The Hardness of Finding Linear Ranking Functions for Lasso Programs

Amir M. Ben-Amram
The Academic College of Tel-Aviv Yaffo
amirben@cs.mta.ac.il

Finding whether a linear-constraint loop has a linear ranking function is an important key to understanding the loop behavior, proving its termination and establishing iteration bounds. If no preconditions are provided, the decision problem is known to be in coNP when variables range over the integers and in PTIME for the rational numbers, or real numbers. Here we show that deciding whether a linear-constraint loop with a precondition, specifically with partially-specified input, has a linear ranking function is EXPSPACE-hard over the integers, and PSPACE-hard over the rationals. The precise complexity of these decision problems is yet unknown. The EXPSPACE lower bound is derived from the reachability problem for Petri nets (equivalently, Vector Addition Systems), and possibly indicates an even stronger lower bound (subject to open problems in VAS theory). The lower bound for the rationals follows from a novel simulation of Boolean programs. Lower bounds are also given for the problem of deciding if a linear ranking-function supported by a particular form of inductive invariant exists. For loops over integers, the problem is PSPACE-hard for convex polyhedral invariants and EXPSPACE-hard for downward-closed sets of natural numbers as invariants.

1 Introduction

The results in this paper relate two basic problems in the analysis of loops: reachability and the existence of a linear ranking function that proves termination of the loop. We only consider the (often used) model in which loops compute over numeric variables (most frequently integer) and their effect is expressed by linear equations or inequalities (constraints).

Termination provers, of which TERMINATOR by Cook, Podelski and Rybalchenko [11] is a prototypical example, are based on the subproblem of proving termination for simple loops with a "stem", the so-called *lasso* (Figure 1). Termination of such loops is established in TERMINATOR by abstracting the loop to linear constraint form and finding a linear ranking function (a function of the state variables which is bounded below and decreases in every iteration). But the algorithm used in TERMINATOR to check for the existence of such a function [27] does not take the effect of the "stem," which is a precondition for the simple loop, into account. We may describe the problem solved by such an algorithm as finding a *universal* ranking function—one that works for any initial state.

There are several works that do take preconditions into account in the algorithm that looks for ranking functions. Early approaches [9, 29] used precomputed invariants, and once these invariants were included in the description of the loop, looked for a universal ranking function. Later, some works attempted to integrate the discovery of *supporting invariants* with the search for a ranking function, e.g., [6, 19]. Other works heuristically find some precondition under which a ranking function can be established, e.g., [10].

I am aware of no published upper or lower bounds on the complexity of precisely answering the question: given a linear-constraint loop with a precondition, does it have a linear ranking function? This contrasts with the well-understood classification of the universal linear ranking-function problem: as a

Adriano Peron and Carla Piazza (Eds.):
Proceedings of the Fifth International Symposium on
Games, Automata, Logics and Formal Verification (GandALF 2014)
EPTCS 161, 2014, pp. 32–45, doi:10.4204/EPTCS.161.6

```
Y := 2;
while X > 0 do
    Y := Y − 1;
    X := X + Y;
    Y := 2 ∗ Y;
```

(a)

$$Y = 2;$$
$$\textit{while } X > 0 \textit{ do}$$
$$X' = X + Y - 1,\ \ Y' = 2Y - 2$$

(b)

Figure 1: A loop with a stem (a): the stem is the straight-line code preceding the while loop. The loop has the ranking function X, but this is only justified when the stem is taken into account. In (b), the loop is written in the formalism of linear constraints.

decision problem (for simplicity we only consider decision problems when referring to a complexity class) it is PTIME over the rationals[1] [2, 27] and coNP-complete over the integers [3].

In this paper, we show that deciding whether a linear-constraint loop with a precondition of a simple form has a linear ranking function is EXPSPACE-hard over the integers, and PSPACE-hard over the rationals. Clearly, these problems are much harder than the universal linear ranking-function problem. In fact, we do not even know if they are decidable!

A possible reaction to the hardness of this problem is to look at a mitigated problem that has been attempted by work already mentioned: the *invariant-supported ranking function*. Instead of asking for a ranking function that holds for the precise set of reachable states, we relax the requirement so that the ranking function has to hold in a set that contains the reachable states, a loop invariant. Moreover, we consider *inductive invariants*: such an invariant is verified by a local condition, that is, a condition on a single loop step, and this condition becomes clearly decidable if the invariant comes from a suitable "effective" class. We shall consider two classes of invariants which seem natural: (1) convex polyhedra, that is, conjunctions of linear constraints, and (2) disjunctive invariants of a very simple form (downward-closed sets—basically a union of boxes with one corner at the origin). For precise definitions see the Section 2. Do they make the problem more tractable? We do not know exactly. But we can show that—over the integers, at least—the problem is certainly not *easy*. We prove PSPACE-hardness for ranking functions supported by inductive invariants which are convex polyhedra, and EXPSPACE-hardness for downward-closed sets.

Thus the results of this paper are four hardness results: two for the general problem with a precondition and two for the invariant-supported problem. In addition, for the integer case (without invariants) we strengthen the hardness result to the claim that the problem is at least as hard as the reachability problem for Vector Addition Systems, a problem for which even a primitive-recursive upper bound is not known (see Section 4 for details). These are the first complexity results for these problems, and hopefully, another contribution is to motivate further research towards their theoretical understanding. At the conclusion of this paper, further discussion of the significance of the results and the open problems will be given.

[1] We say that we solve the problem over the rationals when we consider the state space to consist of all rational-valued points that satisfy the loop constraints, and "over the integers" when only integer points are considered. See Section 2.

