Random Words in a (Weighted) Regular Language: a Free Energy Approach

Cewei Cui and Zhe Dang

School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99164, USA
{ccui,zdang}@eecs.wsu.edu

Abstract. We study random words in a weighted regular language that achieve the maximal free energy using thermodynamics formalism. In particular, typical words in the language are algorithmically generated which have applications in computer security (anomaly detection) and software engineering (test case generation).

1 Introduction

A random walk is essentially a stochastic process, which has been widely used in many disciplines, including mathematics, physics, and finance. A Markov chain is probably the simplest and yet most useful form of random walks, where its ergodicity, hitting time, convergence time etc. have all been well-studied [11,2]. On the other hand, many physical behaviors can be modeled by random walk models, such as Brownian motion, and Ising model. In finance, a famous book “A random walk down wall street” [13] shed light to the path of using mathematical modelling to predict the stock market.

Random walks have applications in Computer Science as well, in particular in using random walks on graph data structures (such as graph represented image and social network). Turning to automata theory, a wide spectrum of applications can also be found, from the earliest Rabin automata (dated back to 1960s) all the way to more modern quantum automata. Since automata are the fundamental model for all computer programs (e.g., an instruction can be understood as a transition), this line of research addresses the problem on, given transition probabilities in a computer program, how the program is going to behave. This is important, since such programs are in turn the backbone of randomized algorithms. However, what we are looking for here concerns a related but essentially different problem:

Given: a program (without probabilities),
Question: what would be the program’s “most random” behaviors?

Clearly, one can re-interpret the problem in various contexts. For instance,

Given: a regular language (with no probabilities),
Question: what are the “most random” words?

As we shall state later, answers to these questions have applications in many areas in Computer Science (such as in network intrusion detection, one would
like to know what the normal/abnormal traffic is, without seeking help from some pre-given probabilistic model – in reality, properly assigning probabilities in a behavioral model is difficult and of course error-prone).

These problems are not new at all, where their most common solutions require computation of a random walk (since it is not given) on a graph using, to our best knowledge, one of the following two main approaches:

1. a symmetric random walk on a graph with uniform probability assignment on branches from the same node;
2. a maximal entropy random walk on a graph to achieve roughly uniform probability on paths.

Turning back to automata theory, the latter approach has been followed by, e.g., Bassett’s maximal entropy stochastic process on timed automata \cite{3}. The random walk model computed in our research is different from the aforementioned two approaches, since we focus on weighted graphs, where a weight can model cost, time, etc for an event. Simply speaking, the computed probability of a path is proportional to its weight. If all paths with the same length take the same weight, it degenerates to the maximal entropy model. In practice, a weight can be used to model time, cost, or risk level, etc., associated with an event.

The theoretical underpinning of our approach comes from thermodynamic formalism \cite{17}, where a weighted regular language (in this paper, we consider the simplest form weighted language where each symbol in the alphabet is given a weight) is the collection of trajectories of a gas molecule moving along the weighted graph defining the minimal deterministic finite automaton (DFA) accepting the language. When the graph is strongly connected, one can compute a random walk – a unique Markov chain $\mu^*$ – that achieves the maximal free energy of the gas molecule, using the theory developed by Gurevich in the 80’s. Clearly, from the computed $\mu^*$, one can generate random words (we call them Gurevich random words) in the weighted language. Hence, we call our approach a free energy one.

The rest of the paper is organized as follows. In Section 2, we present the basics physics behind Gurevich free energy. In Section 3, Gurevich random words in weighted regular languages are proposed and investigated. In Section 4, we study the AEP (Asymptotic Equipartition Property) in a (weighted) regular language and hence typical words in the language can then be computed. In Section 5, we discuss the applications of the random words in software testing and computer security. We conclude the paper in Section 6.

2 Physics of Gurevich Free Energy

When a gas molecule moves, measurements such as its position, speed, etc. can be made at various times. Each such measurement is called a micro-state and thus, we obtain a sequence, called an orbit, of micro-states $\alpha_n = s_0, \ldots, s_n$. There is a change in potential $w(s_i, s_{i+1})$ when the molecule moves from $s_i$ to
The Boltzmann equation defines the relationship between energy and choice:

$$E = k_B \ln \mathcal{C},$$

where the energy $E$ of the molecule can be expressed as the logarithm of the number of choices (of the next micro-state), where $k_B$ is the Boltzmann constant. This is not hard to understand at all: an active molecule carries high energy since it has more choices to move around. Therefore, $e^{w(\alpha_n)}$ (we ignore the Boltzmann constant) will be the total number of choices in the process when the molecule moves on the orbit $\alpha_n$ that is from $s_0$ to $s_n$. Summing up the choices for all the orbits $\alpha_n$ and then taking logarithm, we will obtain the energy of the molecule on orbits with length $n$:

$$\ln \sum_{\alpha_n} e^{w(\alpha_n)}.$$ 

However, this would not give a finite value as $n \to \infty$. We now consider the asymptotic energy (per step)

$$\lambda = \limsup_{n \to \infty} \frac{1}{n} \ln \sum_{\alpha_n} e^{w(\alpha_n)}. \quad (2)$$

Unfortunately, to compute the $\lambda$ is not trivial at all and has been one of the central topics in thermodynamic formalism [17]. A crowning achievement in the decades of research in the formalism is the following Variational Principle [17] (with some side conditions, which are ignored for now for simplicity of our presentation):

$$\limsup_{n \to \infty} \frac{1}{n} \ln \sum_{\alpha_n} e^{w(\alpha_n)} = \sup_{\mu} \left( \int w \, d\mu + H_\mu \right), \quad (3)$$

which can be intuitively understood as the following:

- The LHS, which is the aforementioned energy $\lambda$, can be computed through the RHS;
- To compute the RHS, one would find a $\mu$, a discrete time Markov chain, that maximizes the sum of the average potential change per step and the Kolmogorov-Sinai entropy.

A seminal result is due to Gurevich [8]: the RHS can indeed be achievable by a unique Markov chain $\mu^*$ when the $w(\cdot, \cdot)$ is defined on a finite graph (i.e., the aforementioned micro-state is a node in the graph) that is an SCC (strongly connected component). The Markov chain $\mu^*$ has a unique stationary distribution and assigns every edge of the graph with a transition probability. The RHS is called Gurevich free energy and thus, we call the $\lambda$ as the free energy of the
molecule. We now sketch the Gurevich algorithm in computing the $\lambda$ and the $\mu^*$. Let $G$ be a (directed) graph with nodes in $V$ and edges in $E$. We consider a gas molecule’s moving as a walk on $G$ while a node in $V$ resembles a micro-state of the molecule. And hence, the gas molecule can be observed as a sequence of nodes; i.e., a walk in $G$ (herein, a walk may contain loops). In particular when an edge $e$ in $E$ is associated with a weight $w(e) \in \mathbb{Q}$ (rationals), the weight resembles the potential. Notice that, as a special case, if the $G$ is not weighted graph, we may simply take $w(e) = 0$ for all $e \in E$. We shall note that there are no probability assignments on the edge of $E$ and hence $G$ is just a graph.

Given these, how can we create the “most random” walks on $G$? In our opinion, the walk that we are looking for shall maximize the free energy among all possible walks on $G$; i.e., the Gurevich $\mu^*$ that achieves the superume on the RHS of the aforementioned Variational Principle. The way to compute $\mu^*$ is laid out in the seminal paper of Gurevich[8] where the graph $G$ is required to be strongly connected, and is shown in Algorithm 1. Notice that the free energy rate $\lambda$ in (2) can also be computed as $\lambda = \ln \hat{\lambda}$ where $\hat{\lambda}$ is the Perron number computed in the algorithm.

