




# Randomized Synthesis for Diversity and Cost Constraints with Control Improvisation



Andreas Gittis, Eric Vin, and Daniel J. Fremont<sup>(✉)</sup> 

University of California, Santa Cruz, USA  
{agittis,evin,dfremont}@ucsc.edu



**Abstract.** In many synthesis problems, it can be essential to generate implementations which not only satisfy functional constraints but are also *randomized* to improve variety, robustness, or unpredictability. The recently-proposed framework of control improvisation (CI) provides techniques for the correct-by-construction synthesis of randomized systems subject to hard and soft constraints. However, prior work on CI has focused on qualitative specifications, whereas in robotic planning and other areas we often have quantitative quality metrics which can be traded against each other. For example, a designer of a patrolling security robot might want to know by how much the average patrol time needs to be increased in order to ensure that a particular aspect of the robot’s route is sufficiently diverse and hence unpredictable. In this paper, we enable this type of application by generalizing the CI problem to support quantitative soft constraints which bound the expected value of a given cost function, and randomness constraints which enforce diversity of the generated traces with respect to a given label function. We establish the basic theory of labelled quantitative CI problems, and develop efficient algorithms for solving them when the specifications are encoded by finite automata. We also provide an approximate improvisation algorithm based on constraint solving for any specifications encodable as Boolean formulas. We demonstrate the utility of our problem formulation and algorithms with experiments applying them to generate diverse near-optimal plans for robotic planning problems.

## 1 Introduction

Correct-by-construction synthesis of systems from high-level specifications has become a popular paradigm in fields ranging from circuit design [5] to robotic task planning [25]. Synthesis techniques for many different types of specifications have been developed, especially for temporal logic formulas, which can encode many properties of interest [14]. One less-studied type of specification are *randomness constraints* that require the system’s behavior to be sufficiently random, for instance by being close to a uniform distribution over the set of

---

A. Gittis and E. Vin—The two first authors contributed equally to the paper.

© The Author(s) 2022

S. Shoham and Y. Vizel (Eds.): CAV 2022, LNCS 13372, pp. 526–546, 2022.

[https://doi.org/10.1007/978-3-031-13188-2\\_26](https://doi.org/10.1007/978-3-031-13188-2_26)

allowed behaviors. Such specifications are useful in many applications, as randomness can provide robustness, variety, and unpredictability to a system. For example, fuzz testing tools often use constraints to select classes of inputs which are more likely to trigger bugs, but then search randomly within that class to prevent bias [29]. In robotic planning, a patrolling security robot that uses a fixed plan satisfying its requirements might be vulnerable to exploitation; adding randomness to make its route unpredictable can make exploitation more difficult.

While there has been substantial work on synthesis with stochastic environments (e.g. [2,9]), randomness constraints require the system itself to behave randomly even if the environment is deterministic. Furthermore, unlike most specifications used in synthesis, randomness constraints are properties not of individual behaviors but rather of their distribution, and they cannot be concisely encoded into existing specification formalisms like PCTL [22] and SGL [3]. As a result, synthesis of systems under such constraints requires new techniques.

A recently-proposed paradigm for the correct-by-construction synthesis of systems under randomness constraints is *algorithmic improvisation* [13,15,16]. Algorithmic improvisation comprises a class of synthesis problems whose goal is to construct a randomized algorithm, an *improviser*, satisfying three kinds of constraints: *hard constraints* that the improviser’s output must always satisfy, *soft constraints* that need only be satisfied to a certain (tunable) extent, and *randomness constraints* requiring the output to be sufficiently random. These types of constraints correspond to natural requirements arising for example in robot planning: the hard constraints can encode safety or other functional requirements, the soft constraints can encode notions of efficiency or optimality, and the randomness constraints enforce diversity or unpredictability. The original and most-studied form of algorithmic improvisation is the *control improvisation* (CI) problem (introduced in [12] and formalized in [16,17]), where the improviser generates finite sequences of symbols, the hard constraint is a trace property, the soft constraint requires some trace property hold with at least a desired probability, and the randomness constraint puts upper and lower bounds on the probability of individual outputs. Control improvisation and its extensions have been successfully used for musical improvisation [13], robotic planning [19], human modeling subject to constraints [1], and generating synthetic datasets for testing and training cyber-physical systems with machine learning components [18].

However, the prior work on CI is not general enough to cover many randomized synthesis problems of interest, for two reasons. First, many planning, design space exploration, and other problems come with a *cost function* expressing how optimal a particular solution is; in the setting of generating randomized solutions, the most natural soft constraint would be to require that the *expected cost* of a solution should be low, so that we can obtain a diverse set of near-optimal solutions. In a patrolling robot application, for example, the fastest patrol route might be unique and so predictable, and we then want to know by how much we would need to increase the average patrol time in order to enable a sufficiently-diverse set of routes. The prior work on CI cannot provide such an analysis.

Second, while the CI randomness constraint is sufficient to make the improviser’s exact output unpredictable, it is *not* sufficient to ensure diversity when many outputs are similar to each other. Continuing our patrolling robot example, suppose that the robot has a choice of two rooms to go through: one room is larger, and so there are (say)  $10^6$  possible paths through it, vs. only  $10^3$  through the other room. Even if a perfectly-uniform distribution over all these paths is possible given our other constraints, the robot will end up entering the larger room almost all of the time. But from the point of view of an adversary that wishes to avoid being seen by the robot, the exact path is not relevant: what matters is *which room* the robot will enter, and that is highly predictable. For this application, we need a randomness requirement that enforces diversity not over the output of the improviser, but over some *attribute* of the output.

To enable such applications, in this paper we introduce the concept of *Labelled Quantitative Control Improvisation* (LQCI). This problem extends CI with a soft constraint bounding the expected cost of generated traces, and a randomness constraint requiring near-uniformity of the *label* of a trace, given by an arbitrary label function. We study the theory of LQCI, establishing precise conditions for when an LQCI problem is solvable and a general construction for solving it. We use our construction to develop efficient improvisation algorithms for a broad class of specifications given by finite automata, including common cost functions such as mission time or path length. For specifications not easily encoded to (reasonably-sized) automata, we provide an approximate improvisation algorithm based on constraint solving that handles symbolic specifications encoded as Boolean formulas. We also explore an extension of the LQCI problem for finding the *maximum-entropy* distribution satisfying the other constraints (as in [30]), and develop an algorithm for solving it using convex optimization. Finally, we conduct a case study demonstrating that our approach allows us to formalize and solve realistic robotic planning problems.

In summary, the main contributions of this paper are:

- The labelled quantitative control improvisation problem definition (Sect. 2);
- A characterization of which LQCI problems are solvable, and a general construction for solving them (Sect. 3);
- Efficient improvisation algorithms for finite automata specifications (Sect. 4);
- An approximate algorithm for Boolean formula specifications (Sect. 5);
- An algorithm for maximum-entropy LQCI problems (Sect. 6);
- Experiments using our algorithms for robotic planning (Sect. 7).

We conclude in Sect. 8 with a summary of results and directions for future work. For brevity, we defer full proofs of all results to the Appendix [21].

## 2 Overview and Problem Definition

In this section we formally define the LQCI problem, first using applications to robotic planning and fuzz testing to motivate various aspects of our definitions. We will return to the robotic planning example for our experiments in Sect. 7.