2 Preliminaries

In this section we give basic definitions, regarding linear-constraint loops, linear ranking functions, vector addition systems and inductive invariants.

2.1 Loop representation

We define the loop representation based on linear constraints, which is quite standard.

A *single-path* linear-constraint loop with preconditions (SLC^{ρ} for short) over n variables $x_1,\ldots,x_n$ has the form

$$C\mathbf{x} \leq \mathbf{c}; \quad \textit{while } (B\mathbf{x} \leq \mathbf{b}) \textit{ do } A\begin{pmatrix}\mathbf{x}\\ \mathbf{x}'\end{pmatrix} \leq \mathbf{a}$$

where $\mathbf{x} = (x_1,\ldots,x_n)^{\mathrm{T}}$ and $\mathbf{x}' = (x'_1,\ldots,x'_n)^{\mathrm{T}}$ are column vectors, and for some $p,q,r > 0$, $C \in \mathbb{Z}^{r\times n}$, $B \in \mathbb{Z}^{p\times n}$, $A \in \mathbb{Z}^{q\times 2n}$, $\mathbf{c} \in \mathbb{Z}^r$, $\mathbf{b} \in \mathbb{Z}^p$, $\mathbf{a} \in \mathbb{Z}^q$. The constraint $C\mathbf{x} \leq \mathbf{c}$ is called *the precondition*, and specifies the initial states for the computation of the loop. The set of initial states is denoted by $\mathcal{I}$. The constraint $B\mathbf{x} \leq \mathbf{b}$ is called *the loop condition* (a.k.a. the loop guard) and the last constraint is called *the update*. The update is called *deterministic* if, for a given $\mathbf{x}$ (satisfying the loop condition) there is at most one $\mathbf{x}'$ satisfying the update constraint.

We say that there is a transition from a state $\mathbf{x} \in \mathbb{Q}^n$ to a state $\mathbf{x}' \in \mathbb{Q}^n$, if $\mathbf{x}$ satisfies the condition and $\mathbf{x}$ and $\mathbf{x}'$ satisfy the update. A transition can be seen as a point $\binom{\mathbf{x}}{\mathbf{x}'} \in \mathbb{Q}^{2n}$, where its first n components correspond to $\mathbf{x}$ and its last n components to $\mathbf{x}'$. For convenience, we denote $\binom{\mathbf{x}}{\mathbf{x}'}$ by $\mathbf{x}''$.

The notions of *computation of a loop* and termination are straight-forward. A computation must start at an initial state. Note that when the loop is non-deterministic, termination means that there exists no infinite computation from an initial state. A *reachable state (transition)* is a state (respectively transition) that appears in some computation.

We say that the loop is interpreted *over the rationals* if $\mathbf{x}$ and $\mathbf{x}'$ range over $\mathbb{Q}^n$, and *over the integers* if they range over $\mathbb{Z}^n$. We also say that the loop is *a rational (respectively, integer)* loop. For uniformity of notation, we use $\mathcal{S}$ to denote the state space, without specifying its precise nature.

For purposes of complexity classification, we define the representation of the input to consist of the matrices and vectors that specify the loop, with numbers in binary notation. We often consider a restricted problem, concerning a *loop with partially-specified input*: that means that the precondition is of the form $\bigwedge_{i=1}^{k} x_i = d_i$ for some variables x_i and values d_i. Thus the value of each variable is either specified precisely or left free.

2.2 Ranking functions

We now define linear ranking functions and the decision problem LINRF$^{\rho}$, asking for the existence of a Linear Ranking Function for reachable states (the ρ reminds us of the lasso shape, and is also an initial of "reachability").

An affine linear function $\rho : \mathbb{Q}^n \to \mathbb{Q}$ is of the form $\rho(\mathbf{x}) = \vec{\lambda}\cdot\mathbf{x} + \lambda_0$ where $\vec{\lambda} \in \mathbb{Q}^n$ is a row vector and $\lambda_0 \in \mathbb{Q}$.

DEFINITION 2.1. Given a set $T \subseteq \mathbb{Q}^{2n}$, representing transitions, we say that ρ is a *linear ranking function* (*LRF*) for T if the following hold for every $\binom{\mathbf{x}}{\mathbf{x}'} \in T$:

$$\rho(\mathbf{x}) \geq 0, \tag{1}$$

$$\rho(\mathbf{x}) - \rho(\mathbf{x}') \geq 1. \tag{2}$$

We say that ρ is a *LRF* for a loop (with precondition) if its is a *LRF* for the set of *reachable transitions* of this loop.

DEFINITION 2.2. The decision problem *Existence of a LRF* (with precondition) is defined by

Instance: an SLC^ρ loop.

Question: does there exist a *LRF* for this loop?

The decision problem is denoted by $\text{LINRF}^\rho(\mathbb{Q})$ and $\text{LINRF}^\rho(\mathbb{Z})$ for rational and integer loops respectively.

2.3 Invariants

Consider a loop with initial states $\mathcal{I}$ and transition set $\mathcal{Q}$. We define an *invariant* of the loop to be a set $INV \subseteq \mathcal{S}$ such that all reachable states are in *INV*. We define an *inductive invariant* (sometimes this is just called an invariant) to be a set $INV \subseteq \mathcal{S}$ satisfying the properties of

- Initiation: $\mathcal{I} \subseteq INV$;
- Consecution: if $\binom{\mathbf{x}}{\mathbf{x}'} \in \mathcal{Q}$ then $\mathbf{x} \in INV \Rightarrow \mathbf{x}' \in INV$.

