Labelled Control Improvisation

Eric Vin, Daniel J. Fremont

evin@ucsc.edu, dfremont@ucsc.edu

Department of Computer Science and Engineering
University of California, Santa Cruz

June 2021

Abstract

The classic Control Improvisation problem allows one to extend traditional synthesis problems, by balancing control over what is generated, with how randomly it is generated. In this paper, we propose Labelled Control Improvisation, an extension of classic Control Improvisation that allows for greater control over randomness via the addition of a labelling specification. By assigning one and only one label to every improvisation, we avoid the immediate exponential time complexity of multiple soft constraints, while still allowing for more fine tuned control. We present two different problems, the simpler Labelled Control Improvisation problem, in which explicit randomness bounds are provided, followed by the more natural Maximum Entropy Labelled Control Improvisation problem, which forgoes some of the explicit constraints and instead seeks to generate improvisations in a way that maximizes entropy. We also present several motivating examples that illustrate the usefulness of Labelled Control Improvisation, and provide upper and lower bounds on the complexity of these problems when using a variety of constraint and labelling specifications.

(This document consists of Eric Vin's senior thesis for the Computer Science B.S. at UCSC, with this title page added for publication as a technical report.)

Technical Report # UCSC-SOE-21-09

Labelled Control Improvisation

Eric Vin

evin@ucsc.edu

Department of Computer Science and Engineering
University of California, Santa Cruz

June 2021

A thesis submitted to the faculty of

The University of California, Santa Cruz

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Reading Committee:

Professor Daniel Fremont (Chair)

Professor Seshadhri Comandur

Contents

Abstract

The classic Control Improvisation problem allows one to extend traditional synthesis problems, by balancing control over what is generated, with how randomly it is generated. In this paper, we propose Labelled Control Improvisation, an extension of classic Control Improvisation that allows for greater control over randomness via the addition of a labelling specification. By assigning one and only one label to every improvisation, we avoid the immediate exponential time complexity of multiple soft constraints, while still allowing for more fine tuned control. We present two different problems, the simpler Labelled Control Improvisation problem, in which explicit randomness bounds are provided, followed by the more natural Maximum Entropy Labelled Control Improvisation problem, which forgoes some of the explicit constraints and instead seeks to generate improvisations in a way that maximizes entropy. We also present several motivating examples that illustrate the usefulness of Labelled Control Improvisation, and provide upper and lower bounds on the complexity of these problems when using a variety of constraint and labelling specifications.

Chapter 1

Introduction

In this paper we seek to provide another tool for Algorithmic Improvisation, which is a framework for augmenting a system with randomness while preserving key properties about the system. The core computational problem of Algorithmic Improvisation, Control Improvisation allows one to specify a system that must generate improvisations that meet a hard constraint all the time, a soft constraint within a certain percentage of the time, and be sufficiently random. In this work, we seek to augment classic Control Improvisation (CI) with a labelling specification that assigns one and only label to all improvisations our system generates. We then seek to extend or modify the randomness constraints to apply over labels, instead of merely over all improvisations. This allows us finer control over how exactly we define the randomness requirements in our problem. Without the addition of the soft constraint, this problem could easily be transformed into solving multiple instances of classic CI independently for each set of labelled words. However, our definition includes a soft constraint, like in classic CI, that must be met within a certain tolerance while balancing the randomness requirements of words, which makes for a more complex problem.

We begin by introducing several motivating examples. We then construct a definition for basic Labelled Control Improvisation (LCI), in which one provides marginal and conditional randomness bounds on selecting improvisations with a certain label, along with a soft and hard constraint. Following this, we move on to the more natural Maximum Entropy Labelled Control Improvisation (MELCI), in which the conditional randomness bounds for improvisations are removed and instead the distribution generated is the one that maximizes entropy while meeting all other constraints. We then discuss the time complexities for the LCI and MELCI problems given for different classes of constraint/labelling specifications. Finally, we revisit our motivating examples to provide high level guidance on how one might implement them using the constructions in this paper.

1.1 Related Work

Our work builds off of the work of Fremont et al. [1, 2, 3], which itself generalizes the work of Donzé et al. [4, 5]. More specifically, our work extends the Control Improvisation problem, like the work done by Fremont and Seshia [6] for the reactive case. In addition, our work includes a problem, Maximum Entropy Labelled Control Improvisation, that seeks to calculate an improvising distribution that maximizes entropy, similar to the work done by Vazquez-Chanlatte et al [7] also for the reactive case. In addition, we should note that the LCI problem can be encoded as an instance of the multiple soft constraint control improvisation laid out in [3]. However, with the restriction that an improvisation can have one and only one label, we avoid the exponential runtime encountered in that extension.

1.2 Motivating Examples

In this section we will present several examples in which one could apply classic CI as laid out by Fremont et al. [1, 2, 3]. We will then show how, by labelling our improvisations and enforcing randomness over these labels, we can exert more control over how our improvisations are generated, and thus have an overall more useful improviser.

1.2.1 Robotic Planning

In the original CI robotic planning problem [3, Chapter 6], we seek to find sufficiently random routes for a patrolling robot that meet some mission and safety rules all the time (Hard constraints), along with some efficiency rules that we would like to meet a certain percentage of the time (Soft constraints). Consider a similar robotic planning problem, illustrated in Figure 1.1 in which a robot must patrol a museum with one large main passageway and two smaller side entrances, and ideally catch any thieves before they can make off with one of the exhibits. Our goal is to generate paths through the museum, which take the form of a sequence of moves up, down, left, right, or stop and meet several conditions while being sufficiently random. We assume that the robot can only move through from square to square up, down, left, and right (i.e. no diagonal movement). Impassable walls are indicated by black squares, the 4 objectives, in this case the exhibits, via the $O\#$ notating, and the three main entrances connecting the upper and lower part of the board via the $E\#$ notation.

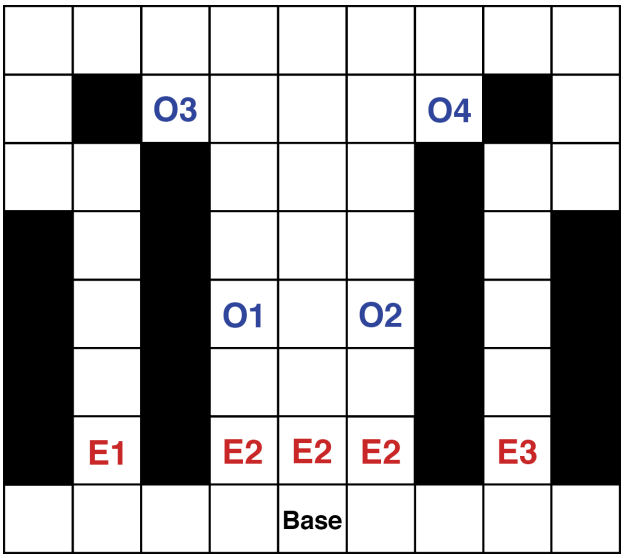

Figure 1.1: Robotic Planning Example

As in the original robotic planning problem, we would like the robot to fulfill several hard and soft conditions:

Hard Conditions:

1. The robot must respect the rules of the terrain (i.e. no entering black tiles or moving off the map)
2. The robot must visit all objectives at least once
3. The robot must start and end at the home base (to maintain consistency between patrols)
4. The path length can be at most 100 tiles (max battery life)

Soft Conditions:

1. The robot's patrol should be less than or equal to 30 tiles at least 75% of the time (so we minimize charging time, and in turn extended periods of downtime that a thief could exploit)

Randomness Conditions:

1. The probability of the robot taking any path meeting the requirements is between some lower and upper randomness bounds, ρ and λ (so the thief has fairly limited information about where our robot will be at any given time)

This provides a patrolling schedule that works fairly well, with one main problem. Potential paths that enter and exit through the wider main entrance, $E2$, tend to be shorter and more numerous compared to the long and narrow side entrances, $E1$ and $E3$. As we only enforce randomness over possible paths and our soft constraint rewards shorter paths,

this means that our improviser will give our robot a relatively low chance of patrolling the side entrances, which a thief could potentially exploit. Ideally, we would want the thief to have limited knowledge of which entrance the robot would enter or leave the building from, by bounding the chances of each entrance being used to a fairly uniform spectrum. This way, it is more difficult for them to plan their entrance/egress without running into the robot in the hallway.

A much more comprehensive solution is one in which we label paths by which entrance they take into the building, and optionally which one they take to leave, for a total of 3 or 9 labels, and then enforce randomness over these as well.

For example, one of the labels could be $(E1, E3)$, indicating the robot entered the building via the $E1$ entrance, patrolled over all objectives, and left via the $E3$ entrance back to its charging station. For our randomness bounds over these labels, we could be as strict as possible and require that the probability of a generated path being in a certain label must be exactly $\frac{1}{9}$ (by setting the lower and upper bound to $\frac{1}{9}$). However,this may render our soft constraint infeasible. If this is the case, we could relax these bounds by requiring that the probability of a generated path being in a certain label be between $\frac{1}{7}$ and $\frac{1}{11}$, which gives our improviser more freedom which it could perhaps use to satisfy the soft constraint.

In this way, we can make sure we don't violate our soft constraint, while balancing picking a random path with other situation specific requirements, in this case the entryway used. If our parameters are infeasible, we would like to know that so we can decide whether to reformulate our problem, give up, or relax our constraints. Labelled Control Improvisation will allow us to do just that.

1.2.2 Fuzz Testing

Another example in which Labelled Control Improvisation provides significantly improved usefulness is fuzz testing. In this example, we will consider fuzz testing a compiler, or really any program, whose input can be described by a unambiguous context free grammar. Our goal is to find inputs in this grammar that cause our compiler to hang indefinitely or crash. To do this, we are interested in generating both inputs that are syntactically correct and inputs that are not syntactically correct but may be close. This allows us to try to maximize the strings that invoke the interior functionality of our program, while also checking to see if undefined inputs are handled gracefully. We also want our inputs to remain relatively short most of the time, so that we can test quickly and find relatively simple failure cases that are easy to understand. Consider the following constraints:

Hard Conditions:

1. All generated inputs must conform to a UCFG composed of all valid inputs and expanded slightly to include some invalid inputs.
2. All generated inputs must be no more than 100 tokens

Soft Conditions:

1. The generated input must be under 50 tokens, 75% of the time.

Randomness Conditions:

1. The probability of generating any input is between some lower and upper randomness bounds, ρ and λ (so that we can ensure we are not neglecting any portion of the space too much)

This would provide a satisfactory generator for a fuzz tester, but it would be useful to have more fine grained control. Say for example that after performing some analysis of our grammar, we become aware of several relatively uncommon syntactical element that trigger complicated functions inside our compiler. It is in our interest to ensure we spend enough time testing the functionality associated with these more complicated, and thus more likely to contain an error, functions. To ensure, this, we could provide a labelling function that partitions our overall grammar into classes, perhaps by which of these syntactical elements they contain. We can then enforce randomness bounds on these labels. This way we can ensure we are spending an appropriate amount of time on the inputs that are more likely to invoke a crash, while ensuring we don't neglect any part of our space and that our inputs are, as a whole, relatively short and simple.

1.3 Classic Control Improvisation

We will now formally introduce the problem we tend to extend in this paper, Control Improvisation. To differentiate it from the two extensions we will introduce, we will refer to it as classic Control Improvisation, or classic CI. In this paper, we will use the definitions laid out by Fremont [3, Section 3.1]. Let a hard specification, $\mathcal{H}$, encode a constraint that we must always satisfy. Let a soft specification, $\mathcal{S}$, encode a constraint that we must satisfy within a certain error tolerance. We can define the I and A sets below.

Definition 1.3.1. Fix a finite alphabet Σ, a hard specification $\mathcal{H}$ over that alphabet, and length bounds $m, n \in \mathbb{N}$. An improvisation is any word $w \in \mathcal{L}(\mathcal{H})$ such that $m \leq |w| \leq n$. Let I denote the set of all improvisations.

Definition 1.3.2. Fix a soft specification $\mathcal{S}$ and an error probability $\epsilon \in [0, 1]$. An improvisation $w \in I$ is admissible if $w \in \mathcal{L}(\mathcal{S})$. Let A denote the the set of all admissible improvisations.

Following from these definitions, we can define a Control Improvisation instance, an improvising distribution, and the Control Improvisation problem.

Definition 1.3.3. (Control Improvisation Instance) In general, we will use the symbol $\mathcal{C}$ to denote an instance of CI, whether it be classic CI, LCI, or MELCI. An instance of classic CI is defined as a tuple, $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, \epsilon, \lambda, \rho)$. Let $\mathcal{H}$, $\mathcal{S}$, m, n, and ϵ be defined as above. Let $\lambda, \rho \in \mathbb{Q}$ such that $0 \leq \lambda \leq \rho \leq 1$.

Definition 1.3.4. (Control Improvisation Improvising Distribution) Given a CI instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, m, n, \epsilon, \lambda, \rho)$, a distribution $D : \Sigma^* \to [0, 1]$ is an improvising distribution if it satisfies all of the following constraints:

1. **Hard Constraint:** $\Pr[w \in I \mid w \leftarrow D] = 1$
2. **Soft Constraint:** $\Pr[w \in A \mid w \leftarrow D] \geq 1 - \epsilon$
3. **Randomness Constraint:** $\forall w \in I,\ \lambda \leq D(w) \leq \rho$

Definition 1.3.5. (Control Improvisation Problem) Fix a CI instance $\mathcal{C}$. We say $\mathcal{C}$ is feasible if an improvising distribution exists. An improviser for $\mathcal{C}$ is a probabilistic algorithm, taking no input and with finite expected runtime, whose output distribution is an improvising distribution.

The Control Improvisation problem is to determine if $\mathcal{C}$ is feasible, and if so to generate an improviser for $\mathcal{C}$. For this problem, we encode m and n in unary and encode λ, ρ, and ϵ in binary.

Throughout this paper we will expand and modify the above definitions into the LCI and MELCI problems.