## 2.1 Motivating Examples

**Robotic Planning.** Consider the problem of generating a path for a package delivery robot, where the robot should efficiently visit various drop-off points, visiting charging stations as necessary along the way. Discretizing the world into a grid, we can represent a path as a finite sequence of north, south, east, and west moves. We might have various requirements for such paths, falling into the three types of constraints of a control improvisation problem described above: hard constraints such as completing mission objectives and not navigating into impassable terrain, soft constraints such as preferring shorter paths, and randomness constraints to ensure the chosen path is unpredictable. However, as we saw in Sect. 1, randomness over paths can be less important than randomness over specific features of a path: here, it might be that charging leaves the robot vulnerable for an extended period, so that it is important to limit the extent to which an adversary can predict ahead of time which charging station will be used. If there are 3 charging stations, then all possible paths are divided into 3 classes, and we want the class of a generated path to be unpredictable; we can formalize this as a *label function* which assigns labels to paths, and require that the distribution over labels be close to uniform. Since we do not want to simply pick a single path from each label class, we can also enforce randomness within each class, either by bounding the conditional probabilities of paths (so that no path is too likely relative to others in its class) or by taking the maximum-entropy distribution that satisfies our randomness-over-labels condition (we will return to this approach in Sect. 6).

For efficiency, we want our robot to use routes which are as fast as possible, taking into account varying terrain. We could model this using a *cost function* assigning numerical costs to each path: here, the total time needed to traverse it. However, as mentioned in Sect. 1, prior work on CI can only encode Boolean soft constraints, such as requiring the cost of a path to be at most 5 with probability at least 0.9. While this does allow for some control over the cost, it requires setting an arbitrary threshold, and otherwise ignores the actual values of the cost; thus, a path of cost 6 is treated no differently than a path of cost  $10^5$ . Instead, we want to bound the *expected* cost of a path, so that both the probabilities of individual paths and their absolute costs are taken into account.

Putting all this together, we define our example planning problem as generating paths through the grid worlds in Fig. 1, subject to the following constraints:

### Hard Constraint:

- (a) The robot must begin in the start cell **S** and must end in the end cell **E**.
- (b) The robot must visit all package drop-off points **O**.
- (c) The robot must charge at a charging station **C**.
- (d) The robot must not enter impassable locations **X**.

### Cost Constraint:

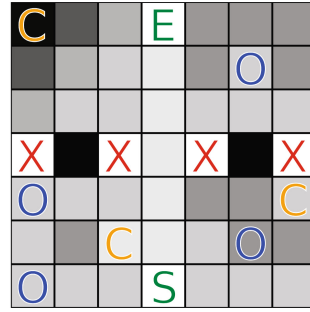
The expected time to complete the mission must be at most a constant  $c$ .

### Randomness over Labels:

For each choice of charging station, the chance that the robot uses that station must be at least  $\lambda$  and at most  $\rho$ .



(a) Small Grid World (6x6)



(b) Large Grid World (7x7)

**Fig. 1.** Grid worlds for our robotic planning example. Darker background indicates higher cost and letters indicate: start and end points (**S**, **E**), impassable locations (**X**), delivery locations (**O**), charging stations (**C**).

### Randomness over Words:

Conditioned on selecting a certain charging station, the probability of picking any path must be at least  $\alpha$  and at most  $\beta$ .

Here, we assume that each grid cell has a cost representing how long it takes to traverse, with the cost of a path (the total mission time) being the sum of the costs of its cells. In Fig. 1, we show higher-cost cells as being darker, with the costs ranging from 0–3 for the small world and 0–10 for the large world. The layout of the map was chosen to admit a variety of different paths, motivated as follows: we envision an impassible river dividing the top and bottom halves of the map, with one low-cost bridge and two high-cost fords. The top-left charging station is a windmill and requires climbing a hill to access; there is also a hydroelectric station next to the river, and an easily-accessible substation near the main north-south road.

**Fuzz Testing.** Prior work has shown that a variety of programs and protocols can be comprehensively tested by randomly sampling from automata encoding constraints on acceptable tests [11]. LQCI allows us to preserve such guarantees while exercising additional control over which tests are generated.

As an example, consider the problem of generating randomized network activity for a set of devices communicating over TCP; this could be useful to test robustness of a network monitoring application or network stack. There are a variety of different constraints we might wish to impose on the sequences of packets we generate: each connection should conform to the TCP protocol, so that the tests are meaningful<sup>1</sup>; tests should exhibit a variety of different behaviors

<sup>1</sup> We might also want to generate tests that *deviate* from the protocol. This could be done in a variety of ways, e.g. modifying our constraints to allow certain types of deviations, or first generating tests that conform to the protocol and subsequently mutating them.

such as successful/failed connections, interleaving of packets between different connections, etc.; and tests should be as short as possible while still exhibiting these different behaviors, so that we can maximize the number of tests we can perform in a given time. These constraints have trade-offs: for example, tests with failed connections that must be retried will necessarily be longer. As in the robotic planning example, we formulate these requirements as cost and label constraints, which allow us to balance our randomness and control needs.

For concreteness, consider the specific example of generating packet traces for 5 systems communicating over TCP. Our hard constraint can enforce that each connection follows the TCP protocol, using an encoding of the operation of the protocol as a finite automaton [24] (we will present efficient algorithms for LQCI with automata specifications below). Our cost function can assign a cost equal to the length of the trace, so that we prefer shorter sequences (whereas if we simply sampled uniformly from the language of the TCP automaton up to some length, longer sequences would be generated more frequently as there are exponentially more of them). Our label function could use two labels, distinguishing traces with connections that terminate cleanly from those that involve system failures and timeouts (we could also further subdivide into several types of failures). There are many more ways for a connection to fail than to terminate cleanly, and these two classes of traces might have significantly different lengths on average, but we want to ensure that our tests cover both cases adequately. By imposing constraints on the expected cost of a trace, as well as randomness constraints over the label and within each label class, we can control test length while enforcing sufficient diversity among the tests. In fact, we will see below that our LQCI algorithms can find the *minimum-cost* distribution consistent with the randomness constraints, thereby allowing us to test as efficiently as possible given coverage requirements.

## 2.2 Problem Definition

To formalize synthesis problems like those described above, we define the LQCI problem. Following the definition of CI [16, 17], we frame the problem as sampling words over a finite alphabet  $\Sigma$  subject to several constraints. We use the general term *specification* to refer to an encoding of a property of words (a language): for example, a deterministic finite automaton (DFA) is a specification, where the DFA accepts a word if and only if it satisfies the specification; a Boolean formula is another kind of specification. The complexity of the LQCI problem will vary depending on the type of specifications used, as we will see later.

**Definition 1.** A Labelled Quantitative Control Improvisation (LQCI) instance over an alphabet  $\Sigma$  is a tuple  $\mathcal{C} = (\mathcal{H}, \mathcal{K}, L, m, n, c, \lambda, \rho, \hat{\alpha}, \hat{\beta})$  which contains:

- $m, n \in \mathbb{N}$ , lower and upper bounds on word length (with  $m \leq n$ );
- $\mathcal{H}$ , a hard specification that must be satisfied by all words;
- $\mathcal{K} : \Sigma^* \rightarrow \mathbb{Q}$ , a cost function mapping words to rational costs;
- $L : \Sigma^* \rightarrow \Omega$ , a label function mapping words to a finite set of labels  $\Omega = \{\ell_1, \dots, \ell_{|\Omega|}\}$ ;

- $c \in \mathbb{Q}^+$ , an upper bound on expected cost;
- $\lambda, \rho \in \mathbb{Q}$ , lower and upper bounds on the marginal probability of selecting a word with a certain label (with  $0 \leq \lambda \leq \rho \leq 1$ );
- $\hat{\alpha}_i, \hat{\beta}_i \in \mathbb{Q}$ , lower and upper bounds on the conditional probability of words in label class  $\ell_i$  (with  $0 \leq \hat{\alpha}_i \leq \hat{\beta}_i \leq 1$  for all  $i$ ).

We note that the specifications and functions above are abstract, and our definition does not make any assumptions about how they will be encoded in a particular problem. For example, the hard constraint  $\mathcal{H}$  over words might be instantiated as the language of a DFA, context-free grammar, etc. Later in the paper we will develop algorithms for solving classes of LQCI instances with specification formalisms that satisfy certain properties.

