The Octatope Abstract Domain for Verification of Neural Networks

Stanley Bak[1], Taylor Dohmen[2], K. Subramani[3], Ashutosh Trivedi[2(✉)], Alvaro Velasquez[2], and Piotr Wojciechowski[3]

[1] Stony Brook University, Stony Brook, NY, USA
[2] University of Colorado, Boulder, CO, USA
ashutosh.trivedi@colorado.edu
[3] West Virginia University, Morgantown, WV, USA

Abstract. Efficient verification algorithms for neural networks often depend on various abstract domains such as *intervals*, *zonotopes*, and *linear star sets*. The choice of the abstract domain presents an expressiveness vs. scalability trade-off: simpler domains are less precise but yield faster algorithms. This paper investigates the *octatope* abstract domain in the context of neural net verification. Octatopes are affine transformations of n-dimensional octagons—sets of unit-two-variable-per-inequality (UTVPI) constraints. Octatopes generalize the idea of zonotopes which can be viewed as an affine transformation of a box. On the other hand, octatopes can be considered as a restriction of linear star set, which are affine transformations of arbitrary $\mathcal{H}$-Polytopes. This distinction places octatopes firmly between zonotopes and star sets in their expressive power, but what about the efficiency of decision procedures?

An important analysis problem for neural networks is the *exact range computation* problem that asks to compute the exact set of possible outputs given a set of possible inputs. For this, three computational procedures are needed: 1) optimization of a linear cost function; 2) affine mapping; and 3) over-approximating the intersection with a half-space. While zonotopes allow an efficient solution for these approaches, star sets solves these procedures via linear programming. We show that these operations are faster for octatopes than the more expressive linear star sets. For octatopes, we reduce these problems to min-cost flow problems, which can be solved in strongly polynomial time using the Out-of-Kilter algorithm. Evaluating exact range computation on several ACAS Xu neural network benchmarks, we find that octatopes show promise as a practical abstract domain for neural network verification.

1 Introduction

The success of deep feed-forward neural networks (DNN) in computer vision and speech recognition has prompted applications in critical infrastructure. These applications range from using pre-trained perception and speech-recognition modules in safety-critical logic (self-driving cars and medical decision making)

M. Chechik et al. (Eds.): FM 2023, LNCS 14000, pp. 454–472, 2023.
https://doi.org/10.1007/978-3-031-27481-7_26

to learning controllers from reinforcement signals [31] to learning succinct representations of formally verified controllers (ACAS Xu). The increasing prevalence of DNNs in safety-, privacy-, and social-critical systems motivates the focus of the formal methods community [3,5,7,34] in developing verification technology to meet the challenge of improving trust in DNNs.

Abstract interpretation [4,11] is a well-established framework for program verification that formalizes the exploration of the program semantics at the granularity provided by the underlying domain. For example, intervals [11] form an abstract domain facilitating analysis in which sets of states are represented as hyperrectangles. Other abstract domains such as difference constraints, octagons (unit-two-variables-per-inequality or UTVPI), and polyhedral (linear constraints) have been successfully deployed for the verification of DNNs. However, the multi-layer architecture of DNNs, when combined with linear function composition followed by a non-linear activation function at each layer, results in the repeated intersection of abstract spaces with linear inequalities. For this reason, abstract domains that do not permit an efficient *affine mapping* suffer in exploring the layered state space of the DNNs.

Zonotopes [29] solve this problem by representing an abstract set as an affine mapping of an interval generator set. For zonotopes, the key operations for DNN verification, such as nonemptiness, optimization, and over-approximation, can be performed via efficient, enumerative procedures. Linear star sets [13,35] generalize zonotopes by representing the generator set using the polyhedral domains. This generalization, while improving the expressiveness, leads to the decision procedures depend upon solving linear programs, which tends to be the performance bottleneck in the overall algorithm. While linear programming is known to be solvable in polynomial time, via a number of celebrated interior-point algorithms [22], there is no known strongly polynomial algorithm. Dantzig's simplex algorithm is a popular algorithm to solve LP and works well in practice, but for general LPs, the time complexity of the simplex algorithm is not polynomial [23], and subexponential lower bounds hold even for randomized pivoting rules [14].

For some subclasses of linear programming problems, more efficient solutions exist. In particular, when the constraints are restricted to difference constraints ($x_i - x_j \leq c$) or UTVPI constraints ($\pm x_i \pm x_j \leq c$), then the duals of the corresponding LPs can be reduced to *minimum cost flow* (MCF) problems [2], for which there exist strongly polynomial time algorithms [17]. The Out-of-Kilter algorithm is one popular algorithm for solving minimum cost flow that also produces a solution to the dual [2]. It runs in time $O((m^2 + m \cdot n \cdot \log n) \cdot U)$ on a network with m arcs and n nodes and maximum supply/demand U. Alternatively, the network simplex algorithm is a specialized version of the simplex algorithm to solve minimum cost flow problems. Unlike standard simplex, network simplex runs in polynomial time [28]. Given its relative efficiency, it is natural to ask: *in neural network verification, is it possible to replace expensive linear programming with min-cost flow calls?*

This question motivates the investigation of sub-classes of star sets that are more general than zonotopes, but enable efficient decision procedures based on

MCF problems. For this purpose, we introduce *octatopes*: sets that can be defined as affine maps of UTVPI constrained sets (octagons [27]). Since octatopes are a special class of star sets, the affine transformation remains efficient. We also study *hexatopes* as the images of difference constrained sets (hexagons [27] or zones [9]). A key contribution of this paper is that the key operations required for verification using octatopes and hexatopes can be performed efficiently using algorithms for MCF problems.

Given that the MCF problem can be solved efficiently via Out-of-Kilter algorithm and network simplex (touted [8] to be 200–300 times faster than simplex), this benefit will translate to the efficiency of octatopes/hexatopes for LP-intensive applications like reachability analysis of neural networks. While the current state-of-the-art implementations of the algorithms for the MCF problem are not as advanced as those for LP, we believe that this will change in light of the proposed application. We implement the octatope and hexatope abstract domains and show their effectiveness on several ACAS Xu networks [20], a popular benchmark for neural network verification.