**Algorithm 1** Computing Gurevich random walk from a weighted graph that is strongly connected

**Require:** $M$ is the adjacency matrix of a strongly connected weighted graph $G = \langle V, E \rangle$. Each entry in $M$, $M_{i,j}$, represents the weight from node $i$ to node $j$ in $G$. If there is no edge from node $i$ to node $j$ in $G$, we simply take the $M_{i,j} = -\infty$.

1: function **ComputeTransitionProbability**($M$)
2: First build a weight matrix $W = \{W_{i,j}\}$
3: for $i, j$ in $\{ M_{i,j} \}$ do
4:   if $M_{i,j} \neq -\infty$ then
5:     $W_{i,j} = e^{M_{i,j}}$
6:   else
7:     $W_{i,j} = 0$
8: end if
9: end for
10: Conduct eigen decomposition on the matrix $W$.
11: Obtain the right eigenvector of matrix $W$, $v = (v_1, v_2, \ldots, v_n)$.
12: Compute the Perron number (i.e., the largest eigenvalue) of $W$, $\hat{\lambda}$.
13: Using Parry measure, obtain the transition probability matrix $P = \{P_{i,j}\}$ that defines the $\mu^*$:
14: for $i, j$ do in $W_{i,j}$
15: $P_{i,j} = \frac{W_{i,j} \cdot v_j}{\hat{\lambda} v_i}$
16: end for
17: end function
3 Gurevich Random Words in a (Weighted) Regular Language

Let $L$ be a regular language on alphabet $\Sigma$. In particular, each symbol $a \in \Sigma$ is associated with a weight $w(a) \in \mathbb{Q}$ (rationals). Our random word generation uses Algorithm 1 (hence we call them Gurevich random words) and therefore relies on a weighted graph. Herein, the graph is a DFA accepting $L$.

Let $M$ be a DFA on alphabet $\Sigma$ and with states in $\mathbb{Q}$ where $q_0$ is the initial state and $F \subseteq Q$ is the set of accepting state. In particular, each symbol $a \in \Sigma$ carries the aforementioned weight $w(a)$. We use $p \xrightarrow{a} q$ to denote the transition from state $p$ to state $q$ while reading input symbol $a$ with weight $w(a)$. Many times we write $p \xrightarrow{a,w(a)} q$ when we want to emphasize the weight $w(a)$.

Now, consider a run of $M$ from the initial state, for some $n > 0$,

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \rightarrow \ldots \xrightarrow{a_n} q_n,$$

where each $q_{i-1} \xrightarrow{a_i} q_i$ is a transition in $M$. The run is called the run on the word $\alpha = a_1a_2\ldots a_n$. As usual, when $q_n$ is an accepting state in the run, we say that $M$ accepts the word $\alpha = a_1a_2\ldots a_n$. We use $L(M)$ to denote all words $\alpha$ accepted by $M$.

We assume that $M$ is cleaned up. That is, every state in $M$ is reachable from the initial state and every state can reach an accepting state (and hence we do not consider the trivial case when $M$ accepts the empty language). An example DFA $M$ is shown in Figure 1.

![Fig. 1. An example DFA with weights $w(a) = 2$, $w(b) = -1$.](image)

How to generate a random word in $L(M)$? Of course, the randomness shall depend on the weights $w(\alpha)$ assigned to symbols $a$ in $\Sigma$. A straightforward way to obtain such random words would be use a most “natural” way to assign transition probabilities on transitions in $M$, each such probability is uniform among all outgoing transitions from each state. For instance, the probability assignments shown in Figure 2 for the DFA in Figure 1. However, there are problems with such uniform probability assignments:

- Though there could be many DFA $M$’s that accept the same $L$, the resulting probability assignments are not the same. Hence, the “randomness” depends on the choice of $M$, instead of $L$. (2). The probability assignments
must be associated with the weights in certain ways such that the resulting “randomness” conforms with established randomness metrics like Shannon’s information rate \( \lambda_L \) (see also our recent work \( \cite{6,7} \)) of the \( L \), under the special case when all weights are 0 (unweighted words). Herein, the rate \( \lambda_L \) is defined (by Shannon) as

\[
\lambda_L = \lim \sup_{n \to \infty} \frac{\log |L_n|}{n},
\]

where \( |L_n| \) is the number of words with length \( n \) in \( L \). For instance, if \( w(a) = w(b) = 0 \) in the example DFA \( M \) shown above, we can compute the information rate of the regular language \( L = a(aa + b)^* \) accepted by the \( M \) is \( \lambda_L = 0.4812 \). However, the entropy rate of the Markov chain as the result of uniform probability assignments, as shown in Figure 2, is 0.4621. The two rates do not conform.

\[
\begin{align*}
qu_0 & \xrightarrow{pr=0.5, a} qu_1, \\
qu_0 & \xrightarrow{pr=0.5, b} qu_1, \\
\end{align*}
\]

**Fig. 2.** Uniform probability assignments for the example DFA.

To address the problems, we first present an algorithm to generate random words in \( L \) based on Gurevich’s Algorithm \( \cite{11} \), where the generated random words achieve the maximal free energy. We considered it as a most natural way to generate random words since, if a word in \( L \) reflects a potential change between measurements in a gas molecule, then the nature tends to make the molecule to be in maximal free energy (actually, in the physical world, the free energy shall be minimal – this is due to the fact that the sign of the energy is flipped in thermodynamics formalism for mathematical convenience, see more details in Sarig’s notes \( \cite{18} \)).

Let \( M \) be the minimal DFA that accepts \( L \). Notice that \( M \) is cleaned up as we have mentioned earlier. It is known that the \( M \) is unique (up to isomorphism on state names) \( \cite{10} \). Applying the weight function \( w \), the DFA \( M \) is also weighted where each transition with input symbol \( a \) is assigned weight \( w(a) \).

We first make the \( M \) be strongly connected. Let \( \Diamond \notin \Sigma \) be a new symbol. For each accepting state \( q \), we add a transition to the initial state \( q_0 \):

\[
q \xrightarrow{\Diamond} q_0.
\]

The resulting DFA is written \( M_\Diamond \). The example automaton in Figure 2 is now modified into the automaton in Figure 3.
One can verify that $M_{\diamond}$ is indeed strongly connected. (Note that for the example automaton in Figure 1 it is already strongly connected. For such an automaton $M$, there is no need in converting them into $M_{\diamond}$. For the ease of presentation, we convert it anyway.) The new symbol carries a weight $w(\diamond)$ that is a pre-given very small negative number (i.e., $\ll$ the minimal weight on $M$). We call these $\diamond$-transitions as $\hat{\epsilon}$-edges while we bear in mind that $\hat{\epsilon} = e^{w(\diamond)}$ is a positive number very close to 0. (Adding such $\hat{\epsilon}$-edges to make a graph strongly connected is not a new idea; it has been used in the Google page rank algorithm [15].)

Next, we convert $M_{\diamond}$ into a graph $G_M$ as follows. Notice that DFA $M$ itself may not be a graph: there could be multiple transitions from a node to another. Each state in $M_{\diamond}$ is also a node in $G_M$. Additionally, $G_M$ has some other nodes shown below. Initially, $G_M$ has no edges. For each transition $p \xrightarrow{a} q$, with $a \in \Sigma \cup \{\diamond\}$, in $M_{\diamond}$, we add a new node $\langle p, a, q \rangle$ and add the following two edges to $G_M$:

- the edge, with weight $w(a)$, from node $p$ to node $\langle p, a, q \rangle$;
- the edge, with weight 0, from node $\langle p, a, q \rangle$ to node $q$.

It is not hard to verify that the resulting weighted graph $G_M$ is strongly connected. Figure 4 shows the result of $G_M$ from the $M_{\diamond}$ in Figure 3.