The restriction to finite traces (via the length bounds  $m$  and  $n$ ) is consistent with prior work on using CI for robotic planning [19]: we frequently want plans that complete within a time limit. Likewise in fuzz testing we want tests of bounded length. Furthermore, as we will see, finite-trace LQCI is still a highly nontrivial problem, so we leave its extension to infinite traces as future work.

Given an LQCI instance, we define several convenient notations:

- $\Sigma^{m:n}$  is all words satisfying the length bounds:  $\{w \in \Sigma^* \mid m \leq |w| \leq n\}$ .
- The set of *improvisations*  $I$  consists of all words satisfying the length bounds and the hard specification. These are all the words which our improviser is allowed to generate.
- Since the length bounds  $m, n$  ensure  $I$  is finite, we can consider the image of  $I$  under  $\mathcal{K}$ , which must also be finite. We will refer to this set of *possible costs* as  $\Theta = \{\theta_1, \dots, \theta_{|\Theta|}\}$  (note that enumerating  $\Theta$  may require an algorithm).
- The *cost class*  $I_{i,k}$  consists of all words with label  $\ell_i$  and cost  $\theta_k$  which satisfy the length bounds and the hard specification, i.e.,  $\{w \in \Sigma^{m:n} \mid w \in \mathcal{L}(H), L(w) = \ell_i, \mathcal{K}(w) = \theta_k\}$ . As the costs of all words in a cost class are equal, we may speak of the *cost* of a cost class without ambiguity.
- The *label class*  $I_i$  consists of all words with label  $\ell_i$  as above but any cost, i.e.,  $\bigcup_{k=1}^{|\Theta|} I_{i,k}$ .
- We write  $\Pr[X(w) \mid w \leftarrow D]$  for the probability (or  $E[\dots]$  for the expected value) of  $X(w)$  given that  $w$  is sampled from distribution  $D$ .

**Definition 2.** *Given an LQCI instance  $\mathcal{C}$ , a distribution  $D$  over  $\Sigma^*$  is an improvising distribution for that instance if it satisfies the following constraints:*

1. **Hard Constraint:**  $\Pr[w \in I \mid w \leftarrow D] = 1$
2. **Cost Constraint:**  $E[\mathcal{K}(w) \mid w \leftarrow D] \leq c$
3. **Randomness over Labels:**  $\forall i \in \{1, \dots, |\Omega|\}, \lambda \leq \Pr[w \in I_i \mid w \leftarrow D] \leq \rho$
4. **Randomness over Words:**  $\forall i \in \{1, \dots, |\Omega|\}, \forall y \in I_i,$   
 $\hat{\alpha}_i \leq \Pr[y = w \mid w \in I_i, w \leftarrow D] \leq \hat{\beta}_i$

We say that an LQCI instance is feasible if there exists an improvising distribution for it (and infeasible otherwise). An improviser for an LQCI instance



is a probabilistic algorithm which takes no input, has finite expected runtime, and whose output distribution is an improvising distribution. Given an LQCI instance  $\mathcal{C}$ , the LQCI problem is then to determine if  $\mathcal{C}$  is feasible, and, if so, to generate an improviser for  $\mathcal{C}$ . Finally, an improvisation scheme for a class of LQCI instances is a probabilistic algorithm with finite expected runtime that solves the LQCI problem for instances in that class.

As described in the preceding sections, the goal of our problem definition is to provide formal guarantees about the randomness of improvisations while respecting the various constraints. In some applications, we may simply wish to maximize randomness: then precise control over the randomness parameters for each label class is not needed, and in fact finding values of  $\hat{\alpha}_i, \hat{\beta}_i$  which maximize randomness while remaining feasible is nontrivial. Building on our analysis of the basic LQCI problem in the next several sections, in Sect. 6 we will introduce a *maximum-entropy* version of LQCI which directly maximizes randomness without requiring  $\hat{\alpha}_i$  and  $\hat{\beta}_i$  to be explicitly specified.

### 3 Feasibility Conditions and the Greedy Construction

In this section, we introduce a greedy construction which will be used to provide necessary and sufficient conditions for an LQCI instance to be feasible. This construction will also form the basis of the improvisation schemes presented later in the paper. For now, we will present the construction without assuming any particular specification formalism and ignoring algorithmic concerns: the description presented here will consider traces one by one and thus be inefficient. The next section will develop efficient implementations of these ideas.

The *greedy LQCI construction* is separated into two phases. In the first phase, the *greedy cost construction*, we define a distribution over each label class individually, greedily optimizing cost by giving as much weight as we can to the cheapest elements while respecting the randomness over words condition. In the second phase, the *greedy label construction*, we define a distribution over labels, greedily assigning maximum marginal probability to the label classes with the cheapest expected costs under the distributions from the first phase while respecting the randomness over labels condition. The intuition is that we want to first make sampling within each label class as cheap as possible, and then sample from the cheapest classes as often as possible, while satisfying the randomness requirements. We will prove below that this greedy approach in fact yields an improvising distribution whenever one exists.

**Toy Example.** We will begin with a toy example which illustrates the idea and correctness of the greedy construction. Suppose we want to sample from words of length 3 ( $m = n = 3$ ) over the binary alphabet  $\Sigma = \{0, 1\}$ , subject to the hard constraint that each word must contain at least one 1. We will have two label classes: words with an odd number of 1s will be in label 1, and those with an even number in label 2. The cost of each word will be its integer value in binary. The label parameters will be  $\lambda = 0.2$  and  $\rho = 1.0$ , so that each label



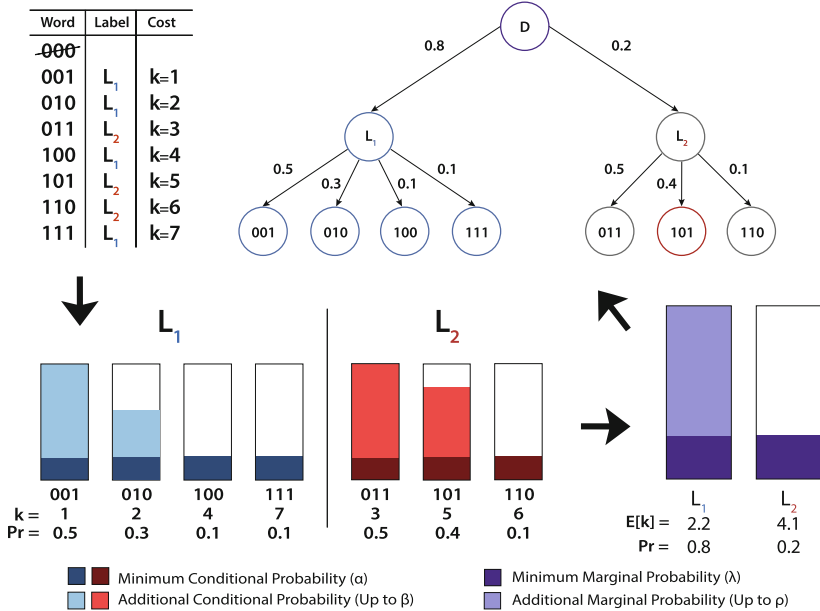
must be sampled from with a probability at least 0.2 and at most 1.0. The word randomness parameters will be  $\hat{\alpha}_1 = \hat{\alpha}_2 = 0.1$  and  $\hat{\beta}_1 = \hat{\beta}_2 = 0.5$ , so that when sampling from a particular label class, each word in the class must be selected with probability at least 0.1 and at most 0.5.

