Automatic Generation of Quality Specifications

Shaull Almagor, Guy Avni, and Orna Kupferman

School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel

Abstract. The logic $\mathrm{LTL}^{\triangledown}$ extends LTL by quality operators. The satisfaction value of an $\mathrm{LTL}^{\triangledown}$ formula in a computation refines the $0/1$ value of LTL formulas to a real value in $[0,1]$. The higher the value is, the better is the quality of the computation. The quality operator $\triangledown_\lambda$, for a quality constant $\lambda \in [0,1]$, enables the designer to prioritize different satisfaction possibilities. Formally, the satisfaction value of a sub-formula $\triangledown_\lambda \varphi$ is λ times the satisfaction value of φ. For example, the $\mathrm{LTL}^{\triangledown}$ formula $G(req \rightarrow (X\,grant \vee \triangledown_{\frac{1}{2}} F\,grant))$ has value 1 in computations in which every request is immediately followed by a grant, value $\frac{1}{2}$ if grants to some requests involve a delay, and value 0 if some request is not followed by a grant.

The design of an $\mathrm{LTL}^{\triangledown}$ formula typically starts with an LTL formula on top of which the designer adds the parameterized $\triangledown$ operators. In the Boolean setting, the problem of automatic generation of specifications from binary-tagged computations is of great importance and is a very challenging one. Here we consider the quantitative counterpart: an $\mathrm{LTL}^{\triangledown}$ query is an $\mathrm{LTL}^{\triangledown}$ formula in which some of the quality constants are replaced by variables. Given an $\mathrm{LTL}^{\triangledown}$ query and a set of computations tagged by satisfaction values, the goal is to find an assignment to the variables in the query so that the obtained $\mathrm{LTL}^{\triangledown}$ formula has the given satisfaction values, or, if this is impossible, best approximates them. The motivation to solving $\mathrm{LTL}^{\triangledown}$ queries is that in practice it is easier for a designer to provide desired satisfaction values in representative computations than to come up with quality constants that capture his intuition of good and bad quality.

We study the problem of solving $\mathrm{LTL}^{\triangledown}$ queries and show that while the problem is NP-hard, interesting fragments can be solved in polynomial time. One such fragment is the case of a single tagged computation, which we use for introducing a heuristic for the general case. The polynomial solution is based on an analysis of the search space, showing that reasoning about the infinitely many possible assignments can proceed by reasoning about their partition into finitely many classes. Our experimental results show the effectiveness and favorable outcome of the heuristic.

1 Introduction

Traditional formal methods are based on a Boolean satisfaction notion – a reactive system satisfies, or not, a given specification. In recent years there is growing need and interest in formalizing and reasoning about quantitative systems and properties. This includes, for example, probabilistic [16], fuzzy [18], and accumulative [9] settings. An exciting direction in this effort is the development of formalisms and methods for reasoning about the *quality* of systems [1,2]. The working assumption in these works is that

N. Sharygina and H. Veith (Eds.): CAV 2013, LNCS 8044, pp. 479–494, 2013.

satisfying a specification is not a yes/no matter. Different ways of satisfying a specification should induce different levels of quality, which should be reflected in the semantics of the specification formalism. In particular, in [2], the authors introduce an extension of linear temporal logic (LTL [19]) by a quantitative layer that enables the designer to prioritize different satisfaction possibilities. In the extended setting, the satisfaction value of a formula in a computation refines the $0/1$ value of LTL formulas to a real value in $[0, 1]$. The higher the value is, the better is the quality of the computation.

The extension uses a family of *propositional quality operators*. A basic such operator is $\triangledown_\lambda$, for a *quality constant* $\lambda \in [0, 1]$ that multiplies the satisfaction value of its operand by λ. We consider the logic $\mathrm{LTL}^\triangledown$, which extends LTL by the $\triangledown_\lambda$ operator. The standard LTL operators are adjusted in $\mathrm{LTL}^\triangledown$ to values in $[0, 1]$: disjunctions are interpreted as $\max$, negation as subtraction from 1, and so on. For example, the satisfaction value of the formula $\psi_1 \vee \triangledown_{\frac{1}{2}} \psi_2$ in a computation π is the maximum between the satisfaction value of ψ_1 in π, and $\frac{1}{2}$ the satisfaction value of ψ_2 in π. As a more elaborate example, consider a system that grants locks to a data structure. The system can grant either a read-only lock or a read-write lock. The quality of the system may be specified as $\mathsf{G}(\mathit{req} \rightarrow \mathsf{X}(\mathit{read\text{-}write} \vee (\triangledown_{\frac{3}{4}} \mathit{read\text{-}only}))$, stating that receiving read-write lock gives satisfaction value of 1, whereas receiving a read-only lock reduces that satisfaction value to $\frac{3}{4}$, In [2], the authors demonstrate the usefulness of the ability to specify quality and solve the model-checking and synthesis problems for $\mathrm{LTL}^\triangledown$.

Already in the Boolean setting, both model checking and synthesis rely on the specification to accurately reflect the designer's intention. One of the criticisms against formal method is that the latter challenge, of coming up with correct specifications, is not much easier than model checking or synthesis. Thus, formal methods merely shift the difficulty of developing correct implementations to that of developing correct specifications [13]. *Property assurance* is the activity of eliciting specifications that faithfully capture designer intent [8,21]. One approach for property assurance is to challenge given specifications with sanity checks like non-validity, satisfiability, and vacuity [15]. More involved quality checks are studied in the PROSYD project [20] [1]. A second approach is that of automatic generation of specifications. This includes ideas from *learning*, where specifications given by means of automata are learned from a sample of behaviors tagged as good or bad [6,17], methods based on a generation of specifications from basic *patterns* [11], and *specification mining*, where specifications are generated by analyzing the runs of the given system [4]. In the novel quantitative setting, there is (yet) no experience in specification design nor tools or methods for property assurance. In this paper, we introduce such a method and study the problem of automatically generating the quality layer in $\mathrm{LTL}^\triangledown$ formulas.

The design of an $\mathrm{LTL}^\triangledown$ formula typically starts with an LTL formula on top of which the designer adds the parameterized $\triangledown$ operators. The underlying assumption behind our approach is that in practice it is easier for a designer to provide desired satisfaction values in representative computations than to come up with quality constants that capture his intuition of high and low quality. This resembles the classical process of learning

[1] A related line of research is that of specification debugging [5], where, in the process of model checking, counterexamples are automatically clustered together in order to make the manual debugging of temporal properties easier.

a hypothesis from tagged samples. Formally, an LTL^{∇} *query* is an LTL^{∇} formula in which some of the quality constants are replaced by variables. A *path constraint* is a pair $\langle \pi, I \rangle$, where π is a lasso-shaped path and $I \subseteq [0,1]$ is a closed interval. Consider an LTL^{∇} query φ with variables in $\mathcal{X}$. For an assignment $f : \mathcal{X} \to [0,1]$, we use φ^f to denote the LTL^{∇} formula obtained from φ by replacing each variable $x \in \mathcal{X}$ by $f(x)$. The LTL^{∇} query problem is to find, given an LTL^{∇} query φ and a set $\mathcal{C}$ of path constraints, an assignment f to the variables in φ so that φ^f satisfies all the constraints (or returns that no such assignment exists). Thus, for all $\langle \pi, I \rangle \in C$, the satisfaction value of φ^f in π is in the interval I. Note that I may (but need not) be a single point. Note that beyond the restrictions on the quality constants in φ^f that follow from the constraints, restrictions may be induced also by repeated occurrences of the same variable. Subtle connections between different constants can be specified too, using nesting. In practice, however, most queries are simple (that is, each variable appears only once) and are free of nesting.