Clearly, an inductive invariant does contain all reachable states. However, frequently, concentrating on inductive invariants makes the verification of an invariant possible—even if the precise set of reachable states could be uncomputable. This depends on the kind of invariants one considers. For example, an often-used type of invariant is *convex polyhedra* [13]. Using a customary representation, e.g., by constraints, the invariant properties are decidable by linear or integer programming (for linear-constraint loops). Another natural class—for loops over the natural numbers—are *downward-closed sets*: sets *INV* such that $\mathbf{x} \leq \mathbf{y}$, $\mathbf{y} \in INV \Rightarrow \mathbf{x} \in INV$. Due to Dickson's lemma, such sets are finitely representable as the downward-closure of a finite set in the lattice $\mathbb{N}_\omega^n$ (adding the element ω allows for unbounded sets in $\mathbb{N}$ to be represented). This makes them useful for analysing certain kinds of programs, notably vector addition systems [17, 21]. We note that, they constitute an elementary kind of disjunctive invariants—each disjunct is of the form $0 \leq \mathbf{x} \leq \mathbf{c}$ where $\mathbf{c} \in \mathbb{N}_\omega^n$.

Since with both of the above classes, verification of an invariant is effective, we call them *effectively inductive invariants*. Our main interest lies in using the invariants to support ranking functions: this means that we look for a ranking function not for the set of reachable transitions, but for the set $\{\binom{\mathbf{x}}{\mathbf{x}'} \mid \mathbf{x} \in INV\}$, which may be larger, but computable.

3 Rational Loops with Preconditions

Most of this section is dedicated to proving the next thoerem, from which we later derive the result on $\text{LINRF}^\rho(\mathbb{Q})$.

THEOREM 3.1. *The following problem is PSPACE-hard: given a (deterministic) rational linear-constraint loop and an initial state, does a specified variable ever get a positive value?*

We prove this by reduction from the halting problem for Boolean programs, namely programs that manipulate a finite number of $\{0,1\}$-valued variables, $X_1, \ldots, X_n$. The program is a list of labeled instructions

$$1{:}I_1, \ldots, m{:}I_m, m{+}1{:}\Box$$

where each instruction I_k is one of the following:

$$incr(X_j) \mid decr(X_j) \mid \textit{if } X_j \textit{ then } k_1 \textit{ else } k_2$$

with $1 \leq k_1, k_2 \leq m+1$ and $1 \leq j \leq n$. A state is of the form $(k, \langle a_1, \ldots, a_n \rangle)$ which indicates that Instruction I_k is to be executed next, and the current values of the variables are $X_1 = a_1, \ldots, X_n = a_n$. In a valid state, $1 \leq k \leq m+1$ and all $a_i \in \{0,1\}$. Any state in which $k = m+1$ is a halting state. For any other valid state $(k, \langle a_1, \ldots, a_n \rangle)$, the successor state is defined as follows.

- If I_k is $incr(X_j)$, then X_j is changed from 0 to 1; if it is already 1, the program aborts. Similarly, $decr(X_j)$ changes a 1 to a 0. In both cases, if execution does not abort, it proceeds at instruction $k+1$.
- If I_k is "*if* X_j *then* k_1 *else* k_2", then the execution moves to instruction k_1 if X_j's value is 1, and to k_2 if it is 0. The values of the variables do not change.

The halting problem is whether the program reaches the halting label $m+1$ when started at the *initial state* $(1, \langle 0, \ldots, 0 \rangle)$ (note that aborting due to an invalid increment or decrement should give a negative answer).

The class of deterministic Boolean programs captures PSPACE computability, and the halting problem for such programs is, therefore, PSPACE-complete (see, e.g., [20], which uses this model up to non-essential differences).

Given a Boolean program P_B, we generate a corresponding SLC^{ρ} loop $\mathrm{T}(P_B)$ by translating the following program, written in pseudo-code with assignments, into linear constraints.

```
while ( 0 ≤ A_1 ≤ 1 ∧ ··· ∧ 0 ≤ A_m ≤ 1 ∧ 0 ≤ X_1 ≤ 1 ∧ ··· ∧ 0 ≤ X_n ≤ 1 ) do {
    N_1 := 0; N_2 := A_1; ... N_m := A_{m-1}; N_{m+1} := A_m + A_{m+1};
    T(1:I_1)
        ⋮
    T(m:I_m)
    A_1 := N_1; ... A_{m+1} := N_{m+1}
}
```

Basically, A_i represents the choice of instruction (the "program counter"), and N_i is a temporary variable used for finding the next instruction (it is modified by jumps, as shown below). $\mathrm{T}(k{:}I_k)$ is a translation of the kth instruction, defined as follows (again, with a mix of assignments and assertions, for readability)

- If $I_k \equiv incr(X_j)$, then $\mathrm{T}(k{:}I_k)$ is $X_j := X_j + A_k$;
- If $I_k \equiv decr(X_j)$, then $\mathrm{T}(k{:}I_k)$ is $X_j := X_j - A_k$;
- If $I_k \equiv \textit{if } X_j > 0 \textit{ then } k_1 \textit{ else } k_2$, then $\mathrm{T}(k{:}I_k)$ involves two dedicated variables, T_k and F_k, as follows:
  $0 \leq T_k \leq A_k$;
  $T_k \leq X_j$;
  $0 \leq F_k \leq A_k$;
  $F_k \leq 1 - X_j$;
  $T_k + F_k \geq A_k$;
  $N_{k+1} := N_{k+1} - A_k$;
  $N_{k_1} := N_{k_1} + T_k$;
  $N_{k_2} := N_{k_2} + F_k$

In the last part, the variables T_k (respectively F_k) represent the choice of the "true" branch (resp. "false") of a conditional branch instruction at label k.