Chapter 2

Labelled Control Improvisation

We begin the contributions by introducing the Labelled Control Improvisation problem. This problem is the simplest to implement, and the one that gives the user maximum control over the resulting distribution. However, this control comes with a trade off, ease of use. In Labelled Control Improvisation one must specify acceptable bounds on randomness for all labels and for words in each label class. This provides an excellent starting point for tackling Labelled Control Improvisation in general, but we will later address a more natural formulation of the problem in Section 3, the Maximum Entropy Labelled Control Improvisation problem.

2.1 Definition

In this section we establish the definitions for the Labelled Control Improvisation problem. In general, one should assume the definitions start from those laid out in Section 1.3. Like before we begin with the definition of an LCI instance.

Definition 2.1.1. (LCI Instance) Let $\mathcal{C} = (\mathcal{H}, \mathcal{S}, L, m, n, \epsilon, \lambda, \rho, \hat{\alpha}, \hat{\beta})$ be a Labelled Control Improvisation, or LCI, instance.

$\mathcal{H}, \mathcal{S}, L, m, n, \epsilon, \lambda$, and ρ are defined as in classic CI. Define L as the encoding of a labelling function $L : \Sigma^* \to \Omega$, where Ω is the corresponding set of labels. Let $\hat{\alpha}$ and $\hat{\beta}$ be two lists of length $|\Omega|$, containing values in $[0, 1] \cap \mathbb{Q}$ where $\forall i \in \{1, \ldots, |\Omega|\},\ \hat{\alpha}_i \leq \hat{\beta}_i$.

For the ith label $\ell_i \in \Omega$, the we can define a set I_i containing all words with the label ℓ_i also accepted by the hard constraint and of the correct length. Formally, $I_i = \{w \in \Sigma^* \mid L(w) = \ell_i\} \cap \mathcal{L}(\mathcal{H}) \cap \Sigma^{m:n}$. Using this, we define $I = \bigcup_{i \in |\Omega|} I_i$. We will refer to each I_i as a label class. As each word in I is assigned one and only one label by L, it follows that the I_i classes partition I.

We can similarly define the A_i classes as $A_i = I_i \cap \mathcal{L}(S)$. Let $A = \bigcup_{i \in |\Omega|} A_i$. Similarly to above, it is also trivial to show that the A_i classes partition A. Also of note and following from the definition is that $A_i \subseteq I_i$.

Using the definition of an LCI instance, we can define an improvising distribution and the LCI problem.

Definition 2.1.2. (LCI Improvising Distribution) Given an LCI instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, L, m, n, \epsilon, \lambda, \rho, \hat{\alpha}, \hat{\beta})$ as described above, a distribution on words $D : \Sigma^* \to [0, 1]$ is an improvising distribution if it satisfies the following constraints,

1. **Hard Constraint:** $\Pr[w \in I \mid w \leftarrow D] = 1$
2. **Soft Constraint:** $\Pr[w \in A \mid w \leftarrow D] \geq 1 - \epsilon$
3. **Randomness over Labels:** $\forall i \in \{1, \ldots, |\Omega|\},\ \lambda \leq \Pr[w \in I_i \mid w \leftarrow D] \leq \rho$
4. **Randomness over Words:** $\forall i \in \{1, \ldots, |\Omega|\},\ \forall y \in I_i,\ \alpha_i \leq \Pr[y = w \mid w \in I_i, w \leftarrow D] \leq \beta_i$

Note that we redefine purpose of the λ and ρ parameters to enforce randomness over the label classes, and we use $\hat{\alpha}$ and $\hat{\beta}$ to enforce randomness within each label class.

For ease of notation we also introduce notation to describe the marginal and conditional probability of a distribution. Let D be an arbitrary distribution over I. $D_M(i)$ is the marginal distribution of selecting a word in I_i and $D_i(w)$ is the conditional probability of selecting w knowing that the word we select will be in I_i.Formally, $D_i(y) = \Pr[y = w \mid w \in I_i, w \leftarrow D]$ and $D_M(i) = \Pr[w \in I_i \mid w \leftarrow D]$. Note that following from the chain rule, for any $w \in I_i$, $D(w) = D_M(i)D_i(w)$

Definition 2.1.3. (LCI Problem) We say that an LCI instance is feasible if there exists an improvising distribution for it. We define an improviser for an LCI instance as a probabilistic algorithm which takes no input and finite expected runtime, whose output distribution is an improvising distribution. Given an LCI instance $\mathcal{C}$, we define the LCI problem as determining if $\mathcal{C}$ is feasible, and if is, generating an improviser for $\mathcal{C}$.

2.2 Necessary and Sufficient Conditions for Labelled Control Improvisation

In this section we seek to establish the intuition for conditions that are necessary and sufficient for an instance of LCI to be solvable. We will then formalize these conditions and prove they are necessary and sufficient.

To begin with, the hard constraint can be trivially satisfied by any distribution over I, and of course can only be satisfied by a distribution over I.

Checking whether or not we can satisfy the λ/ρ randomness constraint is relatively straightforward. We must first check whether there are enough labels, this way no label must necessarily have more than ρ probability of being selected. We also must check that there are not too many labels, such that all labels can have at least λ probability of being selected. Formally, we can write this as $\frac{1}{\rho} \leq |\Omega| \leq \frac{1}{\lambda}$

The $\hat{\alpha}$ and $\hat{\beta}$ randomness constraints are similarly checked. There must be enough words in every I_i, so that we can ensure no word must have more than β conditional probability of being selected. Similarly, there must not be too many words in any I_i, such that all words can be selected with conditional probability at least α. Formally, we can write this as $\forall\ 1 \leq i \leq |\Omega|,\ \frac{1}{\beta} \leq |I_i| \leq \frac{1}{\alpha}$.

The soft constraint is somewhat trickier to reason with. Intuitively however, if the distribution that maximizes the amount of probability given to elements of A without violating the other constraints fails to satisfy the soft constraint, then no other distribution will work either. The optimal way to assign each label distribution D_i is described as the "clamping" method by Fremont [3, p. 28]. First we attempt to assign beta probability β_i to each element of A_i for each i, if we can do so without violating the α_i bound on all the $I_i \backslash A_i$ elements. If we cannot do this, we then assign the minimum α_i to all elements of $I_i \backslash A_i$, and distribute the remaining uniformly over A_i. In this way we maximize the amount each label distribution picks an element in A.

We must also choose a distribution D that maximizes the overall probability of picking an element in A. Here we take the greedy approach of giving as much marginal probability as possible to the label classes that have the highest conditional probability of picking an element in A without violating our randomness requirements. Let $\delta(i)$ be the conditional probability of selecting a word in A_i if we know the word selected is in I_i, or formally $\delta(i) = \sum_{w \in A_i} D_i(w)$. Using our above construction, we set $\delta(i) = 1 - \max(\beta_i|A_i|, 1 - \alpha_i|I_i \backslash A_i|)$. Let H be an ordered list containing all the label indices $\{1, \ldots, |\Omega|\}$, with H_j for any $1 \leq j \leq |\Omega|$ indicating the jth value of H. Let H be ordered in such a way that $\forall\ 1 \leq i < j \leq |\Omega|,\ \delta(H_i) \geq \delta(H_j)$. We also define k as the largest possible number of label distributions that can be assigned ρ probability while still allowing all remaining label distributions to be assigned at least λ probability. We will assign the label classes from index $[1 : k]$ of H marginal probability ρ, and the elements from index $[k + 2 : |\Omega|]$ probability λ, unless $|\Omega| < k + 2$. In making this assignment, there may be a quantity of probability remaining which we will call ν, where $\lambda \leq \nu < \rho$. We assign the element at index $[k + 1]$ to this value ν, unless $|\Omega| < k + 1$. In this way, we maximize the chance of picking an element in A by maximizing the marginal probability of the label classes that have the highest conditional probability of picking strings that are most likely to pick an element of A. It stands to reason that if we cannot satisfy the soft constraint in this way, then no other distribution will satisfy the soft constraint as they will pick elements of A with equal or lower probability. To formalize this, we can write the following inequality, $1 - \epsilon \leq \sum_{i=1}^{k} (\lambda\delta(H_i)) + \nu\delta(H_{k+1}) + \sum_{i=k+2}^{|\Omega|} (\rho\delta(H_i))$.

We now present a formalized version of these conditions, along with a proof that they are necessary and sufficient.

Theorem 2.2.1. *A Labelled Control Improvisation instance is solvable if and only if the following conditions hold,*

1. $\frac{1}{\rho} \leq |\Omega| \leq \frac{1}{\lambda}$

2. $\forall\ 1 \leq i \leq |\Omega|,\ \frac{1}{\beta_i} \leq |I_i| \leq \frac{1}{\alpha_i}$

3. $1 - \epsilon \leq \sum_{i=1}^{k} (\lambda\delta(H_i)) + \nu\delta(H_{k+1}) + \sum_{i=k+2}^{|\Omega|} (\rho\delta(H_i))$

Proof.
(⇒) If the above conditions, hold the labelled control improvisation instance is solvable.

Assume the above conditions hold. We seek to show that there then exists an improvising distribution that satisfies the requirements of the LCI problem. We show this constructively, thereby providing an algorithm which could be encoded in an improviser to define a distribution over all $I_i \backslash A_i$ and A_i classes, from which we would then sample uniformly.

We will first define $D_i(w)$ for each $w \in I_i$ and all i as follows. Let $\epsilon^i_{opt} = \max(1 - \beta_i|A_i|, \alpha_i|I_i \backslash A_i|)$. D_i splits ϵ^i_{opt} probability uniformly over all elements of $I_i \backslash A_i$, and splits $1 - \epsilon^i_{opt}$ probability uniformly over all elements of A_i. Note that by Condition 2, $\alpha_i|I_i \backslash A_i| \leq \alpha_i|I_i| \leq 1$ as $|I_i \backslash A_i| \leq I_i$. This along with the fact that $\alpha_i|I_i \backslash A_i|$ is implicitly greater than 0 and $\epsilon^i_{opt} \geq \alpha_i|I_i \backslash A_i|$ imply $0 \leq \epsilon^i_{opt} \leq 1$.

We will now define $D_M(i)$ for all $i \in \{1, \ldots, |\Omega|\}$. First we will define k, which is the largest possible number of label classes that can be assigned ρ marginal probability while still allowing all remaining label classes to be assigned at least λ marginal probability. Let $k = \left\lfloor \frac{1-|\Omega|\lambda}{\rho-\lambda} \right\rfloor$, unless $\lambda = \rho$ in which case $k = 0$ to simplify our function. We will now define $\delta(i)$ for a label class I_i as $\delta(i) = \sum_{w \in A_i} D_i(w)$. Following from the construction of $D_i(w)$ above, note that $\delta(i) = 1 - \max(1 - \hat{\beta}_i|A_i|, \hat{\alpha}_i|I_i \backslash A_i|)$. Define H as an ordered list containing all the label indices $\{1, \ldots, |\Omega|\}$, with H_j for any $1 \leq j \leq |\Omega|$ indicating the jth value of H. Let H be ordered in such a way that $\forall\ 1 \leq i < j \leq |\Omega|,\ \delta(H_i) \geq \delta(H_j)$, i.e. ordering the indices of the label classes in non-increasing order by their conditional probability of selecting a string in A. For ease of notation, $\forall\ i > |\Omega|,\ \delta(i) = 0$. Finally, define $\nu = 1 - \rho k - \lambda(|\Omega| - k - 1)$ where ν will be the leftover probability after the first k label classes with respect to the ordering of H have been assigned ρ marginal probability and the last $|\Omega| - k - 1$ classes with respect to the ordering of H have been assigned λ marginal probability.

We can then define $D_M(i)$, where i is at index j in H, as follows,

$$D_M(i) = D_M(H_j) = \begin{cases} \rho & j \leq k \\ \nu & j = k+1 \\ \lambda & j > k+1 \end{cases}$$

We illustrate this process in Figure 2.1.

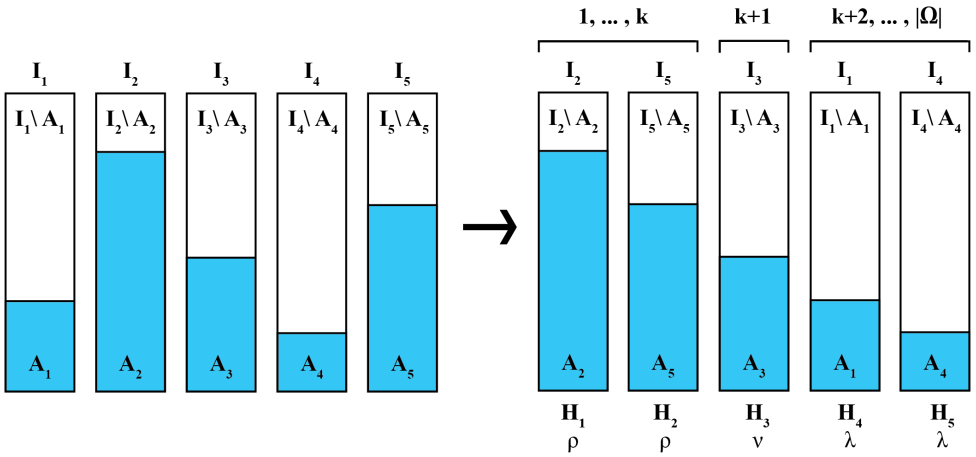

Figure 2.1: LCI Soft Constraint Construction

Note that $\sum_{i=1}^{|\Omega|} D_M(i) = 1$, which we will show formally in the Randomness over Labels section. Now following from the fact that the I_i sets partition I and both D_M and all the D_i are probability distributions, we can define our distribution D for each $w \in I_i$ as $D(w) = D_M(i)D_i(w)$.

For this distribution to be an improvising distribution, we must satisfy all the constraints outlined in the Formalized Labelled Control Improvisation Definition. Below, we prove that we satisfy each one,

Hard Constraint: $(\Pr[w \in I \mid w \leftarrow D] = 1)$

This constraint is trivially satisfied, as by definition D only samples over I, which is only composed of words satisfying the hard constraint.

