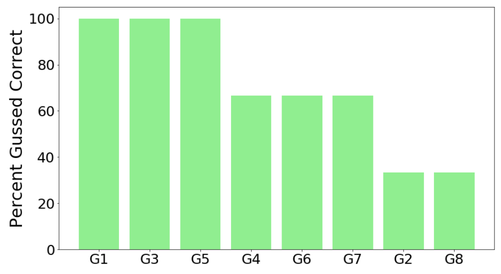
In the table above, "our result" denotes the number of runs where we are able to guess the mutagen successfully; "average result" denotes the average number of successful runs among all groups on this mutagen; "groups' results" denotes the number of groups that can guess this mutagen successfully in all 5 runs, in some of the 5 runs and in no run.

Overall, out of the 15 tournament mutagens, we are able to guess 6 of them successfully in at least one run. This figure is higher than that of 3 other groups (Groups 6, 7 and 8) but lower than that of 4 other groups (Groups 1, 2, 4 and 5). We are only able to guess 1 of them ("ac;gt;g@tcc") successfully in all 5 runs. This suggests that our approach may not be robust enough and is subject to randomness in the test mutagens.

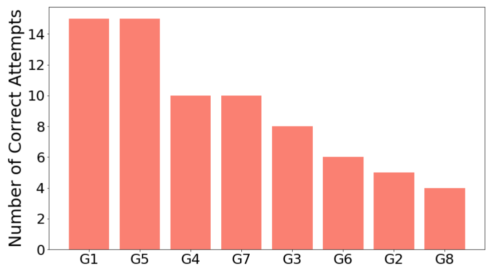
[9]We do not have complete results for $m = 5$ and $m = 15$, though based on partial results it appears that the results with these values of m will probably be very similar to those with $m = 10$.

5.2 Analysis of Results Under the Single-Rule Case

There are 3 single-rule mutagens in the tournament, and we are able to guess one of them ("ac;gt;g@tcc") successfully in all 5 runs. For the other two single-rule mutagens ("c;acg;t@11g" and "a;acgt;c;acgt;g@987atc"), we are able to guess them successfully in at least 1 out of the 5 runs. Together with Groups 1 and 5, we are among the three groups that are able to guess all three single-rule tournament mutagens in at least one run. These results suggest that we are relatively good at guessing mutagens with only one rule, even if the rule has a large window size. We highlight this point in the following charts:



(a) Percent of Single-Rule Mutagens Guessed Correctly at Least Once



(b) Number of Correct Attempts at Single-Rule Mutagens, Out of 15

Figure 8: Summary Visual of Accuracy for Single-Rule Mutagens Across Groups

One aspect that is worth mentioning is that when we guess correctly, we do so fairly efficiently. For the single-rule mutagens with short rules, we are often able to guess them within 20 experiments and 200 guesses, and can occasionally even guess in the first experiment within 10 guesses. In addition, even in the situation where we are unable to guess the mutagen correctly, we may still be able to make a "reasonable" guess with a Jaccard score that is not close to zero. For the mutagen "c;acg;t@11g", we guess it successfully in 2 out of 5 runs, but in the other 3 runs, despite that we are not able to guess it correctly, we can still make guesses with Jaccard scores between 25% and 50%.

With some additional testing, we have found that we are able to guess most single-rule mutagens regardless of window size, though we are more likely to succeed if the window size is smaller. Also, we have made some structural changes to our implementation in order to handle mutagens with multiple rules, and during testing after the tournament, we have found that our previous implementation works better for the single-rule mutagens.