We then run Algorithm 1 on the graph $G_M$. As a result, we obtain transition probabilities on each edge of $G_M$. In particular, suppose that $\theta$ is the transition probability computed on the edge from node $p$ to node $\langle p, a, q \rangle$, with $a \in \Sigma \cup \{\diamond\}$, we now label a transition probability $\theta$ to the transition $p \xrightarrow{a} q$ in the original DFA $M_{\diamond}$. As a result, we obtain a probabilistic DFA $\hat{M}$ where each transition is assigned with a probability. Figure 5 shows the resulting $M$ after we run the Algorithm 1 on the graph $G_M$ shown in Figure 4. One may compare the different probability assignments in Figure 5 and in Figure 2.

Notice that, because of the introduction of additional $\diamond$-transitions, the resulting $\hat{M}$ may not be a perfect Markov chain (after dropping the $\diamond$-transitions). In other words, for an accepting state, the sum of all transition probabilities on all the outgoing transitions from the accepting state could be strictly less than 1 but still very close to 1 because of the choice $e^{w(\diamond)}$ to be very close to 0 (see Figure 5 for an example). We of course can tolerate this, since $\hat{M}$ itself already presents a random algorithm to generate random words from the $L$:
Fig. 4. The weighted graph $G_M$ converted from $M_\Diamond$ in Figure 3, with $w(\Diamond) = -1000$.

Fig. 5. The resulting probabilistic DFA $\hat{M}$ for the example DFA $M$ in Figure 1.
Algorithm 2 Generate a random word on $\hat{M}$ with a given lower bound $N$ on walk

**Require:** $P = \{P_{i,j}\}$ is the transition probability matrix of $\hat{M}$, and $E = \{E_{i,j}\}$ is the transition set of $\hat{M}$

1: function GENERATERANDOMWORD($P$)
2: For every state in $\hat{M}$, build its cumulative transition probability matrix.
3: \hspace{1em} for $q_i$ in $\hat{M}$ do
4: \hspace{2em} Use $P_{i,j}^{\text{cum}}$ to denote its cumulative transition probability.
5: \hspace{2em} $P_{i,j}^{\text{cum}} = P_{i,0} + P_{i,1} + \cdots + P_{i,j}$
6: \hspace{2em} $P_{i,-1}^{\text{cum}} = 0$
7: \hspace{1em} end for
8: Let $q_x = q_{\text{init}}$ and $n = 0$
9: Generate a random number $r \in (0, 1]$
10: \hspace{1em} for $j$ in $\{P_{x,j}^{\text{cum}}\}$ do
11: \hspace{2em} if $r > P_{x,j-1}^{\text{cum}}$ and $r \leq P_{x,j}^{\text{cum}}$ then
12: \hspace{3em} Output the symbol on edge $E_{i,j}$
13: \hspace{3em} \hspace{1em} $n++$
14: \hspace{3em} Let $q_x = q_j$ break from the loop.
15: \hspace{1em} end if
16: \hspace{1em} end for
17: Goto 9 or if $q_x$ is an accepting state and $n > N$ Stop.
18: end function

We use Fig. 5 as an example to explain the Algorithm 2. Let $q_0$ be the initial state. Using the algorithm we have $P_{0,0}^{\text{cum}} = 0$, $P_{0,1}^{\text{cum}} = 1$, $P_{1,0}^{\text{cum}} = 0.9514$, $P_{1,1}^{\text{cum}} = 1$. The following is an example run of the algorithm.

- Current state is $q_0$; generate a random number $r = 0.8$
  - Output the symbol $a$, because $0.8 > P_{0,0}^{\text{cum}}$ and $0.8 \leq P_{0,1}^{\text{cum}}$; Change the current state = $q_1$
- Current state is $q_1$; generate a random number $r = 0.3$.
  - Output the symbol $a$, because $0.3 > P_{1,0}^{\text{cum}}$ and $0.3 \leq P_{1,1}^{\text{cum}}$. Change the current state = $q_0$.
- ... 
  - Current state is $q_1$; generate a random number $r = 0.97$.
  - Output the symbol $b$, because $0.97 > P_{1,0}^{\text{cum}}$ and $0.97 \leq P_{1,1}^{\text{cum}}$. Change the current state = $q_1$.
  - ...

We shall point out that the random words generated by Algorithm 2 is a word in $L^\Diamond = (L\Diamond)^* L$; i.e., a sequence of words in $L$ separated by the $\Diamond$’s. For practical purposes, each such word in $L$ can be treated as a Gurevich random words in $L$. Notice also that this treatment of the random words in $L$ will make it possible to generate Gurevich random words in a finite language $L$ (whose rate, by definition, is $-\infty$).

In the following, we will prove that our algorithm of the random walk is “correct” in the sense that the rate $\lambda_\Diamond$ of the weighted regular language accepted
by $M\hat{\diamond}$ approaches to the rate $\lambda$ of the weighted regular language $L$ as the weight $w(\hat{\diamond})$ assigned on the $\hat{\epsilon}$-edges goes to $-\infty$:

$$\lambda\hat{\diamond} \to \lambda \text{ as } w(\hat{\diamond}) \to -\infty.$$  

(4)

We first need clearly define what the rates are in below:

$$\lambda = \limsup_{n \to \infty} \frac{1}{n} \ln \sum_{\alpha \in L, |\alpha| = n} e^{w(\alpha)},$$

(5)

and

$$\lambda\hat{\diamond} = \limsup_{n \to \infty} \frac{1}{n} \ln \sum_{\alpha \in L^{\hat{\diamond}}, |\alpha| = n} e^{w(\alpha)},$$

(6)

where $L^{\hat{\diamond}} = (L\hat{\diamond})^* L$ is the language accepted by $M\hat{\diamond}$. Recall that we run the Algorithm I on the graph $G_M$ constructed from $M\hat{\diamond}$, so that we finally assign probabilities on the transitions in $\hat{M}$. Notice that, inside the algorithm, the Perron number $\hat{\lambda}$ is also computed. The main result of our earlier paper [4] (the energy rate of a regular language can be computed as the energy rate of the DFA (as a graph), when the graph is strongly connected) shows that

$$\lambda\hat{\diamond} = 2 \ln \hat{\lambda}.$$  

(7)

We shall now thread this earlier result together with the Variational Principle: the probabilistic program we obtained on $G_M$, which is a strongly connected graph without initial and accepting nodes, defines the $\mu^*$ that achieves the RHS of the principle, due to Gurevich [5]:

$$\ln \hat{\lambda} = \limsup_{n \to \infty} \frac{1}{n} \ln \sum_{\alpha \text{ is a walk on } G_M \text{ with length } n} e^{w(\alpha)} = \int w \mu^* + H_{\mu^*},$$

(8)

where the walk $\alpha$ can be between any two nodes. That is, the random walk $\mu^*$ on $G_M$, as the result of the Algorithm I does achieve the rate $\ln \hat{\lambda}$. What’s the difference between the random walk in $\hat{M}$ and the random walk in $G_M$? Each transition $p \xrightarrow{a} q$ in $\hat{M}$, as we have constructed earlier, is the result of two edges in $G_M$: $p \xrightarrow{w(a)} (p, a, q) \xrightarrow{0} q$. Notice that the algorithm has to assign probability 1 to the edge $(p, a, q) \xrightarrow{0} q$ since the outdegree of the node $(p, a, q)$ is 1. Therefore, a random walk in $\hat{M}$ uniquely corresponds to a random walk in $G_M$, while the only difference is the length is shortened by half. Therefore, the random walk defined in $\hat{M}$ does achieve the rate $2 \ln \lambda\hat{\diamond}$. This is also the fact even when the walk in $\hat{M}$ starts from the initial state and ends with an accepting state (as we did in the probabilistic program version of the $\hat{M}$), because of our earlier result in (7). Hence, once we prove the claim in (4), the random words generated from the probabilistic program defined by $\hat{M}$ do achieve the maximal free energy in the Variational Principle.
We now prove the claim in (4). Let $m > -\infty$ be the minimal weight of all transitions in $M$. Consider