Related Work. A growing body of research exists on different methods to verify neural networks [25], including recent tool competitions [6]. Algorithms can be categorized into search, optimization, and reachability solutions. In the space of search procedures, the seminal Reluplex method proposes an extension of the simplex algorithm used for linear programming to handle ReLU networks [20]. This method has been widely adopted and extended by, for example, posing verification as a constraint satisfaction problem [21]. This can then be solved using off-the-shelf Satisfiability Modulo Theory (SMT) solvers like z3 [12]. The use of SMT enables reasoning over different activation functions and topologies.

Interval arithmetic is another popular approach often used to estimate the range of output values given a range of inputs while tracking the input and output ranges of individual activation functions [38]. This can be computed by using linear programming to derive lower and upper bounds for a given node in the network. The work of [18] combines this with symbolic interval propagation and gradient descent to find counter-examples to the over-approximations established by the linear programming solutions. More sophisticated node splitting strategies that account for downstream effects on successor nodes can also be used as part of the symbolic interval propagation phase [19]. Per-neuron split constraints can also further improve efficiency [39].

Optimization solutions to the verification based on ILP have been explored. This is a natural formulation for the verification of neural networks due to the use of affine transformations and the fact that piecewise linear activation functions can be encoded using a set of binary linear constraints [3]. The work in [32] extends similar ideas by estimating the maximum disturbance that is permitted at the input and proposing pre-solve procedures to speed up the solution.

Although solutions based on SMT-solving and mathematical programming are often complete, they require the entire network to be encoded within the corresponding constraints, thereby limiting scalability. In contrast to these search and optimization solutions, the use of reachability analysis for verification of

neural networks has been shown to scale to larger instances at the cost of completeness. Examples of this include the use of zonotope and star set abstract domains. The former can be efficiently employed to compute conservative over-approximations of output bounds of nodes in a network [16], whereas linear programming can be employed for the latter to find tight bounds at the cost of scalability [37]. The work proposed herein seeks to advance the state of verification methods based on reachability analysis by providing tighter over-approximations than zonotopes and more efficient computations than star sets.

2 Preliminaries

Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{Q}$ denote the set of rational numbers. We write $\mathbb{R}^{m\times n}$ for the set of all $m\times n$ dimensional matrices of reals.

For a matrix $M \in \mathbb{R}^{m\times n}$, we write $M(i,\cdot) \in \mathbb{R}^{1\times n}$ and $M(\cdot,j) \in \mathbb{R}^{m\times 1}$ for the i^{th} row vector and j^{th} column vector, respectively, of M, for $1 \leq i \leq m$ and $1 \leq j \leq m$. Similarly, we write $M(i,j)$ for the matrix element at row i and column j. By default, a vector is a column vector and we associate a set of matrices $\mathbb{R}^{m\times 1}$ with the set of vectors $\mathbb{R}^m$.

For a matrix $M \in \mathbb{R}^{m\times n}$ we write $M^{\mathsf{T}} \in \mathbb{R}^{n\times m}$ for its transpose matrix. For a row vector $\boldsymbol{v} \in \mathbb{R}^{1\times n}$, we write $\boldsymbol{v}^{\mathsf{T}} \in \mathbb{R}^n$ for the corresponding (transposed) vector. We write $\mathbf{1}_n$ for the all-ones vector of size n and I for the identity matrix of some fixed dimension (often clear from context). For a (column) vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_n) \in \mathbb{R}^n$ we write $\boldsymbol{v}_i$ for its i^{th} element. For a vector $\boldsymbol{v} \in \mathbb{R}^m$ and scalar $\alpha \in \mathbb{R}$, we write $\alpha \cdot \boldsymbol{v}$ for the vector $(\alpha \cdot \boldsymbol{v}_1, \ldots, \alpha \cdot \boldsymbol{v}_m)$. For two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m$, we write $\boldsymbol{u} \cdot \boldsymbol{v}$ for their dot product, i.e., $\boldsymbol{u} \cdot \boldsymbol{v} = \sum_{i=1}^{m} \boldsymbol{u}_i \cdot \boldsymbol{v}_i$. For two matrices $M \in \mathbb{R}^{m\times n}$ and $N \in \mathbb{R}^{n\times p}$, their product $MN \in \mathbb{R}^{m\times p}$ is defined as $MN(i,j) = M(i,\cdot)^{\mathsf{T}} \cdot N(\cdot,j)$.

We call a function $f : \mathbb{R}^n \to \mathbb{R}^m$ *linear* if $f(\boldsymbol{u}) + f(\boldsymbol{v}) = f(\boldsymbol{u} + \boldsymbol{v})$ and $f(\alpha \cdot \boldsymbol{v}) = \alpha \cdot f(\boldsymbol{v})$ for all scalars $\alpha \in \mathbb{R}$ and vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$. A linear function $f : \mathbb{R}^n \to \mathbb{R}^m$ can be represented as a matrix $A \in M^{m\times n}$ such that f is equivalent to $\boldsymbol{u} \mapsto A\boldsymbol{u}$. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is affine if it is a sum of a linear function and a constant, i.e., $f(\boldsymbol{v}) = A\boldsymbol{v} + \boldsymbol{b}$ for some $A \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$.

2.1 Linear, UTVPI, and Difference Constraints

Let $\boldsymbol{x} = \{x_1, x_2, \ldots, x_n\}$ be a set of real-valued variables with an arbitrary but fixed order. Abusing notation, we represent this set as a vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$. A *linear constraint* over $\boldsymbol{x}$ is a constraint of the form

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b \text{ where } \boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{R}^n \text{ and } b \in \mathbb{R}$$

that represents the set $\{\boldsymbol{v} \in \mathbb{R}^n : \boldsymbol{a} \cdot \boldsymbol{v} \leq b\}$. A *linear constraint system* (LCS)

$$A\boldsymbol{x} \leq \boldsymbol{b} \text{ where } A \in \mathbb{R}^{m\times n} \text{and } \boldsymbol{b} \in \mathbb{R}^m$$

is a conjunction of linear constraints.