We have tested both implementations with some single-run mutagens. 5 runs are performed on each mutagen for each implementation. Some results for our previous implementation are as follows:

Mutagen	Successful Runs	Average Experiments	Average Guesses
c;acg;t@11g	5	11.4	11.4
ac;gt;g@tcc	5	1.0	1.0
a;acgt;c;acgt;g@987atc	5	9.2	9.2
t;a;g@g1ttaca	5	10.6	10.6
c;a;t@52cg	5	9.2	9.2
ac;acgt;ct@78a	5	62.4	62.4
acgt@9876543210	5	6.8	6.8
a;ac;g;ct@ct1a3	5	5.2	5.2
a;ac;g;ct@ct1a7	5	13.2	13.2

In comparison, some results for our most recent implementation are as follows:

Mutagen	Successful Runs	Average Experiments	Average Guesses
c;acg;t@11g	2	9.5	218.0
ac;gt;g@tcc	5	1.0	1.0
a;acgt;c;acgt;g@987atc	1	42.0	1026.0
t;a;g@g1ttaca	5	5.0	102.6
c;a;t@52cg	0	N/A	N/A
ac;acgt;ct@78a	0	N/A	N/A
acgt@9876543210	5	4.4	86.8
a;ac;g;ct@ct1a3	5	4.0	79.6
a;ac;g;ct@ct1a7	2	8.0	180.5

In the cases where our guesser cannot guess a mutagen successfully in all 5 runs, the average numbers of experiments and guesses only take successful runs into account.

It can be seen from these results that compared with our most recent implementation, our previous implementation is more likely to guess the single-rule mutagens successfully. For the mutagens that both of our implementations can guess, our most recent implementation generally needs fewer experiments but more guesses, because in our previous implementation we make only one guess per experiment, whereas in our most recent implementation we make multiple guesses per experiment.

The first three mutagens listed above are the three single-rule mutagens in the tournament. If our previous implementation were used in the tournament to attempt these mutagens, we would be one of the three groups that could guess all single-rule mutagens in all runs (the other two groups are Groups 1 and 5).

Overall, our approach is able to guess single-rule mutagens successfully in most cases. During the structural changes we might have introduced some bugs that cause our most recent implementation to fail with some mutagens. Once these bugs are fixed our most recent implementation should be able to guess most single-rule mutagens as well.

5.3 Analysis of Results Under the Multiple-Rule Case

There are 9 two-rule mutagens in the tournament, of which we can guess 3 successfully in at least one run. Our chances of success do not seem to correlate with the window sizes of the two rules: we are unable to guess a mutagen with two rules of window sizes 1 (“a@t + g@c”), yet we are sometimes able to guess mutagens with two rules of larger window sizes. We believe this is due to the methods we use to come up with the two (or more) rules: we sometimes overlook shorter rules in favor of longer rules, and once we commit to rules to add to our set, we never look back. This greedy approach goes wrong quite often.

There are also 3 three-rule mutagens in the tournament, which we are unable to guess successfully. In general three-rule mutagens are more difficult than single-rule and two-rule ones, and some three-rule tournament mutagens are so difficult that they thwart all 8 groups.

For the tournament each of the 8 groups was asked to submit a mildly difficult mutagen. We submitted a two-rule mutagen ("c;ct;a;t@0a23g + ac;t;actg;ag;ct@c12t4"), and ironically we ourselves are unable to guess it successfully, while 3 other groups (Groups 1, 2 and 4) are able. The reason for submitting a two-rule mutagen was because single-rule mutagens tended to be either too easy or too hard for most groups. The last minute changes we did to our multiple rule guessing implementation left us unable to test enough cases and make improvements.

Overall, we believe that there is still much to be done to improve our performance at guessing multiple-rule mutagens.

6 Ideas for Further Improvement

First and foremost we wish to preserve our best single-rule guesser while also generalizing to multiple rules. This is a matter of debugging the data structures that allow us to generalize, such as lists of belief distributions for P_i and A_i instead of single distributions. Then to *improve* our multiple-rule guesser we would implement the greedy strategy discussed in section 3.4 "Extension to Multiple Rule Guessing". Taking a greedy strategy, it would then also be crucial that we gave our player a way of reversing course after mistakes. A simple reset after some amount of failure is one solution. Another is to reset after all of our candidate rules are too low in probability to be feasible. Another multiple-rule case behavior that we should correct is our player's tendency to guess too many rules. We could, in a probabilistic way, bias our player towards a belief in fewer rules, or else set a hard limit at 3 (the tournament limit).