Figure 2 shows the greedy construction applied to this LQCI instance. Beginning with label 1, we need to construct a probability distribution over the words 001, 010, 100, and 111. We start by assigning 0.1 to each word, since  $\hat{\alpha}_1 = 0.1$ . Then we assign as much additional probability as we can (up to  $\hat{\beta}_1 = 0.5$ ) to the cheapest words first until a total of 1 is reached, as shown in the bottom left of Fig. 2. The result is that there are 3 distinct probabilities within the label class: the minimum  $\hat{\alpha}_1 = 0.1$ , the maximum  $\hat{\beta}_1 = 0.5$ , and the overflow probability 0.3 on the word 010. This process results in a distribution over label 1 with expected cost 2.2, the minimum achievable while satisfying the randomness over words constraint. A similar process yields a distribution of expected cost 4.1 on label 2. Now that we know the minimum expected cost for each label, we should sample from the cheaper label as frequently as possible. Since  $\lambda = 0.2$  and  $\rho = 1.0$ , we sample from label 2 with probability 0.2 (the minimum allowed) and from label 1 with probability 0.8, yielding a distribution over improvisations with expected cost 2.58. Our analysis will show that this is in fact the minimum possible expected cost over all distributions satisfying conditions (1) (3), and (4) in Definition 2. So if the cost bound  $c$  in the LQCI instance is at least this large, then we have an improvising distribution, and otherwise the instance is infeasible.

We now describe the two phases of our construction formally.

**The Greedy Cost Construction.** For a particular label class  $i \in \{1, \dots, |\Omega|\}$ , we proceed as follows. Let  $\delta^i = (\delta^i_1, \dots, \delta^i_{|\Theta|})$  be a list of all the cost classes  $I_{i,k}$  with label  $i$ , sorted in increasing order of cost. Then fix  $o_i = \frac{1 - \hat{\alpha}_i |I_i|}{\hat{\beta}_i - \hat{\alpha}_i}$ , whose floor is the maximum number of words that can be assigned  $\hat{\beta}_i$  probability (the maximum allowed) while still leaving at least  $\hat{\alpha}_i$  probability (the minimum allowed) for each remaining word. Then, moving through the cost classes in the order given by  $\delta^i$ , we assign  $\hat{\beta}_i$  probability to each word in the class, until we get to a class  $\delta^i_r$  where the cumulative number of words so far (including the new class) would exceed  $o_i$ . To this class we assign  $\hat{\beta}_i(o_i - \sum_{k=1}^{r-1} |\delta^i_k|) + \hat{\alpha}_i(\sum_{k=1}^r |\delta^i_k| - o_i)$  probability (spread uniformly over words in the class), the maximum allowed while leaving exactly  $\hat{\alpha}_i$  for each remaining word. Assigning  $\hat{\alpha}_i$  to the remaining words, we obtain a distribution  $D_i$  over the whole label class  $I_i$ .

We note that this process is not well-defined when  $\hat{\alpha}_i = \hat{\beta}_i$  (in which case we simply assign probability  $\hat{\alpha}_i$  to every word in  $I_i$ ) or when  $\hat{\alpha}_i |I_i| > 1$  (in which case the instance is infeasible due to  $\hat{\alpha}_i$  being too large); also, the process does not result in a probability distribution if  $\hat{\beta}_i |I_i| < 1$  (in which case the instance is infeasible due to  $\hat{\beta}_i$  being too small). Except in these cases, we get a well-defined distribution  $D_i$  over  $I_i$  which satisfies conditions (1) and (4) of Definition 2. Moreover, the expected cost of  $D_i$  is minimal among all such distributions, since it assigns as much weight as possible to the words with lowest cost.



**Fig. 2.** Applying the greedy LQCI construction to our toy example. Counter-clockwise from upper left: table of improvisations, the greedy cost construction, the greedy label construction, and the final improvising distribution.

**The Greedy Label Construction.** Given the distributions  $D_i$  for each label class  $I_i$  from the first stage, we now choose a distribution over labels. Following a similar pattern as before, let  $\delta$  be a list of the distributions  $D_i$  sorted in order of increasing expected cost. Then fix  $u = \lfloor \frac{1-|\Omega|\lambda}{\rho-\lambda} \rfloor$ , which is the number of label classes that can be assigned probability  $\rho$  (the maximum allowed) while still leaving at least  $\lambda$  (the minimum allowed) for each remaining class. We assign  $\rho$  probability to the first  $u$  label classes in  $\delta$ . To the next label class we assign probability  $1 - \rho u - \lambda(|\Omega| - u - 1)$ , the maximum allowed while leaving exactly  $\lambda$  for each remaining label class. Finally, we assign  $\lambda$  to all remaining label classes, and call the resulting distribution over labels  $\hat{D}$ . Similar to before, this process will be well-defined and result in a distribution when  $\frac{1}{\rho} \leq |\Omega| \leq \frac{1}{\lambda}$ ; otherwise,  $\rho$  is too small or  $\lambda$  is too large for condition (3) of Definition 2 to be satisfied.

To complete the construction, we obtain a final distribution  $D$  over words by first sampling a label  $i$  from  $\hat{D}$  and then sampling from  $D_i$ . The greedy cost construction ensured that  $D_i$  is defined over the class  $I_i \subseteq I$  and assigns probability between  $\hat{\alpha}_i$  and  $\hat{\beta}_i$  to each word, so  $D$  will satisfy the hard and randomness over words constraints in Definition 2. The greedy label construction ensures that  $\hat{D}$  assigns probability between  $\lambda$  and  $\rho$  to each label, so  $D$  will also satisfy the randomness over labels constraint. Finally, since each phase selects a distribution of minimal cost amongst those satisfying the corresponding constraints, if *any*

improvising distribution exists then  $D$  will have no greater cost, thereby satisfying the cost constraint and being an improvising distribution. Formalizing this argument yields the following theorem (see the Appendix [21] for details):

**Theorem 1.** *An LQCI instance is feasible if and only if all of the following conditions are true:*

1.  $\frac{1}{\rho} \leq |\Omega| \leq \frac{1}{\lambda}$
2.  $\forall i \in \{1, \dots, |\Omega|\}, \frac{1}{\beta_i} \leq |I_i| \leq \frac{1}{\alpha_i}$
3. *The greedy LQCI construction produces a distribution  $D$  whose expected cost is at most  $c$  (i.e.,  $E[\mathcal{K}(w) \mid w \leftarrow D] \leq c$ ).*

We conclude this section with a reminder that the greedy LQCI construction is a *construction* and not a practical algorithm: it defines a distribution but not a practical way to compute it for a specified LQCI instance. With common specification formalisms such as DFAs and Boolean formulas, the number of possible improvisations can easily be exponential in the size of the problem instance. In this case, assigning probabilities to words one at a time as described above in the abstract construction would be highly impractical. Instead, the algorithms we present in the following sections are able to avoid enumerating exponentially-large sets by working with implicit representations to create distributions equal to or approximating the one produced by the greedy LQCI construction.

## 4 Exact LQCI for Automata Specifications

The greedy LQCI construction from Sect. 3 gives us a way to determine if an LQCI instance is feasible and, if so, to build an improvising distribution. Implementing the construction requires several operations—such as computing the size of the label/cost classes—which may or may not be tractable depending on the types of specification used in the instance. In this section, we will identify a sufficient list of operations which yield an efficient generic improvisation scheme for any class of LQCI instances with specifications supporting these operations. Then we will instantiate the scheme for two natural classes of specifications given by deterministic finite automata, obtaining efficient improvisation algorithms.