As an example, consider the following specification: "After a request, an ack should ideally be given immediately and hold for two time units. An ack that holds only for one time unit is also acceptable, provided that it is given within two time units". A designer that wants to formalizes "ideally" and "acceptable" may have a clear idea that he wants to upper bound the satisfaction value of a policy with a single time-unit ack by $\frac{3}{4}$ but may find it difficult to come up with the exact "penalty" for a delay in this case. This situation is captured by the following LTL^{∇} query:

$$\mathsf{G}(req \rightarrow ((\mathsf{X}ack \wedge \mathsf{XX}ack) \vee \nabla_{\frac{3}{4}}(\nabla_x(\mathsf{X}ack) \vee \nabla_y(\mathsf{XX}ack)))).$$

The satisfaction value of an induced LTL^{∇} formula in a computation with an ideal ack policy is 1. In a computation with a single time-unit ack, it is at most $3/4$, to be further tuned down by the assignments to x and y. Grading given behaviors is easier than finding an assignment that captures the designer's intuition. For example, the designer may declare that a computation $(\{req\}, \{ack\})^\omega$ is not that bad, and satisfies the specification with quality $\frac{3}{4}$. Also, the path $(\{req\}, \emptyset, \{ack\})^\omega$ satisfies it with quality in $[\frac{1}{3}, \frac{1}{2}]$. A solution to the corresponding LTL^{∇} query problem suggests an assignment to x and y that satisfies the designer's constraints. For example, $x = 1$ and $y = \frac{1}{2}$. As we discuss in Section 2.2, it is possible to automate not only the generation of quality constraints, but also the generation of LTL^{∇} queries out of LTL formulas.

Before we continue to describe our results, let us review other settings with partially-specified systems or specifications. In the Boolean setting, reasoning about partially-specified systems is useful in automatic partial synthesis [22] and program repair [14]. From the other direction, partially-specified specifications are used for system exploration. In particular, in *query checking* [10], the specification contains variables, and the goal is to find an assignment to the variables with which the explored system satisfies the specification. While the formulation of the problem is similar, the motivation is very different, as the goal is to explore, synthesize, or reason about the system, whereas our goal here is automatic generation of specifications. The fact we consider the quantitative setting makes the underlying considerations and algorithms very different too. In the quantitative setting, related work includes parameterized weighted containment [7], where a partially specified weighted automaton is given, and the goal is to find an assignment to the missing weights such that a containment constraint is met.

An orthogonal research direction is that of parametric real-time reasoning [3]. There, the quantitative nature of the automata origins from real-time constraints, the semantics is very different, and the goal is to find restrictions on the behavior of the clocks such that the automata satisfy certain properties.

We start by showing that in general, the $\mathrm{LTL}^{\triangledown}$ query problem is NP-hard. Checking whether a suggested assignment satisfies the set of constraints can be done in polynomial time, suggesting that the problem is in NP. One, however, also has to consider the domain and representation of the interval constraints and the assignment. For example, the only solution to the query $\triangledown_x \triangledown_x p$ and the constraint $\langle \{p\}^\omega, \frac{1}{2} \rangle$ assigns to x the value $1/\sqrt{2}$, which is irrational and thus does not have a finite representation in a binary expansion. For common queries, in particular simple queries without nesting of variables, we are able to prove that a "short" satisfying assignment exists, making the problem NP-complete.

We proceed to study a fragment of the problem, where the $\mathrm{LTL}^{\triangledown}$ queries are simple and the set of constraints includes a single computation. We show that in this case, the problem can be solved in polynomial time.[2] Our polynomial algorithm is based on an analysis of the search space, showing that the infinitely many possible assignments to each of the variables x in the query can be rounded up to linearly many ones, depending on the desired satisfaction value and the structure of the formula inside which x is nested. The induced space of possible assignments can be then searched efficiently.

Finally, we use the case of a single constraint in a heuristic for the general case. To do so, it is convenient to consider the problem in a geometrical perspective: consider an $\mathrm{LTL}^{\triangledown}$ query φ over k variables and a set of constraints $\mathcal{C}$. Every constraint $\langle \pi, I \rangle \in \mathcal{C}$ induces a set $S_{\langle \pi, I \rangle, \varphi} \subseteq [0,1]^k$ of solutions to that constraint. The $\mathrm{LTL}^{\triangledown}$ query problem for $\mathcal{C}$ amounts to finding a point $f \in \bigcap_{\langle \pi, I \rangle \in \mathcal{C}} S_{\langle \pi, I \rangle, \varphi}$. [3] The geometrical perspective makes it easy to define an optimization problem: we seek an assignment that minimizes the sum (over $\langle \pi, I \rangle \in \mathcal{C}$) of distances to $S_{\langle \pi, I \rangle, \varphi}$. We suggest three heuristics for finding an assignment, and evaluate them according to two *loss function*, namely ways to define distances in the $[0,1]^k$ space: number of constraints satisfied, and distance in $\| \cdot \|_2$. The three heuristics combine a consideration of external assignments as well as the center of gravity of the assignments for the underlying constraints. We implemented our algorithms and examined the quality constants generated for various specifications. As detailed in Section 6, our results show the effectiveness and favorable outcome of the approach and algorithms.

Due to lack of space, most proofs are omitted and can be found in the full version at the authors' URLs.

2 The Logic $\mathrm{LTL}^{\triangledown}$

The logic $\mathrm{LTL}^{\triangledown}$ is a multi-valued logic that extends the linear temporal logic LTL with a parameterized quality operator ∇_λ. The logic, along with model-checking and

[2] We note that both requirements, of simple queries and a singleton constraint are needed; removing one of them we are back to NP-hardness.

[3] The main difficulty in solving the $\mathrm{LTL}^{\triangledown}$ query problem is that even for a single constraint, this set may not be convex, and therefore not amenable to methods of convex analysis.

synthesis algorithms for it, was introduced in [2]. We start by defining its syntax and semantics. Let AP be a set of Boolean atomic propositions[4]. An LTL$^{\nabla}$ formula is one of the following:

- $\texttt{True}$, $\texttt{False}$, or p, for $p \in AP$.
- $\neg\varphi$, $\varphi \vee \psi$, $\nabla_\lambda \varphi$, $\mathsf{X}\varphi$, or $\varphi \mathsf{U} \psi$, for LTL$^{\nabla}$ formulas φ and ψ, and a *quality constant* $\lambda \in [0,1]$.

The semantics of LTL$^{\nabla}$ is defined with respect to infinite computations over AP. Each position in the computation corresponds to a valuation to the atomic propositions, thus a computation is a word $\pi = \pi_0, \pi_1, \ldots \in (2^{AP})^\omega$. We use π^i to denote the suffix $\pi_i, \pi_{i+1}, \ldots$ of π. The semantics maps a computation π and an LTL$^{\nabla}$ formula φ to the satisfaction value of φ in π, denoted $[\![\pi, \varphi]\!]$. The satisfaction value is in $[0,1]$, defined by induction on the structure of φ as described in Table 1 below. As with LTL, we use $\mathsf{F}\psi$ ("eventually") and $\mathsf{G}\psi$ ("always") as abbreviations for $\texttt{True}\mathsf{U}\psi$ and $\neg\mathsf{F}\neg\psi$, respectively, as well as the standard Boolean abbreviations $\wedge$ and $\rightarrow$.