Soft Constraint: $(\Pr[w \in A \mid w \leftarrow D] > 1 - \epsilon)$

To prove that $\Pr[w \in A \mid w \leftarrow D] > 1 - \epsilon$, we can equivalently show,

$$1 - \epsilon \leq \sum_{w \in A} D(w)$$

Before we continue, recall that $\delta(i)$ is defined as $\delta(i) = 1 - \epsilon^i_{opt}$. As ϵ^i_{opt} is by definition the amount of probability allocated to $I_i \backslash A_i$, and $I_i \backslash A_i$ and A_i partition I_i which is what D_i is defined over, it follows that $1 - \epsilon^i_{opt}$ is the probability allocated over all elements of A_i by D_i. Combining these two, we can conclude that $\delta(i)$ is in fact the probability of the distribution returning an element of A_i, or A as $A_i \subseteq A$ and $(I_i \backslash A_i) \cap A = \emptyset$. As the list H contains every element of $\{1, \ldots, |\Omega|\}$, albeit in an arbitrary order, it follows that the $\sum_{i=1}^{|\Omega|} D_M(i)\delta(H_i)$ equals the total probability of generating an element of A. We perform this transformation below,

$$\begin{aligned}
\sum_{w \in A} D(w) &= \sum_{i=1}^{|\Omega|} \sum_{w \in A_i} D_M(i)D_i(w) \\
&= \sum_{i=1}^{|\Omega|} D_M(i)\delta(i) \\
&= \sum_{i=1}^{k} D_M(i)\delta(i) + \sum_{i=k+1}^{k+1} D_M(i)\delta(i) + \sum_{i=k+2}^{|\Omega|} D_M(i)\delta(i) \\
&= \sum_{i=1}^{k} \lambda\delta(H_i) + \nu\delta(H_{k+1}) + \sum_{i=k+2}^{|\Omega|} \rho\delta(H_i)
\end{aligned}$$

Note that this is well defined, as if $k + 1 > |\Omega|$, then $\delta(H_{k+1}) = 0$ nullifying the term, and if $k + 2 > |\Omega|$, there are no integers satisfying the summation so it is also nullified. Combining the two above equations, we can see that we have satisfied the Soft Constraint if $1 - \epsilon < \sum_{i=1}^{k} (\lambda\delta(H_i)) + \nu\delta(H_{k+1}) + \sum_{i=k+2}^{|\Omega|} (\rho\delta(H_i))$. However, as we have assumed Condition 1 above, we know this is true, and we have therefore satisfied the Soft Constraint.

Randomness over Labels: $(\forall\, 1 \leq i \leq |\Omega|,\ \lambda \leq \Pr[w \in I_i \mid w \leftarrow D] \leq \rho)$

First, recall that $D_M(i) = \Pr[w \in I_i \mid w \leftarrow D]$. Looking at our construction above, $\forall\, 1 \leq i \leq |\Omega|,\ D_M(i) \in \{\lambda, \rho, \nu\}$. It is trivial to see that λ and ρ satisfy our randomness constraint over labels. However, we must prove that $\lambda \leq \nu \leq \rho$. Recall that we assume Condition 1, which states $\frac{1}{\rho} \leq |\Omega| \leq \frac{1}{\lambda}$.

We will separate out the case where $\lambda = \rho$, as that directly implies $\nu = \lambda$.

$$
\begin{aligned}
\nu &= 1 + \lambda k - \rho k - |\Omega|\lambda + \lambda \\
&= 1 + \lambda k - \lambda k - |\Omega|\lambda + \lambda \\
&= 1 - |\Omega|\lambda + \lambda \\
&= 1 + \lambda(1 - |\Omega|) \\
&\geq 1 + \lambda(1 - \frac{1}{\lambda}) \\
&= 1
\end{aligned}
\qquad
\begin{aligned}
\nu &= 1 + \lambda k - \rho k - |\Omega|\lambda + \lambda \\
&= 1 + \rho k - \rho k - |\Omega|\rho + \rho \\
&= 1 - |\Omega|\rho + \rho \\
&= 1 + \rho(1 - |\Omega|) \\
&\leq 1 + \rho(1 - \frac{1}{\rho}) \\
&= 1
\end{aligned}
$$

For the general case, consider the following manipulations,

$$
\begin{aligned}
\nu &= 1 + \lambda k - \rho k - |\Omega|\lambda + \lambda \\
&= 1 + \lambda \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor - \rho \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor - |\Omega|\lambda + \lambda \\
&= 1 + (\lambda - \rho) \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor - |\Omega|\lambda + \lambda \quad \text{(Recall by definition } \lambda \leq \rho\text{)}
\end{aligned}
$$

$$
\begin{aligned}
&\geq 1 + (\lambda - \rho)\frac{1 - |\Omega|\lambda}{\rho - \lambda} - |\Omega|\lambda + \lambda \\
&= 1 + (\lambda - \rho)\frac{|\Omega|\lambda - 1}{\lambda - \rho} - |\Omega|\lambda + \lambda \\
&= 1 + |\Omega|\lambda - 1 - |\Omega|\lambda + \lambda \\
&= \lambda \\
\therefore \quad \nu &\geq \lambda
\end{aligned}
\qquad
\begin{aligned}
&= 1 + (\lambda - \rho)\left(\frac{|\Omega|\lambda - 1}{\lambda - \rho} - \iota\right) - |\Omega|\lambda + \lambda \qquad \text{(For some } \iota < 1\text{)} \\
&= 1 + |\Omega|\lambda - 1 - \iota\lambda + \epsilon\rho - |\Omega|\lambda + \lambda \\
&= -\iota\lambda + \iota\rho + \lambda \\
&= \iota(\rho - \lambda) + \lambda \\
&< \rho - \lambda + \lambda \\
&= \rho \\
\therefore \quad \nu &< \rho
\end{aligned}
$$

We must also prove that $\sum_{i=1}^{|\Omega|} D_M(i) = 1$. We ignore the case where $p \neq q$ as it directly implies $k = 0, |\Omega| = \frac{1}{\lambda}, \nu = \lambda$, trivializing the problem. Consider the following transformations,

$$
\begin{aligned}
\sum_{i=1}^{|\Omega|} D_M(i) &= \rho k + \nu + \lambda(|\Omega| - k - 1) \\
&= \rho \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor + 1 + \lambda k - \rho k - \lambda|\Omega| + \lambda + \lambda(|\Omega| - \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor - 1) \\
&= \rho \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor + 1 + \lambda k - \rho k - \lambda|\Omega| + \lambda + \lambda|\Omega| - \lambda \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor - \lambda \\
&= \rho \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor + 1 + \lambda k - \rho k - \lambda \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor \\
&= 1 + (\rho - \lambda) \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor + (\lambda - \rho)k \\
&= 1 + (\rho - \lambda) \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor + (\lambda - \rho) \left\lfloor \frac{1 - |\Omega|\lambda}{\rho - \lambda} \right\rfloor \\
&= 1
\end{aligned}
$$

We must also show that $0 \leq k \leq |\Omega|$. For these operations, assume $\lambda \neq \rho$ as in this case k trivially satisfies the bounds. Recall that by assumption, Condition 1 binds $|\Omega|$ as follows, $\frac{1}{\rho} \leq |\Omega| \leq \frac{1}{\lambda}$.

$$
\begin{aligned}
k &= \left\lfloor \frac{1-|\Omega|\lambda}{\rho-\lambda} \right\rfloor \\
&\leq \frac{1-|\Omega|\lambda}{\rho-\lambda} \\
&\leq \frac{1-\frac{\lambda}{\rho}}{\rho-\lambda} \\
&= \frac{\rho-\lambda}{\rho(\rho-\lambda)} \\
&= \frac{1}{\rho} \\
&\leq |\Omega|
\end{aligned}
\qquad\qquad
\begin{aligned}
k &= \left\lfloor \frac{1-|\Omega|\lambda}{\rho-\lambda} \right\rfloor \\
&\geq \left\lfloor \frac{1-\frac{\lambda}{\lambda}}{\rho-\lambda} \right\rfloor \\
&= 0
\end{aligned}
$$

Combining all this, we can see that we assign a marginal probability between λ and ρ to each label, which adds up to one. We can also see that k, ν, and $(n-k-1)$ must be positive, except in the case where $n=k$ in which case ν and $n-k-1$ cancel each other out and every label is assigned ρ marginal probability. We have also shown that these probabilities add up to 1, indicating we have satisfied Randomness Over Labels.

Randomness Over Words: $\quad (\forall\ 1 \leq i \leq |\Omega|,\ \forall y \in I_i,\ \alpha_i \leq \Pr[y = w \mid w \in I_i, w \leftarrow D] \leq \beta_i)$

There are two possibilities for the distribution over any D_i, either **(A)**, $\epsilon^i_{opt} = 1-\beta_i|A_i|$, or **(B)**, $\epsilon^i_{opt} = \alpha_i|I_i \backslash A_i|$. We will show that in either case all elements of I_i satisfy the lower and upper randomness constraints.

(A) $\epsilon^i_{opt} = 1-\beta_i|A_i|$

Showing that all elements in A_i satisfy the constraint is simple, as by definition we distribute $1-\epsilon^i_{opt} = \beta_i|A_i|$ uniformly over all elements of A_i. Thus $\forall w \in A_i$, $D_i(w) = \frac{1-\epsilon^i_{opt}}{|A_i|} = \frac{\beta_i|A_i|}{|A_i|} = \beta_i$. This satisfies both the upper and lower bound requirements for the constraint over all elements of A_i.
Since $\epsilon^i_{opt} = 1-\beta_i|A_i|$, we can also conclude that $1-\beta_i|A_i| \geq \alpha_i|I_i \backslash A_i|$ by the max function. As we are distributing ϵ^i_{opt} uniformly over all elements $\forall w \in I_i \backslash A_i$, $D_i(w) = \frac{\epsilon^i_{opt}}{|I_i \backslash A_i|} = \frac{1-\beta_i|A_i|}{|I_i \backslash A_i|} \geq \frac{\alpha_i|I_i \backslash A_i|}{|I_i \backslash A_i|} = \alpha_i$. In addition, by Condition 2, $\frac{1}{\beta_i} \leq I_i$. In addition, as $A_i \subseteq I_i$, we can conclude that $|I_i \backslash A_i| = |I_i| - |A_i|$. Therefore, $\forall w \in I_i \backslash A_i$, $D_i(w) = \frac{\epsilon^i_{opt}}{|I_i \backslash A_i|} = \frac{1-\beta_i|A_i|}{|I_i|-|A_i|} \leq \frac{1-\beta_i|A_i|}{\frac{1}{\beta_i}-|A_i|} = \beta_i \frac{1-\beta_i|A_i|}{1-\beta_i|A_i|} = \beta_i$. Thus we have shown that both the upper and lower bound requirements are satisfied for all elements of $A_i \backslash I_i$.

(B) $\epsilon^i_{opt} = \alpha_i|I_i \backslash A_i|$

Showing that all elements in $I_i \backslash A_i$ is simple, as we are distributing $\alpha_i|I_i \backslash A_i|$ uniformly over all elements of $I_i \backslash A_i$. Thus, $\forall w \in I_i \backslash A_i$, $D_i(w) = \frac{\alpha_i|I_i \backslash A_i|}{|I_i \backslash A_i|} = \alpha_i$. This satisfies both the upper and lower bound requirements for the constraint over all elements of $I_i \backslash A_i$.
Since $\epsilon^i_{opt} = \alpha_i|I_i \backslash A_i|$, it follows by the definition of the max function that $\alpha_i|I_i \backslash A_i| > 1-\beta_i|A_i|$. Again, note that as $A_i \subseteq I_i$, we can conclude that $|I_i \backslash A_i| = |I_i| - |A_i|$ and that by condition 2, $|I_i| \leq \frac{1}{\alpha_i}$. As we are distributing $1-\epsilon^i_{opt}$ uniformly over all elements of A_i, $\forall w \in A_i$, $D_i(w) = \frac{1-\epsilon^i_{opt}}{|A_i|} = \frac{1-\alpha_i|I_i|+\alpha_i|A_i|}{|A_i|} = \frac{1-\alpha_i|I_i|}{|A_i|} + \alpha_i \geq \frac{1-\alpha_i \frac{1}{\alpha_i}}{|A_i|} + \alpha_i = \alpha_i$. Similarly, $\forall w \in A_i$, $D_i(w) = \frac{1-\epsilon^i_{opt}}{|A_i|} = \frac{1-\alpha_i|I_i \backslash A_i|}{|A_i|} \leq \frac{1-(1-\beta_i|A_i|)}{|A_i|} = \frac{\beta_i|A_i|}{|A_i|} = \beta_i$. Thus we have shown that all elements in A_i satisfy the upper and lower bound requirements.

Finally, it is implicit by the definition that $\sum_{w \in I_i} D_i(w) = 1$, as in either case all remaining probability is assigned.

($\Leftarrow$) If the labelled control improvisation instance is solvable, the above conditions hold.

Assume that an instance $\mathcal{C}$ of the labelled control improvisation is solvable. This implies there exists a distribution, which we will call D', that satisfies the 4 constraints listed in the definition. Let D'_M be the marginal probability function

and $D'_i(w)$ be the conditional probability function as defined above. We will now show that $\mathcal{C}$ must also satisfy the 3 conditions enumerated above.

1. $\frac{1}{\rho} \leq |\Omega| \leq \frac{1}{\lambda}$

   Note that a probability distribution must sum to 1, and that a label class must have between λ and ρ marginal probability of being selected in an improvising distribution, so,

$$\begin{aligned} 1 &= \sum_{i=1}^{|\Omega|} D'_M(i) \leq \rho|\Omega| \\ \therefore \quad \frac{1}{\rho} &\leq |\Omega| \\ 1 &= \sum_{i=1}^{|\Omega|} D'_M(i) \geq \lambda|\Omega| \\ \therefore \quad |\Omega| &\leq \frac{1}{\lambda} \end{aligned}$$

   As we can see, $\frac{1}{\rho} \leq |\Omega| \leq \frac{1}{\lambda}$.