Our precondition defines an initial state that corresponds to the initial state of P_B. More precisely, in the initial state, all variables are set to 0, except $A_1 = 1$. All auxiliary variables (N_k, T_k, F_k) are set to the appropriate values according to their constraints, or to 0 if unconstrained. The essential arguments to complete the justification of the reduction are given by the following lemma.

LEMMA 3.2. *In every (rational-valued) state reachable from the initial state, it holds that*

1. *all variables have values in* $\{0,1\}$.
2. $T_k = 1$ *if and only if* $A_k = 1$, *instruction k is a branch on* X_j *and* $X_j = 1$.
3. $F_k = 1$ *if and only if* $A_k = 1$, *instruction k is a branch on* X_j *and* $X_j = 0$.
4. *At most one variable* $A_k = 1$.
5. *A state where all of* $A_1, \ldots, A_m$ *are 0 is only reached when a transfer to label* $m+1$ *has been simulated. Only in such a state is* $N_{m+1} = 1$.
6. *When a state where* $N_{m+1} = 1$ *is reached, the program idles in this state.*

Proof. The proof requires induction on the number of transitions from the initial state. The initial state was chosen to satisfy these properties. For the induction step, we first prove (2) and (3), which follow quite easily (as the reader may check) from the assumption that A_k and X_j are either 0 or 1 (which we have by the induction hypothesis). Given these facts, one can check that for any state in which (1) and (4) hold, the variables X_i remain in $\{0,1\}$ (in fact, at most one of them is modified), proving that (1) holds in the next step. For the variables N_i, since initially they are zero, it is easy to see that it always holds that just one of them will be a 1 (using (4) and (5)), which implies (4) and (5) for the next state. Finally, (6) is easy to verify. □

Essentially, the lemma shows that the constraint loop simulates P_B in lockstep (i.e., every transition of P_B is simulated by a transition of the loop), except that normal halting becomes an infinite loop in a state where $N_{m+1} = 1$. Theorem 3.1 follows immediately. By modifying the constructed program slightly, we obtain

COROLLARY 3.3. *The* $\text{LINRF}^{\rho}(\mathbb{Q})$ *problem is PSPACE-hard, even when restricted to deterministic loops.*

Proof. We add another variable Y, initially unbounded, and the constraints:

$$Y > 0,\ Y' = Y - 1 + N_{m+1}\,.$$

It is easy to see that if the original Boolean program does *not* halt, our SLC^{ρ} loop will halt from the specified initial state, and Y is a ranking function. If the Boolean program *does* halt, our SLC^{ρ} loop does not, and therefore, has no ranking function (of any kind). □

The fact that our loop either has the specified ranking function, or does not halt at all, is significant: it means that the existence of any "termination witness" (like *LRF*) which can handle this loop (in particular, any witness which encompasses single-variable *LRF*s) will also be PSPACE-hard. On the other hand, we can distinguish our problem from termination in the following sense.

COROLLARY 3.4. *The* $\text{LINRF}^{\rho}(\mathbb{Q})$ *problem is PSPACE-hard even if restricted to deterministic loops that do terminate.*

Proof. We add another variable R, initially unbounded, and the constraints:

$$R' = R - 1,\ Y' \leq Y + R.$$

Now, the loop will always halt, since R must eventually be negative and force Y to decrease. But, when the Boolean program halts, the loop can go through several iterations in which Y does not decrease (as long as R is still positive); therefore, Y is still not a ranking function. Neither can we form a ranking function using other variables. Specifically, we cannot use R, because it has no lower bound; and the rest of the variables do not change in such iterations. □

4 Integer Loops with Preconditions

The constructions in this section are inspired by the simulation of Petri nets by SLC^{ρ} loops, used for Theorem 6.1 in [4], which states that the termination problem for such loops is EXPSPACE-hard. The hardness result is based on Lipton's reduction from halting of counter programs with exponential space;[2] this was originally used by Lipton [24] to prove hardness of some decision problems in Petri nets. First, we give the necessary definitions.

4.1 VAS and Petri nets

A vector addition system is a type of program which maintains n counter variables (variables of non-negative integer value), so that a state $\mathbf{x}$ is a vector of non-negative integers. A state-transition is of the form $\mathbf{x}' = \mathbf{x} + \mathbf{v}_i$, where $\mathbf{v}_i$ is chosen non-deterministically among given *displacement vectors* $\mathbf{v}_1, \ldots, \mathbf{v}_k$, and subject to the constraint that all variables remain non-negative. For purposes of complexity classification, we define the representation of a VAS as input to be the list of vectors, with numbers in binary notation. We denote the jth element of $\mathbf{v}_i$ by $\mathbf{v}_i[j]$.

A Petri net is a very similar model, and for convenience we present it here using the terminology of VAS. Then, the difference lies in the definition of transitions: a possible transition is specified by *two* vectors, $\mathbf{v}_i^-$ and $\mathbf{v}_i^+$, both non-negative, and its effect is described by $\mathbf{x}' = \mathbf{x} - \mathbf{v}_i^- + \mathbf{v}_i^+$, provided $\mathbf{x} - \mathbf{v}_i^-$ is non-negative. One may think of $\mathbf{v}_i^-$ as a requirement for the enabling of transition i.