Definition 1 (Interval Constraint Systems). *An* interval constraint *is a linear constraint of the form*

$$a_i \leq x_i \leq b_i \text{ where } a_i, b_i \in \mathbb{Q}.$$

An interval constraint system (ICS) is a conjunction of interval constraints. An ICS is a unit hypercube if $a_i = -1$ *and* $b_i = 1$ *for all* $1 \leq i \leq n$ *and we denote it as* $-\mathbf{1}_n \leq \boldsymbol{x} \leq \mathbf{1}_n$.

Definition 2 (Difference Constraint Systems). *A* difference constraint *is a linear constraint of the form*

$$x_i - x_j \leq b_i \text{ where } b_i \in \mathbb{Q}.$$

A difference constraint system (DCS) is a conjunction of difference constraints.

Definition 3 (UTVPI Constraint System). *A Unit Two Variable Per Inequality (*UTVPI*) constraint is a linear constraint of the form*

$$a_i \cdot x_i + a_j \cdot x_j \leq b_{ij} \text{ where } a_i, a_j \in \{-1, 0, +1\} \text{ and } b_{ij} \in \mathbb{Q}.$$

A UTVPI *constraint system (UCS) is a conjunction of* UTVPI *constraints.*

A UTVPI constraint $a_i \cdot x_i + a_j \cdot x_j \leq b$ is said to be an absolute constraint if $a_i = 0$ or $a_j = 0$. An absolute constraint can be converted into constraints of the form: $a_i \cdot x_i + a_j \cdot x_j \leq b_{ij}$, where both a_i and a_j are non-zero. Note that a UTVPI constraint $a_i \cdot x_i + a_j \cdot x_j \leq b_{ij}$, $b_{ij} \in \mathbb{Q}$ is a difference constraint if $a_i = -a_j$. The constant that bounds a UTVPI constraint is called the defining constant. For instance, the defining constant for the constraint $x_1 - x_2 \leq 9$ is 9.

2.2 Minimum Cost Network Flow Problem

When optimizing a linear function over DCS or UCS, its dual program can be reduced to the *minimum cost flow* (MCF) problem [2], for which there exist strongly polynomial time algorithms [17]. We review the Out-of-Kilter algorithm (Algorithm 1) for MCF that also produces a solution to the dual [2].

A *flow network* $\boldsymbol{G} = (G = (V, E), c, a, d)$ is a directed graph G with capacity $c : E \to \mathbb{R}_{\geq 0}$ and cost $a : E \to \mathbb{R}$ associated with every edge (arc) and demand $d : V \to \mathbb{R}$ associated with every vertex (node). We assume that $\sum_{v \in V} d(v) = 0$. The *minimum cost flow* (MCF) problem can be stated as follows:

$$\text{Minimize} \sum_{(u,v) \in E} f(u, v) \cdot a(u, v)$$

subject to:

$$\sum_{u \in V} f(u, v) - \sum_{u \in V} f(v, u) = d(v) \qquad \text{for all } v \in V,$$
$$0 \leq f(u, v) \leq c(u, v) \qquad \text{for all } (u, v) \in E$$

Algorithm 1. OUT-OF-KILTER($\boldsymbol{G} = (G = (V, E), c, a, d)$)

Initialize the potential as $\pi \leftarrow 0$.
Let f be a flow in $\boldsymbol{G}$.
Construct the residual network $\boldsymbol{G}_f$.
Compute the kilter number $k(u, v)$ of each edge (u, v) in $\boldsymbol{G}_f$.
while ($\boldsymbol{G}_f$ contains an edge with positive kilter number) **do**
    Select an edge (u, v) in $\boldsymbol{G}_f$ with positive kilter number.
    Let the weight of each edge (u, v) in $\boldsymbol{G}_f$ be $\max\{0, c^\pi(u, v)\}$.
    For $w \in V \setminus \{u, v\}$, let $l(w)$ be the weight of the least weight path from v to w.
    Let P be a shortest path from v to u.
    For each node w, set $\pi(w) \leftarrow \pi(w) - l(w)$.
    if ($c^\pi(u, v) < 0$) **then**
        $Q \leftarrow P \cup \{(u, v)\}$.
        $\delta \leftarrow \min_{(u,v)\in Q} r(u, v)$.
        Augment δ units of flow along Q.
        Update f and $\boldsymbol{G}_f$.
return f.

Out-of-Kilter Algorithm. A pseudocode for the Out-of-Kilter algorithm is given as Algorithm 1. It starts with a possibly infeasible flow and iteratively modifies this flow in a way that decreases the infeasibility of the solution and moves it closer to optimality. Each step of the algorithm consists of solving a shortest path problem and augmenting the flow along the shortest path. It operates on the residual network $\boldsymbol{G}_f$ corresponding to the current flow f. This residual network is constructed as follows. For each edge $(v_i, v_j) \in E$:

1. **Feasible Edges.** If $f(u, v) < c(u, v)$, we add the edge (u, v) with a residual capacity of $r(u, v) = c(u, v) - f(u, v)$ and cost $a(u, v)$. If $f(u, v) > 0$, we add the edge (v, u) with a residual capacity of $r(v, u) = f(u, v)$ and cost $-a(u, v)$.
2. **Lower-Infeasible Edges.** If $f(u, v) < 0$, we add the edge (u, v) with a residual capacity of $r(u, v) = -f(u, v)$ and cost $a(u, v)$.
3. **Upper-Infeasible Edges.** If $f(u, v) > c(u, v)$, we add the edge (v, u) with a residual capacity of $r(v, u) = f(u, v) - c(u, v)$ and cost $-a(u, v)$.