2. $\forall\ 1 \leq i \leq |\Omega|,\ \frac{1}{\beta_i} \leq |I_i| \leq \frac{1}{\alpha_i}$

   Again, note that a probability distribution must sum to 1, and that a word must have between α_i and β_i conditional probability of being selected, so,

$$\begin{aligned} 1 &= \sum_{w \in I_i} D'_i(w) \leq \beta_i|I_i| \\ \therefore \quad \frac{1}{\beta_i} &\leq |I_i| \\ 1 &= \sum_{w \in I_i} D'_i(w) \geq \alpha_i|I_i| \\ \therefore \quad |I_i| &\leq \frac{1}{\alpha_i} \end{aligned}$$

   As we can see, $\frac{1}{\beta_i} \leq |I_i| \leq \frac{1}{\alpha_i}$.

3. $1 - \epsilon \leq \sum_{i=1}^{k} (\lambda\delta(H_i)) + \nu\delta(H_{k+1}) + \sum_{i=k+2}^{|\Omega|} (\rho\delta(H_i))$

   We will prove this statement by arguing that the improvising distribution constructed above in the proof that our conditions are sufficient is optimal in terms of maximizing the probability of selecting an element from A. Once we have shown this, we can conclude that if there exists an improvising distribution, it must select elements in A with equal or less probability than our improvising distribution. Thus our constructed distribution must also be a valid distribution for $\mathcal{C}$ implying that the above inequality holds.

   We will first show that the constructed distribution is optimal via a transformation argument. Let $\delta'(i)$ be the marginal probability of selecting a string in A_i from D', conditional on that string being a member of I_i.

   As D' is an improvising distribution, we know that $\Pr[w \in A \mid w \leftarrow D'] \geq 1 - \epsilon$. Using our above definitions, we know that $\Pr[w \in A \mid w \leftarrow D']$ is exactly $\sum_{i=1}^{|\Omega|} D'_M(i)\delta'(i)$. We formalize this below,

$$\begin{aligned} 1 - \epsilon &\leq \sum_{i=1}^{|\Omega|} \sum_{w \in A_i} D'_M(i) D'_i(w) \\ &= \sum_{i=1}^{|\Omega|} D'_M(i)\delta'(i) \end{aligned}$$

Now we will show that for each $\delta'(i)$, we can apply the clamping method for assigning distribution used in our construction above, and get an equal or greater probability of selecting an element. To this end, we must show that $\forall\ 1 \leq i \leq |\Omega|,\ \delta'(i) \leq 1 - \max(1 - \beta_i|A_i|, \alpha_i|I_i \backslash A_i|)$. Assume for the sake of contradiction that for some i, $\delta'(i) > 1 - \max(1 - \beta_i|A_i|, \alpha_i|I_i \backslash A_i|)$. Consider the following cases,

Case 1: $1 - \beta_i|A_i| \geq \alpha_i|I_i \backslash A_i|$

$$\begin{aligned}\delta'(i) &> 1 - (1 - \beta_i|A_i|) \\ &= \beta_i|A_i|\end{aligned}$$

This however is a contradiction, as by definition we can distribute no more than $\beta|A_i|$ probability over all A_i.

Case 2: $\alpha_i|I_i \backslash A_i| \geq 1 - \beta_i|A_i|$

$$\begin{aligned}\delta'(i) &> 1 - \alpha_i|I_i \backslash A_i| \\ &\geq 1 - (1 - \beta_i|A_i|) \\ &= \beta_i|A_i|\end{aligned}$$

Again, this is a contradiction by the same reason as above.

Thus we can change each $\delta'(i)$ to the probability obtained with the clamping method, and have equal or larger probability of selecting an element in A_i. Note that we have already shown that the clamping method always satisfies the conditions if there are an appropriate number of labels, which we have shown while proving the necessity of the other conditions. Using this information, for every i define $\delta(i)$ to the probability of selecting an element of A_i, conditional on that element being in I_i, that we obtain from the clamping method. From this, we can also define a list H. Let H be a list of the indices, i, for each label, sorted in non increasing order by the value of $\delta(i)$. Overall, it is clear that $\sum_{i=1}^{|\Omega|} D'_M(i)\delta'(i) \leq \sum_{i=1}^{|\Omega|} D'_M(i)\delta(i)$.

Now we will show that the method that we use to assign the marginal probability to each label class in our constructive method is optimal in terms of maximizing the marginal probability of selecting an element in A. In particular we will show this over D', but with the newly obtained $\delta(i)$. Consider the following manipulation, and repeat it as many times as possible: for the earliest element of H, H_i that is not already equal to ρ, set $D'_M(H_i)$ to ρ and subtract probability from any and as many $D'_M(H_j),\ i < j$ as necessary to ensure that $\sum_{i=0}^{|\Omega|} D'_M(i) = 1$ and $\forall\ 1 \leq i \leq |\Omega|,\ D'_M(i) \geq \lambda$. As our list H is in non increasing order by the value of $\delta(i)$, for any $1 \leq i < j, \delta(i) \geq \delta(j)$. Thus for any ω subtracted from $D'_M(j)$ and added to $D'_M(i)$,

$$\begin{aligned}D'_M(H_i)\delta(H_i) + D'_M(H_j)\delta(H_j) &= D'_M(H_i)\delta(H_i) + (D'_M(H_j) + \omega - \omega)\delta(H_j) \\ &= D'_M(H_i)\delta(H_i) + \omega\delta(H_j) + (D'_M(H_j) - \omega)\delta(H_j) \\ &\leq (D'_M(H_i) + \omega)\delta(H_i) + (D'_M(H_j) - \omega)\delta(H_j)\end{aligned}$$

As we can see, each time we do this the total sum is equal to or greater to what it was before. Once this precess terminates, let D_M be the resulting marginal probability distribution. Note that we do not violate any of the Randomness over Labels constraints, $\sum_{i=1}^{|\Omega|} D_M(i)$ remains the same, we never assign $D_M(i)$ for any i to a value higher than ρ, and we ensure via our conditions that we do not reduce an $D_M(i)$ for any i below λ.

We must now show that $k = \left\lfloor \frac{1-|\Omega|\lambda}{\rho-\lambda} \right\rfloor$ labels will end up with marginal probability ρ after we complete this process. This can be shown simply by reverting to its derivation, $1 - k\rho \geq (|\Omega| - k)\lambda$, where k is the smallest natural number satisfying the inequality. We then compute k as defined above for this problem, which fits this inequality, ensuring that when we have assigned k label distributions probability ρ, we still have enough remaining to assign all the remaining label distributions probability at least λ. We then perform a similar iteration of the process above one final time to assign any remaining probability to the next $D_M(i)$, resulting in it having probability ν and all label distributions after it having probability λ. By the same logic as above, this must also result in a greater or equal sum. Thus, we have shown that $\sum_{i=1}^{|\Omega|} D'_M(i)\delta(i) \leq \sum_{i=1}^{|\Omega|} D_M(i)\delta(i) = \sum_{i=1}^{k} (\lambda\delta(H_i)) + \nu\delta(H_{k+1}) + \sum_{k+2}^{|\Omega|} (\rho\delta(H_i))$.

Combining all of the results above, we can see that,

$$1-\epsilon \leq \sum_{i=1}^{|\Omega|} \sum_{w \in A_i} D'_M(i) D'_i(w) = \sum_{i=1}^{|\Omega|} D'_M(i)\delta'(i) \leq \sum_{i=1}^{|\Omega|} D'_M(i)\delta(i) \leq \sum_{i=1}^{k} (\lambda\delta(H_i)) + \nu\delta(H_{k+1}) + \sum_{k+2}^{|\Omega|} (\rho\delta(H_i))$$

And thus we satisfy Condition 3.

■

Of importance in the previous theorem is that it is constructive, meaning that we have also outlined a clearly polynomial time algorithm to compute an improvising. We formalize this in the theorem below.

Theorem 2.2.2. *(LCI Construction) Given $|I_i \backslash A_i|$ and $|A_i|$ for all $i \in \{1, \ldots, |\Omega|\}$ for an LCI instance $\mathcal{C}$, there is an algorithm to compute an LCI improvising distribution in polynomial time of the size of the LCI instance.*

Proof. This theorem follows immediately from Theorem 2.2.1. The procedure outlined in the proof that the conditions are sufficient provides an algorithm for constructing an improvising distribution. This algorithm can clearly be done in polynomial time, and the only inputs required are $|I_i \backslash A_i|$ and $|A_i|$ for all $i \in \{1, \ldots, |\Omega|\}$. ■

Chapter 3

Maximum Entropy Labelled Control Improvisation

As we have seen in the Labelled Control Improvisation problem, the sheer number of parameters can be somewhat unwieldy to work with. To resolve this, we seek to simplify the parameters to the Labelled Control Improvisation problem by removing the $\hat{\alpha}$ and $\hat{\beta}$ parameters. Without these parameters however, we begin to encounter trivial and overly simplistic solutions, as we no longer enforce randomness inside each label class. Consider an example with 3 label classes, each containing 1,000 words. Assume each label class has exactly one word in A. One possible distribution could simply return only the three words in A, with whatever marginal probability over the labels dictated by λ and ρ. However, the ϵ for our soft constraint might allow us to return quite a few words from I without violating the soft constraint. Obviously if we seek greater randomness, this second solution is preferable. To solve this problem, we introduce the Maximum Entropy Labelled Control Improvisation problem.

3.1 Definition

We now define a MELCI instance, and from that a MELCI improvising distribution and the MELCI Problem.

Definition 3.1.1. (MELCI Instance) Let $\mathcal{C} = (\mathcal{H}, \mathcal{S}, L, m, n, \epsilon, \lambda, \rho, \tau)$ be a Maximum Entropy Labelled Control Improvisation, or MELCI, instance. Assume all the parameters are defined as in Section 2.1, and note that a MELCI instance has the same form as an LCI instance without the $\hat{\alpha}$ and $\hat{\beta}$ parameters, and with the addition of τ, a positive rational number.

Definition 3.1.2. (MELCI Improvising Distribution) Let H be the distribution entropy function. Given a MELCI Instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, L, m, n, \epsilon, \lambda, \rho, \tau)$, a distribution on words $D : \Sigma^* \to [0, 1]$ is an improvising distribution if it satisfies the following constraints,

1. **Hard Constraint:** $\Pr[w \in I \mid w \leftarrow D] = 1$
2. **Soft Constraint:** $\Pr[w \in A \mid w \leftarrow D] \geq 1 - \epsilon$
3. **Randomness over Labels:** $\forall\, 1 \leq i \leq |\Omega|,\ \lambda \leq \Pr[w \in I_i \mid w \leftarrow D] \leq \rho$

Definition 3.1.3. (MELCI Problem) We say that a MELCI instance is feasible if there exists an improvising distribution for it. We define an improviser for an MELCI instance as a probabilistic algorithm which takes no input and finite expected runtime, whose output distribution is an improvising distribution such that the entropy of the output distribution is at most τ less than the maximum entropy improvising distribution (an additive error bound). Given a MELCI instance $\mathcal{C}$, we define the MELCI problem as determining if $\mathcal{C}$ is feasible, and if is, generating an improviser for $\mathcal{C}$ and the entropy of the improviser's output distribution.

We can now address how to solve the MELCI problem.

3.2 Construction of Maximum Entropy Distribution

3.2.1 Initial Problem

Recall that $D_M(i)$ is defined as the marginal probability of picking a word in I_i, and $D_i(w)$ is defined as the conditional probability of selecting a word $w \in I_i$ given that we know we will select a word from I_i. In this construction, we seek to maximize the entropy equation, which we write in expanded form below,

$$\sum_{i=1}^{|\Omega|} \sum_{w \in I_i} -D_M(i)D_i(w) \cdot \log(D_M(i)D_i(w))$$

In doing so we must respect the following constraints,

$$\begin{aligned}
\sum_{i=1}^{|\Omega|} \sum_{w \in A_i} D_M(i)D_i(w) &\geq 1 - \epsilon \\
\forall\, i \in \{1, \ldots, |\Omega|\},\ \lambda \leq D_M(i) &\leq \rho \\
\sum_{i=1}^{|\Omega|} D_M(i) &= 1 \\
\forall\, i \in \{1, \ldots, |\Omega|\},\ \sum_{w \in I_i} D_i(w) &= 1 \\
\forall\, i \in \{1, \ldots, |\Omega|\},\ D_M(i) &\geq 0 \\
\forall\, i \in \{1, \ldots, |\Omega|\},\ \forall w \in I_i,\ D_i(w) &\geq 0
\end{aligned}$$

As we can see, this is none other than a continuous constrained optimization problem over the variables $D_M(i)$ and $D_i(w)$ for all $i \in \{1, \ldots, |\Omega|\}$ and all $w \in I$.

3.2.2 Bi-Uniform Label Class Distribution

We can simplify this problem by simplifying all the assignments of D_i for every word in I_i into two assignments, the probability distributed over A_i and the probability distributed over $I_i \backslash A_i$. We will prove this below after proving a lemma to help us.