4.2 Lipton's reduction

Let us first recall Lipton's reduction (a good reference is [15]). Given an exponential-space counter program P, the reduction constructs a Petri net N_P that has the following behavior when started at an appropriate initial state. N_P has two kinds of computations, *successful* and *failing*. Failing computations are caused by taking non-deterministic branches which are not the correct choice for simulating P. Failing computations always halt. The (single) successful computation simulates P faithfully. If (and only if) P halts, the successful computation reaches a state in which a particular flag, say *HALT*, is raised (that is, *HALT* is a counter which is incremented for the first time from 0 to 1). This flag is never raised in failing computations. Thus, the reduction proves hardness of a problem which we may call *eventual positivity*:

THEOREM 4.1. *It is EXPSPACE-hard to decide, for a Petri net with a given initial state $\mathbf{x}_0$, whether there is a reachable state in which $x_n > 0$.*

[2] The space complexity measure for counter programs is the number of bits necessary to maintain the counters in binary notation.

Note that this problem is a special case of *coverability* (given $\mathbf{x}_0$ and another vector $\mathbf{y}$, is there a reachable state $\mathbf{x}$ such that $\mathbf{x} \geq \mathbf{y}$?). It is easy to adapt the reduction to also show hardness of state reachability (is $\mathbf{y}$ reachable from $\mathbf{x}_0$?).

4.3 Application to the LRF problem

By translating Petri nets to SLC^{ρ} loops, we obtain a question on eventual positivity in such loops with a partially-specified input. It is easy enough to transform this question to a question on the existence of a *LRF*. Thus we obtain

THEOREM 4.2. $\text{LINRF}^{\rho}(\mathbb{Z})$ *is EXPSPACE-hard for partially-specified input.*

Proof. Let a Petri net be given, having n counter variables and m displacement vectors, along with an initial state $\mathbf{x}_0$. We construct a SLC^{ρ} loop having variables $X_1,\ldots,X_n$, that represent the counters, and flags $A_1,\ldots,A_m$, that represent the choice of the next transition and change non-deterministically. The loop guard is $X_1 \geq 0 \wedge \cdots \wedge X_n \geq 0$. The initial state for our loop is the given initial state (for the X_i) and zeros for the A_i. The transition relation of the loop implements the Petri-net transitions in a straightforward way, specifically, it is the conjunction of the following three conjunctions

$$\begin{aligned}
\Delta &\equiv \bigwedge_{k=1}^{m} (A'_k \geq 0) \wedge (A'_1 + ... + A'_m = 1) \\
\Psi &\equiv \bigwedge_{i=1}^{n} (X_i \geq \sum_{k=1}^{m} \mathbf{v}_k[i] \cdot A'_k) \\
\Phi &\equiv \bigwedge_{i=1}^{n} (X'_i = X_i - \sum_{k=1}^{m} \mathbf{v}_k^-[i] \cdot A'_k + \sum_{k=1}^{m} \mathbf{v}_k^+[i] \cdot A'_k)
\end{aligned}$$

where Δ ensures that one and only one A'_k will be a 1, Ψ ensures that the transition chosen is enabled, and Φ implements the effect of the transition.

To reduce to the $\text{LINRF}^{\rho}(\mathbb{Z})$ problem, we add another variable Y, and the constraints:

$$Y > 0,\ Y' \leq Y - 1 + X_n.$$

In addition, we add a new transition to our VAS (and encode it in our loop); the new transition is enabled when X_n is positive, and does not modify the state (so it loops forever).

It is easy to see now that if the original counter program does *not* halt, our SLC^{ρ} loop will halt from the specified initial state, because variable Y will hit its lower bound; in fact, Y is a ranking function. If the original counter program *does* halt, our SLC^{ρ} loop does not, and therefore, has no ranking function (of any kind).

We conclude that determining if the constructed loop has a linear ranking function is as hard as deciding whether the counter machine that the Petri net simulates halts, that is, EXPSPACE-hard. □

Note that a reduction from counter programs with unbounded counter values would have proved undecidability. Unfortunately, such a reduction has not yet been found. The reduction from VAS suceedes, essentially, because it is a kind of counter program in which a transition cannot be conditioned on a zero-test.

4.4 A reduction from Reachability

In the Reachability problem for Petri nets/VAS, we are given an initial state **s** and are asked whether a given target state **t** is reachable from **s**. We can also reduce to $\text{LINRF}^{\rho}(\mathbb{Z})$ from the reachability problem. This observation may be of interest since the latter problem is generally presumed to be harder than coverability, which provided our EXPSPACE lower bound [14, 16, 23] (at least, it is certain that reachability too is EXPSPACE-hard, so the same lower bound follows. Hence, in terms of the resulting lower bound, the next theorem supersedes the previous one. However, the previous reduction is not entirely redundant as it is useful for a proof to be given later in Section 5).

Intuitively, the reduction operates as follows: a program simulates the VAS and tries to check if the target vector **t** is reached. When this happens, it results in an infinite execution.

THEOREM 4.3. *There is polynomial-time reduction of the VAS reachability problem to* $\text{LINRF}^{\rho}(\mathbb{Z})$*, with a partially-specified initial state.*

Proof. Let a VAS (of dimension n, and with m displacement vectors), and the vectors **s**, **t** be given. We assume (with no loss of generality) that the VAS is designed so that a computation from **s** will never reach the zero vector.