Table 1. The semantics of LTL$^{\nabla}$

Formula	Satisfaction Value
$[\![\pi, \texttt{True}]\!]$	1
$[\![\pi, \texttt{False}]\!]$	0
$[\![\pi, p]\!]$	1 if $p \in \pi_0$ 0 if $p \notin \pi_0$
$[\![\pi, \neg\varphi]\!]$	$1 - [\![\pi, \varphi]\!]$
$[\![\pi, \varphi \vee \psi]\!]$	$\max([\![\pi, \varphi]\!], [\![\pi, \psi]\!])$
$[\![\pi, \nabla_\lambda \varphi]\!]$	$\lambda \cdot [\![\pi, \varphi]\!]$
$[\![\pi, \mathsf{X}\varphi]\!]$	$[\![\pi^1, \varphi]\!]$
$[\![\pi, \varphi \mathsf{U} \psi]\!]$	$\max_{i \geq 0}\{\min\{[\![\pi^i, \psi]\!], \min_{0 \leq j < i} [\![\pi^j, \varphi]\!]\}\}$

Evaluating LTL$^{\nabla}$ Formulas on Lasso Computations. We say that a computation π is a *lasso* if $\pi = u \cdot v^\omega$, for finite computations $u, v \in (2^{AP})^*$ with $v \neq \epsilon$. We refer to u as the *prefix* of the lasso and to v as its *cycle*. The standard bottom-up labeling algorithm for model checking LTL formulas with respect to lasso computations can be easily extended to LTL$^{\nabla}$. The algorithm is based on the simple observation that if $[\![\pi^i, \psi]\!]$ is known for all $i \geq 0$ and subformulas ψ of φ, then it is possible to calculate, in time linear in $|u| + |v|$, the values $[\![\pi^i, \varphi]\!]$, for all $i \geq 0$. Indeed, the periodicity of π implies that there are only $|u| + |v|$ different suffixes to consider, and, by the semantics of LTL$^{\nabla}$, the satisfaction value of φ can be easily inferred from the satisfaction value of its subformulas. The only non-trivial case is when $\varphi = \psi_1 \mathsf{U} \psi_2$, but also there, one can start with the satisfaction value of ψ_2 and then repeatedly go back the lasso checking for every suffix whether, taking the satisfaction value of ψ_1 into an account, it is worthwhile to postpone the satisfaction of the eventuality. To conclude, we have the following.

Proposition 1. *Given an* LTL$^{\nabla}$ *formula* φ *and finite computations* $u, v \in (2^{AP})^*$*, calculating* $[\![u \cdot v^\omega, \varphi]\!]$ *can be done in time* $O(|\varphi| \cdot (|u| + |v|))$.

[4] As discussed in Remark 1 (Section 4), it is possible to extend the definition as well as our results to weighted atomic propositions with values in $[0,1]$.

2.1 LTL^{∇} Queries

Let $\mathcal{X}$ be a finite set of variables. An LTL^{∇} *query* (over $\mathcal{X}$) is an LTL^{∇} formula in which some of the quality constants are replaced with variables from $\mathcal{X}$. For example, $\varphi = \mathsf{G}(req \rightarrow ((\nabla_{x_1} X(read \vee \nabla_{x_2} X\, read)) \vee (\nabla_{x_3} X\, write) \vee (\nabla_{\frac{3}{4}} halt)))$ is an LTL^{∇} query over $\{x_1, x_2, x_3\}$. We say that an LTL^{∇} query is *simple* if each of its variables occurs only once. The *depth* of an LTL^{∇} query φ is the maximal nesting depth of variables in φ. For example, φ above is simple and is of depth 2. Note that, as in φ above, not all quality constraints are replaced by variables. For an LTL^{∇} query φ, we denote by $var(\varphi)$ the set of variables $x \in \mathcal{X}$ such that $\nabla_x \psi$ is a subformula of φ. Given an assignment $f : \mathcal{X} \rightarrow [0,1]$, we define φ^f to be the LTL^{∇} formula obtained from φ by replacing every occurrence of $x \in \mathcal{X}$ with $f(x)$. Note that an assignment f as above prioritizes the different possible ways to satisfy the specification. In φ above, the assignment to x_1 and x_3 reflects the priority of the designer as to whether a read lock or a write lock is granted after a request, and the assignment to x_2 reflects the cost of a delayed read lock.

Consider an LTL^{∇} query φ. A *path constraint* is a pair $\langle \pi, I \rangle$ such that $\pi \in (2^{AP})^{\omega}$ and $I \subseteq [0,1]$ is a closed interval; that is, $[a,b]$ for $0 \leq a \leq b \leq 1$. A *lasso constraint* is a path constraint in which π is a lasso. When the interval I is a single point, thus $a = b$, we only state the point in the specification of the constraint.

The LTL^{∇} query problem is to decide, given an LTL^{∇} query φ and a set $\mathcal{C}$ of lasso constraints, whether there exists an assignment f to $var(\varphi)$ such that $[\![\pi, \varphi^f]\!] \in I$ for all $\langle \pi, I \rangle \in \mathcal{C}$. We then say that the assignment f is a *solution* to $\langle \varphi, \mathcal{C} \rangle$.

2.2 Generating LTL^{∇} Queries

Our algorithms for solving the LTL^{∇} query problem takes as input an LTL^{∇} query, which is up to the designer to write. While the semantics of LTL^{∇} is easy to understand and use, there are some caveats one should be aware of when designing LTL^{∇} formulas and queries. In this section we demonstrate methods to soundly design LTL^{∇} queries. Moreover, the proposed methods can be automated, so that the designer may actually start with an LTL formula, rather than an LTL^{∇} query.

Typically, LTL^{∇} formulas are obtained from LTL formulas by adding the ∇_λ operator to various components. A common pattern for LTL specifications is a conjunction of properties. Consider a conjunction $\alpha \wedge \beta$. Assume that the satisfaction of β is less crucial than that of α. Specifically, if only β holds, we want the satisfaction value to be 0, but if only α holds, the satisfaction value is $\frac{3}{4}$. Note that a naive introduction of the ∇_λ operator may result in an undesirable behavior. In particular, according to the semantics of LTL^{∇}, the satisfaction value of $\alpha \wedge \nabla_{\frac{3}{4}} \beta$ is at most $\frac{3}{4}$, and the contribution of α, for values above $\frac{3}{4}$, is irrelevant. We now demonstrate two sound methods for adding ∇_λ operators.

As discussed in Section 1, different ways of satisfying a formula induce different qualities. A conjunction has only one way to be satisfied, thus in order to prioritize its components we decompose its components into a disjunction of cases, on which we apply the ∇_λ operator. For example, $\alpha \wedge \beta$ becomes $(\nabla_{\lambda_1}(\alpha \wedge \beta)) \vee (\nabla_{\lambda_2} \alpha) \vee$

$(\triangledown_{\lambda_3}\beta)$. After this transformation, the λ_i quality constants reflect the *gain* from the corresponding disjuncts.

The second method is to use negations to dualize the behavior of $\triangledown_\lambda$. Consider the formula $\neg\triangledown_{(1-\lambda)}\neg\varphi$ for some formula φ. Note that $[\![\pi, \neg\triangledown_{(1-\lambda)}\neg\varphi]\!] = 1-(1-\lambda)(1-[\![\pi,\varphi]\!])$. We use the abbreviation $\blacktriangledown_\lambda\varphi = \neg\triangledown_{(1-\lambda)}\neg\varphi$. In particular, if $[\![\pi,\varphi]\!] = 1$, then $\blacktriangledown_\lambda\varphi = 1$, and if $[\![\pi,\varphi]\!] = 0$, then $\blacktriangledown_\lambda\varphi = \lambda$. The operator $\blacktriangledown_\lambda$ does work well with conjunctions. For example, $\alpha \wedge \beta$ becomes $(\blacktriangledown_{\lambda_1}\alpha) \wedge (\blacktriangledown_{\lambda_2}\beta)$, with λ_i indicating the *loss* when the corresponding conjuncts do not hold. In the full version we formalize this intuition.

3 Solving the $\mathrm{LTL}^\triangledown$ Query Problem

In this section we study the complexity of the $\mathrm{LTL}^\triangledown$ query problem and show that it is NP-hard. As follows from Proposition 1, given an $\mathrm{LTL}^\triangledown$ query φ, a set $\mathcal{C}$ of lasso constraints, and an assignment $f : var(\varphi) \to [0,1]$, it is possible to check in linear time whether f is a solution for $\langle\varphi,\mathcal{C}\rangle$. Indeed, for each of the lasso constraints $\langle\pi, I\rangle \in \mathcal{C}$ we can calculate $[\![\pi,\varphi^f]\!]$ and verify that it is in I. This suggests that the $\mathrm{LTL}^\triangledown$ query is in NP, as given a witness assignment f, we can verify it efficiently. Membership in NP, however, also requires the witness f to be polynomial in the φ and $\mathcal{C}$.