Lemma 3.2.1. *A uniform distribution over a finite set of words is a distribution that achieves maximum entropy.*

Proof. Let D_{opt} be the uniform distribution over a set of words S. Using the definition of entropy,

$$\begin{aligned}
H(D_{opt}) &= -\sum_{w \in I} D_{opt}(w) \log(D_{opt}(w)) \\
&= \sum_{w \in I} -\left(\frac{1}{|I|}\right) \log\left(\frac{1}{|I|}\right) \\
&= \sum_{w \in I} -\left(\frac{1}{|I|}\right) (\log(1) - \log(|I|)) \\
&= \sum_{w \in I} \left(\frac{1}{|I|}\right) \log(|I|) \\
&= \log(|I|)
\end{aligned}$$

Now, recall Jensen's inequality, which states, for a real concave function f, numbers $x_1, \ldots, x_n$ in f's domain, and positive weights $a_i, \ldots, a_n$.

$$f\left(\frac{\sum a_i x_i}{\sum a_i}\right) \geq \frac{\sum a_i f(x_i)}{\sum a_i}$$

In addition, recall that entropy is a concave function [8, Theorem 2.7.3]. Noting this, let $f(x) = -x\log(x)$. Let D be any distribution over S. Again using the definition of entropy,

$$\begin{aligned}
H(D) &= \sum_{w\in I} D(w)\log(D(w)) \\
&= \sum_{w\in I} f(D(w)) \\
&= |I|\left(\frac{\sum_{w\in I} f(D(w))}{|I|}\right) \\
&= |I|\left(\frac{\sum_{w\in I} f(D(w))}{\sum_{w\in I} 1}\right) \\
&\leq |I| f\left(\frac{\sum_{w\in I} D(w)}{\sum_{w\in I} 1}\right) \qquad \text{(Jensen's Inequality)} \\
&= |I| f\left(\frac{1}{|I|}\right) \\
&= -(|I|)\left(\frac{1}{|I|}\right)\log\left(\frac{1}{|I|}\right) \\
&= -\log\left(\frac{1}{|I|}\right) \\
&= -(\log(1) - \log(|I|)) \\
&= \log(|I|)
\end{aligned}$$

As we can see $H(D) \leq H(D_{opt})$, thus showing that D_{opt}, the uniform distribution, is a maximum entropy distribution over I. ■

Theorem 3.2.2. *Let I_i be a label class for which we require a distribution that has P_{A_i} probability of choosing a word in A_i and $P_{I_i\setminus A_i}$ probability of choosing a word in $I_i\setminus A_i$. D_{Opt} is such a distribution that achieves maximum entropy,*

$$D_{Opt}(w) = \begin{cases} \frac{P_{A_i}}{|A_i|} & w \in A_i \\ \\ \frac{P_{I_i\setminus A_i}}{|I_i\setminus A_i|} & w \in I_i\setminus A_i \end{cases}$$

Proof. Let D be any distribution over a label class I_i that has P_{A_i} probability of choosing a word in A_i and $P_{I_i\setminus A_i}$ probability of choosing a word in $I_i\setminus A_i$. Using the definition of entropy,

$$\begin{aligned}
H(D) &= \sum_{w\in I_i} -D(w)\log(D(w)) \\
&= \sum_{w\in A_i} -D(w)\log(D(w)) + \sum_{w\in A_i\setminus I_i} -D(w)\log(D(w))
\end{aligned}$$

Consider two distributions D_{A_i} and $D_{I_i\setminus A_i}$ as defined below,

$$D_{A_i}(w) = \begin{cases} \frac{1}{P_{A_i}} D(w) & w \in A_i \\ 0 & w \notin A_i \end{cases}$$

$$D_{I_i \backslash A_i}(w) = \begin{cases} \frac{1}{P_{I_i \backslash A_i}} D(w) & w \in I_i \backslash A_i \\ 0 & w \notin I_i \backslash A_i \end{cases}$$

Recall that $P_{A_i} = \Pr[w \in A_i \mid w \leftarrow D]$ and $P_{I_i \backslash A_i} = \Pr[w \in I_i \backslash A_i \mid w \leftarrow D]$. From the above we can see that $D_{A_i}(w) = \Pr[w = y \mid y \in A_i, y \leftarrow D]$ and $D_{I_i \backslash A_i}(w) = \Pr[w = y \mid y \in I_i \backslash A_i, y \leftarrow D]$.

Using these definitions and observations, along with Lemma 3.2.1, we can continue our calculations from above,

$$\begin{aligned}
&= \sum_{w \in A_i} -D(w) \log(P_{A_i} D_{A_i}(w)) + \sum_{w \in A_i \backslash I_i} -D(w) \log(P_{I_i \backslash A_i} D_{I_i \backslash A_i}(w)) \\
&= \sum_{w \in A_i} -D(w) \log(P_{A_i}) + \sum_{w \in A_i} -D(w) \log(D_{A_i}(w)) + \sum_{w \in A_i \backslash I_i} -D(w) \log(P_{I_i \backslash A_i}) + \sum_{w \in A_i \backslash I_i} -D(w) \log(D_{I_i \backslash A_i}(w)) \\
&= \sum_{w \in A_i} -P_{A_i} D_{A_i}(w) \log(P_{A_i}) + \sum_{w \in A_i} -P_{A_i} D_{A_i}(w) \log(D_{A_i}(w)) \\
&\quad + \sum_{w \in A_i \backslash I_i} -P_{I_i \backslash A_i} D_{I_i \backslash A_i}(w) \log(P_{I_i \backslash A_i}) + \sum_{w \in A_i \backslash I_i} -P_{I_i \backslash A_i} D_{I_i \backslash A_i}(w) \log(D_{I_i \backslash A_i}(w)) \\
&= -P_{A_i} \log(P_{A_i}) + P_{A_i} \sum_{w \in A_i} -D_{A_i}(w) \log(D_{A_i}(w)) - P_{I_i \backslash A_i} \log(P_{I_i \backslash A_i}) + P_{I_i \backslash A_i} \sum_{w \in A_i \backslash I_i} -D_{I_i \backslash A_i}(w) \log(D_{I_i \backslash A_i}(w)) \\
&= -P_{A_i} \log(P_{A_i}) + P_{A_i} H(D_{A_i}) - P_{I_i \backslash A_i} \log(P_{I_i \backslash A_i}) + P_{I_i \backslash A_i} H(D_{I_i \backslash A_i}) \\
&\leq P_{A_i} \log(|A_i|) + P_{I_i \backslash A_i} \log(|I_i \backslash A_i|) - P_{A_i} \log(P_{A_i}) - P_{I_i \backslash A_i} \log(P_{I_i \backslash A_i}) \qquad \text{(Lemma 3.2.1)}
\end{aligned}$$

Now we calculate the entropy of D_{Opt}.

$$\begin{aligned}
H(D_{Opt}) &= \sum_{w \in I_i} -D_{Opt}(w) \log(D_{Opt}(w)) \\
&= \sum_{w \in A_i} -D_{Opt}(w) \log(D_{Opt}(w)) + \sum_{w \in I_i \backslash A_i} -D_{Opt}(w) \log(D_{Opt}(w)) \\
&= \sum_{w \in A_i} -\frac{P_{A_i}}{|A_i|} \log\left(\frac{P_{A_i}}{|A_i|}\right) + \sum_{w \in I_i \backslash A_i} -\frac{P_{I_i \backslash A_i}}{|I_i \backslash A_i|} \log\left(\frac{P_{I_i \backslash A_i}}{|I_i \backslash A_i|}\right) \\
&= -P_{A_i} \log\left(\frac{P_{A_i}}{|A_i|}\right) - P_{I_i \backslash A_i} \log\left(\frac{P_{I_i \backslash A_i}}{|I_i \backslash A_i|}\right) \\
&= P_{A_i} \log\left(|A_i|\right) + P_{I_i \backslash A_i} \log\left(|I_i \backslash A_i|\right) - P_{A_i} \log\left(P_{A_i}\right) - P_{I_i \backslash A_i} \log\left(P_{I_i \backslash A_i}\right)
\end{aligned}$$

As we can see, the entropy of D_{Opt} is always greater than or equal to any other distribution that meets the requirements. Therefore, D_{Opt} is a maximum entropy distribution over a label class I_i that has P_{A_i} probability of choosing a word in A_i and $P_{I_i \backslash A_i}$ probability of choosing a word in $I_i \backslash A_i$. ■

Using these results, we can simplify our optimization problem, as we already know the optimal distribution in each label class. All that remains to be chosen is P_{A_i}, the amount of probability each conditional distribution in a label class has of selecting a word in A (which also fixes $P_{I_i \backslash A_i}$). Incorporating the results from Theorem 3.2.2, we can perform the following transformations to our maximization problem,

$$
\begin{aligned}
&\sum_{i=1}^{|\Omega|}\sum_{w\in I_i} -D_M(i)D_i(w)\cdot\log(D_M(i)D_i(w)) \\
&= \sum_{i=1}^{|\Omega|}\left(\sum_{w\in A_i} -D_M(i)D_i(w)\cdot\log(D_M(i)D_i(w)) + \sum_{w\in I_i\backslash A_i} -D_M(i)D_i(w)\cdot\log(D_M(i)D_i(w))\right) \\
&\leq \sum_{i=1}^{|\Omega|}\left(\sum_{w\in A_i} -D_M(i)\frac{P_{A_i}}{|A_i|}\cdot\log\left(D_M(i)\frac{P_{A_i}}{|A_i|}\right) + \sum_{w\in I_i\backslash A_i} -D_M(i)\frac{P_{I_i\backslash A_i}}{|I_i\backslash A_i|}\cdot\log\left(D_M(i)\frac{P_{I_i\backslash A_i}}{|I_i\backslash A_i|}\right)\right) \qquad \text{(Theorem 3.2.2)} \\
&= \sum_{i=1}^{|\Omega|}\left(-D_M(i)P_{A_i}\cdot\log\left(D_M(i)\frac{P_{A_i}}{|A_i|}\right) - D_M(i)P_{I_i\backslash A_i}\cdot\log\left(D_M(i)\frac{P_{I_i\backslash A_i}}{|I_i\backslash A_i|}\right)\right) \\
&= \sum_{i=1}^{|\Omega|}\left(-D_M(i)P_{A_i}\cdot\log\left(D_M(i)\right) - D_M(i)P_{A_i}\cdot\log\left(\frac{P_{A_i}}{|A_i|}\right) - D_M(i)P_{I_i\backslash A_i}\cdot\log\left(D_M(i)\right) - D_M(i)P_{I_i\backslash A_i}\cdot\log\left(\frac{P_{I_i\backslash A_i}}{|I_i\backslash A_i|}\right)\right) \\
&= \sum_{i=1}^{|\Omega|} -D_M(i)P_{A_i}\left(\log\left(D_M(i)\right)+\log\left(P_{A_i}\right)-\log\left(|A_i|\right)\right) - D_M(i)P_{I_i\backslash A_i}\left(\log\left(D_M(i)\right)+\log\left(P_{I_i\backslash A_i}\right)-\log\left(|I_i\backslash A_i|\right)\right) \\
&= \sum_{i=1}^{|\Omega|} -D_M(i)P_{A_i}\left(\log\left(D_M(i)P_{A_i}\right)-\log\left(|A_i|\right)\right) - D_M(i)P_{I_i\backslash A_i}\left(\log\left(D_M(i)P_{I_i\backslash A_i}\right)-\log\left(|I_i\backslash A_i|\right)\right) \\
&= -\sum_{i=1}^{|\Omega|}\left(D_M(i)P_{A_i}\log\left(D_M(i)P_{A_i}\right) - D_M(i)P_{A_i}\log\left(|A_i|\right) + D_M(i)P_{I_i\backslash A_i}\log\left(D_M(i)P_{I_i\backslash A_i}\right) - D_M(i)P_{I_i\backslash A_i}\log\left(|I_i\backslash A_i|\right)\right)
\end{aligned}
$$

Thus our new optimization problem is,

$$
\text{Max } -\sum_{i=1}^{|\Omega|} D_M(i)P_{A_i}\log\left(D_M(i)P_{A_i}\right) - D_M(i)P_{A_i}\log\left(|A_i|\right) + D_M(i)P_{I_i\backslash A_i}\log\left(D_M(i)P_{I_i\backslash A_i}\right) - D_M(i)P_{I_i\backslash A_i}\log\left(|I_i\backslash A_i|\right)
$$

Subject to,

$$
\begin{aligned}
&\sum_{i=1}^{|\Omega|} D_M(i)P_{A_i} \geq 1-\epsilon \\
\forall\ i\in\{1,\ldots,|\Omega|\},\ &\lambda \leq D_M(i) \leq \rho \\
&\sum_{i=1}^{|\Omega|} D_M(i) = 1 \\
\forall\ i\in\{1,\ldots,|\Omega|\},\ &D_M(i)\geq 0 \\
\forall\ i\in\{1,\ldots,|\Omega|\},\ &P_{A_i}+P_{I_i\backslash A_i}=1 \\
\forall\ i\in\{1,\ldots,|\Omega|\},\ &\text{if } |A_i|\neq 0, P_{A_i}\geq 0 \\
\forall\ i\in\{1,\ldots,|\Omega|\},\ &\text{if } |I_i\backslash A_i|\neq 0, P_{I_i\backslash A_i}\geq 0 \\
\forall\ i\in\{1,\ldots,|\Omega|\},\ &\text{if } |A_i|= 0, P_{A_i}= 0 \\
\forall\ i\in\{1,\ldots,|\Omega|\},\ &\text{if } |I_i\backslash A_i|= 0, P_{I_i\backslash A_i}= 0
\end{aligned}
$$

Note that we must add two additional constraints to ensure that if a label class has no words in A_i or $I_i\backslash A_i$, then we do not assign any probability to P_{A_i} or $P_{I_i\backslash A_i}$ respectively.

3.2.3 Optimization of Distribution

We will now present all of these formulas in canonical form while simplifying the problem further. Without these changes, the soft constraint in this problem is not convex. To transform the problem into a convex one, we define $D(A_i) = D_M(i)P_{A_i}$ and $D(I_i \backslash A_i) = D_M(i)P_{I_i \backslash A_i}$, and restate the problem in terms of these new variables. We can also restrict the domain of all variables $D(A_i)$ and $D(I_i \backslash A_i)$ to $[0, 1]$. Note that for any i, $D_M(i) = D(A_i) + D(I_i \backslash A_i)$. In addition, for simplicity we assume that $\log(x)$ is the natural logarithm.

In its final state, the optimization problem takes the form below, optimizing over $D(A_i)$ and $D(I_i \backslash A_i)$ for every i, of which there are $|\Omega|$, for a total of $2|\Omega|$ variables and up to $5|\Omega| + 2$ constraints.

$$\text{Min } f = \sum_{i=1}^{|\Omega|} \left(D(A_i) \log \left(D(A_i) \right) + D(I_i \backslash A_i) \log \left(D(I_i \backslash A_i) \right) - D(A_i) \log \left(|A_i| \right) - D(I_i \backslash A_i) \log \left(|I_i \backslash A_i| \right) \right)$$

subject to

$$-\sum_{i=1}^{|\Omega|} D(A_i) + (1 - \epsilon) \leq 0 \tag{C1}$$

$$\forall\ i \in \{1, \ldots, |\Omega|\},\ -D(A_i) - D(I_i \backslash A_i) + \lambda \leq 0 \tag{C2}$$

$$\forall\ i \in \{1, \ldots, |\Omega|\},\ D(A_i) + D(I_i \backslash A_i) - \rho \leq 0 \tag{C3}$$

$$\forall\ i \in \{1, \ldots, |\Omega|\}, -D(A_i) \leq 0 \tag{C4}$$

$$\forall\ i \in \{1, \ldots, |\Omega|\}, -D(I_i \backslash A_i) \leq 0 \tag{C5}$$

$$\sum_{i=1}^{|\Omega|} \left(D(A_i) + D(I_i \backslash A_i) \right) - 1 = 0 \tag{C6}$$

$$\forall\ i \in \{1, \ldots, |\Omega|\},\ \text{if } |A_i| = 0, D(A_i) = 0 \tag{C7}$$

$$\forall\ i \in \{1, \ldots, |\Omega|\},\ \text{if } |I_i \backslash A_i| = 0, D(I_i \backslash A_i) = 0 \tag{C8}$$

In general we would like an algorithm to compute this problem numerically with clear bounds on runtime and a guarantee of convergence. We will provide such a method in Section 3.3.