For each vertex v in the residual network, the algorithm maintains a potential $\pi(v)$ and for each edge (u, v) with cost $a(u, v)$, it maintains the reduced cost $a^\pi(u, v) = c(u, v) - \pi(u) + \pi(v)$. Additionally, for each edge in the residual network, it maintains a kilter number $k(u, v)$ which is 0 if $c^\pi(u, v) \geq 0$ and is the residual capacity $r(u, v)$ if $c^\pi(u, v) < 0$. This kilter number represents the change in flow required so that each edge satisfies its optimality condition.

Note that the node potentials π and reduced costs c^π corresponding to the optimal flow f are the optimal solution of the dual problem [1]. The Out-of-Kilter algorithm runs in time $O((m^2 + m \cdot n \cdot \log n) \cdot D)$ on a network with m edges and n vertices and maximum demand D.

2.3 Verification of Neural Networks

A rectified linear unit (ReLU) is a commonly used activation function $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ defined as $\sigma(x) = \max\{x, 0\}$. We can generalize this function from scalars to vectors as $\sigma : \mathbb{R}^n \rightarrow \mathbb{R}^n$ in a straightforward fashion by applying ReLU componentwise. In this paper, we primarily work with feedforward neural networks (NN) with ReLU activation units. We focus on networks with k fully-connected layers, also called multi-layer perceptrons, where each layer i is defined with a weight matrix W_i and a bias vector b_i of appropriate size and is followed by a ReLU.

Formally, a *neural network* can be viewed as a function $f : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^{n_o}$, where n_i is the number of inputs and n_o is the number of outputs. Given an input $y_0 \in \mathbb{R}^{n_i}$, a neural network will compute an output $y_k \in \mathbb{R}^{n_o}$ as follows:

$$\begin{aligned} x^{(1)} &= W_1 y_0 + b_1, & y_1 &= \sigma(x^{(1)}) \\ x^{(2)} &= W_2 y_1 + b_2, & y_2 &= \sigma(x^{(2)}) \\ &\vdots \\ x^{(k)} &= W_k y_{k-1} + b_k, & y_k &= \sigma(x^{(k)}) \end{aligned}$$

We call y_{i-1} and y_i the input and output of the i-th layer, respectively, and $x^{(i)}$ the intermediate values at layer i. This setup is the most typical situation considered for neural network verification tools [6], although extensions have been made to other layer types [33,36] and activation functions [30].

Definition 4 (Exact Range Computation Problem). *Given a neural network implementing the function $f : \mathbb{R}^{n_i} \rightarrow \mathbb{R}^{n_o}$ and an input set $\mathcal{I} \subseteq \mathbb{R}^{n_i}$, the exact range computation problem is to compute the set*

$$\mathsf{Range}(f, \mathcal{I}) = \{y_k \mid y_k = f(y_0),\ y_0 \in \mathcal{I}\}.$$

of possible outputs of the network.

The exact range computation problem can be used to solve the open-loop neural network verification problem defined next.

Definition 5 (Open-Loop Neural Network Verification). *Given an input set $\mathcal{I} \subseteq \mathbb{R}^{n_i}$, an unsafe set $\mathcal{U} \subseteq \mathbb{R}^{n_o}$, and a neural network that computes f, the open-loop neural network verification problem asks if $\mathsf{Range}(f, \mathcal{I}) \cap \mathcal{U} = \emptyset$.*

As is typical with the state-of-practice in DNN verification, we restrict the input and unsafe sets to ones defined with linear constraints,

$$\begin{aligned} \mathcal{I} &= \{\boldsymbol{x} \mid A_i \boldsymbol{x} \leq b_i, \boldsymbol{x} \in \mathbb{R}^{n_i}\}, \text{ and} \\ \mathcal{U} &= \{\boldsymbol{x} \mid A_u \boldsymbol{x} \leq b_u, \boldsymbol{x} \in \mathbb{R}^{n_o}\}. \end{aligned}$$

The popular ACAS Xu neural network verification benchmarks [20] match these assumptions, and will be used in our evaluation.

Although abstraction and refinement methods are often more efficient for verifying neural networks [5], the performance of the exact range computation problem is important for the following two reasons. First, as more refinement needs to be done, the performance of abstraction-refinement will approach that of exact range computation. Efficient exact range computation is therefore essential for efficient abstraction-refinement analysis. Second, exact range computation methods are building blocks for other types of verification problems, such as closed-loop verification of neural-network control systems [26], which often arise in reinforcement learning applications. In these cases, over-approximating the range of a network is too imprecise for analysis over many control cycles, and such analysis loses the relationship between the inputs and the outputs of a network, creating issues similar to the dependency problem in interval arithmetic [34]. In future work, we plan to explore over-approximation methods as well as abstraction-refinement approaches that use octatopes, following similar work done for zonotopes and other abstract domains [15].

3 Abstract Domains: Octatopes and Hexatopes

Both a zonotope and a star set may be viewed as an n-dimensional image of a polytope—which we refer to as the *kernel*—under affine transformation. For zonotopes the kernel is a hypercube, while for linear star sets the kernel is a set defined by an LCS. In this section, we introduce octatopes and hexatopes as generalizations of zonotopes where the kernel is restricted to be a set defined by a UCS and a DCS, respectively. This section also studies algorithms for operations over octatopes and hexatopes required for the verification of neural networks.