$$\limsup \frac{1}{n} \ln \sum_{|\alpha|=n, \alpha \in L} e^{w(\alpha) - mn} = \lambda - m,$$

where $\lambda = \limsup_{n \to \infty} \frac{1}{n} \ln \sum_{|\alpha|=n, \alpha \in L} e^{w(\alpha)}$ is defined in (5). Note that the $\lambda \geq -\infty$. (When $L$ is a finite language, the $\lambda = -\infty$. Otherwise, the $\lambda$ is finite and $\lambda \geq m$.) Fix a small $\epsilon > 0$. Then, by definition, there is an $N_\epsilon > 0$ such that

$$\forall n > N_\epsilon, \quad E^w(L_n) - nm := \sum_{|\alpha|=n, \alpha \in L} e^{w(\alpha) - nm} \leq e^{n(\lambda - m + \epsilon)},$$

where $L_n$ is the set of words in $L$ with length $n$. Notice that the term $E^w(L_n) - nm$ is defined as 0 when $L_n = \emptyset$. We now consider $M_\diamondsuit$ with the negative number $w(\diamondsuit)$ satisfying that $\hat{\epsilon} = e^{w(\diamondsuit)}$ will make the following two items true:

- For each $l \leq N_\epsilon$,

$$E^w(L_l) \hat{\epsilon} \leq \epsilon l + 1,$$

where $E^w(L_l) := \sum_{|\alpha|=n, \alpha \in L} e^{w(\alpha)}$ and therefore,

$$E^w(L_l) - ml \hat{\epsilon} \leq \epsilon$$

is also true.

- $\hat{\epsilon} \leq \epsilon$

Clearly, $M_\diamondsuit$ accepts a new language

$$L_\diamondsuit := (L\diamondsuit)^\ast L$$

whose energy rate is, by definition in (4),

$$\lambda_\diamondsuit := \limsup \frac{1}{n} \ln \sum_{|\alpha|=n, \alpha \in L_\diamondsuit} e^{w(\alpha)}$$

$$= m + \limsup_{n \to \infty} \frac{1}{n} \ln \sum_{|\alpha|=n, \alpha \in L_\diamondsuit} e^{w(\alpha) - nm}.$$

Notice that the $\lambda_\diamondsuit$ is a finite number since $L_\diamondsuit$ is not a finite language (as mentioned earlier, we do not consider the trivial case when $L = \emptyset$ and, furthermore, the null word is taken out of the $L$). Now we focus on estimating the term $\sum_{|\alpha|=n, \alpha \in L_\diamondsuit} e^{w(\alpha) - nm}$ with $n > N_\epsilon$. Notice that each $\alpha$ in $L_\diamondsuit$ with length $n$ takes the following form:

$$\alpha_{e_1} \circ \alpha_{e_2} \circ \alpha_{e_3} \circ \ldots$$

\[\begin{array}{c}
\xrightarrow{l_1} \\
\xrightarrow{l_2} \\
\xrightarrow{l_3}
\end{array}\]
where $\alpha$ contains, say, $k$ diamondsuits for some $0 \leq k \leq n$. Each non-diamondsuit block has its own length, say, $l_1, l_2, \ldots, l_{k+1}$, with $l_1 + l_2 + l_3 + \cdots + l_{k+1} = n - k$. Notice that $l_i > 0$ for each $i$, recalling that the null word is taken out of $L$. In this case, each block is either "short" (i.e., the length $\leq N_\epsilon$) or "long" (i.e., the length $> N_\epsilon$).

Suppose that, among the $k + 1$ blocks of lengths $l_1, l_2, \ldots, l_{k+1}$ respectively, there are $r$ short ones and $R$ long ones with $r + R = k + 1$. Furthermore, the total length of the long ones is denoted by $\text{LongLength} \leq n$. Of course, when $L$ is a finite language, there are no long blocks (when $N_\epsilon$ is large enough).

It is left to the reader to verify the following two cases:

Case 1. $L$ is an infinite language and hence $+\infty > \lambda - m > 0$. In this case, the term to be estimated
\[
\sum_{|\alpha|=n, \alpha \in L^\diamond} e^{w(\alpha) - mn}
= \sum_{0 \leq k \leq n, l_1 + \cdots + l_{k+1} = n - k, l_i > 0} e^{w(L_{l_1}) - ml_1} \cdot e^{w(L_{l_2}) - ml_2} \cdot \cdots \cdot e^{w(L_{l_{k+1}}) - ml_{k+1}}
\]
(Using (13))
\[
\leq \sum_{0 \leq k \leq n, l_1 + \cdots + l_{k+1} = n - k, l_i > 0} e^{\text{LongLength}(\epsilon + \lambda - m)} \cdot R
\leq \sum_{0 \leq k \leq n, l_1 + \cdots + l_{k+1} = n - k, l_i > 0} \epsilon^{k+1} e^{n(\epsilon + \lambda - m)}
\leq \sum_{0 \leq k \leq n, l_1 + \cdots + l_{k+1} = n - k, l_i \geq 0} \epsilon^{k+1} e^{n(\epsilon + \lambda - m)}
= \sum_{k=0}^{n} \binom{n}{k} \epsilon^k e^{n(\epsilon + \lambda - m)}
= (1 + \epsilon)^n \epsilon e^{n(\epsilon + \lambda - m)}.
\]

Therefore,
\[
\lambda^\diamond = m + \limsup_{n \to \infty} \frac{1}{n} \ln \sum_{|\alpha|=n, \alpha \in L^\diamond} e^{w(\alpha) - mn}
\leq m + \limsup_{n \to \infty} \frac{1}{n} \ln (1 + \epsilon)^n \epsilon e^{n(\epsilon + \lambda - m)}
= m + \ln(1 + \epsilon) + \epsilon + \lambda - m
= \ln(1 + \epsilon) + \epsilon + \lambda.
\]

Case 2. $L$ is a finite language and hence $\lambda = -\infty$. Noticing that we do not have any long blocks when $N_\epsilon$ is big enough. In this case, we estimate the following term instead
\[
\sum_{|\alpha|=n, \alpha \in L^\diamond} e^{w(\alpha)}
\]
\[
\sum_{0 \leq k \leq n,l_1 + \cdots + l_{k+1} = n-k,l_i > 0} E^{w(L_{i_1})} \times \hat{\epsilon} \times E^{w(L_{i_2})} \times \hat{\epsilon} \times \ldots E^{w(L_{i_{k+1}})}
\]

(because all blocks are short, we have the following, using (19))

\[
\leq \sum_{0 \leq k \leq n,l_1 + \cdots + l_{k+1} = n-k,l_i > 0} \epsilon^n \epsilon^{k+1}
\]

\[
\leq \sum_{0 \leq k \leq n,l_1 + \cdots + l_{k+1} = n-k,l_i > 0} \epsilon^n \epsilon^{k+1}
\]

\[
= \sum_{k=0}^{n} \binom{n}{k} \epsilon^k \epsilon^n
\]

\[
= (\epsilon(1 + \epsilon))^n.
\]

In this case, therefore,

\[
\lambda_\Diamond = \limsup_{n \to \infty} \frac{1}{n} \ln \sum_{|\alpha| = n, \alpha \in L^{\Diamond}} e^{w(\alpha)}
\]

\[
\leq \limsup_{n \to \infty} \frac{1}{n} \ln (\epsilon(1 + \epsilon))^n
\]

\[
= \ln \epsilon(1 + \epsilon).
\]

In Case 1, \(\lambda_\Diamond \to \lambda\) as \(\epsilon \to 0\) (and hence \(w(\Diamond) \to -\infty\)). In Case 2, \(\lambda_\Diamond \to \lambda = -\infty\) as \(\epsilon \to 0\) (and, also, hence \(w(\Diamond) \to -\infty\)). That is, the claim in (4) is valid since \(\lambda_\Diamond\) is monotonic in \(w(\Diamond)\) according to the definition in (6).