The latter requirement adds to the picture considerations like the domain and representation of the interval constraints. A natural suggestion is to assume that all intervals I are of the form $[a,b]$ for rational numbers $0 \le a, b \le 1$, given by their binary expansion. As we now demonstrate, things are involved already in this case. To see why, consider the query $\triangledown_x\triangledown_x p$ and the constraint $\langle\{p\}^\omega, \frac{1}{2}\rangle$. The single solution to the problem is f with $f(x) = 1/\sqrt{2}$. But $1/\sqrt{2}$ is irrational, and therefore its binary expansion is infinite. Thus, while it is possible that the problem is in NP, describing a witness for an input requires a more sophisticated way of encoding solution, which is of debatable interest to the CAV community. As good news, in Section 4.1 we show that for typical instances of the problem, namely simple queries of depth 1, short witnesses exist, making the problem NP-complete for them. The proof requires results we develop in Section 4. Here, we describe the lower bound.

Theorem 1. *The* $\mathrm{LTL}^\triangledown$ *query problem is NP-hard.*

Proof: We describe a reduction from 3-SAT. Let $\theta = (l_1^1 \vee l_2^1 \vee l_3^1) \wedge ... \wedge (l_1^k \vee l_2^k \vee l_3^k)$ be a 3-CNF formula. We construct an $\mathrm{LTL}^\triangledown$ query φ and a set $\mathcal{C}$ of constraints such that θ is satisfiable iff there is a solution for $\langle\varphi,\mathcal{C}\rangle$. Let $X = \{x_1, ..., x_m\}$ be the set of variables that appear in θ. We define $AP = \{p_1, n_1, ..., p_m, n_m\}$ and $\mathcal{X} = \{y_1, z_1, \ldots, y_m, z_m\}$. Intuitively, the proposition p_i (resp. n_i) stands for "the variable x_i appears positively (resp. negatively) in the clause", and we define the query and the constraints so that the variable y_i (resp. z_i) is assigned 1 when x_i is assigned `True` (resp. `False`).

We define $\varphi = \mathsf{G}(\triangledown_{y_1}p_1 \vee \triangledown_{z_1}n_1 \vee ... \vee \triangledown_{y_m}p_m \vee \triangledown_{z_m}n_m)$. We first have to ensure that in every solution to the query, at least one of the variables $\{y_i, z_i\}$ gets value 0, for all $1 \le i \le m$. This is done by the constraint $\langle\pi_i, 0\rangle$, with $\pi_i = \{p_i\}\{n_i\}(AP)^\omega$. Note that in order for $[\![\pi_i,\varphi]\!]$ to be 0, it must be that either $[\![\{p_i\}, \triangledown_{y_i}p_i]\!] = 0$ or $[\![\{n_i\}, \triangledown_{z_i}n_i]\!] = 0$, implying that indeed at least one of the variables $\{y_i, z_i\}$ has value 0.

The family of m constraints above guarantees that a solution f to the query induces a truth assignment to X: the variable x_i is assigned `True` iff $f(y_i) = 1$. It is left to ensure that f induces a satisfying assignment. This is done by the constraint $\langle \pi, 1 \rangle$, where $\pi = \{s_1^1, s_2^1, s_3^1\} \cdots \{s_1^k, s_2^k, s_3^k\} \cdot (AP)^\omega$ is such the i-th position corresponds to the i-th clause and ensures that at least one of its literals gets value `True`. Accordingly, s_j^i is p_t if $l_j^i = x_t$ and is n_t if $l_j^i = \neg x_t$. Note that in order for $[\![\pi, \varphi]\!]$ to be 1, it must be that $[\![\{s_1^i, s_2^i, s_3^i\}, \triangledown_{y_t} p_t]\!] = 1$ or $[\![\{s_1^i, s_2^i, s_3^i\}, \triangledown_{z_t} z_t]\!] = 1$, for some t such that x_t appears in the i-th clause. If x_t appears positively in the clause, then one of the s_j^i's is p_t, and if x_t appears negatively, then one of them is n_t. Thus, in a solution f, one of the corresponding variables – that is, y_t in the first case and n_t in the second, is assigned 1.

It is easy to see that the reduction is polynomial.

□

Theorem 1 motivates a study of special easy cases of the LTL$^\triangledown$ query problem. Since the reduction in the proof of Theorem 1 uses a query with multiple constraints, a natural candidate is the case of a single constraint. Lemma 1 below hints that this case is not easier.

Lemma 1. *Let φ be a simple* LTL$^\triangledown$ *query over a set $\mathcal{X}$ of variables and let $\mathcal{C}$ be a set of lasso constraints of the form $\langle \pi, 1 \rangle$ or $\langle \pi, 0 \rangle$. Then, there exists an* LTL$^\triangledown$ *query φ' over $\mathcal{X}$ and a lasso π such that for every assignment $f : \mathcal{X} \to [0,1]$, we have that f is a solution to $(\varphi', \{\langle \pi, 0 \rangle\})$ iff f is a solution to $\langle \varphi, \mathcal{C} \rangle$. In addition, the length of the prefix of π is the length of the longest prefix of a lasso in $\mathcal{C}$, and the length of its cycle is the* lcm *(least common multiple) of the lengths of the cycles in the lassos in $\mathcal{C}$.*

Proof: Let AP be the set of atomic propositions in φ, and let $\mathcal{C} = \{\langle u_1 \cdot v_1^\omega, c_1 \rangle, \ldots, \langle u_k \cdot v_k^\omega, c_k \rangle\}$. We define AP' as k disjoint copies of AP, thus $AP' = AP \times \{1, \ldots, k\}$. We define φ_j to be the LTL$^\triangledown$ query obtained from φ by replacing each atomic proposition $p \in AP$ by the atomic proposition $\langle p, j \rangle \in AP'$. Let $u, v \in 2^{AP'}$ be such that for all $1 \leq j \leq k$, the projection of $u \cdot v^\omega$ on $AP \times \{j\}$ agrees with $u_j \cdot v_j^\omega$. It is easy to define u and v as above by taking u of length $m = \max_i |u_i|$ and v of length $\ell = lcm(|v_1|, ..., |v_k|)$. Indeed, the labeling of u by $AP \times \{j\}$ is obtained by concatenating to u_j a prefix of v_j^ω of length $m - |u_j|$ and the labeling of v by $AP \times \{j\}$ is then obtained from v_j' by the corresponding shift of $v_j^{\ell/|v_j|}$.

Now, we define $\varphi' = \bigwedge_{j=1}^k \psi_j$, where ψ_j is either φ_j, in case $c_j = 1$, or is $\neg\varphi_j$, in case $c_j = 0$. Consider an assignment $f : \mathcal{X} \to [0,1]$. Since φ' is defined as a conjunction, then the constraint $\langle u \cdot v^\omega, 1 \rangle$ is met for φ' iff $[\![u \cdot v^\omega, \psi_j^f]\!] = 1$ for all $1 \leq j \leq k$, which, by the definition of ψ_j, holds iff f is a solution to $\langle \varphi, \{\langle u_j \cdot v_j^\omega, c_j \rangle\} \rangle$. To conclude, f is a solution to $\langle \varphi', \{\langle u \cdot v^\omega, 1 \rangle\} \rangle$ iff f is a solution to $\langle \varphi, \mathcal{C} \rangle$. □

While the single lasso constructed in Lemma 1 may be exponential in the original constraints, examining the lassos that are used in the reduction in the proof of Theorem 1, we see that they all have cycles of length 1. Therefore, Lemma 1 together with the reduction there imply that the special case of a single constraint is not easy. Formally, we have the following.

Theorem 2. *The* $\mathrm{LTL}^{\triangledown}$ *query problem is NP-hard even for the case of a single lasso constraint.*

4 A Feasible Special Case

While Theorem 2 implies that the $\mathrm{LTL}^{\triangledown}$ query problem is hard already for a single constraint, the transformation described in the proof of Lemma 1 generates formulas that are not simple. Indeed, the transformation is based on a relation between multiple constraints and multiple occurrences of a variable. In this section we show that in a setting with both limitations, the $\mathrm{LTL}^{\triangledown}$ query problem can be solved efficiently. Formally, we prove the following.