Following the description of the preceding section, we can see that for a given LQCI instance, the operations listed below are sufficient to complete the greedy LQCI construction and sample from the resulting distribution:

**Definition 3.** (*Sufficient Operations*) *Given an LQCI instance  $\mathcal{C}$ :*

1. *Compute the list of possible costs  $\Theta$ .*
2. *For each  $i \in \{1, \dots, |\Omega|\}$  and  $k \in \Theta$ , compute  $|I_{i,k}|$ .*
3. *For each  $i \in \{1, \dots, |\Omega|\}$  and  $k \in \Theta$ , sample uniformly from  $I_{i,k}$ .*

If we can implement these operations in polynomial time, we can build a polynomial-time improvisation scheme in the sense of [16, 17], i.e., an algorithm which solves the LQCI problem in polynomial time, and whose generated improvisers themselves run in polynomial (expected) time. To do this we first compute the list of possible costs and the size of each  $I_{i,k}$ . We then perform a modified version of the greedy construction which assigns probabilities to entire cost classes instead of individual words. As each word in a class has the same label and cost, we can satisfy our cost and randomness requirements with a distribution that assigns the same probability to every word within a class. Then to implement placing probability  $p$  on each word of  $I_{i,k}$  without enumerating this potentially exponentially-large set, we simply choose the set with probability  $p|I_{i,k}|$  and then sample uniformly from it (see the Appendix [21] for a detailed argument).

**Theorem 2.** *Suppose for a class of LQCI instances the operations in Definition 3 can be performed in polynomial time (in the size of the instance). Then there is a polynomial-time improvisation scheme for that class.*

One broad class of specifications to which this scheme can apply is deterministic finite automata (DFAs): for example, we can encode the specifications from our robotic planning example as DFAs. While a DFA can encode the hard specification  $\mathcal{H}$  directly, encoding cost and label functions is not as clear. We consider two natural encodings: most simply, we can label each state of the DFA with an integer, assigning the associated label/cost to words ending at that state.

**Theorem 3.** *Consider the class of LQCI instances where  $\mathcal{H}$  is a DFA,  $\mathcal{K}$  and  $L$  are given by DFAs which output an integer cost/label associated with the state they end on, the length bounds are given in unary and all other numerical parameters in binary. This class has a polynomial-time improvisation scheme.*

*Proof (Sketch).* Operation (1) is trivial. For (2) and (3), we can easily construct DFAs accepting all improvisations with a given label and cost, then apply classical techniques for counting/sampling from the language of a DFA [23].  $\square$

To capture cost functions like path length or mission time (as in our planning example), we consider a second encoding using weighted DFAs: states are again labeled with integers, but the cost is now given by *accumulating* costs from every state passed through. Here, the number of possible costs can grow linearly with the largest cost of a single state, and so be exponential in the size of the (binary) encoding; as a result we only obtain a *pseudopolynomial* improvisation scheme by applying Theorem 2. The algorithm can still be feasible, however, when the magnitude of possible costs is not too large, as we will see in Sect. 7.

**Theorem 4.** *Consider the class of LQCI instances as in Theorem 3 but where  $\mathcal{K}$  is given by a weighted DFA, i.e. summing the integer costs associated with each state of a DFA accepting path (with multiplicity). This class has a pseudopolynomial improvisation scheme.*

*Proof (Sketch).* We can perform operation (1) by dynamic programming over the states and word lengths up to the length bound  $n$ . If the maximum cost of a state in the DFA for  $\mathcal{K}$  is  $M$ , then the cost of an improvisation is at most  $M(n+1)$ ; so for (2) and (3) we can build DFAs of size  $\text{poly}(M, n)$  recognizing  $I_{i,k}$  and then apply counting/sampling as above. If state costs were encoded in unary, the operations above would take polynomial time and Theorem 2 would apply. Converting from binary to unary yields a pseudopolynomial scheme.  $\square$

## 5 Approximate LQCI for Symbolic Specifications

The LQCI algorithms for DFAs that we developed in the previous section cover many useful specifications; however, as we will see in Sect. 7, even fairly simple specifications can require very large automata when represented explicitly. In this section we propose an algorithm that avoids such blowup by working with *symbolic* specifications given by Boolean formulas. We cannot use our scheme of Theorem 2 directly, because counting the number of solutions of a Boolean formula is  $\#P$ -hard. Nevertheless, we will show that by leveraging recent advances in SAT solving, we can *approximately* solve LQCI to any desired accuracy.

We consider LQCI instances with specifications given by Boolean formulas, whose variables encode traces and costs; for modeling convenience, we also allow a vector of auxiliary variables  $z$ . Specifically, we assume we are given:

- a conjunctive normal form (CNF) formula  $h(x, z)$  such that  $\exists z.h(x, z)$  holds if and only if the bitvector  $x$  encodes a trace satisfying the hard constraint;
- a CNF formula  $\ell(x, y, z)$  such that  $\exists z.\ell(x, y, z)$  holds if and only if trace  $x$  has the label encoded by the bitvector  $y$ ;
- a CNF formula  $k(x, y, z)$  such that  $\exists z.k(x, y, z)$  holds iff trace  $x$  has cost  $y$  (a positive integer).

We further assume that the instance has only a polynomial number of labels, although there can be exponentially-many costs.

Given such an instance, we can readily build a CNF formula  $\phi_i(x, y, z)$  which is satisfiable iff  $x$  encodes a word which has length between  $m$  and  $n$ , satisfies the hard constraint, belongs to label  $i$ , and has cost  $y$ . The solutions  $x$  for a particular choice of  $i$  and  $y$  comprise the associated cost class, so that the operations we need for the greedy construction are instances of the *model counting* and *uniform generation* problems for SAT.<sup>2</sup> Recent work has yielded practical algorithms based on SAT solvers which solve these problems approximately [7, 27]<sup>3</sup>:

<sup>2</sup> Since we do not want to count over the auxiliary variables  $z$ , we actually require *projected* counting/sampling, which the algorithms we use can also perform [7, 17].

<sup>3</sup> We note that UniGen [6, 7] is not strictly speaking an almost-uniform generator as in Definition 4 since it only supports sufficiently-large tolerances; for theoretical results, one can substitute the algorithm of [4] to do *exact* (projected) uniform sampling.

**Definition 4.** ([7]) *An approximate counter is a probabilistic algorithm  $\mathcal{C}$  which given a CNF formula  $F$  with set of solutions  $R_F$ , a tolerance  $\tau > 0$ , and a confidence  $1 - \delta \in [0, 1)$  guarantees that*

$$\Pr [|R_F|/(1 + \tau) \leq \mathcal{C}(F, \tau, 1 - \delta) \leq (1 + \tau)|R_F|] \geq 1 - \delta.$$

*An almost-uniform generator  $\mathcal{G}$  is a probabilistic algorithm that, given  $F$  as above and a tolerance  $\epsilon > 0$ , guarantees that for every  $y \in R_F$ , we have*

$$1/((1 + \epsilon)|R_F|) \leq \Pr[\mathcal{G}(F, \epsilon) = y] \leq (1 + \epsilon)/|R_F|.$$

We can modify our greedy construction to work with only approximate counting/sampling as follows. If the cost bitvector has  $|y|$  bits, the cost of a word is between 1 and  $2^{|y|}$ . To avoid enumerating exponentially-many cost classes for label  $i$ , we group words into “cost buckets” by subdividing this interval into powers of  $r$  for some  $r > 1$ , i.e.  $[1, r), [r, r^2), \dots, [r^{b-1}, r^b)$ . We will have  $b = O(\log_r(2^{|y|})) = O(|y|/\log r)$  buckets, and we can estimate the size of bucket  $j$  by approximately counting solutions to  $\exists z. [\phi_i(x, y, z) \wedge (r^j \leq y < r^{j+1})]$ . We will then use these estimates to choose a distribution over buckets, following the intuition of the greedy cost construction that we should assign the most probability to buckets with lowest estimated cost, but with some adjustments to bound the error that approximate sampling introduces.