The Gurevich’s Algorithm (11) is known efficient, numerically. However, the translating from a regular language to a DFA is not when we use textbook algorithms [10]. In fact, many online tools are available for the translation and in many practical cases, it does not seem terribly inefficient. Of course, it is meaningful future research to, efficiently, construct a random walk program directly from an NFA accepting the \(L\) while achieving a reasonably good energy rate that is close to the \(\lambda\).

4 AEP for a (Weighted) Regular Language

Asymptotic Equipartition Property (AEP) is defined in Shannon’s information theory to say that sequences generated from certain random sources, with probability approaching to 1, share roughly the same information rate (that is the rate of the source). In literature, those sequences whose rate is close to the rate of the source are called typical sequences. In this section, we generalize the concept to a weighted regular language while the source is the Markov chain \(\hat{M}\) constructed from the given regular language \(L\) in the previous section. Because the \(\hat{M}\) defines the unique \(\mu^*\) in the Variational Principle, we can then define typical words in a weighted regular language. Surprisingly, among all sequences, only a small
number of sequences are typical even though they will show up with probability close to 1. This also says that, if we understand that a finite automaton is a probabilistic device to generate words in a regular language by assigning transition probabilities on the state transitions and that the probability assignments actually make the device achieve Gurevich free energy, then, with probability asymptotically 1, the device will only generate a “small” number of words, and each such word is typical. If we understand that the regular language \( L \) is used to specify a person’s normal behavior, then typical behavior only takes a small portion of the \( L \) — most behavior in \( L \) is not typical. Identifying such typical words has a lot of immediate applications in computer science, such as the ones listed below.

– In software testing, we may use the above idea to build two kinds of test suites: typical test suite and non-typical test suite. Typical test suite is used to verify the programs implement basic and commonly used functionalities correctly. Non-typical test suite creates challenges testing cases to check whether programs work correctly in the extreme cases.

– In computer security, the typical words concept are similar to the Pareto principle (a.k.a. 80-20 rule). Most behaviors are normal behaviors and the types of normal behaviors only take a small portion of all behaviors types. While the abnormal behaviors occur rarely, the number of types regarding abnormal behaviors is much larger than that of normal ones. Thus, we may use typical words to improve existing intrusion detection systems.

We start with claiming AEP on aperiodic (i.e., there are two (nested) loops at the same node who lengths are co-prime) and strongly connected graphs. Then, we apply the claim on weighted regular languages.

### 4.1 AEP on a weighted, aperiodic and strongly connected graph

Let \( G \) be a weighted graph. We assume that \( G \) is aperiodic and is an SCC. Let Gurevich Markov chain \( \mu^* \) obtained from running Algorithm 1 on the \( G \) be \( p(\cdot) \), where for each edge \( t \) in \( G \), \( p(t) \) is the transition probability of \( t \), and \( w(t) \) is the given weight associated with \( t \). Since the obtained \( p(\cdot) \) is ergodic, we assume that \( \pi(\cdot) \) is the unique stationary distribution where \( \pi(v) \) is the stationary probability on node \( v \) in \( G \), for each node \( v \). We shall use \( \pi(t) \) to denote \( \pi(v) \) where \( v \) is the starting node of the edge \( t \). A walk \( \alpha \) of \( G \) is a sequence of edges

\[
\alpha = t_1 \cdots t_n
\]

for some \( n \) such that the ending node of \( t_i \) equals the starting node of \( t_{i+1} \) for each \( i \). For the walk \( \alpha \), we use \( P(\alpha) \) to denote the probability that \( \alpha \) is actually walked; i.e., \( P(\alpha) = \pi(t_1)p(t_1)\cdots p(t_n) \).

Let \( \epsilon > 0 \). We say that the walk \( \alpha \) in (11) is \( \epsilon \)-typical (of \( G \)), if

\[
|\lambda_\alpha - \lambda_G| \leq \epsilon,
\]
where $\lambda_G$ (recalling that $\lambda_G$ is the $\ln \hat{\lambda}$ where $\hat{\lambda}$ is the Perron number computed in Algorithm 1) is the free energy rate of $G$, and $\lambda_\alpha$ is the free energy rate of the walk $\alpha$; i.e.,

$$
\lambda_\alpha = \frac{\sum_{i=1}^{n} w(t_i) - \ln p(t_i)}{|\alpha|}.
$$

We define $T^n_{\epsilon,G}$ to be set of $\epsilon$-typical walks in $G$ with length $n$. Then, we claim that

$$
\lim_{n \to \infty} \sum_{\alpha \in T^n_{\epsilon,G}} P(\alpha) = 1,
$$

where $p$ is defined above from the Gurevich Markov chain on $G$. Its proof is as follows.

**Proof.** Notice that the Gurevich Markov chain $p(t)$ where $t$ is an edge in $G$, is ergodic, and hence, for any fixed $n$, we use $X_n$ to denote a random variable over all walks $t_1t_2 \ldots t_n$ on $G$ with length $n$. Then,

$$
\mathbb{E}\{\lambda_{X_n}\} = \sum_{t_1 \ldots t_n \text{ is a walk}} \frac{\sum_{i=1}^{n} w(t_i) - \ln p(t_i)}{n} P(t_1 \ldots t_n)
$$

$$
= \frac{1}{n} \sum_{t_1 \ldots t_n \text{ is a walk}} \sum_{i=1}^{n} [w(t_i) - \ln p(t_i)] \pi(t_1)p(t_1) \ldots p(t_n)
$$

$$
= \frac{1}{n} \sum_{t_1 \ldots t_n \text{ is a walk}} [w(t_1) - \ln p(t_1)] \pi(t_1)p(t_1) \ldots p(t_n) + \cdots
$$

$$
= \frac{1}{n} \sum_{t_1 \ldots t_n \text{ is a walk}} [w(t_1) - \ln p(t_1)] \pi(t_1)p(t_1) \ldots p(t_n)
$$

$$
+ \frac{1}{n} \sum_{t_1 \ldots t_n \text{ is a walk}} [w(t_2) - \ln p(t_2)] \pi(t_1)p(t_1) \ldots p(t_n)
$$

(Using the fact that $\pi(\cdot)$ is the stationary distribution)

$$
= \frac{1}{n} \sum_{t \text{ is an edge}} [w(t) - \ln p(t)] \pi(t)p(t) + \cdots
$$

$$
+ \frac{1}{n} \sum_{t \text{ is an edge}} [w(t) - \ln p(t)] \pi(t)p(t)
$$

$$
= \sum_{t \text{ is an edge}} w(t) \pi(t)p(t) - \pi(t)p(t) \ln p(t)
$$

15
(Using the RHS of the Variational Principle)

\[ = \lambda_G \]

Hence, from the law of large numbers for ergodic Markov chain, we have

\[ \lim_{n \to \infty} \text{Prob}\{|\lambda_{X_n} - \lambda_G| \leq \epsilon\} = 1. \]

The claim follows.

### 4.2 AEP for a weighted regular language

Let \( L \) be a weighted regular language. In the construction presented in Section 3, the strongly connected graph \( G_M \) is run on Gurevich’s Algorithm. However, in order to use the AEP results in Section 4.1, the graph must be aperiodic. This can be easily resolved as follows.

We now consider a new language \( \tilde{L} = L(\heartsuit \spadesuit + \diamondsuit) \) where \( \heartsuit, \spadesuit, \diamondsuit \) are new symbols with weight 0. Suppose that \( M \) is the minimal DFA accepting \( L \). We now construct \( \tilde{M} \) by modifying the \( M \) as follows

- Add a new state named \( F \);
- For each accepting state \( q \) of \( M \), add a state \( \spadesuit q \) as well as three transitions \( q \rightarrow \spadesuit q, \spadesuit q \rightarrow F, q \rightarrow F \);
- Make \( F \) be the only accepting state of \( \tilde{M} \).