Theorem 3. *The* $\mathrm{LTL}^{\triangledown}$ *query problem for simple queries and a single constraint can be solved in polynomial time.*

In Section 5, we show that Theorem 3 and the algorithm developed for its proof are useful in approximation and heuristic algorithms for the general case.

Let φ be a simple $\mathrm{LTL}^{\triangledown}$ query over AP and $\mathcal{X}$, and let π be a computation. Let $k = |\mathcal{X}|$. Consider the function $\mu_{\pi,\varphi} : [0,1]^k \to [0,1]$ defined by $\mu_{\pi,\varphi}(f) = [\![\pi, \varphi^f]\!]$.

We start with some useful observations.

Lemma 2. *For all computations* π *and* $\mathrm{LTL}^{\triangledown}$ *queries* φ*, the function* $\mu_{\pi,\varphi}$ *is continuous. That is, for every infinite sequence* $(a_n)_{n=1}^{\infty}$ *of points in* $[0,1]^k$ *such that* $\lim_{n\to\infty} a_n = a$*, it holds that* $\lim_{n\to\infty}(\mu_{\pi,\varphi}(a_n)) = \mu_{\pi,\varphi}(a)$.

Consider a variable $x \in \mathcal{X}$. Since φ is simple, the variable x is either *positive* in φ, in case the subformula $\triangledown_x \psi$ is in the scope of an even number of negation, or is *negative* in φ, otherwise. We refer to the positivity or negativity of x in φ as its *polarity* in φ.

Lemma 3. *For all computations* π *and* $\mathrm{LTL}^{\triangledown}$ *queries* φ*, the function* $\mu_{\pi,\varphi}$ *is monotonic in each variable. Specifically, for every variable* $x \in \mathcal{X}$*, if* x *is positive in* φ *then* $\mu_{\pi,\varphi}$ *is increasing with* x *and if* x *is negative in* φ *then* $\mu_{\pi,\varphi}$ *is decreasing with* x*.*

We note that proofs of Lemmas 2 and 3 are by induction on the structure of φ.

The idea behind our polynomial algorithm is to limit the search space for a satisfying assignment. Before defining the limited search space, let us first observe that an $\mathrm{LTL}^{\triangledown}$ formula has finitely (in fact, linearly many) possible satisfaction values. We define the set of possible values of φ, denoted $val(\varphi)$, by induction on the structure of φ as follows.

- If $\varphi = p \in AP$, then $val(p) = \{0, 1\}$.
- If $\varphi = \psi_1 \vee \psi_2$ or $\varphi = \psi_1 U \psi_2$, then $val(\varphi) = val(\psi_1) \cup val(\psi_2)$.
- If $\varphi = \neg\psi$, then $val(\varphi) = \{1 - v : v \in val(\psi)\}$.
- If $\varphi = \mathsf{X}\psi$, then $val(\varphi) = val(\psi)$.
- If $\varphi = \triangledown_\lambda \psi$, then $val(\varphi) = \{\lambda \cdot v : v \in val(\psi)\}$.

It is easy to prove that for every path π it holds that $[\![\pi, \varphi]\!] \in val(\varphi)$.

We start by defining the limited search space for $\mathrm{LTL}^{\triangledown}$ queries of depth 1. We will later generalize the definition to all depths. Let φ be a simple $\mathrm{LTL}^{\triangledown}$ query of depth 1. For $x \in var(\varphi)$ and $c \in [0,1]$ we define the set of relevant values for x with respect to φ and c, denoted $val(x, \varphi, c)$, by induction on the structure of φ as follows.

- If $\varphi = \nabla_x \psi$, for a LTL$^\nabla$ formula ψ, then $val(x, \varphi, c) = \{\frac{c}{v} : v \in val(\psi) \text{ and } v \geq c\}$. Note that since φ is of nesting depth 1, then ψ has no variables, and this is the base case for the induction.
- If $\varphi = \psi_1 \vee \psi_2$ or $\varphi = \psi_1 U \psi_2$, then $val(x, \varphi, c) = val(x, \psi_i, c)$, for the single $i \in \{1, 2\}$ such that $x \in var(\psi_i)$.
- If $\varphi = \mathsf{X}\psi$, then $val(x, \varphi, c) = val(x, \psi, c)$.
- If $\varphi = \neg\psi$, then $val(x, \varphi, c) = val(x, \psi, 1 - c)$.
- If $\varphi = \nabla_\lambda \psi$, we distinguish between two cases. If $\lambda \geq c$, then $val(x, \varphi, c) = val(x, \psi, \frac{c}{\lambda})$. Otherwise, $val(x, \varphi, c) = \emptyset$.

Lemma 4 below justifies the restricted search space. Consider a value $u \in [0, 1]$. Let $up_{x,\varphi,c}(u)$ and $down_{x,\varphi,c}(u)$ be the "rounding" up and down of u to the nearest value in $val(x, \varphi, c)$. Formally, $up_{x,\varphi,c}(u) = \min\{v : v \in val(x, \varphi, c) \text{ and } v \geq u\}$ and $down_{x,\varphi,c}(u) = \max\{v : v \in val(x, \varphi, c) \text{ and } v \leq u\}$.

Consider an assignment $f : \mathcal{X} \to [0, 1]$. For a variable $x \in \mathcal{X}$, define the assignments $f^+_{x,\varphi,c}$ and $f^-_{x,\varphi,c}$ as the assignments obtained from f by leaving the assignments to all variables except x unchanged and rounding the value of x up or down to the nearest value in $val(x, \varphi, c)$. The decision whether to round the value of x up or down depends on the $+$ and $-$ indication as well as in the polarity of x in φ. Formally, we have the following.

$$f^+_{x,\varphi,c}(x) = \begin{cases} up_{x,\varphi,c}(f(x)) & \text{if } x \text{ is positive in } \varphi, \\ down_{x,\varphi,c}(f(x)) & \text{if } x \text{ is negative in } \varphi, \end{cases}$$

and dually (switch positive and negative) for $f^-_{x,\varphi,c}(x)$.

Lemma 4. *Consider a simple* LTL$^\nabla$ *query* φ *of depth* 1 *and an assignment* f *to* $var(\varphi)$. *Let* $c \in [0, 1]$, *and let* π *be a computation. Then,*

1. *If* $[\![\pi, \varphi^f]\!] \leq c$, *then* $[\![\pi, \varphi^{f^+_{x,\varphi,c}}]\!] \leq c$.
2. *If* $[\![\pi, \varphi^f]\!] \geq c$, *then* $[\![\pi, \varphi^{f^-_{x,\varphi,c}}]\!] \geq c$.

Note that by the monotonicity of LTL$^\nabla$ queries, increasing the value of a variable x that appears positively in φ can only increase the satisfaction value of φ (and dually for reducing the value of x or for the case of a variable that appears negatively). The claim in Lemma 4, however, is different and is much stronger, as it states that we can actually increase the value of a variable that appears positively without increasing the value of φ. More precisely, if the satisfaction value of φ^f in π is below c, then we can round the value of x up to the closest value in $val(x, \varphi, c)$ and still keep the satisfaction value below c.

We can now prove that the restriction of the search space to values in $val(x, \varphi, c)$ is allowed.

Lemma 5. *Let* φ *be a simple* LTL$^\nabla$ *query of depth* 1, *let* $c \in [0, 1]$, *and let* π *be a path. If there exists an assignment* f *such that* $[\![\pi, \varphi^f]\!] = c$, *then there also exists an assignment* g *such that for every* $x \in var(\varphi)$ *it holds that* $g(x) \in val(x, \varphi, c)$ *and* $[\![\pi, \varphi^g]\!] = c$.

Proof: Consider a simple LTL$^{\triangledown}$ query φ of depth 1 and an assignment f to $var(\varphi)$. Let $c \in [0,1]$, and let π be a computation. By the monotonicity of $\mu_{\pi,\varphi}$, Lemma 4 implies that if $[\![\pi, \varphi^f]\!] = c$, then $[\![\pi, \varphi^{f^+_{x,\varphi,c}}]\!] = c$.