3.1 Zonotopes and Linear Star Sets

An n-dimensional *zonotope* $Z = \langle \boldsymbol{c}, G \rangle$ is the image of a p-dimensional hypercube under an affine transformation $\mathbb{R}^p \to \mathbb{R}^n$. Given a center $\boldsymbol{c} \in \mathbb{R}^n$ and a set of generator vectors $\{\boldsymbol{g}_1, \ldots, \boldsymbol{g}_p \in \mathbb{R}^n\}$ forming a matrix $G = \begin{bmatrix} \boldsymbol{g}_1 \cdots \boldsymbol{g}_p \end{bmatrix} \in \mathbb{R}^{n \times p}$, the semantics of Z are defined as

$$[\![Z]\!] = \{G\boldsymbol{x} + \boldsymbol{c} \,:\, -\mathbf{1}_p \leq \boldsymbol{x} \leq \mathbf{1}_p\}.$$

Linear star sets generalize zonotopes by letting the kernel be defined by an LCS. Formally, an n-dimensional star set $S = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ is the image of a p-dimensional polytope $A\boldsymbol{x} \leq b$ under an affine transformation $\mathbb{R}^p \to \mathbb{R}^n$. Given a center $\boldsymbol{c} \in \mathbb{R}^n$ and a set of generator vectors $\{\boldsymbol{g}_1, \ldots, \boldsymbol{g}_p \in \mathbb{R}^n\}$ that form a matrix $G = \begin{bmatrix} \boldsymbol{g}_1 \cdots \boldsymbol{g}_p \end{bmatrix} \in \mathbb{R}^{n \times p}$, the semantics of S are defined as

$$[\![S]\!] = \{G\boldsymbol{x} + \boldsymbol{c} \,:\, A\boldsymbol{x} \leq \boldsymbol{b}\}.$$

The following theorems [35] provide the foundational results on linear star sets that are leveraged in neural network verification.

Theorem 1 (Affine Transformations of Linear Star Sets). *The linear star sets are closed under affine transformation, i.e., given a linear star set* $S = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ *and an affine map* $f(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{d}$ *on* $[\![S]\!]$*, the image* $S_{[W,\boldsymbol{d}]} = \{f(\boldsymbol{x}) : \boldsymbol{x} \in [\![S]\!]\}$ *is equal to* $[\![S']\!]$ *for a linear star set* $S' = \langle \boldsymbol{c'}, G', A, \boldsymbol{b} \rangle$ *where*

$$\boldsymbol{c'} = W\boldsymbol{c} + \boldsymbol{d} \text{ and } G' = \begin{bmatrix} W\boldsymbol{g}_1 \cdots W\boldsymbol{g}_p \end{bmatrix}.$$

Theorem 2 (Linear Optimization Over Linear Star Sets). *The optimization of a linear function* f *over a linear star set* S *reduces to linear programming.*

Theorem 3 (Intersection of Linear Star Sets and Half-Spaces). *The intersection of a star set* $S = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ *and half space* $\{\boldsymbol{y} \mid H\boldsymbol{y} \leq \boldsymbol{h}\}$ *is another star set* $S' = \langle \boldsymbol{c}, G, A', \boldsymbol{b'} \rangle$ *where* $A'\boldsymbol{x} \leq \boldsymbol{b'}$ *are the conjunction of constraints*

$$A\boldsymbol{x} \leq \boldsymbol{b} \text{ and } HG\boldsymbol{x} \leq \boldsymbol{h} - H\boldsymbol{c}.$$

Next, we extend the notion of zonotopes to define octatopes and hexatopes and develop a series of results, analogous to Theorem 1 to 3, that provide the theoretical framework for the application of these abstract domains to the verification of neural networks.

3.2 Octatopes and Hexatopes

Definition 6 (Octatopes and Hexatopes). *An octatope is an* n*-dimensional star set* $\langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ *where the kernel constraints* $A\boldsymbol{x} \leq \boldsymbol{b}$ *form a UCS. A hexatope is similarly defined as an* n*-dimensional star set* $\langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ *where the kernel constraints* $A\boldsymbol{x} \leq \boldsymbol{b}$ *form a DCS.*

Our first result mirrors Theorem 1 and establishes closure under affine mappings for octatopes and hexatopes.

Theorem 4. *Octatopes and Hexatopes are closed under affine transformation.*

Proof. From Theorem 1 it follows for an octatope (hexatope) $S = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ and an affine mapping $f(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{d}$, that $S_{[W,d]} = \{W\boldsymbol{x} + \boldsymbol{d} : \boldsymbol{x} \in [\![S]\!]\}$ is a star set $S' = \langle \boldsymbol{c'}, G', A, \boldsymbol{b} \rangle$ where

$$\boldsymbol{c'} = W\boldsymbol{c} + \boldsymbol{d} \text{ and } G' = \begin{bmatrix} W\boldsymbol{g}_1 \cdots W\boldsymbol{g}_p \end{bmatrix}.$$

Since this transformation does not change the kernel, the resulting set remains an octatope (hexatope). □

3.3 Linear Optimization Over Octatopes and Hexatopes

By Theorem 2, linear optimization over linear star sets can be done in polynomial time. Our next result shows that linear optimization over octatopes and hexatopes can be done in *strongly* polynomial time.

Theorem 5. *The linear optimization problem for octatopes and hexatopes can be solved in strongly polynomial time via a reduction to the MCF problem.*

Proof. We reduce the optimization problem for octatopes to a similar problem for hexatopes. Consider an n-dimensional octatope $O = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ which is the image of a p-dimensional UCS-defined set. Here $\boldsymbol{c} \in \mathbb{R}^n$ is the center and vectors $\{\boldsymbol{g}_1, \boldsymbol{g}_2, \ldots, \boldsymbol{g}_p \in \mathbb{R}^n\}$ are the generators. In order to optimize a linear function f over $[\![O]\!]$, it suffices to optimize the composition of functions $\boldsymbol{x} \mapsto G\boldsymbol{x} + \boldsymbol{c}$ and f over the UTVPI constrained set $A\boldsymbol{x} \leq \boldsymbol{b}$. We describe a method to find the linear optimum of an arbitrary linear objective function over a UCS.