In Section 3, the strongly connected graph \( G_M \) is constructed by splitting every transition in \( M \) into two transitions (see Figure 4). We can construct \( G_{\tilde{M}} \) similarly but without splitting all the newly added edges in \( \tilde{M} \). For instance, for the example DFA \( M \) in Figure 1, the \( G_{\tilde{M}} \) is shown in Figure 6, which can be compared with Figure 4 where the difference can be seen.

![Fig. 6. The weighted graph \( G_{\tilde{M}} \) converted from \( M \).](image-url)
Clearly, $G_{\hat{M}}$ is aperiodic.

From $\hat{L}$, the graph $G_{\hat{M}}$ is constructed accordingly. Consequently, as described in Section 3, a Markov chain $\mu^*$ that assigns transition probability $p(t)$ to an edge $t$ in the graph $G_{\hat{M}}$ is computed using Algorithm 1. We now first recall the following the definition in Section 4.1. Let $K$ be a (large) number and $\epsilon > 0$ be a small number. Notice that, herein, the (negative) weight $w(\diamond)$ in $G_{\hat{M}}$ gives $\hat{\epsilon} = e^{w(\diamond)}$. A walk $T$ in $G_{\hat{M}}$ with length $K$ is $(\epsilon, \hat{\epsilon})$-typical if

$$|\lambda_{G_{\hat{M}}} - \frac{1}{K} \sum_{1 \leq i \leq k} w(t_i) - \ln p(t_i)| < \epsilon,$$

where $\lambda_{G_{\hat{M}}}$ the free energy rate of the graph $G_{\hat{M}}$. Though $G_{\hat{M}}$ is a graph (without “initial” and “accepting” nodes), the initial/accepting state in $M$ can be uniquely located in $G_{\hat{M}}$ through the construction of $G_{\hat{M}}$. Hence, we can still say that a walk $T$ is initial when it starts with the initial node that corresponds to the initial state of $M$. Since the Markov chain $\mu^*$ computed on the graph $G_{\hat{M}}$ is ergordic, and hence,

$$\lim_{K \to \infty} \text{Prob}\{T : T \text{ is an initial walk with length } K\} = 1. \quad (13)$$

Using the AEP result in Section 4.1, we then have

$$\lim_{K \to \infty} \text{Prob}\{T : T \text{ is an initial and } (\epsilon, \hat{\epsilon})\text{-typical walk with length } K\} = 1. \quad (14)$$

For each initial and $(\epsilon, \hat{\epsilon})$-typical walk $T$, one can uniquely recover, by reversing the construction from $M$ all the way to the graph $G_{\hat{M}}$, a word $\sigma(T)$ that is a prefix of a word in $(\hat{L} \diamond)^* \hat{L}$. Noticing that the $\alpha$ is in the form of

$$\alpha_1 \# \cdots \# \alpha_k$$

for some $k$, where each $\alpha_i \in L$ (except for the last $\alpha_k$, see below) and the $\#$ is either $\clubsuit \spadesuit$ or $\heartsuit \diamondsuit$. The last $\alpha_k$, when ended with an auxiliary symbol (like $\spadesuit, \heartsuit$, or $\diamondsuit$), we can simply remove those symbols and obtain a word (also denoted as $a_k$) in $L$. When, however, $\alpha_k$ is not ended with an auxiliary symbol, we may append the $\alpha_k$ with a bounded length word so that the resulting word (still denoted as $\alpha_k$) will be in $L$. In this latter case, one can think the $T$ is appended with a few more edges and the resulting $T$’s typicalness will not be affected if the length $K$ of $T$ is taken large enough.

We shall now say that the *cluster* (a multiset) of words in $L$

$$\{\alpha_1, \cdots, \alpha_k\}$$

obtained from the $T$ is $(\epsilon, \hat{\epsilon}, K)$-typical in $L$ when the $T$ of length $K$ is $(\epsilon, \hat{\epsilon})$-typical. We use the cluster instead of the sequence $\alpha_1, \cdots, \alpha_k$ here since the ordering of the appearance of the $\alpha_i$’s in $\sigma(T)$ has no effect for the typicalness (it is left to the reader to verify).

Surprisingly, computing an $(\epsilon, \hat{\epsilon}, K)$-typical cluster in $L$ is not as obvious as it looks. A direct implementation would include the following steps:
1 Construct the $M$ (minimal DFA for $L$);
2 Exercise all the steps in Section 3 to obtain $G_{\tilde{M}}$;
3 Run Algorithm 1 on $G_{\tilde{M}}$ so that transition probabilities can be assigned to the edges in $G_{\tilde{M}}$ and, at the same time, obtain the free energy rate $\lambda_{G_{\tilde{M}}}$;
4 Find an initial walk $T$ (which may contain loops and nested loops) on the graph $G_{\tilde{M}}$ so that the constraint in (12) is satisfied;
5 We recover and return a cluster constructed from the $T$ as described in above.

Though Step 1 is, in theory, not efficient. However, for practical purpose, as we have mentioned at the end of Section 3, it may not be bad at all using some well established online tools. The main difficulty is, however, Step 4. This is because all the initial walks $T$ that we are looking for form a semilinear language whose Parikh map is not so easy to compute. One possible way is sketched as follows. One can construct a system of integer linear/modular constraints, through loop analysis, to characterize the semilinear set (on edge counts) of all the initial walks on the graph (see, e.g., the paper [20], and then, translate the constraint in (12) into linear constraints on the counts of individual edge counts, and finally, an integer linear constrain solver may be used to obtain a solution (as a vector of counts of edges). One final step would be to recover the vector to a walk $T$. This whole process is terribly inefficient and not practical at all. Unfortunately, this is the best algorithm we can come up with, for now, since the problem in Step 4 is essentially asking an algorithm to decide whether there is a walk on a graph whose vector of edge counts satisfies a given linear constraint system. In [5], we investigated “typicalness” in a setting where we used branching factor of a graph to approximate information rate (instead of energy rate) of the graph, where a model-checker SPIN [9] was used to identify a typical path of a graph. This model-checking approach might serve an alternative way to resolve Step 4, practically.

Luckily, we shall notice that, as shown in (14), for a large $K$, the initial and typical walks take probability asymptotically 1. That is, if we just randomly walk on the Markov chain $\mu^{*}$ assigned on the graph $G_{\tilde{M}}$, we will have probability close to 1 to obtain a desired $T$. Hence, the following random algorithm is a far more practical and efficient solution to Step 4:

4.1 Assign transition probabilities to edges in $G_{\tilde{M}}$ according to the computed Markov chain $\mu^{*}$ in Step 3;
4.2 Treat the graph now as a probabilistic program and by running the program from the initial node, we create a random walk of length $T$;
4.3 If $T$ satisfies (12), return, else goto Step 4.2;

One should be able to obtain the expected number of rounds on the loop between Step 4.2 and Step 4.3, though analyzing the mixing time of the ergordic Markov chain $\mu^{*}$, but we leave this for future work.

4.3 Special cases when a typical cluster is a singleton

One shall notice that, through out the paper, if the minimal DFA $M$ of the regular language $L$ is already strongly connected and aperiodic, we do not need any
\(\diamondsuit/\odot/\heartsuit\)-edges in the \(G_M\) and hence, the \(\heartsuit\)-edges are not needed and consequently, an obtained typical cluster will always contain one word (i.e., the cluster is a singleton); in this special case, we can say that the word is typical in \(L\). Such a regular language would inherently a “looped” language such as \((ab+c)^*\).

There is another special case when \(L\) is prefix-closed that is very useful in practice. In this case, every state in the minimal DFA \(M\) accepting \(L\) is an accepting state. Such a prefix-closed \(L\) can be used to specify applications like network traffic (a prefix of the traffic stream is also a traffic stream).