Since there are only a small number of possible short rules up to window size 2, we could further improve our performance on multiple-rule mutagens with such short rules by guessing all possible combinations of short rules (up to 3 rules) that are consistent with the observations in the first experiment. If none of these combinations is the actual mutagen, we then move on to guessing mutagens with longer rules.

We could also choose to relax our intentions of handling the ambiguity of disjunction in the case of multiple rules. Then we could assume all rules will have no disjunction and add rules to our guessing set after collecting irrefutable evidence. We could obtain such incontrovertible evidence after eliminating disjunction because of the drastic reduction of ambiguity in evidence.

7 Conclusion

Beginning with a written formalization of the problem and a causal graph [10] granted us a powerful framework. With such a mathematical structure, we were able to derive a number of independence assumptions about the component elements of rules.

[10]See section 2

In order to manage the complexity of the task, we dissected the problem into sub-problems. To do this we:

1. Created a single-rule guesser that could later be adapted to multiple-rule guessing
2. Derived the probability of rules from the probability of their components
3. Integrated distributions deduced from single blocks of evidence with the entire set of prior evidence
4. Constrained the search space to a fixed window size, and incremented when it was reasonable or necessary to do so

By translating our probabilistic framework and modular ideas into code, we were able to build a somewhat consistent and efficient single-rule guesser for the tournament. Moreover, although it wasn't necessary for the tournament, we were able to handle rules with disjunction without treating them as multiple rules. This is an arguably more challenging task that also preserves the full probabilistic behavior of mutagens. We also managed to generalize to multiple-rule guessing, albeit with limited success. Although we have much room for improvement, we believe we have laid a strong foundation.

8 Group Member Individual Contributions

All group members have provided general ideas and algorithms for this project. In addition, some particular contributions from each group member are as follows:

Juan Correa: Probabilistic modeling of the problem, implemented player's main loop and most likely rule estimation. Attempted extension to multiple-rule-mutagen guessing.

John Daciuk: Wrote sections 1, 2.2, 2.3, 3 and 7 of the report. Contributed to section 6 and created graphs for the appendix. Contributed to a handful of helper functions for core player behavior.

Patrick Kwok: Implemented the initial structure of the guesser, implemented the algorithm for designing experiments, conducted tests and analyzed results. Also wrote the corresponding parts of this report.

9 Acknowledgments

We would like to thank Professor Kenneth Ross and the rest of the class for the valuable discussion both during and after class, and Chengyu Lin, a teaching assistant for this class, for providing a simulator on which we develop our guesser and test our implementation.

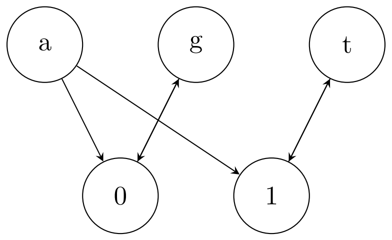
We would like to thank some other groups for providing inspirations for some of the ideas in our implementation. In particular we would like to thank Group 1 for the idea of designing experiments, Group 4 for the idea of guessing multiple times in each experiment, and Group 7 for the idea of moving to guessing multiple rules after single-rule guesses.

For the testing process, we would like to thank other groups for providing sample mutagens and (particularly Groups 1 and 4) allowing us to test our sample mutagens on their implementations. This is an important part of this project, since it allows us to have an estimation on the relative difficulties of different mutagens.