We construct a constraint loop simulating it as follows. The loop has variables $X_1,\ldots,X_n,X_{n+1}$ and $A_1,\ldots,A_{m+2}$. The guard is

$$X_1 \geq 0 \wedge \cdots \wedge X_{n+1} \geq 0 \wedge A_1 \geq 0 \wedge \cdots \wedge A_{m+2} \geq 0 \wedge \sum_i A_i = 1$$

and the update is the conjunction of the following constraints,

$$X'_j = X_j + (\sum_{i=1}^{m} \mathbf{v}_i[j] \cdot A_i) - \mathbf{t}[j] \cdot A_{m+1} \qquad \text{for } j = 1,\ldots,n, \tag{3}$$

$$A'_1 \geq 0 \wedge \cdots \wedge A'_{m+2} \geq 0 \wedge \sum_i A'_i = 1, \tag{4}$$

$$X'_{n+1} = X_{n+1} - (\sum_{j=1}^{n} X_j), \tag{5}$$

$$A'_{m+2} \geq A_{m+1} + A_{m+2}\,. \tag{6}$$

The initial state for our loop is just the given initial state **s** (for the X_i) and unspecified for the A_i.

Explanation: As before, the variables $X_1,\ldots,X_n$ and $A_1,\ldots,A_m$ are used to simulate the VAS. The X's represent the state vector **x**, and A's are flags which change non-deterministically to indicate the next transition. This simulation goes on as long as A_{m+1} or A_{m+2} have not turned on, and as long as it does go on, X_{n+1} descends, by (5). If A_{m+1} turns on, the target vector **t** is substracted from $(X_1 \ldots X_n)$, so such a transition is only enabled if this vector **x** is at least as large as **t**. Suppose that this happens. Then by (6), later transitions are forced to have $A_{m+2} = 1$, so they do not simulate the loop any longer, and the X's do not change, except for X_{n+1}, which continues to decrease if and only if **x** at the start of this phase was *not equal* to **t**.

Hence, if **t** is reachable from **s**, it is possible to run into a non-terminating computation where nothing decreases, and the loop has no *LRF*. Otherwise, X_{n+1} keeps decreasing, even when $A_{m+2} = 1$, so it constitutes a *LRF*.

We should note that it is possible to turn A_{m+2} on without passing through $A_{m+1} = 1$, and in this case X_{n+1} keeps decreasing regardless of the reachability question, so our reduction remains correct. □

4.5 Ranking versus termination

As in Section 3, we can see that our reduction yields a loop which either has the specified ranking function, or does not halt at all, which means that the existence of any "termination witness" (like *LRF*) which can handle the loop (in particular, any witness which encompasses single-variable *LRF*s) will also be PSPACE-hard. On the other hand, we can show (using the same trick as in Corollary 3.4) that the $\text{LINRF}^{\rho}(\mathbb{Z})$ problem is EXPSPACE-hard even if restricted to loops that do terminate.

4.6 Deterministic loops

Both of the above hardness results also hold for deterministic constraint loops. In order to do that, we need to "determinize" the loop constructed in the reduction. The technique is from [4], and consists of adding an uninitialized variable, whose value is used as an "oracle," to guide the non-deterministic choices. In the case that there is a computation which makes the value of X_n positive (in our first reduction) or the value of $\sum_{j=1}^{n} X_j$ zero (for the second), there will be a value for the oracle variable that guides the computation to this state. See [4, Sect. 6.1] for more details.

5 Hardness for Invariant-Supported LRFs

In this section we turn to the problem of invariant-supported LRFs, namely the decision problem defined as follows:

> **Instance:** an SLC^{ρ} loop.
>
> **Question:** does there exist an inductive invariant *INV* (of a particular class) for this loop, such that there is a *LRF* for $\{\binom{\mathbf{x}}{\mathbf{x}'} \mid \mathbf{x} \in INV\}$?

We give two hardness results, depending on the type of invariant: PSPACE-hardness for convex polyhedra and EXPSPACE-hardness for downward-closed sets over $\mathbb{N}^n$. Both are derived from the constructions of earlier sections, by noticing that if there is a ranking function, there is an invariant to support it. Both address integer loops only.

THEOREM 5.1. *For deterministic integer SLC^{ρ} loops, deciding whether convex polyhedral invariant exists which supports a LRF for the loop is PSPACE-hard.*

Proof. We use the reduction from halting of Boolean programs (Section 3). We claim that when there is a *LRF* (which would consist of the variable Y as shown in the proof of Corollary 3.3), then there is a convex polyhedral invariant supporting it. Indeed, assume that the Boolean program does not halt. Let $\mathcal{R}$ be the set of reachable states of the SLC^{ρ} loop constructed. In all these states, the variables are 0-1 valued (except for Y, which is unbounded). Note that the convex hull of a set V of 0-1 vectors includes no other integer vectors besides V. Thus, as we are considering an integer loop, the convex hull of $\mathcal{R}$ represents $\mathcal{R}$ precisely, and constitutes an invariant (the set of reachable states is clearly an inductive invariant) which supports the *LRF*. □

THEOREM 5.2. *For deterministic SLC^{ρ} loops over the natural numbers, deciding whether a downward-closed invariant exists which supports a LRF for the loop is EXPSPACE-hard.*

Proof. We reuse the reduction from the Eventual Positivity problem of Petri nets (Theorem 4.2). We claim that when there is a *LRF* (which would consist of the variable Y as shown in the proof), then there

is a downward-closed invariant supporting it. Let $\mathcal{R}$ be the set of reachable states form the given initial states, and consider its downward-closure $\check{\mathcal{R}}$. It clearly contains the initial states, and it is easy enough to verify that it is closed under the transition relation (recall that $\mathcal{R}$ is closed, by definition). Thus $\check{\mathcal{R}}$ constitutes an invariant; it supports the *LRF* Y because X_n is zero in all these states. □

As in previous sections, both of the above hardness results are also valid when restricting to programs that do terminate.

6 Related work

Termination analysis has been the subject of many papers (too many to list here), but, in addition to works already mentioned, the following works seem closely related.