Recall that, in Section 4.1 the AEP is established on the Markov chain \(G_M\) (with transition probabilities in \(\mu^*\)) constructed from the regular language \(L\). We assume that \(L\) is an infinite language (and therefore its energy rate \(\lambda\) is finite) and hence the energy rate \(\lambda_\delta\) of \(G_M\) approaches the finite number \(\lambda\) as \(w(\diamondsuit) \to -\infty\). In Section 4.2 we use the notation \(\sigma(T)\) to denote the unique word “recovered” from a walk on the graph \(G_M\). The walk \(T\) is called accepting if \(T\) starts with \(q_0\) and ends with the node \(F\), where in between, it never passes the node \(F\). Such an accepting walk \(T\) can also be understood as the following: the Markov chain starts with \(q_0\) and walks along the \(T\). When it ends at the node \(F\), it is also the first hitting time (the length of \(T\)) for the node \(F\). We use \(\tau\) to denote the random variable where \(\text{Prob}\{\tau = n\}\) is the probability that there is an accepting walk \(T\) with length \(n\). There are some existing results on the distribution of \(\tau\). For instance, from Maciuca and Zhu [12], explicit formulas are given (see formulas (2.2) in the paper) on \(E(\tau)\) through the fundamental matrix of the Markov chain. Notice that, herein, the \(\tau\) is for the Markov chain starting from the node \(q_0\) instead of starting from the unique stationary distribution \(\pi(\cdot)\). The \(\tau\) can be approximated well with an exponential distribution (Prop 3.7 in [12]). Hence, with high probability, the \(\tau\) can not be too bigger than the mean \(E(\tau)\). Even though it is still difficult to estimate the value of \(E(\tau)\), we can have a clue of it from the well known result: when the Markov chain starts from the unique stationary distribution \(\pi(\cdot)\), \(E(\tau) = \frac{1}{\pi(F)}\).

Furthermore, \(|\pi(F) \cdot w(\diamondsuit)| < c\) for some constant \(c\) when \(w(\diamondsuit) \to -\infty\). (This can be shown by the fact \(\lambda_\delta \to \lambda > -\infty\) as \(w(\diamondsuit) \to -\infty\) and by the ergodicity theorem of the Markov chain; we omit the details.) Therefore, \(E(\tau)\) should be at the same magnitude of \(-w(\diamondsuit)\) (which is big according to our choice of \(w(\diamondsuit)\)). Therefore, for a large \(N\) (but less than \(O(-w(\diamondsuit))\)), the set \(S_N\) of all initial (i.e., starting from \(q_0\)) walks \(T\) (with length \(N\)) that is a prefix of an accepting walk should take a high probability (among all initial walks with length \(N\)) since, as we have said, the probability that \(N < \tau\) is high. Notice that all initial walks with length \(N\) are with probability close to 1 (when \(N\) is large), and also from the AEP in Section 4.1 we shall expect the following: all \((\epsilon, \cdot, N)\)-typical walks \(T\) in the \(S_N\) take a high probability. Notice that since \(L\) is prefix closed, each such typical \(T\) uniquely corresponds to a word in \(L\) and hence each such word can be called typical. Of course, the above analysis is very informal since it is very difficult to make a precise estimate on the hitting times. But, we think this shall good enough for a practitioner to successively (with high probability — that is \(1-\delta\) where the small \(\delta\) depends on \(\epsilon, w(\diamondsuit)\) but not on \(N\), and hence it is merely an
approximation of the AEP) generate a long typical word from the prefix-closed $L$ as follows: (on given $\epsilon, \hat{\epsilon} = e^{w(\hat{\epsilon})}, N$)

Step 1. Construct the Markov chain $G_{\hat{M}}$ (with transition probabilities in $\mu^*$), from the minimal automaton $M$ for the $L$;

Step 2. Treat $G_{\hat{M}}$ as a probabilistic program and starts to walk from the node $q_0$;

Step 3. When the program hits state $F$, goto Step 2. When the program hits the length $N$, if the walk so far is indeed $(\epsilon, \hat{\epsilon}, N)$-typical (using its definition in Section 4.1), then return the word (in $L$) recovered from the walk as a typical word, else goto Step 2.

4.4 Discussions on a typical cluster in a weighted regular language

What are the practical implications for a typical cluster? One can interpret it in the following way. Suppose that we are allowed to pick words from a bag $L$ of words. This bag can be finite or infinite. It is required to pick words whose total length is (roughly) $K$ that is a large given number. What would be the most typical way to pick? A typical cluster obtained in Section 4.2 fits this purpose.

One can also re-interpret the cluster under various context. For instance, given a software under test, one may generate a test suite (a set of test cases, where each test case is simply an input sequence) from a given requirements specification which can be a regular language (on input/output events) or a finite automaton to accept the language. Then, what would be a typical test suite? (and according, what would be a non-typical test suite?) Notice that weights can be naturally assigned with each event to indicate for instance the cost of running a test on the event. A typical cluster obtained in Section 4.2 may also serve this purpose. One other example is abnormal detection in computer security or in social security. Suppose that $A$ is a device (such as a network, a medical monitor, etc.,) or a malicious person that we already have a specification of $A$’s “normal” behavior. Such a specification can be simply a finite automaton or a regular language, to define normal or legal sequences of activities. Each activity can itself associated with some physical quantity such as time, money spent, or even risk level. However, a difficulty in computer security is to identify abnormal in normal. This is because an abnormal behavior can be completely legal but sometimes, it is a prelude to a terrible attack. (For instance, a person circulating a police office three times a day is legal but abnormal.) It is often the case that a set of behaviors (i.e., sequences of activities) are obtained through, for instance, probing a network traffic, or targeted surveillance. Consequently, we may apply the algorithms in Section 4.2 to decide whether such a set is typical or non-typical, which has great importance in practice.

Finally, we would like to add a few words on the abnormal detection in normal mentioned earlier. In reality, it is very difficult to give a specification for normal; in many cases the specification in partial, error-prone, or even missing. In the case that without a specification, can we still detect normal/abnormal? In other words, can we solve the following problem:
Given a number of large finite sets $S_1, \ldots, S_k$ (some or all of the sets can be singletons, see Section 4.3) of weighted activity sequences (on a known weighted activity alphabet $\Sigma$);

Goal: identify those sets that are typical/non-typical (i.e., normal/abnormal).

Notice that, herein, there is no automaton or regular language given. Our solutions are as follows.

Step a. For each set $S_i$, every sequence in $S_i$ obtain a random but unique id $\eta_{ij}$ in $[1, |S_i|]$. Now, we use the $\beta_{ij}$ to represent the sequence whose id is $\eta_{ij}$. Notice that each activity symbol in $\beta_{ij}$ is weighted. Then, we concatenate all the sequences $\beta_{ij}$ to a new long sequence $\beta_i$. (For example, suppose that the set $S_1$ has three sequence $abc$, $def$, and $hij$. Let $\beta_{11} = def$, $\beta_{12} = hij$ and $\beta_{13} = abc$. Then, $\beta_1 = defhijabc$)

Step b. For each $\beta_i$, for every symbol $a$, we count its occurrences in $\beta_i$, $\#_{\beta_i}(a)$. Then, estimate the probability of symbol $a$, i.e., $Pr(a) = \frac{\#_{\beta_i}(a)}{|\beta_i|}$. So, we have $\int wd\mu \approx \sum_a \frac{\#_{\beta_i}(a)}{|\beta_i|}$.

Step c. Run Lempel-Ziv compression algorithm (such as the one implemented in 7z [1]) on each $\beta_i$ so that we obtain the reciprocal of the compression ratio which is an estimation of the entropy $H_{\beta_i}$ (converted in natural logarithm). Then, we have $\lambda_i = \sum_{a \in \Sigma_{\beta_i}} \frac{\#_{\beta_i}(a)}{|\beta_i|} + H_{\beta_i}$;

Step d. Compute $\lambda = \frac{1}{k} \sum \lambda_i;

Step d. For the given $\epsilon$ and for each $i$, if $|\lambda - \lambda_i| < \epsilon$, report that $S_i$ is typical/normal, else report that $S_i$ is non-typical/abnormal.