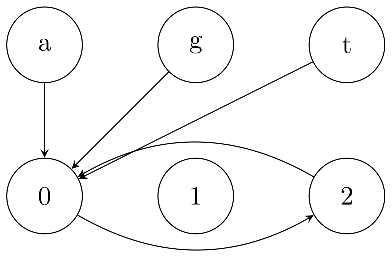
10 Appendix: Additional Rule Visualizations

Let the nodes labeled with numbers represent the sequence across which the mutation happens. The arrows going into the sequence specify the pattern that must be matched for the mutation to happen. The arrows leaving the sequence determine what the sequence will become after the mutation.

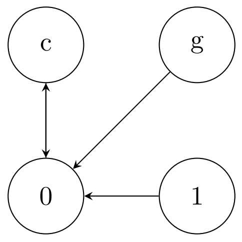
10.1 Rule: ag; at $\rightarrow$ gt



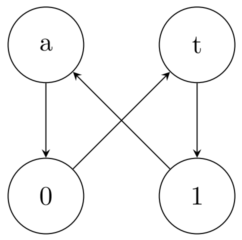
10.2 Rule: agt $\rightarrow$ 210



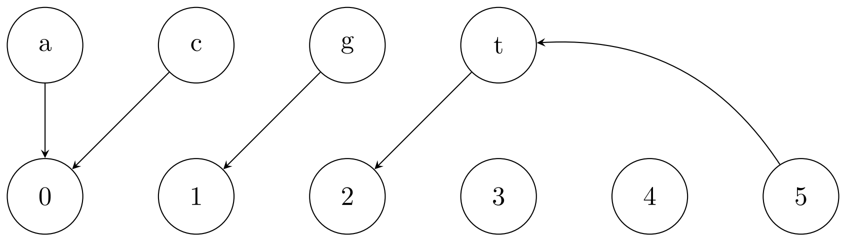
10.3 Rule: cg; → c0



10.4 Rule: a;t → ta

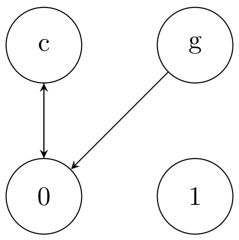


10.5 Rule: ac; g; t → 01234t



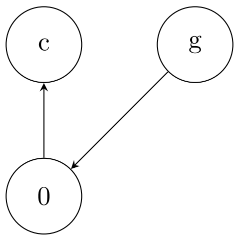
11 Appendix: Simplifying Rules

11.1 Rule: cg;agtc → c1

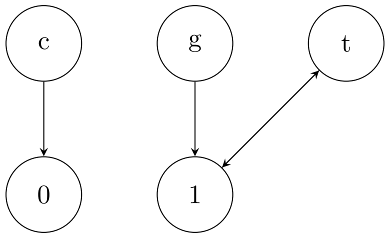


First, we may like to write this rule as cg; → c1, where nothing after the semicolon denotes a match with anything. When a node can match anything, we can say that the mutation has no matching requirement there.

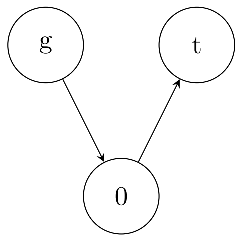
We could simplify this rule to "g → c". Notice that in doing so we would be removing a cycle from the graph. Generally, if we can remove edges from a graph we will say it's simpler. Removing cycles often does not grant an equivalent mutation, but it's worth considering when we can remove cycles. Also notice that we remove the "1" node because the mutation is not required to match there and that node will not mutate. In other words, it has no incoming or outgoing edges. If the node was not on one of the ends of the sequence we would leave it as a placeholder. So now the resulting graph becomes:



11.2 Rule: c;gt → 0t



We can eliminate node 0 because it's dangling all the way to one side with no outgoing edges. Once we're left with a single sequence node, we can then eliminate the cycle. We then get the equivalent rule: g → t



References

[Koller and Friedman, 2009] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*, volume 2009.

[Pearl, 2000] Pearl, J. (2000). *Causality: Models, Reasoning, and Inference.* Cambridge University Press, New York, NY, USA, 2nd edition.