Let f be such that $[\![\pi, \varphi^f]\!] = c$. The assignment g is obtained by repeating the following process for all variables $x \in var(\varphi)$ in an arbitrary order: if $f(x) \in val(x, \varphi, c)$, then $g(x) = f(x)$. Otherwise, we define $g(x)$ to be $f^+_{x,\varphi,c}(x)$. By the above, $[\![\pi, \varphi^g]\!] = c$. □

Consider a simple LTL$^{\triangledown}$ query φ over $\mathcal{X}$. By Lemma 5, the search for a solution f to a single constraint with a point interval c involves a search in finitely many possible assignments – these that map each variable $x \in var(\varphi)$ to values in $val(x, \varphi, c)$. By Lemma 3 (monotonicity of assignments), the search can combine a binary search for the assignment for each $x \in var(\varphi)$ with an ordering of the different variables. We will get back to this point when we describe our experimental results in Section 6.

The search described above assumes simple queries of depth 1 and constraints with point intervals. We now remove both assumptions. Let f^{max} and f^{min} be the assignments that maximizes and minimizes the value of φ. Thus, $f^{max}(x)$ is 1 if x appears positively in φ and is 0 otherwise, and dually for f^{min}. We define the LTL$^{\triangledown}$ formulas $\varphi^{max} = \varphi^{f^{max}}$ and $\varphi^{min} = \varphi^{f^{min}}$. Note that $\triangledown_1\psi$ and $\triangledown_0\psi$ subformulas can be replaced by ψ and `False`, respectively.

For a simple LTL$^{\triangledown}$ query φ (of an arbitrary depth), let $\varphi^\star$ be the LTL$^{\triangledown}$ formula obtained from φ by replacing every subformula of the form $\triangledown_x\psi$ by $\triangledown_x\psi^{max}$. Note that $\varphi^\star$ is of depth 1. By Lemma 3 (monotonicity of assignments), for every computation π, LTL$^{\triangledown}$ query ψ, and assignment f, we have $[\![\pi, \psi^f]\!] \leq [\![\pi, \psi^{max}]\!]$. Thus, replacing ψ by ψ^{max} may cause the satisfaction value of ψ to go above a desired bound. As we show below, however, in this case we can play with the assignment to x in order "tune down" the satisfaction value of ψ^{max}.

Lemma 6. *Let φ be a simple* LTL$^{\triangledown}$ *query, and let $\langle \pi, I\rangle$ be a constraint. The* LTL$^{\triangledown}$ *query $\langle \varphi, \langle \pi, I\rangle\rangle$ has a solution iff the query $\langle \varphi^\star, \langle \pi, I\rangle\rangle$ has a solution.*

Lemma 6 implies that we can use our algorithm also for formulas of depth greater than 1 by applying the algorithm to $\varphi^\star$. It is left to extend the algorithm to handle interval constraints. That is, given a simple LTL$^{\triangledown}$ query φ and a lasso constraint $\langle \pi, I\rangle$ such that $I \subseteq [0,1]$, our goal is to decide whether $\langle \varphi, \langle \pi, I\rangle\rangle$ has a solution. We do this as follows. First, compute $a = [\![\pi, \varphi^{min}]\!]$ and $b = [\![\pi, \varphi^{max}]\!]$. If $I \cap [a,b] = \emptyset$, then, as $[\![\pi, \psi^{min}]\!] \leq [\![\pi, \psi^f]\!] \leq [\![\pi, \psi^{max}]\!]$, we can conclude that there is no solution to $\langle \varphi, \langle \pi, I\rangle\rangle$. Otherwise, by the continuity of $\mu_{\pi,\varphi}$, every value in $c \in [a,b]$ can be attained by $\mu_{\pi,\varphi}$, so one can choose any value $c \in I \cap [a,b]$, and find a solution to $\langle \varphi, \langle \pi, c\rangle\rangle$.

Remark 1. Our definition of computations assumes Boolean atomic propositions. It is easy to extend our setting and results to computations over *weighted atomic propositions*. Let WP be a finite set of weighted atomic propositions over some domain D. The domain D may be infinite (say, the natural numbers) and different propositions may be over different domains. Each position in the computation is then a function in D^{WP}, and the semantics of LTL$^{\triangledown}$ is as in the Boolean case, except that $[\![\pi, p]\!]$, for $p \in WP$

is $\pi_0(p)$. The solution of the corresponding LTL$^\triangledown$ query is similar to the one described above, except that the definition of $val(\psi)$, and consequently also $val(x, \varphi, c)$, should be adjusted to reflect the fact the weighted propositions in ψ can take values in D. Since each lasso commutation has only finitely many positions, the number of the values to be considered is still linear in the input to the problem.

4.1 NP Completeness of the LTL$^\triangledown$ Query Problem for Simple Queries

Lemma 5 gives us an upper bound to complete Theorem 1 for simple LTL$^\triangledown$ queries of depth 1. For an LTL$^\triangledown$ query φ, a set of constraints $\mathcal{C} = \{\langle \pi_i, [a_i, b_i]\rangle\}_{i=1}^m$, and a variable x, define $val(x, \varphi, \mathcal{C}) = \bigcup_{i=1}^m val(x, \varphi, b_i)$. That is, $val(x, \varphi, \mathcal{C})$ includes the restricted search space of the upper end points of the intervals in the constraints in $\mathcal{C}$.

Lemma 7. *Let φ be a simple* LTL$^\triangledown$ *query of depth* 1*, and let $\mathcal{C}$ be a set of constraints. If there exists a solution f for $\langle \varphi, \mathcal{C}\rangle$, then there also exists a solution g such that for every $x \in var(\varphi)$, it holds that $g(x) \in val(x, \varphi, \mathcal{C})$.*

Since the binary expansion of the values in $val(x, \varphi, \mathcal{C})$ is polynomial in φ and $\mathcal{C}$ (with intervals given by their binary expansion), Lemma 7 implies that a witness solution to a simple LTL$^\triangledown$ query of depth 1 is polynomial. Since witnesses can be verified in linear time, we can conclude with the following.

Theorem 4. *The* LTL$^\triangledown$ *query problem for simple queries of depth* 1 *is NP-complete.*

5 Approximations and Heuristics

In this section we discuss two heuristic schemes for the LTL$^\triangledown$ query problem. The motivation is twofold. First, our heuristic algorithms run in polynomial time, whereas, as studied in Section 3, the problem is NP-hard. Second, in case a query does not have a solution, it is helpful to find a sub-optimal, or partial, solution. In order to study sub-optimal solutions, we should first formalize the LTL$^\triangledown$ query problem as an optimization problem. To do so, it is convenient to consider the problem in a geometrical perspective: for an LTL$^\triangledown$ query φ and a set of constraints $\mathcal{C}$, let $k = |var(\varphi)|$. Every constraint $\langle \pi, I\rangle$ induces a set $S_{\langle \pi, I\rangle, \varphi} \subseteq [0, 1]^k$ of solutions to $\langle \pi, I\rangle$. The LTL$^\triangledown$ query problem then amounts to finding a point $f \in \bigcap_{\langle \pi, I\rangle \in \mathcal{C}} S_{\langle \pi, I\rangle, \varphi}$. We note that the main difficulty in solving the LTL$^\triangledown$ query problem is that even for a single constraint, this set may not be convex, and therefore not amenable to methods of convex analysis. Indeed, consider for example the query $\psi = (\triangledown_x p) \vee (\triangledown_y q)$ with the constraint $\langle \{p, q\}^\omega, 1\rangle$. We have that $S_{\mathcal{C}, \psi} = ([0, 1] \times \{1\}) \cup (\{1\} \times [0, 1])$, which is not convex.