Let $\mathbf{U}$ be a UCS and let f be an objective function we are maximizing. First we convert [24] the UCS $\mathbf{U}$ into a DCS $\mathbf{D}$. The first part of the conversion creates the variables x_i^+ and x_i^- in $\mathbf{D}$ for each variable x_i in $\mathbf{U}$. Then, each constraint in $\mathbf{U}$ is converted as follows:

1. Each constraint of the form $x_i + x_j \leq b_{ij}$ becomes
$$x_i^+ - x_j^- \leq b_{ij} \text{ and } -x_i^- + x_j^+ \leq b_{ij}.$$
2. Each constraint of the form $x_i - x_j \leq b_{ij}$ becomes
$$x_i^+ - x_j^+ \leq b_{ij} \text{ and } -x_i^- + x_j^- \leq b_{ij}.$$
3. Each constraint of the form $-x_i + x_j \leq b_{ij}$ becomes
$$x_i^- - x_j^- \leq b_{ij} \text{ and } -x_i^+ + x_j^+ \leq b_{ij}.$$
4. Each constraint of the form $-x_i - x_j \leq b_{ij}$ becomes
$$x_i^- - x_j^+ \leq b_{ij} \text{ and } -x_i^+ + x_j^- \leq b_{ij}.$$
5. Each constraint of the form $x_i \leq b_i$ becomes
$$x_i^+ - x_i^- \leq 2 \cdot b_i.$$
6. Each constraint of the form $-x_i \leq b_i$ becomes
$$x_i^- - x_i^+ \leq 2 \cdot b_i.$$

Observe that $x_i = \frac{1}{2}(x_i^+ - x_i^-)$ satisfies the original UCS. Thus, we can consider this as the problem maximizing the objective function over variables $\frac{1}{2}(x_i^+ - x_i^-)$ of the DCS $\mathbf{D}$. Note that the problem of maximizing a linear objective function over a DCS is the dual of a minimum cost flow problem. Since the Out-of-Kilter algorithm also solves the dual to the minimum cost flow problem [1], running the Out-of-Kilter algorithm on the dual of the DCS optimization problem will also solve the DCS optimization problem. For a UCS with m constraints, this process takes $O((n^2 + m \cdot n \cdot \log m) \cdot C)$ time where C is the largest absolute value of any coefficient in the objective function. □

Algorithm 2. UTVPIBoundingBox($\mathbf{U}, l$)

Input: UCS $\mathbf{U}$ and constraint l
Output: A UTVPI bounding box $\mathbf{U}'$

1: $\mathbf{U}' \leftarrow \emptyset$
2: **for all** pairs of variables x_i, x_j in $\mathbf{U}$ **do**
3: Let $u_{ij}^{+-} = \max_{\mathbf{U}\cup\{l\}} x_i - x_j$ and add constraint $x_i - x_j \leq u_{ij}^{+-}$ to $\mathbf{U}'$
4: Let $u_{ij}^{-+} = \max_{\mathbf{U}\cup\{l\}} x_j - x_i$ and add constraint $x_j - x_i \leq u_{ij}^{-+}$ to $\mathbf{U}'$
5: Let $u_{ij}^{++} = \max_{\mathbf{U}\cup\{l\}} x_i + x_j$ and add constraint $x_i + x_j \leq u_{ij}^{++}$ to $\mathbf{U}'$
6: Let $u_{ij}^{--} = \max_{\mathbf{U}\cup\{l\}} -x_i - x_j$ and add $-x_i - x_j \leq u_{ij}^{--}$ to $\mathbf{U}'$
7: Let $u_i^{+} = \max_{\mathbf{U}\cup\{l\}} x_i$ and add constraint $x_i \leq u_i^{+}$ to $\mathbf{U}'$
8: Let $u_i^{-} = \max_{\mathbf{U}\cup\{l\}} -x_i$ and add constraint $-x_i \leq u_i^{-}$ to $\mathbf{U}'$
9: **return** $\mathbf{U}'$.

Emptiness Checking. We also consider the feasibility problem for octatopes. That is, the problem of deciding whether an octatope is empty. The emptiness of an octatope can be decided in $O(n \cdot m)$ time and $O(n+m)$ space where n is the number of generator variables and m is the number of generator constraints. It is easy to see that an octatope (hexatope) is empty if and only if the UTVPI constraints of its kernel are unsatisfiable as linear mappings over polytopes that are monotone with respect to set inclusion. The complexity then follows from results on checking the feasibility of UTVPI constraint systems [24].

3.4 Intersection of Octatopes/Hexatopes and Half-Spaces

It follows from Theorem 3 that the intersection of an octatope $O = \langle \boldsymbol{c}, G, A, \boldsymbol{b} \rangle$ and half space $\{\boldsymbol{y} \mid H\boldsymbol{y} \leq \boldsymbol{h}\}$ is a star set $O' = \langle \boldsymbol{c}, G, A', \boldsymbol{b'} \rangle$ where the constraints $A'\boldsymbol{x} \leq \boldsymbol{b'}$ are the conjunction of UCS constraints $A\boldsymbol{x} \leq \boldsymbol{b}$ and the hyperplane $HG\boldsymbol{x} \leq \boldsymbol{h} - H\boldsymbol{c}$. In the rest of this section, we show how an over-approximation of this intersection can be represented as UCS constraints. The treatment for hexatopes is similar, and hence omitted.

We formalize this problem as the UTVPI *bounding box problem*. Given a UCS $\mathbf{U}$ and an arbitrary linear constraint l, a UTVPI *bounding box* is a UCS $\mathbf{U}'$, such that every solution to $\mathbf{U} \cup \{l\}$ is a solution to $\mathbf{U}'$. For a given UCS $\mathbf{U}$ and constraint l, a *tightest* UTVPI bounding box is a bounding box of $\mathbf{U} \cup \{l\}$ that is contained within every other bounding box of $\mathbf{U} \cup \{l\}$. Thus, a UTVPI bounding box of a UCS $\mathbf{U}$ and constraint l is a UCS that overestimates the solution space of $\mathbf{U} \cup \{l\}$. A tightest bounding box is a UCS that overestimates the solution space the least. Each of the linear programs used to construct $\mathbf{U}'$ can be solved (with L bits of precision) in $O(n^{2.38} \cdot L)$ time [10]. Since finding the UTVPI bounding box requires solving $O(n^2)$ linear programs, the UTVPI bounding box can be found in $O(n^{4.38} \cdot L)$ time.