Notice that the reason why this approach works is due to the fact that Lempel-Ziv can be used to approach the free energy rate after the transcoding in Step a is applied. However, it does need a side condition (for Lempel-Ziv algorithm): the source that generate the sequences must be stationary and ergodic, which is the same as the constraints that our graph $G_{M'}$ satisfies. We leave this in the journal version of the paper for a more detailed presentation.

Of course, the nontypical/abnormal detection algorithms we presented so far on the case that the language or the automaton is presented and on the case that the language or the automaton is missing also give a way to clustering weighted stream data with a model or without a model (unsupervised). Currently, our PhD student William Hutton is conducting experiments on using the algorithms to detect abnormal TCP/IP handshaking traffic for the case when a model is present and the case when the model is not available. In the future, we would like to generalize the experiments to a broader setting so that more practical effectiveness results can be obtained. For the completeness of this paper, we present a small but real-word example to see how one would use the algorithms in software testing.

5 Applications: typical and non-typical test cases

Model-based software testing generates test codes from a labeled transition system, served as the model of the system under test. ModelJunit [14] is one such
testing tool in which one can specify a labeled transition system and the tool may also produce test cases automatically. We now look at an example, which is taken from an online presentation [16].

This transition system is intended to model an implementation of a simple text box where a user can fill in “Hello” or clear the text in the box. A test sequence is a sequence of events on the user inputs, which is the sequence of labels on a walk on the graph shown above. To ease our presentation, we take the state “sleep” as the accepting state. (So, we consider a complete cycle of the implementation.) We call the transition system as the DFA $M$. Notice that the DFA is already minimal, strongly connected and aperiodic. Hence, following the discussions at the beginning of Section 4.3 we can directly compute the transition probabilities, using the Markov chain that achieves the free energy rate of $L(M)$, on transitions in $M$ without introducing any $\hat{\epsilon}$-edges, as shown in the following figure, which is a probabilistic program denoted as $\hat{M}$ (Here, we implicitly assume that the input labels share the same weight (i.e., $= 1$).):

From this, we can see the test cases can be generated automatically when the $\hat{M}$ is run as a probabilistic program.
Notice that, not all test cases are born equal. From the AEP theorem, the program, asymptotically with probability 1, will generate a test case that is always typical. However, there are indeed nontypical test cases, which are also valuable (i.e., those test cases may reveal a fault that is not possible to detect when the system under test is exercised “normally”). Therefore, a “good” test suite shall not only include typical test cases, but also include nontypical test cases. We now look at the following example test suite. \( T_1 \) is the set of the following four test cases.

\[
\begin{align*}
T_1 &= \text{StartHelloExit}, \\
T_2 &= \text{StartHelloClearExit}, \\
T_3 &= \text{StartClearExit}, \\
T_4 &= \text{StartHelloHelloExit}.
\end{align*}
\]

Notice that, by walking each test case on the graph, every branch (transition) is exercised at least one. Hence, the suite \( T_1 \) achieves 100\% branch-coverage.

We now take a small \( \epsilon = 0.1 \) and verify that \( T_1, T_2, T_3, T_4 \) are all \( \epsilon \)-typical. Next, we define the following suite \( T_2 \) of four test cases:

\[
\begin{align*}
T_{1}' &= \text{StartHelloExit}, \\
T_{2}' &= \text{StartExitStartExitStartExit}, \\
T_{3}' &= \text{StartHelloHelloHelloHelloExit}, \\
T_{4}' &= \text{StartHelloClearClearExit}.
\end{align*}
\]

Again, we can compute that \( T_{1}', T_{3}', T_{4}' \) are \( \epsilon \)-typical but, \( T_{2}' \) is not \( \epsilon \)-typical. Notice that \( T_2 \) also achieves 100\% branch coverage. That is, \( T_2 \) may have a better chance to find “corner faults”. Therefore, our approach can also be used to evaluate an existing test suite and see if it contains a reasonable portion of typical test cases. We shall also point out that the discussions made in Section 4.4 can also be applied to identifying typical/non-typical test cases in a given test suite even when the model is not given.

Finally, we shall point out that when the weights assigned to the input labels are changed, so are the test cases’ typicalness (since the free energy rate and the Markov chain are accordingly changed). For instance, if we assign \( w(\text{Clear}) = 4, w(\text{Hello}) = 5, w(\text{Start}) = 1, w(\text{Exit}) = 2 \) (the weight of an input label can be used to measure, e.g., the cost associated with the run of the system under test when the input label is executed.) and \( \epsilon \) is the same as above, then we can verify that the test case \( T_{3}' \) becomes typical.

6 Conclusions

We study random words in a weighted regular language that achieve the maximal free energy using thermodynamics formalism. In particular, typical words in the language are algorithmically generated which have applications in computer security (anomaly detection) and software engineering (test case generation). In the future, we may continue the effort to apply our approaches to larger scale real world applications.
References

1. 7zip. [http://www.7-zip.org/]
2. David Aldous and Jim Fill. Reversible markov chains and random walks on graphs, 2002.
3. Nicolas Basset. A maximal entropy stochastic process for a timed automaton. *Information and Computation*, 243:50–74, 2015.
4. Cewei Cui and Zhe Dang. A free energy foundation of semantic similarity in automata and languages. In *Similarity Search and Applications - 9th International Conference, SISAP 2016, Tokyo, Japan, October 24-26, 2016. Proceedings*, pages 34–47, 2016.
5. Cewei Cui, Zhe Dang, and Thomas R. Fischer. Typical paths of a graph. *Fundam. Inform.*, 110(1-4):95–109, 2011.
6. Cewei Cui, Zhe Dang, Thomas R. Fischer, and Oscar H. Ibarra. Execution information rate for some classes of automata. *Information and Computation*, 246:20–29, 2016.
7. Cewei Cui, Zhe Dang, Thomas R. Fischer, and Oscar H. Ibarra. Information rate of some classes of non-regular languages: An automata-theoretic approach. *Information and Computation*, (to appear).
8. B. M. Gurevich. A variational characterization of one-dimensional countable state gibbs random fields. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 68(2):205–242, 1984.
9. Gerard J. Holzmann. The model checker spin. *IEEE Transactions on software engineering*, 23(5):279–295, 1997.
10. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.
11. Laszlo Lovasz. Random walks on graphs: A survey. *Combinatorics, Paul Erdos in Eighty*, 2, 1993.
12. Romeo Maciuca and Song-Chun Zhu. First hitting time analysis of the independence metropolis sampler. *Journal of Theoretical Probability*, 19(1):235–261, 2006.
13. Burton Gordon Malkiel. *A random walk down Wall Street: including a life-cycle guide to personal investing*. WW Norton & Company, 1999.
14. ModelJunit. [https://sourceforge.net/projects/modeljunit/]
15. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.
16. Kyra Hameleers Richid Kherrazi and Adrian Iankov. Hands-on experience model based testing with spec explorer. [https://www.slideshare.net/Rachid99/handson-experience-model-based-testing-with-spec-explorer]
17. D. Ruelle. *Thermodynamic Formalism: The Mathematical Structure of Equilibrium Statistical Mechanics*. Cambridge Mathematical Library. Cambridge University Press, 2004.
18. Omri M. Sarig. Lecture notes on thermodynamic formalism for topological markov shifts, 2009.
19. C. E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, 1949.
20. Gaoyan Xie, Cheng Li, and Zhe Dang. Linear reachability problems and minimal solutions to linear diophantine equation systems. *Theor. Comput. Sci.*, 328(1-2):203–219, 2004.