There are some tools which, like TERMINATOR, use a counterexample-directed approach which naturally calls for the analysis of lasso programs. Examples include [12, 18, 28].

Bagnara et al. [2] give a clear exposition on the computation of universal ranking functions, comparing [27] with previous solutions [25, 29] that use essentially the same approach. Recently, several works addressed the generation of more complex termination proofs, in particular, involving lexicographically-decreasing tuples of linear functions. The universal problem for linear-constraint loops is analysed in [3], while preconditions have been taken into account in some works: Bradley et al. [6, 7] search for supporting invariant using constraint solving (for multi-path loops). Alias et al. [1] handle control-flow graphs of any form, but require precomputed invariants. Brockschmidt et al. [8] use an iterative method in which invariant generation is guided by the needs of the termination prover. They use a separate safety checker to provide them, while Larraz et al. [22] use the constraint-solving approach to find supporting invariants together with the ranking functions, but using iterative improvement as in the latter work.

Regarding the computation of *termination preconditions*, Bozga et al. [5] show that for loops specified by *octagonal relations* a precondition for *termination* can be computed in polynomial time. But the proof does not necessarily produce a linear ranking function.

7 Concluding Remarks

We have established lower bounds (that is, hardness results) on the complexity of the linear-ranking problem with preconditions, first in its general form and then when restricted to *LRF*s supported by two forms of effectively inductive invariants. In fact, our lower bounds hold for the linear-ranking function verification problem:

> **Instance:** an SLC^{ρ} loop and an affine-linear function f.
>
> **Question:** is f a *LRF* for this loop?

Moreover, the lower bounds hold for a simple kind of precondition, namely a partially-specified input. Even for this case, we do not have upper bounds, and obtaining them seems extremely difficult. We still have no answer to the following intriguing questions: *Is any of the decision problem studied here any easier than termination for the corresponding class of loops? And are they equivalent to reachability?*

An interesting open problem results from restricting the update, say to *affine linear*: $\mathbf{x}' = A'\mathbf{x} + \mathbf{a}'$. We note that the notorious *positivity problem for linear recurrence sequences* [26] translates easily to *LRF* verification—to check whether x_1 is always positive, add a variable x_{n+1} with $x'_{n+1} = x_{n+1} - x_1$, put

$x_{n+1} \geq 0$ in the guard and ask whether $f(\mathbf{x}) = x_{n+1}$ is a *LRF*. It would be interesting to know whether a reduction to *LRF existence* can also be found.

An interesting direction for further research may be to find out the implications of fixing the number of variables. For our VAS problem it is known to make the problem solvable in polynomial space [14, Corollary 3.4.5]. Our lower bounds do not give any significant result in this case (and clearly, the problem does get easier for sufficiently small n: at least for $n = 1$ it does!).

Another, very natural, idea is to restrict the invariants to a fixed (or polynomial) number of conjuncts or disjuncts. In fact, all the works using the constraint-solving approach are based on such a restriction. But is the problem tractable now? There is again an intriguing lack of results. For conjunctions of a given number of linear constraints, Bradely, Manna and Sipma [6] show decidability in exponential time, but only over the reals (it is not clear whether results would be different over the rationals). Heizmann et al. [19] show a polynomial-time procedure for loops over the reals or rationals, when the invariant is a single half-space; and they prove completeness only under an additional restriction. So there is a long way ahead.

References

[1] Christophe Alias, Alain Darte, Paul Feautrier & Laure Gonnord (2010): *Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs*. In Radhia Cousot & Matthieu Martel, editors: *Static Analysis Symposium, SAS'10, LNCS* 6337, Springer, pp. 117–133, doi:10.1007/978-3-642-15769-1_8.

[2] Roberto Bagnara, Fred Mesnard, Andrea Pescetti & Enea Zaffanella (2012): *A new look at the automatic synthesis of linear ranking functions*. *Inf. Comput.* 215, pp. 47–67, doi:10.1016/j.ic.2012.03.003.

[3] Amir M. Ben-Amram & Samir Genaim (2014): *Ranking Functions for Linear-Constraint Loops*. *Journal of the ACM*. Accepted.

[4] Amir M. Ben-Amram, Samir Genaim & Abu Naser Masud (2012): *On the Termination of Integer Loops*. *ACM Trans. Program. Lang. Syst.* 34(4), pp. 16:1–16:24, doi:10.1145/2400676.2400679.

[5] Marius Bozga, Radu Iosif & Filip Konecný (2014): *Deciding Conditional Termination*. Technical Report. Available at `http://arxiv.org/abs/1302.2762`.

[6] Aaron Bradley, Zohar Manna & Henny Sipma (2005): *Linear Ranking with Reachability*. In Kousha Etessami & Sriram Rajamani, editors: *Computer Aided Verification, Lecture Notes in Computer Science* 3576, Springer Berlin / Heidelberg, pp. 247–250, doi:10.1007/11513988_48.

[7] Aaron R. Bradley, Zohar Manna & Henny B. Sipma (2005): *The Polyranking Principle*. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi & Moti Yung, editors: *Proc. 32nd International Colloquium on , Languages and Programming, Lecture Notes in Computer Science* 3580, Springer Verlag, pp. 1349–1361, doi:10.1007/11523468_109.

[8] Marc Brockschmidt, Byron Cook & Carsten Fuhs (2013): *Better Termination Proving through Cooperation*. In Natasha Sharygina & Helmut Veith, editors: *Computer Aided Verification, CAV 2013, Lecture Notes in Computer Science* 8044, Springer, pp. 413–429, doi:10.1007/978-3-642-39799-8_28.