For each label class  $i$  with randomness parameters  $\alpha$  and  $\beta$ , we apply a modified form of the greedy cost construction, shown in Algorithm 1. We start in lines 1–3 by using model counting as above (with a tolerance  $\tau$  and confidence  $1 - \delta$  to be specified later) to find estimates  $c_k$  of the size of each bucket  $k$ , and corresponding lower bounds  $p_k$  on how much probability the bucket would have received in the exact greedy construction (the extra  $1 + \tau$  factor accounting for possibly overestimating the size of the bucket). If these lower bounds total more than 1, then we know there are too many improvisations for the instance to be feasible (assuming the model counts are within their tolerance) and we return false on line 4. Otherwise, on lines 5–7 we proceed as in the greedy construction, starting from the cheapest bucket, increasing the assigned probability per word to  $(1 + \tau)\beta$  until a probability of 1 is reached. The factor of  $1 + \tau$  ensures that, even if the model counts have underestimated the size of the cheaper buckets, we still assign them at least as much probability as the exact greedy construction would. Next, line 8 checks if there are too few improvisations, similarly to line 4. Finally, we return our distribution over buckets, as well as a lower bound on its expected cost that we will use next.

If Algorithm 1 does not return false for any label class, then we complete our approximate LQCI algorithm by running the greedy label construction from Sect. 3, using the lower bounds from Algorithm 1 as the expected cost of each label class. As before, we declare the instance infeasible if the construction fails or if its expected cost exceeds the cost bound  $c$ . Otherwise, we obtain a distribution over all the cost buckets; our improviser then simply chooses a bucket from this distribution and applies almost-uniform sampling to sample a word from it.

Choosing the bucket count and counting/sampling tolerances appropriately, our algorithm can approximate an improvising distribution to within arbitrarily-small multiplicative error, using polynomially-many calls to a SAT solver:

---

**Algorithm 1.** ApproximateGreedyCost( $i, \alpha, \beta, r, b, \tau, \delta$ )

---

```

1: for  $k = 1$  to  $b$  do
2:    $c_k := \#SAT(\exists z. \phi_i(x, y, z) \wedge (r^{k-1} \leq y < r^k), \tau, 1 - \delta)$ 
3:    $p_k := \alpha c_k / (1 + \tau)$ 
4: if  $\sum_{j=1}^b p_j > 1$  then return False
5: for  $k = 1$  to  $b$  do
6:    $p_k := \min((1 + \tau)\beta c_k, 1 - \sum_{j \neq k} p_j)$ 
7:   if  $\sum_{j=1}^b p_j = 1$  then break
8: if  $\sum_{j=1}^b p_j < 1$  then return False
9:  $Lo := \sum_{j=1}^b p_j r^{j-1}$ 
10: return  $\{p_j\}_{j=1}^b, Lo$ 

```

---

**Theorem 5.** *There is an algorithm which, given a Boolean LQCI instance  $\mathcal{C}$ , a cost tolerance  $\zeta > 0$ , a randomness tolerance  $\gamma > 0$ , and a confidence  $1 - \delta \in [0, 1)$ , runs in  $\text{poly}(|\mathcal{C}|, 1/\zeta, 1/\gamma, \log(1/\delta))$  time relative to an NP oracle and either returns  $\perp$  or an algorithm sampling from a distribution  $\tilde{D}$  over words. With probability at least  $1 - \delta$ , if  $\perp$  is returned then  $\mathcal{C}$  is infeasible, and otherwise:*

1. **Hard Constraint:**  $\Pr[\mathcal{H}(w) \mid w \leftarrow \tilde{D}] = 1$
2. **Cost Constraint:**  $E[\mathcal{K}(w) \mid w \leftarrow \tilde{D}] \leq (1 + \zeta)c$
3. **Randomness over Labels:**  $\forall i \in \{1, \dots, |\Omega|\}, \lambda \leq \Pr[w \in I_i \mid w \leftarrow \tilde{D}] \leq \rho$
4. **Randomness over Words:**  $\forall i \in \{1, \dots, |\Omega|\} \forall y \in I_i,$   
 $\hat{\alpha}_i / (1 + \gamma) \leq \Pr[y = w \mid w \in I_i, w \leftarrow \tilde{D}] \leq (1 + \gamma)\hat{\beta}_i$

## 6 Maximum-Entropy LQCI

Our LQCI definition requires providing conditional probability bounds for every label, which while allowing maximal control of the distribution, can be unwieldy to use. However, if we drop conditional bounds entirely, trivial solutions with unnecessarily-poor randomness can appear. For example, consider an LQCI instance with parameters  $\lambda = 0.5, \rho = 0.5, \hat{\alpha} = (0, \dots, 0), \hat{\beta} = (1, \dots, 1)$ . With this choice, any distribution will satisfy the randomness over words constraint, and all labels have the same marginal probability of being selected. Then assume that we have two labels, costs  $\Theta = (1, 2)$ , and cost bound  $c = 1.5$ , along with the following cost class sizes:  $|I_{1,1}| = 1, |I_{2,1}| = 1, |I_{1,2}| = 1000, |I_{2,2}| = 1000$ . Now simply assigning 50% probability to  $I_{1,1}$  and 50% probability to  $I_{2,1}$  is an improvising distribution. Assigning 25% probability to all 4 classes is also an improvising distribution, and clearly preferable from the perspective of randomness. Unfortunately, without a nontrivial randomness over words constraint, we have no way to push the improviser to select the second distribution. To enforce this, we introduce the concept of entropy from information theory.

**Definition 5.** *Given a discrete random variable  $X$  with a set of outcomes  $\Omega$  and probabilities  $p : \Omega \rightarrow [0, 1]$ , the entropy of  $X$  is  $H(X) = -\sum_{x \in \Omega} p(x) \lg p(x)$ .*



To obtain a problem formulation that maximizes randomness without requiring probability bounds for each class, we invoke the Principle of Maximum Entropy: amongst all improvising distributions (without a randomness over words constraint), we should select the one with the highest entropy (as first proposed for reactive CI in [30]). This yields a notion of Maximum-Entropy LQCI:

**Definition 6.** A Maximum-Entropy LQCI (MELQCI) instance is an LQCI instance where  $\hat{\alpha} = (0, \dots, 0)$  and  $\hat{\beta} = (1, \dots, 1)$ . A  $\tau$ -improviser for a MELQCI instance  $\mathcal{C}$  is an improviser (as in LQCI) whose output distribution has entropy at most  $\tau$  less than the maximum-entropy improvising distribution for  $\mathcal{C}$ . We define the MELQCI problem as, given an instance  $\mathcal{C}$  and  $\tau > 0$ , determining if  $\mathcal{C}$  is feasible, and, if so, generating a  $\tau$ -improviser for  $\mathcal{C}$ .

We can solve MELQCI efficiently in the same cases as LQCI:

**Theorem 6.** Given a class of MELQCI instances for which one can perform the operations in Definition 3 in polynomial time, there is a polynomial-time algorithm which given an instance from the class and a  $\tau > 0$ , computes a  $\tau$ -improviser.

*Proof.* (Sketch). Once cost class sizes have been computed as in Theorem 2, the search for the desired distribution over cost classes can be formulated as an optimization problem with a separable convex objective (the entropy of the distribution) and linear constraints (improviser constraints). This problem can be solved in time polynomial in the size of the instance and  $\log(1/\tau)$  [10].

As in Sect. 4, we can transform this algorithm into a *pseudopolynomial* scheme for accumulated-cost DFA specifications.

## 7 Experiments

We ran several experiments on the robotic planning problems from Sect. 2 (code available at [20]). These experiments aim to demonstrate that we can encode practical problems as LQCI instances solvable using our algorithms, highlight the relative advantages/disadvantages of our exact/approximate algorithms, and show the necessity of the label function in ensuring meaningful randomness.