3.3 Algorithm for Computing Maximum Entropy Distribution

It will first prove useful to us to show that our problem takes the form of a convex optimization problem. It is immediately clear that all of our inequality and equality constraints (C1-C8) are affine. This in turn implies that they are convex. We will now show that f is also a convex function.

Theorem 3.3.1. *f is convex.*

Proof. The sum of convex functions is convex, so we must simply prove that each function in the summation is convex. For any $i \in \{1, \ldots, |\Omega|\}$, the corresponding function in the summation is of the form,

$$g(D(A_i), D(I_i \backslash A_i)) = D(A_i) \log \left(D(A_i) \right) + D(I_i \backslash A_i) \log \left(D(I_i \backslash A_i) \right) - D(A_i) \log \left(|A_i| \right) - D(I_i \backslash A_i) \log \left(|I_i \backslash A_i| \right)$$

Repeating this step again, we see that g is again composed of a summation of convex functions. It is easy to see that $(x \log(x))'' = \frac{1}{x}$. As $\frac{1}{x}$ is positive for any $x \in [0, 1]$, our domain, $x \log(x)$ is a convex function over our domain. The two other components are affine, and therefore trivially convex. Thus g is a convex function for any i, and as f is a sum of convex functions, f is also convex. ■

We can now present a formal procedure for calculating the Maximum Entropy Labelled Control Improvisation.

Theorem 3.3.2. *Given $|I_i \backslash A_i|$ and $|A_i|$ for all $i \in \{1, \ldots, |\Omega|\}$ for a MELCI instance $\mathcal{C}$, there is an algorithm to compute a MELCI improvising distribution within τ of the maximum entropy distribution or show that no feasible point exists in time polynomial in the size of $|\Omega|$ and $\log(\frac{1}{\tau})$.*

Proof. Looking at the procedure described in Theorem 2.2.2 where we lay out our MELCI optimization problem, we can see that the only inputs required are $|I_i \backslash A_i|$ and $|A_i|$ for all $i \in \{1, \ldots, |\Omega|\}$. As we know these quantities by assumption, we have all the inputs required for the specified procedures.

We must first determine whether or not our MELCI instance is feasible. To do this, we must first observe that a MELCI improvising distribution has all the constraints of an LCI distribution with the exception of the Randomness over Words constraint. Thus from our MELCI instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, L, m, n, \epsilon, \lambda, \rho, \tau)$, we can create a LCI instance $\mathcal{C}' = (\mathcal{H}, \mathcal{S}, L, m, n, \epsilon, \lambda, \rho, \hat{\alpha}, \hat{\beta})$ where $\hat{\alpha} = (0, \ldots, 0)$ and $\hat{\beta} = (1, \ldots, 1)$. As any distribution satisfies the Randomness over Words constraint for this choice of $\hat{\alpha}$ and $\hat{\beta}$, it follows that $\mathcal{C}$ is feasible if and only if $\mathcal{C}'$ is feasible. Thus we can use the Theorem 2.2.2 to arrive at a feasible distribution for both $\mathcal{C}$ and $\mathcal{C}'$ or determine that both problems are infeasible in polynomial time. Going forward we will assume the problem is feasible.

We will now present an algorithm to compute a MELCI improvising distribution within τ of the maximum entropy distribution using the procedure laid out by Chubanov [9]. To use this method, we must first show that our optimization problem takes the form of an objective function that is a separable convex function and entirely linear constraints. We have already shown convexity in Theorem 3.3.1, and using the same argument in that theorem we can see that f is clearly separable into convex functions of one variable. We can also clearly see that all our constraints are linear. We must also show that we can compute our objective function in polynomial time, but this is again trivial. As our problem meets the required specification and we have an an initial feasible point as computed via the LCI construction above, we can use Chubanov's method to solve our problem. The resulting time complexity is polynomial in the size of $|\Omega|$, as both the number of variables and the number of constraints is bounded by this number, and $\log(\frac{1}{\tau})$. The resulting approximate distribution is guaranteed to be feasible and have entropy at most τ less the maximum entropy improvising distribution. ■

Chapter 4

Time Complexity for LCI and MELCI Problems

In this section, we will explore the time complexities of the LCI and MELCI problems for different choices of constraint and labelling specifications. We will provide upper bounds on the complexity of a broad group of feasible problems, and provide more specific bounds for several groups of general specification choices.

4.1 Definitions

We begin this section by defining a specification, which is any formalism that defines a language of words over an alphabet. Some examples might be DFAs, NFAs, CFGs, etc... In both LCI and MELCI, we take in a variety of parameters, but the three with which we will concern ourselves in this section are $\mathcal{H}$, $\mathcal{S}$, L as they will determine the time complexity of our problem. We use the notation, LCI(SPEC_1, SPEC_2, SPEC_3) and MELCI(SPEC_1, SPEC_2, SPEC_3), where SPEC_1, SPEC_2, and SPEC_3 are classes of specifications. In our notation, LCI(SPEC_1, SPEC_2, SPEC_3) and MELCI(SPEC_1, SPEC_2, SPEC_3) imply a problem instance where $\mathcal{H}$ is encoded as an element of SPEC_1, $\mathcal{S}$ is encoded as an element of SPEC_2, and L can be decomposed into indicator functions of the class SPEC_3 for each label.

4.2 Construction of Labelling Function

In our notation above, we have left the format of the labelling function L rather abstract. We do this because we lack a sufficiently general framework for labelling functions. Instead we assume in general that they can be decomposed in polynomial time into polynomially many labels in the size of L, and treat them by the decision specifications they decompose into. In this section however, we highlight a some possible choices of labelling functions that preserve these properties and decompose into usable specifications.

- **DFA:** A labelling function that decomposes into DFAs might simply be a DFA along with a mapping that links each accepting state to a label. First note that there are clearly only polynomially many labels in the size of L, as there can be at most one label per accepting state. In addition, we can decompose L into all label indicator functions in polynomial time by copying over the DFA for each label, keeping only the accepting states corresponding to the current label and setting all other accepting states to non accepting states.
- **UCFG:** In this case, the simplest example is one in which we bundle multiple UCFGs into one function, each labelling the words they accept. However, in this case it is essential that no word is accepted by more than one of the UCFGs, as otherwise it is an improperly defined labelling function.

In general, we assume there are reasonable labelling functions that support decomposition in the ways we describe below.

4.3 Essential Operations

We can now generalize the process of solving the LCI and MELCI problems. First we must compute $|I_i|$ and $|I_i \backslash A_i|$ based off of the three specifications. Using these computed values, we can calculate our distribution over the I_i and $I_i \backslash A_i$ sets using the LCI construction, in Theorem 2.2.2, or the MELCI algorithm, in Section 3.3. Finally, we use this distribution to pick a I_i or $I_i \backslash A_i$ set, and then sample uniformly at random over that set.

We can now layout the operations that our specifications must support for us to be able to follow the framework laid out above.

Definition 4.3.1. (Required Specification Operations)

1. **Labelling Function Decomposition:** Given a labelling function $L : \Sigma^* \to \Omega$, compute the indicator function of the class SPEC_3 for each label in Ω. In other words, for each label $\ell_i \in \Omega$ we must be able to compute a specification $L_i : \Sigma^* \to \{\top, \bot\}$, belonging to the class SPEC_3, such that L_i accepts a word $w \in \Sigma^*$ if and only if $L(w) = \ell_i$.
2. **Counting:** Given specifications $\mathcal{H} \in \text{SPEC}_1, \mathcal{S} \in \text{SPEC}_2, L_i \in \text{SPEC}_3$ and $m, n \in \mathbb{N}$ in unary, compute $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(L_i)} \cap \Sigma^{m:n}|$ and $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(L_i) \cap \Sigma^{m:n}|$.
3. **Uniform Sampling:** Given specifications $\mathcal{H} \in \text{SPEC}_1, \mathcal{S} \in \text{SPEC}_2, L_i \in \text{SPEC}_3$ and $m, n \in \mathbb{N}$ in unary, sample uniformly at random from $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(L_i)} \cap \Sigma^{m:n}$ and $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(L_i) \cap \Sigma^{m:n}$.

If we can implement the above operations efficiently, we can solve the LCI and MELCI problems efficiently. In general, as the space of possible labelling functions is rather abstract, we will assume *Labelling Function Decomposition* in most of our Theorems. We will formalize this in Theorem 4.4.1

4.4 Polynomial Relative to an Oracle Time Improvisation Schemes

We can now show an important result relating the time complexity of an improvisation scheme to the complexity of implementing the above operations.

Theorem 4.4.1. *(Polynomial Oracle Time Complexity) Suppose we have an instance of* $\text{LCI}(\text{SPEC}_1, \text{SPEC}_2, \text{SPEC}_3)$ *or* $\text{MELCI}(\text{SPEC}_1, \text{SPEC}_2, \text{SPEC}_3)$*, in which all specifications support the relevant operations described in Definition 4.3.1. Suppose further that the operations can be done in polynomial time relative to an oracle $\mathcal{O}$ (expected time for uniform sampling) and that $|\Omega|$ is polynomial in the size of L. Then there is an improvisation scheme that is polynomial-time relative to $\mathcal{O}$ for* $\text{LCI}(\text{SPEC}_1, \text{SPEC}_2, \text{SPEC}_3)$ *or* $\text{MELCI}(\text{SPEC}_1, \text{SPEC}_2, \text{SPEC}_3)$*.*

Proof. We have already shown and proven correct polynomial time distribution constructions for LCI (Theorem 2.2.2) and MELCI (Section 3.3). However, these constructions assume we have access to $|I_i \backslash A_i|$ and $|A_i|$ for all $i \in \{1, \ldots, |\Omega|\}$. Therefore, our first step is calculating these values.

Our first step is to compute the L_i specifications from L. By assumption we know that there will only be polynomially many label indicator functions with respect to the size of L. Again by assumption, we can now compute $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(L_i)} \cap \Sigma^{m:n}|$ and $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(L_i) \cap \Sigma^{m:n}|$ for each i. Note that by definition, $|I_i \backslash A_i| = |\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(L_i)} \cap \Sigma^{m:n}|$ and $|A_i| = |\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(L_i) \cap \Sigma^{m:n}|$. We assume the label function decomposition takes polynomial time relative to $\mathcal{O}$, and that the set counting operation takes polynomial time relative to $\mathcal{O}$, for each label of which there are polynomially many. By the closure of polynomials, this means that all these calculations take polynomial time relative to $\mathcal{O}$.

Using these values, we can compute either the LCI or MELCI distribution using the appropriate construction, in polynomial time of the input. As the input is bounded by the number of labels, which is polynomial, computing the distribution must also take polynomial time.

Finally, we must select a I_i or $I_i \backslash A_i$ set for some i using our calculated distribution. We then sample uniformly over our selected class, which takes polynomial time relative to $\mathcal{O}$ by assumption.

As each of the three stages of our improvisation scheme take polynomial time relative to $\mathcal{O}$, our improvisation scheme as a whole takes polynomial time relative to $\mathcal{O}$. ■

4.5 Upper Bound on Time Complexity for Improvisation Schemes

Using Theorem 4.4.1, we can now establish an upper bound on the time complexity of most problems we will encounter. Specifically, we can bound the time complexity of our improvisation scheme in cases were membership in the language of a SPEC, or label assignment if it is a labelling function, can be decided in polynomial time relative to a PH oracle and the number of labels is bounded polynomially in the size of the input.