Theorem 6. *Let* $\mathbf{U}$ *be a UCS and let* l *be an arbitrary linear constraint. The UCS* $\mathbf{U}'$*, constructed by Algorithm 2, is a* UTVPI *bounding box of* $\mathbf{U} \cup \{l\}$.

Proof. Let $\mathbf{x}^*$ be a solution to $\mathbf{U} \cup \{l\}$. Let $a_i \cdot x_i + a_j \cdot x_j \leq u_{ij}$ be an arbitrary constraint in $\mathbf{U}'$. By construction of $\mathbf{U}'$, we have $u_{ij} = \max_{\mathbf{U} \cup \{l\}} a_i \cdot x_i + a_j \cdot x_j$.

Since $\mathbf{x}^*$ is a solution of $\mathbf{U} \cup \{l\}$, $a_i \cdot x_i^* + a_j \cdot x_j^* \leq u_{ij}$. This means that $\mathbf{x}^*$ satisfies the constraint $a_i \cdot x_i + a_j \cdot x_j \leq u_{ij}$. Since the constraint $a_i \cdot x_i + a_j \cdot x_j \leq u_{ij}$ was chosen arbitrarily, $\mathbf{x}^*$ is a solution to $\mathbf{U}'$. Note that $\mathbf{x}^*$ was an arbitrary solution to $\mathbf{U} \cup \{l\}$. Thus, every solution to $\mathbf{U} \cup \{l\}$ is a solution to $\mathbf{U}'$. Consequently, $\mathbf{U}'$ is a UTVPI bounding box of $\mathbf{U} \cup \{l\}$. □

We now show that $\mathbf{U}'$ is a tightest UTVPI bounding box of $\mathbf{U} \cup \{l\}$. Note that $\mathbf{U} \cup \{l\}$ must have a tightest bounding box. Consider two bounding boxes $\mathbf{U}_1$ and $\mathbf{U}_2$ of $\mathbf{U} \cup \{l\}$. Let $\mathbf{U}^*$, be the UCS formed by combining the constraints in $\mathbf{U}_1$ and $\mathbf{U}_2$. Note that $\mathbf{U}^*$ is also a bounding box of $\mathbf{U} \cup \{l\}$. Additionally, every solution to $\mathbf{U}^*$ is a solution to both $\mathbf{U}_1$ and $\mathbf{U}_2$. Thus, if $\mathbf{U} \cup \{l\}$ has two incomparable bounding boxes, then a new bounding box can be constructed that is tighter than both.

Theorem 7. *Let* $\mathbf{U}$ *be a UCS and let* l *be a linear constraint. The UCS* $\mathbf{U}'$*, produced by Algorithm 2, is a tightest* UTVPI *bounding box of* $\mathbf{U} \cup \{l\}$.

Proof. Assume for the sake of contradiction, that $\mathbf{U}'$ is not a tightest UTVPI bounding box of $\mathbf{U} \cup \{l\}$. Thus, there exist a UTVPI bounding box $\mathbf{U}''$ and a point $\mathbf{x}^*$ such that $\mathbf{x}^*$ is a solution to $\mathbf{U}'$, but not a solution to $\mathbf{U}''$. This means that there is a UTVPI constraint $a_i \cdot x_i + a_j \cdot x_j \leq b$ in $\mathbf{U}''$ that is violated by $\mathbf{x}^*$.

Let $u_{ij} = \max_{\mathbf{U} \cup \{l\}} a_i \cdot x_i + a_j \cdot x_j$. Since $\mathbf{U}''$ is a UTVPI bounding box of $\mathbf{U} \cup \{l\}$, every solution to $\mathbf{U} \cup \{l\}$ is a solution to $\mathbf{U}''$. Thus, every solution to $\mathbf{U} \cup \{l\}$ satisfies the constraint $a_i \cdot x_i + a_j \cdot x_j \leq b$. This means that

$$\max_{\mathbf{U} \cup \{l\}} a_i \cdot x_i + a_j \cdot x_j$$

is bounded from above by b. Thus, u_{ij} exists and $u_{ij} \leq b$.

By the construction of $\mathbf{U}'$, the constraint $a_i \cdot x_i + a_j \cdot x_j \leq u_{ij}$ is in $\mathbf{U}'$. However, $\mathbf{x}^*$ is a solution to $\mathbf{U}'$ such that $a_i \cdot x_i^* + a_j \cdot x_j^* > b \geq u_{ij}$. This is a contradiction. Thus, $\mathbf{U}'$ must be a tightest UTVPI bounding box of $\mathbf{U} \cup \{l\}$. □

4 Range Computation for Neural Nets with Prefilters

The exact range computation problem from Definition 4 can be solved using linear star sets (see Algorithms 1 and 2 in earlier work for a full review [7]).

The neural network function f as defined in Sect. 2.3 is a piece-wise affine function of the inputs. The range computation proceeds using geometric set operations. The initial set of states is represented as a linear star set and propagated through each layer of the network. To go from the output of one layer to the vector of intermediate values at the next layer, an affine transformation operation is performed on the set. The effect of the ReLU activation in a layer is handled iteratively for each neuron. The set of states is potentially split along the neuron input constraint $y_i = 0$, into a negative region and a positive region,

using a half-space intersection operation. The negative region is then projected to zero to match the semantics of a ReLU. The two sets are then considered independently for the remaining neurons in the layer, as well as the rest of the layers in the network. For a given input set, not all neurons require splitting the set in two, since the input constraints may restrict inputs to be strictly positive or negative. To check this, before splitting we first optimize over the set in the direction of the intermediate value $x_j^{(i)}$ corresponding to a specific neuron j in layer i. If splitting occurs, the two sets are treated independently and propagated through the remaining neurons in the layer, possibly requiring further splitting in the remaining parts of the network.

After applying a number of optimizations, the bottleneck of exact range computation with star sets is the use of LP solving to compute the input bounds for each neuron [7]. To improve analysis speed, rather than speeding up LP solving—which is a well-studied problem where further progress is likely to be difficult—we instead seek methods that can reduce the number of LPs needed.