As a minimal experiment, we used a  $6 \times 6$  grid world with a small range of costs (0–3 per cell, 8–39 for paths); we compared against a  $7 \times 7$  grid world with a much larger range of costs (0–9 per cell, 38–137 for paths).<sup>4</sup> We encoded the specifications in Sect. 2 both as DFAs for our exact LQCI and MELQCI algorithms, and as Boolean formulas for our approximate LQCI algorithm. The Boolean encodings were obtained by formulating the specifications in the SMT theory of bitvectors, and bit-blasting them with Z3 [26]; the resulting formulas had several thousand variables and tens of thousands of clauses. We used UniGen3 [7, 27] for uniform generation with its default tolerance<sup>5</sup> of 17, and an

<sup>4</sup> A larger  $8 \times 8$  map exceeded our 24-hour wallclock timeout for all exact and approximate experiments.

<sup>5</sup> UniGen3 cannot guarantee a multiplicative error of less than 7.48 [6]; see footnote 3.

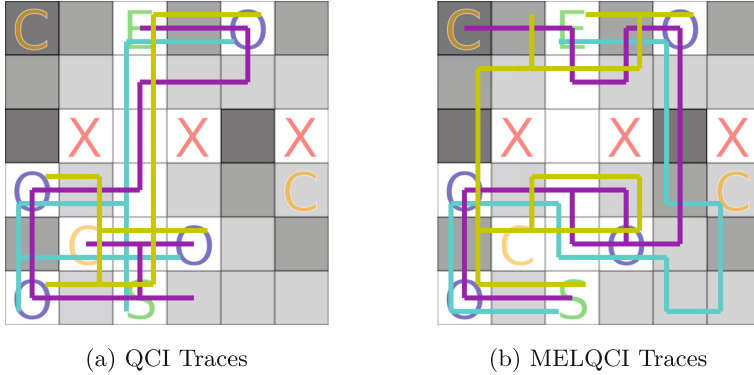
**Table 1.** Experiment parameters and improviser construction times (in minutes).

| Map   | Problem Type | $(\lambda, \rho)$ | $(\hat{\alpha}_i, \hat{\beta}_i)$ | $r$ | $\gamma$        | $\delta$ | Wall Time                        | CPU Time    |
|-------|--------------|-------------------|-----------------------------------|-----|-----------------|----------|----------------------------------|-------------|
| 6 × 6 | Exact QCI    | (0, 3e-5)         | (1, 1)                            |     |                 |          | 540                              | 5568        |
|       | Exact LQCI   | (0, 1e-5)         | (0.3, 0.4)                        |     | N/A             |          | 444                              | 6102        |
|       | Exact MELQCI | N/A               | (0.3, 0.4)                        |     |                 |          | 444 <sup>a</sup>                 | 6102        |
|       | Approx. LQCI | (0, 1e-5)         | (0.3, 0.4)                        | 1.2 | 10 <sup>2</sup> | 0.2      | 23.7 ± 0.6                       | 93.3 ± 1.4  |
|       | Approx. LQCI | (0, 1e-5)         | (0.3, 0.4)                        | 1.2 | 10 <sup>3</sup> | 0.2      | 21.2 ± 0.7                       | 81.5 ± 1.1  |
|       | Approx. LQCI | (0, 1e-5)         | (0.3, 0.4)                        | 1.2 | 10 <sup>4</sup> | 0.2      | 20.2 ± 0.7                       | 78.4 ± 3.4  |
| 7 × 7 | Exact QCI    | (0, 3e-5)         | (1, 1)                            |     |                 |          |                                  |             |
|       | Exact LQCI   | (0, 1e-5)         | (0.3, 0.4)                        |     | N/A             |          | Timed out<br>(24-hour wall time) |             |
|       | Exact MELQCI | N/A               | (0.3, 0.4)                        |     |                 |          |                                  |             |
|       | Approx. LQCI | (0, 1e-5)         | (0.3, 0.4)                        | 1.2 | 10 <sup>2</sup> | 0.2      | 42.8 ± 2.1                       | 186.1 ± 3.9 |
|       | Approx. LQCI | (0, 1e-5)         | (0.3, 0.4)                        | 1.2 | 10 <sup>3</sup> | 0.2      | 38.8 ± 8.8                       | 152.6 ± 9.0 |
|       | Approx. LQCI | (0, 1e-5)         | (0.3, 0.4)                        | 1.2 | 10 <sup>4</sup> | 0.2      | 38.8 ± 9.7                       | 145.5 ± 9.5 |

<sup>a</sup> The LQCI/MELQCI runtimes were nearly identical, since MELQCI reuses the LQCI computations and adds a convex optimization step, which took negligible time.

in-development version of **ApproxMC** [8, 27, 28] for approximate model counting with tolerances of 1.4, 6.7, and 23.25, so that the overall  $\gamma$  values were 10<sup>2</sup>, 10<sup>3</sup>, and 10<sup>4</sup>. To put these values into context, the small/large maps had on the order of 10<sup>7</sup>/10<sup>9</sup> improvisations, and we required that no word have  $> \rho = 10^{-5}$  probability of being selected. Therefore, with our tightest/loosest  $\gamma$  we are guaranteed that no word will be more than 0.1%/10% of the distribution respectively. The confidence was set to 0.8 ( $\delta = 0.2$ ), ApproxMC’s default confidence. Each model counting call however required a much higher confidence to achieve an overall  $\delta$  of 0.2.

For the small/large maps respectively we used length bounds of (1,25)/(1,30) and cost bounds of 30/50. We used label probability bounds of (0.3, 0.4) throughout, except for unlabeled “QCI” experiments. The experiments were run on a 64-core machine with 188 GB of RAM; we used 62 parallel threads, unless this exhausted RAM, in which case we used 16 threads. The experiments are summarized in Table 1; due to significant runtime variability for the approximate experiments, we report means and standard deviations over 10 repetitions. For all exact experiments which completed within the 24-h wallclock timeout, RAM usage was  $\leq 6$  GB per thread, and the average time to sample an improvisation was  $\leq 1$  ms; all approximate experiments required  $\leq 250$  MB RAM per thread and took  $\sim 20$  s to sample an improvisation.



**Fig. 3.** Randomly-selected traces generated by the QCI/MELQCI improvisers for the  $6 \times 6$  map. Note that all the QCI traces use the same charging station.

We can draw several conclusions from these results. Improviser construction with the exact algorithm is significantly more expensive than with the approximate algorithm, in both CPU time and RAM. This is not surprising, as the exact encodings resulted in enormous DFAs which, for the large map, approached  $10^{10}$  states. Conversely, sampling is much faster for the exact algorithm, with no SAT queries required. We can also see that the approximate algorithm can be used to practically solve problems that are infeasible to solve exactly, such as the large-map problem. We expect new developments in the relatively young field of approximate model counting/sampling will further speed up our algorithm.

Visualizing several randomly-chosen traces from our exact QCI and MELQCI experiments in Fig. 3, we can see the importance of labels. In unlabeled QCI, the robot always charged at the substation near the main road due to the lower expected cost of such paths. In contrast, MELQCI yielded a near-uniform distribution over the charging stations. This increase in diversity was not free, with the average cost rising to 21.4 for MELQCI from 8.7 for QCI. This trade-off demonstrates how LQCI allows us to balance the need for control over our improvisations with the need for meaningful diversity (not merely randomness) by choosing appropriate label functions.

## 8 Conclusion

In this paper, we introduced *labelled quantitative control improvisation* as a framework allowing correct-by-construction synthesis of randomized systems whose behavior must be diverse with respect to a label function and near-optimal with respect to a cost function. We studied the theory of LQCI problems and developed algorithms for solving them for broad classes of specifications encoded as finite automata or Boolean formulas. Our experiments demonstrated how our framework can be used to formalize and solve realistic robotic planning problems.

There are a number of clear directions for future work. Scalability is an evident concern: our experiments show that our algorithms can require substantial resources to solve even relatively small LQCI problems. While LQCI with Boolean formulas is a difficult  $\#P$ -hard problem, our algorithms will directly benefit from future progress in model counting; our DFA algorithms could also be improved through the use of abstraction to reduce state-space explosion. We also plan to explore generalizations of our algorithms, such as extending our approximate scheme to MELQCI and to problems with exponentially-many labels, as well as potentially infinite traces. Finally, we are investigating extensions of the LQCI problem to *reactive* settings with adversarial environments, and to *black-box* settings for design-space exploration and other problems where we do not have complete models for the cost function and other constraints.