Theorem 4.5.1. *(Upper Bound on Time Complexity for Improvisation Scheme) Suppose that membership in a language $\mathcal{X} \in \text{SPEC}_1 \cup \text{SPEC}_2$ can be decided in polynomial time relative to a $\mathbb{PH}$ oracle. In addition, suppose given $\mathcal{Y} \in \text{SPEC}_3$, the label assigned to a word by $\mathcal{Y}$ can be computed in polynomial time relative to a $\mathbb{PH}$ oracle and that there are at most polynomially many labels in the size of* SPEC_3. *Then* $\text{LCI}(\text{SPEC}_1, \text{SPEC}_2, \text{SPEC}_3)$ *and* $\text{MELCI}(\text{SPEC}_1, \text{SPEC}_2, \text{SPEC}_3)$ *have improvisation schemes that are both polynomial time relative to a $\#\mathbb{P}$-hard oracle.*

Proof. By assumption there is a polynomial time algorithm relative to a $\mathbb{PH}$ oracle, $\text{M}^{\text{PH}}(\text{w}, \mathcal{X})$, that for $\mathcal{X} \in SPEC_1 \cup \text{SPEC}_2$ decides whether $w \in \mathcal{L}(\mathcal{X})$. We also assume there is a polynomial time algorithm relative to a $\mathbb{PH}$ oracle, $\text{N}^{\text{PH}}(\text{w}, \mathcal{Y})$, that for $\mathcal{Y} \in \text{SPEC}_3$, returns the label assigned to w by $\mathcal{Y}$. Because of this, we can define I_i, A_i, and $I_i \backslash A_i$ in terms of our assumed algorithms,

$$I_i = \{w \in \Sigma^* \mid m \leq |w| \leq n \wedge \text{M}^{\text{PH}}(w, \mathcal{H}) = 1 \wedge \text{N}^{\text{PH}}(w, \mathcal{L}) = \ell_i\}$$
$$A_i = I_i \cap \{w \in \Sigma^* \mid \text{M}^{\text{PH}}(w, \mathcal{S}) = 1\}$$

Note that all words in I_i and A_i have length at most n, which means we can verify length in time polynomial of the size of the input $\mathcal{C}$, as the length bounds are input in unary. Combining this with the knowledge that $\text{M}^{\mathbb{PH}}(\text{w}, \mathcal{X})$ and $\text{N}^{\mathbb{PH}}(\text{w}, \mathcal{X})$ both have polynomial runtime relative to a $\mathbb{PH}$ oracle, we can compute whether a word is in I_i or A_i in polynomial time with respect to a $\mathbb{PH}$ oracle. We can then define three classes of relations, which are each defined for any i,

$$R_{I_i} = \{(C, w) \mid w \in I_i\}$$
$$R_{A_i} = \{(C, w) \mid w \in A_i\}$$
$$R_{I_i \backslash A_i} = \{(C, w) \mid w \in I_i \backslash A_i\}$$

We have already shown that the first two relations can be verified in polynomial time with a $\mathbb{PH}$ oracle, and it is clearly seen that the third can be also be verified in polynomial time with a $\mathbb{PH}$ oracle by computing whether w is in I_i and not A_i. Because of this, all of these relations are $\mathbb{NP}^{\mathbb{PH}}$-relations, of which there are polynomially many. Thus it follows that computing $|I_i|$, and $|I_i \backslash A_i|$ are $\#\mathbb{P}^{\mathbb{PH}}$ problem. Using the relativized algorithms of Jerrum et al. [10] or Bellare et al [11], we can also sample uniformly from these sets in polynomial expected time relative to a $\#\text{P}^{\text{PH}}$ oracle. We can now use a process similar to the one outlined in Theorem 4.4.1 to show that there is an improvisation scheme for $\text{LCI}(\text{SPEC}_1, \text{SPEC}_2, \text{SPEC}_3)$ and $\text{MELCI}(\text{SPEC}_1, \text{SPEC}_2, \text{SPEC}_3)$ that is polynomial time relative to a $\#\mathbb{P}^{\mathbb{PH}}$ oracle (As we must complete two operations that take polynomial time relative to $\#\mathbb{P}^{\mathbb{PH}}$, and one operation that takes polynomial time).

From this point we can use the results established in the second half of [3, (Theorem 3.3)], to further reduce this scheme to simply using a $\#\mathbb{P}$ oracle. Thus we have derived an improvisation scheme for LCI or MELCI that is polynomial-time relative to a $\#\mathbb{P}$ oracle. ■

With these results, we have established an upper bound on the time complexity of most practical problems we will encounter.

4.6 Time Complexity for Choice of Specification

In this section, we will look at various choices of specification, and the resulting time complexity.

Theorem 4.6.1. *(Time Complexity with Exclusively DFA Specifications)*
If L supports Labelling Function Decomposition in polynomial time and $|\Omega|$ is bounded polynomially by the size of L, $\text{LCI}(\text{DFA},\text{DFA},\text{DFA})$ *and* $\text{MELCI}(\text{DFA},\text{DFA},\text{DFA})$ *have polynomial time improvisation scheme.*

Proof. By assumption, we can decompose in polynomial time L into $|\Omega|$ DFAs L_i that accepts a word $w \in \Sigma^*$ if and only if $L(W) = \ell_i$. In addition, we assume that the number of labels is bounded polynomially by the size of the input.

As all our specifications are DFAs, we must now show that DFAs support the *Counting* and *Uniform Sampling* operations. We will first show that we can find DFA specifications that have the language $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(L_i)}$ and $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(L_i)$ for each i. This follows simply from using the standard intersection and negation constructions, all of which take polynomial time in the size of the input DFAs. As DFAs are closed under all these constructions, we must merely show that DFAs support the *Counting* and *Uniform Sampling* operations.

1. **Counting:** Here, we can use the dynamic programming algorithm of Hickey and Cohen [12], using the counting the algorithm does as a preliminary step to its uniform sampling. We can use this algorithm to count, for any c, $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(L_i)} \cap \Sigma^c|$ or $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(L_i) \cap \Sigma^c|$ in polynomial time. Repeating this and summing the results for any $c \in \{m, \ldots, n\}$, the number of which is bounded polynomially, we arrive at $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(L_i)} \cap \Sigma^{m:n}|$ or $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(L_i) \cap \Sigma^{m:n}|$, in polynomial time.
2. **Uniform Sampling:** Here again we use the dynamic programming algorithm of Hickey and Cohen [12] to sample uniformly over the words accepted by a DFA. As laid out in [3, (Theorem 3.4)], we calculate the number of words of each possible length. We then pick a word length to sample uniformly over with weight proportional to that classes share of the total number of words. Thus we can sample uniformly over all words accepted by $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \overline{\mathcal{L}(L_i)}$ or $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(L_i)$ of the correct length in polynomial time.

We have shown that we can compute all the operations laid out in Theorem 4.4.1 in polynomial time (we can add a polynomial oracle for the sake of completeness, but $\mathbb{P}^{\mathbb{P}} = \mathbb{P}$ so it is redundant). Thus we have a polynomial time improvisation scheme for LCI(DFA,DFA,DFA) and MELCI(DFA,DFA,DFA). ■

Theorem 4.6.2. *(Time Complexity with one UCFG Specification and two DFAs Specifications) Let exactly one of* SPEC_1, SPEC_2, SPEC_3 *be the class of UCFGs, and the remaining two specifications be the class of DFAs. If L supports Labelling Function Decomposition in polynomial time and* $|\Omega|$ *is bounded polynomially by the size of L,* LCI(DFA,DFA,DFA) *and* MELCI(DFA,DFA,DFA) *have polynomial time improvisation scheme.*

Proof. By assumption, we can decompose, in polynomial time, L into $|\Omega|$ DFAs or UCFGs L_i that accept a word $w \in \Sigma^*$ if and only if $L(W) = \ell_i$ (Depending on whether SPEC_3 is the class of DFAs or UCFGs). In addition, we assume that the number of labels is bounded polynomially by the size of the input. We will now show that we can perform the *Counting* and *Uniform Sampling* operations in polynomial time.

1. **Counting:** We must first determine $|I_i|$ and $|A_i|$. We can compute UCFGs with the languages $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(L_i)$ and $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(L_i)$ for each i. We can do this in polynomial time using the algorithm described in [3, Lemma 3.1]. Then using the counting technique from Hickey and Cohen [12] to compute I_i and A_i for each i. From there, we can determine that $|I_i \backslash A_i| = |I_i| - |A_i|$. Thus we can compute both $|I_i \backslash A_i|$ and $|A_i|$ in polynomial time.
2. **Uniform Sampling:** To sample uniformly from each I_i is simple, as we can simply use the algorithms of Hickey and Cohen [12] to do so in polynomial time. To sample uniformly from $I_i \backslash A_i$ is significantly more intricate, but we can still do so in polynomial time using the procedure outlined in [3, Theorem 3.8]. We must simply take the intersection of our two DFA specifications, which results in an output DFA. We then we can follow the technique for sampling the intersection of a DFA and a UCFG outlined in [3, Theorem 3.8].

We have shown that we can compute all the operations laid out in Theorem 4.4.1 in polynomial time. Thus we have a polynomial time improvisation scheme for cases where there is one UCFG specification and two DFA specifications. ■

Theorem 4.6.3. *(Time Complexity with two UCFG Specifications and one DFA Specification) Let exactly two of* SPEC_1, SPEC_2, SPEC_3 *be the class of UCFGs, and the remaining specification be the class of DFAs.* LCI*(*SPEC_1, SPEC_2, SPEC_3*) and* MELCI*(*SPEC_1, SPEC_2, SPEC_3*) are both #ℙ-hard .*

Proof. To prove this for the LCI case, we will reduce #UCFG-Int [3, Definition 3.7], which we know to be #ℙ-hard [3, Lemma 3.2], to solving for the feasibility of LCI(SPEC_1, SPEC_2, SPEC_3).

Consider an instance of #UCFG-Int, $(\mathcal{H}, \mathcal{S}, 1^n)$, where we must compute $|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|$. Let L decompose into the trivial DFA that accepts all words. Consider the LCI instance $\mathcal{C}_N = (\mathcal{H}, \mathcal{S}, L, 0, n, 0, 0, 1, 0, \frac{1}{N})$. Note that this LCI instance samples over all words of length less than or equal to n, that are accepted by $\mathcal{H}$, $\mathcal{S}$. As there is only label that accepts all words and it has trivial randomness constraints, $\mathcal{C}_N$ is only feasible if there are more than N words in

$\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \Sigma^{\leq n}$. As described in [3, Theorem 3.9], we can now perform a binary search type operation to exactly compute $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \Sigma^{\leq n}|$ in polynomial time.

Now, we can make the observation that it does not matter which specifications are associated with the hard and soft constraints or labelling function. This is because we sample over the intersection of the languages of our three specifications, as there is only one label and our ϵ is set to 0. As long as one of them is the trivial DFA that accepts all words and the other two are the UCFGs from our #UCFG-Int, we get the same result. As we have provided a clearly polynomial time reduction, it follows that LCI(SPEC_1, SPEC_2, SPEC_3) is #$\mathbb{P}$-hard .

To prove this for the MELCI case, we will perform a similar reduction from #UCFG-Int to solving for a MELCI(SPEC_1, SPEC_2, SPEC_3) with at least a certain entropy. Again consider an instance of #UCFG-Int, $(\mathcal{H}, \mathcal{S}, 1^n)$, where we must compute $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \Sigma^{\leq n}|$. Through Lemma 3.2.1, we know that the maximum entropy a distribution of size s can have is $\log(s)$ (ignoring choice of log basis), achievable through the uniform distribution over all elements of I. Thus if the entropy of a distribution over a set is greater than $\log(s)$, it follows that there must be more than s elements in that set. Let L decompose into the trivial DFA that accepts all words. Consider the MELCI instance $\mathcal{C} = (\mathcal{H}, \mathcal{S}, L, 0, n, 0, 0, 1, \tau)$. As there is only one label with trivial randomness constraints, $\mathcal{C}_N$ is feasible no matter what. As $\epsilon = 0$, our improviser must sample exclusively over strings accepted by $\mathcal{H}$ and $\mathcal{S}$ (and of course the trivial DFA created by L). Recall that part of the MELCI problem is returning the entropy of the maximum entropy distribution calculated. As there are no non-trivial randomness constraints and the soft constraint must be satisfied all the time by our ϵ parameter choice, the maximum distribution is clearly a uniform distribution over $|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|$.

Using the above and the problem definition, we know that the returned entropy will be in the range $[\log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|) - \tau, \log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|)]$ which means that it will clearly be greater than or equal to $\log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|) - \tau$. As we seek to transform this max entropy into the size of the set, we must now pick τ such that $e^{\log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|) - \tau} \geq e^{\log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n} - 1|)}$ which allows us to determine exactly how large $|\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{S}) \cap \Sigma^{\leq n}|$ is simply by rounding up $\log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|) - \tau$ to the next integer. We show how to do this below,

$$
\begin{aligned}
e^{\log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|) - \tau} &> e^{\log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}| - 1)} \\
e^{\log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|)} e^{-\tau} &> e^{\log(|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}| - 1)} \\
|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}| e^{-\tau} &> |\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}| - 1 \\
e^{-\tau} &> \frac{|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}| - 1}{|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|} \\
-\tau &> \log\left(\frac{|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}| - 1}{|\mathcal{H} \cap \mathcal{S} \cap \Sigma^{\leq n}|}\right)
\end{aligned}
$$

We now simplify τ to something easily computable.

$$
\begin{aligned}
\tau &< -\log\left(\frac{|\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|-1}{|\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|}\right)\\
&= \log\left(\frac{|\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|}{|\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|-1}\right) \qquad \text{(Assume } |\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|\geq 2\text{)}\\
&\leq \log\left(\frac{\sum_{i=1}^{n}|\Sigma|^n}{\sum_{i=1}^{n}|\Sigma|^n-1}\right)\\
&= \log\left(\frac{\frac{1-|\Sigma|^{n+1}}{1-|\Sigma|}}{\frac{1-|\Sigma|^{n+1}}{1-|\Sigma|}-1}\right)\\
&= \log\left(\frac{1-|\Sigma|^{n+1}}{1-|\Sigma|^{n+1}-1+|\Sigma|}\right)\\
&= \log\left(\frac{1-|\Sigma|^{n+1}}{1-|\Sigma|^{n+1}-1+|\Sigma|}\right)\\
&= \log\left(\frac{1-|\Sigma|^{n+1}}{|\Sigma|-|\Sigma|^{n+1}}\right)\\
&= \log\left(\frac{|\Sigma|^{n+1}-1}{|\Sigma|^{n+1}-|\Sigma|}\right)
\end{aligned}
$$

Note that we assume $|\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|\geq 2$. If $|\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|=0$, we will never use τ as there are no improvising distributions for $MELCI$, so the we will return that the problem is not feasible. We account for the case where $|\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|=1$ there exists only one feasible distribution which is over exactly one element, thus meaning it has a maximum entropy of 0. As a distribution over two elements has a maximum entropy of log(2) we can simply set $\tau=\frac{1}{2}\min(\log(2),\log\left(\frac{|\Sigma|^{n+1}-1}{|\Sigma|^{n+1}-|\Sigma|}\right))$, ensuring that that we will always generate a distribution with non-zero entropy if $|\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|>1$. Thus with our choice of τ, we can always compute the entropy of our constructed MELCI problem, raise e to its power, and round it to the next highest number, resulting in the exact size of $|\mathcal{H}\cap\mathcal{S}\cap\Sigma^{\leq n}|$. Note that even without the fact that our algorithm relies only logarithmically on τ, we still remain in polynomial time for our computation as the number of bits needed to represent τ is bounded polynomially in the size $|\Sigma|$.