Using the geometrical perspective, we consider the following optimization problem: for an LTL$^\triangledown$ query φ and a set of constraints $\mathcal{C}$, find an assignment that minimizes the sum (over $\langle \pi, I\rangle \in \mathcal{C}$) of distances to $S_{\langle \pi, I\rangle, \varphi}$. We still have some freedom in choosing a distance function. Since we evaluate an assignment by the satisfaction value it induces on a computation, it is natural to use the obtained satisfaction value as an underlying metric for the distance function. Traditionally, such distance functions are known as *loss functions*, and we consider two common ones here. Let $x, y \in [0, 1]$.

- $0/1$-*loss*, defined by $\ell_{0/1}(x,y) = 0$ if $x = y$, and $\ell_{0/1}(x,y) = 1$ if $x \neq y$.
- $\|\cdot\|_2$-*loss*, defined by $\|x,y\|_2 = |x-y|$.

Consider a loss function ℓ and an assignment $f \in [0,1]^k$. Recall that for a closed interval $A \subseteq [0,1]$ and $x \in [0,1]$, we have $\ell(x,A) = \min\{\ell(x,y) : y \in A\}$. We define $dist_\ell(\varphi, \mathcal{C}, f) = \frac{1}{|\mathcal{C}|}\sum_{\langle\pi,I\rangle\in\mathcal{C}} \ell([\![\pi, \varphi^f]\!], I)$. That is, the average loss. Then, the $\mathrm{LTL}^\triangledown$ query optimization problem is to find $\arg\min_{f\in[0,1]^k}\{dist(\varphi,\mathcal{C},f)\}$, namely an assignment that minimizes the loss. Accordingly, for $\ell_{0/1}$, the problem is to find an assignment that maximizes the number of satisfied constraints, and in $\|\cdot\|_2$-loss we want to minimize the geometrical distance.

Note that for both loss functions ℓ, an assignment f is a solution iff $dist_\ell(\varphi,\mathcal{C},f) = 0$. Hence, the problem of finding the optimum is NP-hard. As Theorem 2 shows, the $\mathrm{LTL}^\triangledown$ query problem is NP-hard even when a single constraint is allowed. Accordingly, since a nontrivial approximation of the number of satisfied constraints must be greater than 0, it is NP-hard to even approximate the problem (to any ratio) under $\ell_{0/1}$.

Our use of $\mathrm{LTL}^\triangledown$ queries for generating quality specifications makes $\ell_{0/1}$ less appealing. Indeed, it takes us back to the Boolean setting. Still it involves some nice theoretical aspects. In particular, as we show in the full version, it suggests a naive $\frac{1}{2}$-approximation (that is, a guarantee that at most hard of the constraints are satisfied) for the case the constraints are all pure upper- or lower-bounds.

As we mentioned above, the set of solutions (even when it is not empty) may not be convex, which is the underlying reason for the hardness of the problem. We suggest a polynomial-time heuristic algorithm, based on our ability to solve the $\mathrm{LTL}^\triangledown$ query problem efficiently for a single constraint (Theorem 3). Given a simple $\mathrm{LTL}^\triangledown$ query φ and a set $\mathcal{C}$ of constraints, we find, for every constraint $\langle\pi, I\rangle \in \mathcal{C}$, an assignment $f_{\langle\pi,I\rangle}$ such that $[\![\pi, \varphi^{f_{\langle\pi,I\rangle}}]\!] \in I$. Let $F = \{f_{\langle\pi,I\rangle} : \langle\pi,I\rangle \in \mathcal{C}\}$. Intuitively, every point $f \in F$ "represents" the set $S_{\langle\pi,I\rangle,\varphi}$ for one of the constraints. Our algorithm combines the points in F in order to obtain a single assignment. If the representation is "good enough", we hope to get a good assignment. There are several ways to obtain a single assignment from F. Our algorithm uses the following three.

- *Minimum assignment*, denoted f_{min}: For every $x \in var(\varphi)$, if x is positive in φ, then $f_{min}(x) = \min_{f\in F} f(x)$; otherwise $f_{min}(x) = \max_{f\in F} f(x)$.
- *Maximum assignment*, denoted f_{max}: For every $x \in var(\varphi)$, if x is positive in φ, then $f_{max}(x) = \max_{f\in F} f(x)$; otherwise $f_{max}(x) = \min_{f\in F} f(x)$.
- *Center of gravity*, denoted f_{CoG}: For every $x \in var(\varphi)$, we define $f_{CoG}(x) = \frac{1}{|F|}\sum_{f\in F} f(x)$. It is easy to prove that the center of gravity is the point that minimizes the square-distance from $f_1, ..., f_m$. That is, $f_{CoG} = \arg\min\{\sum_{i=1}^m \|f - f_i\|_2^2 : f \in [0,1]^k\}$, where $\|x\|_2 = \sqrt{\sum_{i=1}^k x_i^2}$ for $x \in [0,1]^k$. This motivates using this as a heuristic, as it minimizes some sort of distance to the constraints.

Before we proceed to the experimental results, we show that in theory, all three methods may perform poorly.

Example 1. Recall the example $\psi = (\triangledown_x p) \vee (\triangledown_y q)$ with the set of constraints $\mathcal{C} = \{\langle\{p\}^\omega, 1\rangle, \langle\{q\}^\omega, 1\rangle\}$. Clearly, the assignment $(1,1)$ is a solution. Assume that our

polynomial time algorithm returns $F = \{(1,0),(0,1)\}$; that is, each point is a solution to a single constraint. The minimum assignment heuristic then gives $f_{min} = (0,0)$, for which $dist_{\ell_{0/1}}(\psi, \mathcal{C}, f_{min}) = dist_{\|\cdot\|_2}(\psi, \mathcal{C}, f_{min}) = 1$. The CoG heuristic cannot do quite as badly, being an average. However, in the example above we have $f_{CoG} = (\frac{1}{2}, \frac{1}{2})$ for which $dist_{\ell_{0/1}}(\psi, \mathcal{C}, f_{CoG}) = 1$ and $dist_{\|\cdot\|_2}(\psi, \mathcal{C}, f_{CoG}) = \frac{1}{2}$. It is easy to verify that taking the query $\neg\psi$ with the same constraints may result again in $F = \{(1,0),(0,1)\}$, whereas the optimum now is $(0,0)$, and $f_{max} = (1,1)$, again giving a bad lower bound.

6 Experimental Results

In this section we present experimental results demonstrating our heuristic algorithm for solving the LTL$^\nabla$ query problem. We evaluate the quality of the heuristic under the two loss functions (namely $\ell_{0/1}$ and $\|\cdot\|_2$). As our benchmark we use LTL formulas from [12], to which we add a quality layer, as well as queries constructed manually. As constraints, we use accepting paths in the automaton for the corresponding LTL formulas, as well as manually generated computations. An example of a query is a specification for a traffic light. The atomic propositions are $\{N, E, W, S\}$, standing for a green light for traffic coming from North, East, West, and South, respectively. We assume traffic crosses the junction and makes no turns, and so the specification allows N and S, as well as W and E, to hold simultaneously, but not N and W, nor S and E, and so on. Thus, the specification includes the property $\psi = (G(N \vee S) \rightarrow (\neg E \wedge \neg W)) \wedge (G(E \vee W) \rightarrow (\neg N \wedge \neg S))$. We want the traffic light to direct the traffic efficiently, so we require that at least one direction has a green light. This is specified by the conjunct $G\theta$, for $\theta = (S \vee N \vee W \vee E)$. We may also be satisfied by $FG\theta$. But while ψ is a crucial safety requirement, the conjuncts involving θ only concern the efficiency of the traffic light. Thus, we can tune them down using the LTL$^\nabla$ formula $\psi \wedge \blacktriangledown_{0.4} G\theta \wedge \blacktriangledown_{0.9} FG\theta$ (as discussed in Section 2.2, we use the abbreviation $\blacktriangledown_\lambda \varphi = \neg\nabla_{(1-\lambda)}\neg\varphi$ to indicating the *loss* when the corresponding conjuncts do not hold). Note that we prefer a junction that is never empty over a junction that is only eventually never empty. But this is not the end of the story. We may want to prioritize the different directions. Deciding the priorities and their combination with the external tuning down of the "efficiency requirement" may be a difficult task. So, we replace θ by $\theta' = (\blacktriangledown_{x_1} S) \vee (\blacktriangledown_{x_2} N) \vee (\blacktriangledown_{x_3} W) \vee (\blacktriangledown_{x_3} E)$, leaving the priorities as variables. The obtained LTL$^\nabla$ query is φ_5 in the table. Examples of constraints we use are $\langle(\{N,S\},\{E,W\})^\omega, 1\rangle$ and $\langle(\{N,S\},\emptyset,\{E,W\})^\omega, 0.4\rangle$.