**Acknowledgements.** The authors thank Skyler Stewart for designing Fig. 2, and several anonymous reviewers for their helpful comments. This work was supported in part by DARPA contract FA8750-20-C-0156 (SDCPS).

## References

1. Akkaya, I., Fremont, D.J., Valle, R., Donzé, A., Lee, E.A., Seshia, S.A.: Control improvisation with probabilistic temporal specifications. In: First IEEE International Conference on Internet-of-Things Design and Implementation, IoTDI 2016, Berlin, Germany, 4–8 April 2016, pp. 187–198. IEEE Computer Society (2016). <https://doi.org/10.1109/IoTDI.2015.33>, <https://doi.org/10.1109/IoTDI.2015.33>
2. Almagor, S., Kupferman, O.: High-quality synthesis against stochastic environments. In: Talbot, J.M., Regnier, L. (eds.) 25th EACSL Annual Conference on Computer Science Logic (CSL 2016). Leibniz International Proceedings in Informatics (LIPIcs), vol. 62, pp. 28:1–28:17. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2016). <https://doi.org/10.4230/LIPIcs.CSL.2016.28>, <http://drops.dagstuhl.de/opus/volltexte/2016/6568>
3. Baier, C., Brázdil, T., Größer, M., Kučera, A.: Stochastic game logic. *Acta informatica* pp. 1–22 (2012)
4. Bellare, M., Goldreich, O., Petrank, E.: Uniform generation of NP-witnesses using an NP-oracle. *Inf. Comput.* **163**(2), 510–526 (2000)
5. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weighofer, M.: Specify, compile, run: hardware from PSL. In: Proceedings of the 6th International Workshop on Compiler Optimization meets Compiler Verification (COCV 2007). *Electronic Notes in Theoretical Computer Science*, vol. 190, pp. 3–16. Elsevier (2007). <https://doi.org/10.1016/j.entcs.2007.09.004>, <http://www.sciencedirect.com/science/article/pii/S157106610700583X>
6. Chakraborty, S., Fremont, D.J., Meel, K.S., Seshia, S.A., Vardi, M.Y.: On parallel scalable uniform SAT witness generator. In: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS), pp. 304–319 (4 2015)
7. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Balancing scalability and uniformity in sat-witness generator. In: Proceedings of Design Automation Conference (DAC), pp. 60:1–60:6, June 2014

8. Chakraborty, S., Meel, K.S., Vardi, M.Y.: Algorithmic improvements in approximate counting for probabilistic inference: from linear to logarithmic sat calls. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), July 2016
9. Chen, T., Forejt, V., Kwiatkowska, M., Simaitis, A., Wiltsche, C.: On stochastic games with multiple objectives. In: Chatterjee, K., Sgall, J. (eds.) MFCS 2013. LNCS, vol. 8087, pp. 266–277. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40313-2\\_25](https://doi.org/10.1007/978-3-642-40313-2_25)
10. Chubanov, S.: A polynomial-time descent method for separable convex optimization problems with linear constraints. *SIAM J. Optim.* **26**(1), 856–889 (2016). <https://doi.org/10.1137/14098524x>
11. Denise, A., Gaudel, M.C., Gouraud, S.D., Lassaigne, R., Oudinet, J., Peyronnet, S.: Coverage-biased random exploration of large models and application to testing. *Int. J. Softw. Tools Technol. Transfer* **14**(1), 73–93 (2011). <https://doi.org/10.1007/s10009-011-0190-1>
12. Donze, A., Libkind, S., Seshia, S.A., Wessel, D.: Control improvisation with application to music. Tech. Rep. UCB/EECS-2013-183, EECS Department, University of California, Berkeley (Nov 2013). <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-183.html>
13. Donzé, A., Valle, R., Akkaya, I., Libkind, S., Seshia, S.A., Wessel, D.: Machine improvisation with formal specifications. In: Music Technology meets Philosophy - From Digital Echos to Virtual Ethos: Joint Proceedings of the 40th International Computer Music Conference, ICMC 2014, and the 11th Sound and Music Computing Conference, SMC 2014, Athens, Greece, 14–20 September 2014. Michigan Publishing (2014). <http://hdl.handle.net/2027/spo.bbp2372.2014.196>
14. Finkbeiner, B.: Synthesis of reactive systems. In: Esparza, J., Grumberg, O., Sickert, S. (eds.) Dependable Software Systems Engineering. NATO Science for Peace and Security Series, D: Information and Communication Security, vol. 45, pp. 72–98. IOS Press, Amsterdam (2016)
15. Fremont, D.J.: Algorithmic improvisation. Thesis (2019). <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-133.pdf>
16. Fremont, D.J., Donzé, A., Seshia, S.A., Wessel, D.: Control improvisation. In: Harsha, P., Ramalingam, G. (eds.) 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015). Leibniz International Proceedings in Informatics (LIPIcs), vol. 45, pp. 463–474. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2015). <https://doi.org/10.4230/LIPIcs.FSTTCS.2015.463>, <http://drops.dagstuhl.de/opus/volltexte/2015/5659>
17. Fremont, D.J., Donzé, A., Seshia, S.A.: Control improvisation (2017). <https://arxiv.org/abs/1704.06319>
18. Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: a language for scenario specification and scene generation. In: McKinley, K.S., Fisher, K. (eds.) Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, 22–26 June 2019, pp. 63–78. ACM (2019). <https://doi.org/10.1145/3314221.3314633>, <https://doi.org/10.1145/3314221.3314633>
19. Fremont, D.J., Seshia, S.A.: Reactive control improvisation. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 307–326. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-96145-3\\_17](https://doi.org/10.1007/978-3-319-96145-3_17)

20. Gittis, A., Vin, E., Fremont, D.J.: Randomized synthesis for diversity and cost constraints with control improvisation (artifact). <https://doi.org/10.5281/zenodo.6558391>
21. Gittis, A., Vin, E., Fremont, D.J.: Randomized synthesis for diversity and cost constraints with control improvisation (2022). <https://arxiv.org/abs/2206.02775>
22. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. *Formal Aspects Comput.* **6**(5), 512–535 (1994)
23. Hickey, T., Cohen, J.: Uniform random generation of strings in a context-free language. *SIAM J. Comput.* **12**(4), 645–655 (1983). <https://doi.org/10.1137/0212044>
24. Kozierek, C.M.: The TCP/IP guide. [http://tcpipguide.com/free/t\\_TCPOperationalOverviewandtheTCPFiniteStateMachineF-2.htm](http://tcpipguide.com/free/t_TCPOperationalOverviewandtheTCPFiniteStateMachineF-2.htm), (Accessed Jan 21 2022)
25. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. *IEEE Trans. Rob.* **25**(6), 1370–1381 (2009)
26. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78800-3\\_24](https://doi.org/10.1007/978-3-540-78800-3_24)
27. Soos, M., Gocht, S., Meel, K.S.: Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling. In: Proceedings of International Conference on Computer-Aided Verification (CAV), July 2020
28. Soos, M., Meel, K.S.: Arjun: an efficient independent support computation technique and its applications to counting and sampling. *CoRR* abs/2110.09026 (2021). <https://arxiv.org/abs/2110.09026>
29. Sutton, M., Greene, A., Amini, P.: Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley (2007)
30. Vazquez-Chanlatte, M., Junges, S., Fremont, D.J., Seshia, S.: Entropy-guided control improvisation (2021). <https://doi.org/10.15607/RSS.2021.XVII.051>, <https://doi.org/10.15607/RSS.2021.XVII.051>

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