Making the same observation as above regarding which specifications are associated with the hard and soft constraints or labelling function, we can see that this process is easily generalized for any assignment of specifications where two specifications are UCFGs and one is a DFA. As all of this can clearly be done in polynomial time, we have a polynomial time reduction from #UCFG-Int to MELCI(SPEC_1, SPEC_2, SPEC_3) and thus MELCI(SPEC_1, SPEC_2, SPEC_3) is #$\mathbb{P}$-hard . ■

Theorem 4.6.4. *(Time Complexity with Exclusively UCFG Specifications)* LCI*(*SPEC_1*,* SPEC_2*,* SPEC_3*) and* MELCI*(*SPEC_1*,* SPEC_2*,* SPEC_3*) are* #$\mathbb{P}$*-hard .*

Proof. We will show this by reducing LCI(SPEC_1, SPEC_2, SPEC_3) and MELCI(SPEC_1, SPEC_2, SPEC_3) to LCI(UCFG, UCFG, UCFG) and MELCI(UCFG, UCFG, UCFG) where exactly two of SPEC_1, SPEC_2, SPEC_3 are the class of UCFGs, and the remaining specification is the class of DFAs. We will do this by noting that all DFAs can be converted to unambiguous regular grammars, which are in fact unambiguous context free grammars. Thus we can reduce any instance of LCI(SPEC_1, SPEC_2, SPEC_3) and MELCI(SPEC_1, SPEC_2, SPEC_3) to LCI(UCFG, UCFG, UCFG) and MELCI(UCFG, UCFG, UCFG) by encoding the DFA specification as a UCFG. As we have provided a clearly polynomial time reduction, it follows that LCI(UCFG, UCFG, UCFG) and MELCI(UCFG, UCFG, UCFG) are #$\mathbb{P}$-hard . ■

Theorem 4.6.5. *(Time Complexity with DFA Specifications and one or more NFA Specifications) Let one or more of* SPEC_1*,* SPEC_2*,* SPEC_3 *be the class of NFAs, and the remaining specification be the class of DFAs. Then* LCI*(*SPEC_1*,* SPEC_2*,* SPEC_3*) and* MELCI*(*SPEC_1*,* SPEC_2*,* SPEC_3*) are* #$\mathbb{P}$*-hard .*

Proof. As shown by Kannan et al. [13], determining $|\mathcal{L}(\mathcal{M})\cap\Sigma^l|$ for an NFA $\mathcal{M}$ over an alphabet Σ and $l\in\mathbb{N}$ in unary is #$\mathbb{P}$-hard . We will provide a polynomial time reduction from this problem to LCI(SPEC_1, SPEC_2, SPEC_3). Define $\mathcal{T}$

as the trivial DFA that accepts all words and define a labelling function L to be a function that is decomposed into a single DFA mapping all words to a single label ℓ. For some $N \in \mathbb{N}$, consider an LCI instance $\mathcal{C}_N = (\mathcal{M}, \mathcal{T}, L, l, l, 0, 0, 1, 0, \frac{1}{N})$. This LCI instance generates only words in $\mathcal{L}(M) \cap \mathcal{L}(T) = \mathcal{L}(M)$ and of length l as $\epsilon = 0$ and $n = m = l$. As no word can have more than $\frac{1}{N}$ probability of being generated, it follows that our LCI instance is satisfiable if and only if $|\mathcal{L}(M) \cap \Sigma^l| \geq N$. Using this, like in Theorem 4.6.3, we can perform a binary search to find the solution to $|\mathcal{L}(M) \cap \Sigma^l| \geq N$ in polynomial time.

We will now provide a similar reduction from determining $|\mathcal{L}(\mathcal{M}) \cap \Sigma^l|$ for an NFA $\mathcal{M}$ over an alphabet Σ and $l \in \mathbb{N}$ in unary to MELCI(SPEC$_1$, SPEC$_2$, SPEC$_3$). Define a labelling function L to be a function that is decomposed into a single DFA mapping all words to a single label ℓ. Consider a MELCI instance $\mathcal{C} = (\mathcal{M}, \mathcal{T}, L, l, l, 0, 0, 1, \tau)$. As in Theorem 4.6.3 the labelling constraints are trivial and apply to only one label, meaning we can ignore them. Also, as $\epsilon = 0$, we only consider strings also in $\mathcal{T}$. Thus our distribution will sample exclusively over strings in $\mathcal{L}(\mathcal{M}) \cap \Sigma^l$. Using the arguments made in Theorem 4.6.3, we can pick an appropriate τ, compute the entropy, bring e to its power, and round up to the nearest integer to exactly compute $|\mathcal{L}(\mathcal{M}) \cap \Sigma^l|$. The only difference would be that instead of having to create an upper bound accounting for every string in Σ of length less than or equal to l, here we must only consider Σ^l in our bounds. Again we can clearly do this in polynomial time as the number of bits needed to represent τ is again bounded polynomially and our algorithm terminates in polynomial time.

Much like in Theorem 4.6.3, which specifications are associated with the hard/soft constraints or labelling function does not matter, as long as one of them is an NFA and the rest are DFAs. As we have provided a polynomial time reduction, it follows that LCI(SPEC$_1$, SPEC$_2$, SPEC$_3$) and MELCI(SPEC$_1$, SPEC$_2$, SPEC$_3$) are #$\mathbb{P}$-hard .

We can also note that since any DFA is also an NFA, the above instance can be reduced to any LCI or MELCI instance with more NFA specifications replacing the current DFA specifications. This means that any LCI or MELCI instance with one or more NFA specifications with all remaining specifications being DFAs is a #$\mathbb{P}$-hard problem, which generalizes our proof to the theorem stated above. ■

Theorem 4.6.6. *(Time Complexity with DFA Specifications and one or more CFG Specifications) Let one or more of* SPEC$_1$, SPEC$_2$, SPEC$_3$ *be the class of CFGs, and the remaining specification be the class of DFAs. Then* LCI(SPEC$_1$, SPEC$_2$, SPEC$_3$) *and* MELCI(SPEC$_1$, SPEC$_2$, SPEC$_3$) *are #$\mathbb{P}$-hard .*

Proof. We can show this by reducing an instance LCI(SPEC$_1$', SPEC$_2$', SPEC$_3$') and MELCI(SPEC$_1$', SPEC$_2$', SPEC$_3$'), where at least one of SPEC$_1$', SPEC$_2$' ,SPEC$_3$' is an NFA and the remaining specifications are DFAs, to instances of LCI(SPEC$_1$, SPEC$_2$, SPEC$_3$) and MELCI(SPEC$_1$, SPEC$_2$, SPEC$_3$). This can be done trivially as any regular language encoded by an NFA can be represented by an equivalent CFG. As we have provided a clearly polynomial time reduction, it follows that LCI(SPEC$_1$, SPEC$_2$, SPEC$_3$) and MELCI(SPEC$_1$, SPEC$_2$, SPEC$_3$) must both be #$\mathbb{P}$-hard . ■

Below, we present Tables 4.1, 4.2, and 4.3 summarizing the time complexity results above.

SPEC$_1$ \ SPEC$_2$	DFA	UCFG	CFG	NFA
DFA	$\mathbb{P}$	$\mathbb{P}$		
UCFG	$\mathbb{P}$	#$\mathbb{P}$		
CFG			#$\mathbb{P}$	
NFA				#$\mathbb{P}$

Table 4.1: SPEC$_3$ = DFA

SPEC$_1$ \ SPEC$_2$	DFA	UCFG	CFG	NFA
DFA	$\mathbb{P}$			
UCFG		#$\mathbb{P}$		
CFG			#$\mathbb{P}$	
NFA				#$\mathbb{P}$

Table 4.2: SPEC$_3$ = UCFG

$SPEC_1$ \ $SPEC_2$	DFA	UCFG	CFG	NFA
DFA	$\#\mathbb{P}$			
UCFG		$\#\mathbb{P}$		
CFG			$\#\mathbb{P}$	
NFA				$\#\mathbb{P}$

Table 4.3: $SPEC_3$ = CFG/NFA

4.7 Complexity of Motivating Examples

We will now determine the complexity of the motivating examples we laid out in Section 1.2.

4.7.1 Robotic Planning

We begin by declaring our alphabet Σ as the possible movements in each direction or lack thereof, {North, East, South, West, Stop}. We can encode our hard constraints as a DFA where each tile is a state, with transitions accounting for terrain rules. We can then copy these into a tree like structure where once the robot has visited an objective, it must visit one of the remaining objectives to move to a copy deeper down the tree. The start state will be the root base state, and the accepting states are the base states in the leaves of this tree, which are only reachable when the robot has visited all objectives and returned to base. Our soft constraint can be encoded by simply accepting any sequence of symbols of length 30 or less. Finally, our labelling function can be encoded as a DFA in a similar tree structure as the hard constraint, but this time only allowing movement down the tree by passing through an entrance/exit. We will ignore cases where a robot uses an entrance/exit multiple times for simplicity, so we can simply have a two level tree with the base state in each leaf being an accepting state annotated with the label of the entrance/exit the robot took to get there. This leaves us with a total of 9 labels.

As one can clearly see from the above, our robotic planning example takes the form LCI(DFA, DFA, DFA) or MELCI(DFA, DFA, DFA), implying we have a polynomial time improvisation scheme for it by Theorem 4.6.1.

4.7.2 Fuzz Testing

Here, we can define our hard constraint as our grammar that allows all valid inputs and perhaps some invalid inputs. We can satisfy the length requirement by setting $m = 0, n = 100$. Our soft constraint can be encoded by simply accepting any sequence of symbols of length 50 or less. For our labelling function, we could perhaps consider three key symbols that activate complicated functions in our program along with a label for functions with invalid symbols. We can then create 8 labels representing which of the symbols are in a generated word, along with a label indicating a program with symbols that do not exist in well formed programs. We can then create a relatively simple label function that recognizes these using a DFA with label annotated accepting states.

Based off the above, our fuzz testing problem takes the form LCI(UCFG, DFA, DFA) or MELCI(UCFG, DFA, DFA), implying we have a polynomial time improvisation scheme for it by Theorem 4.6.2.

Chapter 5

Conclusion

In this paper, we introduced Labelled Control Improvisation in the form of the Labelled Control Improvisation problem and the Maximum Entropy Labelled Control Improvisation problem, both of which allow us to include a labelling specification to label the improvisations we generate. In both problems, this allows us finer control over the randomness of our improviser. LCI gives us total control over all of the randomness requirements, while MELCI allows us to specify the desired marginal probability bounds over the label classes, and computes the maximum entropy improvising distribution that satisfies those bounds. We calculate the runtime for some general classes of specifications, providing upper and lower bounds on most possible choices. We also include several motivating examples showcasing the extra control we gain through Labelled Control Improvisation.

In future work, we plan to augment LCI further, replacing the soft constraint with a cost function. In all of our motivating examples, we measure some sort of cost that we would like to keep minimal within a certain tolerance. It is a natural extension of this goal to replace the soft constraint with a cost function that allows us more specific control over the "cost" of various improvisations. Using this extension, we hope to address the problem of Design Improvisation, involving generating different improvisations of various types, or labels, while keeping the expectation of the "cost" of the improvisations within a certain threshold.

Bibliography

[1] Daniel J. Fremont et al. "Control Improvisation". In: *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*. Ed. by Prahladh Harsha and G. Ramalingam. Vol. 45. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 463–474. ISBN: 978-3-939897-97-2. DOI: 10.4230/LIPIcs.FSTTCS.2015.463. URL: http://drops.dagstuhl.de/opus/volltexte/2015/5659.

[2] Daniel J. Fremont, Alexandre Donzé, and Sanjit A. Seshia. *Control Improvisation*. 2017. arXiv: 1704.06319 [cs.FL]. URL: https://arxiv.org/abs/1704.06319.

[3] Daniel J. Fremont. "Algorithmic Improvisation". University of California Berkeley, 2019. URL: https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-133.pdf.

[4] Alexandre Donze et al. "Machine Improvisation with Formal Specifications". In: 40th International Computer Music Conference (ICMC). Sept. 2014. URL: https://quod.lib.umich.edu/i/icmc/bbp2372.2014.196/1.

[5] Alexandre Donze et al. *Control Improvisation with Application to Music*. Tech. rep. UCB/EECS-2013-183. EECS Department, University of California, Berkeley, Nov. 2013. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-183.html.

[6] Daniel J. Fremont and Sanjit A. Seshia. "Reactive Control Improvisation". In: *Computer Aided Verification - 30th International Conference*. Ed. by Hana Chockler and Georg Weissenbacher. Vol. 10981. Lecture Notes in Computer Science. Springer, 2018, pp. 307–326. DOI: 10.1007/978-3-319-96145-3_17. URL: https://doi.org/10.1007/978-3-319-96145-3%5C_17.

[7] Marcell Vazquez-Chanlatte et al. *Entropy-Guided Control Improvisation*. 2021. arXiv: 2103.05672 [cs.RO].

[8] Joy M. Thomas Thomas M. Cover. *Elements of Information Theory*. 2006.

[9] Sergei Chubanov. "A Polynomial-Time Descent Method for Separable Convex Optimization Problems with Linear Constraints". In: *SIAM Journal on Optimization* 26.1 (2016), pp. 856–889. DOI: 10.1137/14098524x.

[10] Mark R. Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. "Random generation of combinatorial structures from a uniform distribution". In: *Theoretical Computer Science* 43 (1986), pp. 169–188. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(86)90174-X. URL: https://www.sciencedirect.com/science/article/pii/030439758690174X.

[11] Mihir Bellare, Oded Goldreich, and Erez Petrank. "Uniform Generation of NP-Witnesses Using an NP-Oracle". In: *Information and Computation* 163.2 (2000), pp. 510–526. ISSN: 0890-5401. DOI: https://doi.org/10.1006/inco.2000.2885. URL: https://www.sciencedirect.com/science/article/pii/S0890540100928852.

[12] Timothy Hickey and Jacques Cohen. "Uniform Random Generation of Strings in a Context-Free Language". In: *SIAM Journal on Computing* 12.4 (1983), pp. 645–655. DOI: 10.1137/0212044.

[13] Sampath Kannan, Z. Sweedyk, and Steve Mahaney. "Counting and Random Generation of Strings in Regular Languages". In: *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 1995, pp. 551–557. ISBN: 0898713498.