[9] Michael Colón & Henny Sipma (2001): *Synthesis of Linear Ranking Functions*. In: *7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lecture Notes in Computer Science* 2031, Springer, pp. 67–81, doi:10.1007/3-540-45319-9_6.

[10] Byron Cook, Sumit Gulwani, Tal Lev-Ami, Andrey Rybalchenko & Mooly Sagiv (2008): *Proving conditional termination*. In: *Computer Aided Verification, CAV'08, Lecture Notes in Computer Science* 5123, Springer, pp. 328–340, doi:10.1007/978-3-540-70545-1_32.

[11] Byron Cook, Andreas Podelski & Andrey Rybalchenko (2006): *Termination proofs for systems code*. In Michael I. Schwartzbach & Thomas Ball, editors: *Programming Language Design and Implementation, PLDI'06*, ACM, pp. 415–426, doi:10.1145/1133981.1134029.

[12] Byron Cook, Abigail See & Florian Zuleger (2013): *Ramsey vs. Lexicographic Termination Proving*. In Nir Piterman & Scott A. Smolka, editors: *Tools and Algorithms for the Construction and Analysis of Systems,TACAS 2013, Lecture Notes in Computer Science* 7795, Springer, pp. 47–61, doi:10.1007/978-3-642-36742-7_4.

[13] Patrick Cousot & Nicholas Halbwachs (1978): *Automatic Discovery of Linear Restraints Among Variables of a Program*. In: *Conference Record of the Fifth annual ACM Symposium on Principles of Programming Languages*, ACM, ACM, pp. 84–96, doi:10.1145/512760.512770.

[14] Stéphane Demri (2011): *Decidable Problems for Counter Systems*. Available at `http://www.lsv.ens-cachan.fr/~demri/esslli2010-lecture-notes.pdf`. Revised lecture notes from a course at ESSLLI 2010, Copenhagen.

[15] Javier Esparza (1998): *Decidability and Complexity of Petri Net Problems—An Introduction*. In Wolfgang Reisig & Grzegorz Rozenberg, editors: *Lectures on Petri Nets, Vol. I: Basic Models, LNCS* 1491, Springer-Verlag (New York), Dagstuhl, Germany, pp. 374–428, doi:10.1007/3-540-65306-6_20.

[16] Javier Esparza & Mogens Nielsen (1994): *Decidability Issues for Petri Nets*. Technical Report RS-94-8, BRICS, Department of Computer Science, University of Aarhus.

[17] Michel H. T. Hack (1979): *Decidability questions for Petri Nets*. Ph.D. thesis, MIT.

[18] William R Harris, Akash Lal, Aditya V Nori & Sriram K Rajamani (2011): *Alternation for termination*. In: *Static Analysis Symposium, SAS 2011, LNCS* 6337, Springer, pp. 304–319, doi:10.1007/978-3-642-15769-1_19.

[19] Matthias Heizmann, Jochen Hoenicke, Jan Leike & Andreas Podelski (2013): *Linear Ranking for Linear Lasso Programs*. In Dang Hung & Mizuhito Ogawa, editors: *Automated Technology for Verification and Analysis, Lecture Notes in Computer Science* 8172, Springer International Publishing, pp. 365–380, doi:10.1007/978-3-319-02444-8_26.

[20] Neil D. Jones (1997): *Computability and Complexity From a Programming Perspective*. Foundations of Computing Series, MIT Press.

[21] Richard M Karp & Raymond E Miller (1969): *Parallel program schemata*. *Journal of Computer and system Sciences* 3(2), pp. 147–195, doi:10.1016/S0022-0000(69)80011-5.

[22] Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell & Albert Rubio (2013): *Proving termination of imperative programs using Max-SMT*. In: *Formal Methods in Computer-Aided Design, FMCAD 2013*, IEEE, pp. 218–225, doi:10.1109/FMCAD.2013.6679413.

[23] Jérôme Leroux (2011): *Vector Addition System Reachability Problem: A Short Self-contained Proof*. In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, ACM, New York, NY, USA, pp. 307–316, doi:10.1145/1926385.1926421.

[24] Richard J. Lipton (1976): *The Reachability Problem Requires Exponential Space*. Technical Report 63, Yale University.

[25] Frédéric Mesnard & Alexander Serebrenik (2008): *Recurrence with affine level mappings is P-time decidable for CLP(R)*. *TPLP* 8(1), pp. 111–119, doi:10.1017/S1471068407003122.

[26] Joël Ouaknine & James Worrell (2014): *Positivity problems for low-order linear recurrence sequences*. In: *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, SIAM, doi:10.1137/1.9781611973402.27.

[27] Andreas Podelski & Andrey Rybalchenko (2004): *A Complete Method for the Synthesis of Linear Ranking Functions*. In Bernhard Steffen & Giorgio Levi, editors: *Verification, Model Checking, and Abstract Interpretation, VMCAI'04*, *LNCS* 2937, Springer, pp. 239–251, doi:10.1007-978-3-540-24622-0_20.

[28] Andreas Podelski & Andrey Rybalchenko (2007): *ARMC: the logical choice for software model checking with abstraction refinement*. In: *Practical Aspects of Declarative Languages*, *LNCS* 4354, Springer, pp. 245–259, doi:10.1007/978-3-540-69611-7_16.

[29] Kirack Sohn & Allen Van Gelder (1991): *Termination detection in logic programs using argument sizes (extended abstract)*. In: *Proceedings of the Tenth ACM SIGACT-SIGMOD-SOGART Symposium on Principles of Database Systems (PODS), May 1991, Denver, Colorado*, ACM Press, pp. 216–226, doi:10.1145/113413.113433.