We implemented the algorithm in Python and ran it on an Intel® Core i5 2.53GHz machine. The code can be found in: http://www.cs.huji.ac.il/∼guya03/CAV13/.

In Table 2 we evaluate the quality of results in the $\ell_{0/1}$ loss function. The results show that the algorithm preforms very well compared to the optimum. We ran the heuristic algorithm 40 times, each time with a different random ordering of the variables (recall that the algorithm for the case of a single constraint chooses variables according to an arbitrary order, which affects the resulting assignment). In each run we calculate, f_{min} and f_{max} in each iteration. We evaluate the assignments by checking how many constraints they satisfy, and output the best assignment. The running time of the algorithm that is shown in the table is the total running time, which is still negligible compared

to the optimal algorithm's running time.[5] In order to find the optimal assignment we go over all the assignments that use values in the restricted search space, hence the very high running times.

Table 2. Evaluating the heuristic under the $\ell_{0/1}$ loss

Query	$\|\mathcal{C}\|$	$\|\mathcal{X}\|$	Algorithm		Optimum	
			# of constraints satisfied	running time	# of constraints satisfied	Running time
φ_1	8	4	5	10sec	6	35sec
φ_2	6	4	3	27sec	3	151sec
φ_3	8	4	1	37sec	2	202sec
φ_4	4	5	2	11sec	2	260sec
φ_5	11	8	2	21sec	6	620sec

We continue to evaluate the heuristic under the $\|\cdot\|_2$ loss function, as shown in Table 3. In this approach we have no optimum to compare with (as we do not even know that the problem is in NP). Instead, we perform *sanity checks* and compare our results to them. As in the previous table, we run the algorithm 40 times and calculate f_{min}, f_{max}, and f_{CoG}. We evaluate the assignments by calculating $dist_{\|\cdot\|_2}(\varphi, \mathcal{C}, f_{min})$, $dist_{\|\cdot\|_2}(\varphi, \mathcal{C}, f_{max})$, and $dist_{\|\cdot\|_2}(\varphi, \mathcal{C}, f_{CoG})$, which we present in the table. Our first sanity check is the *Cross validation* technique, which is a widely used technique in machine learning. We partition the constraints $\mathcal{C}$ into two sets: $\mathcal{C}_1$ and $\mathcal{C}_2$. We find an assignment f_1 using the constraints $\mathcal{C}_1$. Then, we evaluate the assignments on the constraints $\mathcal{C}_2$. That is, we calculate $dist_{\|\cdot\|_2}(\varphi, \mathcal{C}_2, f_1)$. In machine learning, the goal of this technique is, given a training set, to assess the quality of a hypothesis. The difference between the learning scenario and our case is that there, the training set is chosen uniformly from a certain distribution, whereas our constraints are manually chosen by the designer. Thus, the accuracy of this assessment is strongly affected by the dependencies between the constraints. This makes cross-validation unreliable at times.

Table 3. Evaluating the heuristic in the distance-minimization approach

Query	$\|\mathcal{C}\|$	$\|\mathcal{X}\|$	Distance			Sanity checks			
			Min	Max	CoG	Min	Max	CoG	random
φ_1	8	4	0.031	0.031	0.129	0.05	0.2	0.218	0.230
φ_2	6	4	0.333	0.166	0.219	0.466	0.466	0.466	0.322
φ_3	8	4	0.229	0.104	0.162	0.191	0.275	0.4	0.200
φ_4	4	5	0.05	0.05	0.075	0.175	0.075	0.225	0.096
φ_5	11	8	0.1	0.173	0.1	0.34	0.06	0.127	0.348

In our second sanity check, we calculate $dist_{\|\cdot\|_2}(\varphi, \mathcal{C}, f)$ for a random assignment f. We repeat this test 10 times and take the average distance, thus approximating the expectancy of the distance.

[5] In practice, the lassos in the constraints are typically short, as the designer grades them manually according to specific behaviors he has in mind. We still challenged our implementation with both short and long lassos, and running time was not an issue – what we care here more is the quality of the assignment returned.

As seen in the table, there is no clear winner between the minimal, maximal, and center of gravity mechanism. As the running time of the algorithm is very short, there is no reason not to find all three points and choose the best one for the specific instance of the problem. As described above, the first sanity check returns mixed results. However, our heuristic significantly out-performs the random assignment.

References

1. Almagor, S., Boker, U., Kupferman, O.: Discounting in LTL. TR (2013), http://leibniz.cs.huji.ac.il/tr/1300.pdf
2. Almagor, S., Boker, U., Kupferman, O.: Formalizing and reasoning about quality. ICALP 2013 (2013)
3. Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: Proc. 25th STOC, pp. 592–601 (1993)
4. Ammons, G., Bodík, R., Larus, J.R.: Mining specifications. In: POPL, pp. 4–16 (2002)
5. Ammons, G., Mandelin, D., Bodk, R., Larus, J.R.: Debugging temporal specifications with concept analysis. In: Proc. PLDI, pp. 182–195. Springer (2003)
6. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. 75(2), 87–106 (1987)
7. Avni, G., Kupferman, O.: Parameterized Weighted Containment. In: Pfenning, F. (ed.) FOSSACS 2013. LNCS, vol. 7794, pp. 369–384. Springer, Heidelberg (2013)
8. Bloem, R., Cavada, R., Pill, I., Roveri, M., Tchaltsev, A.: RAT: A tool for the formal analysis of requirements. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 263–267. Springer, Heidelberg (2007)
9. Boker, U., Chatterjee, K., Henzinger, T.A., Kupferman, O.: Temporal specifications with accumulative values. In: Proc. 26th LICS, pp. 43–52 (2011)
10. Chan, W.: Temporal-logic queries. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 450–463. Springer, Heidelberg (2000)
11. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE, pp. 411–420 (1999)
12. Jobstmann, B., Bloem, R.: Optimizations for LTL synthesis. In: Proc. FMCAD, pp. 117–124 (2006)
13. Jobstmann, B., Galler, S., Weiglhofer, M., Bloem, R.: Anzu: A tool for property synthesis. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 258–262. Springer, Heidelberg (2007)
14. Jobstmann, B., Griesmayer, A., Bloem, R.: Program repair as a game. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 226–238. Springer, Heidelberg (2005)
15. Kupferman, O.: Sanity checks in formal verification. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006. LNCS, vol. 4137, pp. 37–51. Springer, Heidelberg (2006)
16. Kwiatkowska, M.Z.: Quantitative verification: models techniques and tools. In: ESEC/SIGSOFT FSE, pp. 449–458 (2007)
17. Madhusudan, P.: Learning algorithms and formal verification (Invited tutorial). In: Cook, B., Podelski, A. (eds.) VMCAI 2007. LNCS, vol. 4349, pp. 214–214. Springer, Heidelberg (2007)
18. Moon, S., Lee, K., Lee, D.: Fuzzy branching temporal logic. IEEE Transactions on Systems, Man, and Cybernetics, Part B 34(2), 1045–1055 (2004)
19. Pnueli, A.: The temporal semantics of concurrent programs. TCS 13, 45–60 (1981)
20. PROSYD. The Prosyd project on property-based system design, http://www.prosyd.org
21. Roveri, M.: Novel techniques for property assurance. TR PROSYD FP6-IST-507219 (2007)
22. Solar-Lezama, A., Rabbah, R.M., Bodík, R., Ebcioglu, K.: Programming by sketching for bit-streaming programs. In: Proc. PLDI, pp. 281–294 (2005)