In earlier work, zonotope abstract domains have been considered for this task. Rather than just propagating star sets through a network, we also propagate a zonotope overapproximation that we use in a *prefiltering* step. Recall that before splitting we first need to optimize over the set in the direction of the intermediate value $x_j^{(i)}$. Before optimizing over the star set using LP, we first optimize over the zonotope abstraction prefilter. If the zonotope abstraction can prove that the inputs are strictly positive or negative, than we are guaranteed the exact result from the LP will be strictly positive or negative as well (as the zonotope is an overapproximation of the star set). This allows us to avoid LP, as optimization over zonotopes can be done efficiently using a simple loop.

The reason zonotope analysis is not exact is that zonotopes do not support general half-space intersections when sets must be split. Instead, two approaches have been considered. The easiest option is to ignore intersections, which is fast but can cause significant overapproximation error in the abstraction [15,36]. Alternatively, we can perform *domain contraction*, which is to search for zonotopes that more tightly overapproximate the intersection. Different approaches for domain contraction are possible, ranging from reasoning methods over individual constraints to more accurate approaches that use LP solving on the star set in the generator coefficient space [7]. Although the LP approach uses the expensive operation we are trying to reduce, it can result in an overall reduction of LPs, as the neuron input bounds can be computed more accurately.

This work proposes using octatope abstract domains as a prefilter. As described earlier, optimization over octatopes can be done more efficiently than general LP solving. The greater expressiveness of octatopes compared with zonotopes means that we can hope to further reduce the number of LPs needed with the star set when computing a neuron's input bounds for splitting. We evaluate this impact in our experiments. In terms of handling intersections when splitting sets, octatopes (like zonotopes) cannot exactly support any general half-space intersection operation. This means that a domain contraction step may be necessary to ensure tight overapproximation.

5 Experimental Results

We next evaluate the potential savings in LP computation to computing neuron input bounds during exact range computation for neural networks. Our evaluation is performed on several benchmarks from the ACAS Xu benchmark suite [20], specifically focusing on property 3 and 4 where earlier work has shown exact range computation is tractable [7]. We generally report number of LPs for different operations rather than runtime, as the runtime is influenced by other factors such code optimizations and the choice of LP solver.

First, we examine the number of LPs needed to perform neuron input range computation, for different choices of prefilter abstract domain. The LP calls to find the neuron input ranges is the bottleneck of the overall range computation algorithm, so its reduction is of particular importance. The results are in Table 1. The Star-Only approach uses only LP solving with no prefiler, and therefore has the highest number of LPs. The next column, Zonotope-NC corresponds to the case where zonotope prefilters are used, but no domain contraction is performed (halfspace intersections are ignored). This has a significant reduction on the number of LP calls, for example in the first row with property 3 and network 1–6, where the number of LP calls is reduced from 91 K to 11 K. Using domain contraction with zonotopes, Zono-C, further reduces this to around 3.3 K. The more precise domains with hexatopes and octatopes can further reduce this to around 2.6 K and 2.5 K, respectively. The minimum column is computed by seeing how many bounds computations could not be eliminated as they correspond to cases where the input to a neuron truly can be either positive or negative. Even a perfect prefilter could not eliminate these LPs, as prefilters only eliminate cases where splitting is impossible. Other approaches could be considered to remove these LPs, such as tracking specific witness input points that can prove a neuron can have both positive and negative inputs, which we may consider in future work. Overall, the proposed octatope abstract domain has the potential to reduce the number of unnecessary LPs significantly in exact range computation.

When using the new abstract domain, however, there is a trade-off where extra operations are needed to perform domain contraction as well as to optimize within the abstract domains. We used a witness-tracking approach [5], where for each constraint a witness point was included that was in the star set and on the boundary of the constraint. When new intersections are performed, each witness point is checked to see if it is now excluded from the set. When points are excluded, new witness points get generated by solving an LP in the direction of the constraint, which may tighten the constraint. This results in the tight abstract domains, but can be expensive when many constraints are possible. For hexatopes and octatopes, the number of possible constraints is quadratic in the number of variables (ACAS Xu has 5 input variables).

Table 2 shows the number of LPs needed for each example when performing domain contraction. Star-Only and Zono-NC do not perform domain contraction, and so have 0 LPs for this operation. As expected, the more complex the abstract domain, the more operations are needed. This is due to the contraction method

Table 1. Number of LP calls to find neuron input bounds for different abstract domain prefilters on various ACASXu properties and networks.

Prop	Net	Star-Only	Zono-NC	Zono-C	Hex	Oct	Minimum
3	1–6	91762	11152	3382	2635	2571	1886
3	2–7	77896	9365	2921	2240	2198	1626
3	3–5	80988	8990	2711	2131	2092	1710
3	5–2	54758	15523	7762	6820	6704	3779
4	1–4	53036	7736	2597	2389	2330	1926
4	2–7	38748	3851	1249	888	861	753
4	5–9	68750	8814	2952	2286	2151	1591

Table 2. Number of LP calls for the domain contraction step for different abstract domain prefilters for various ACAS Xu properties and networks.

Prop	Net	Star-Only	Zono-NC	Zono-C	Hex	Oct
3	1–6	0	0	12765	38400	115200
3	2–7	0	0	12280	36840	110520
3	3–5	0	0	10407	31230	93690
3	5–2	0	0	21249	63750	191250
4	1–4	0	0	11828	35493	106476
4	2–7	0	0	5533	16620	49860
4	5–9	0	0	9906	29730	89190

performed, where the number of possible LPs needed at a domain contraction step increases as the number of possible constraints increases.

In terms of the performance of network simplex for optimizing within the octatope domain, the engineering aspect of the problem also requires further development. When computing the range of network 2–7 with the input set from property 4, the UTVPI constraints were optimized 38748 times. When using the commercial LP solver Gurobi on these constraints, each call took on average of 0.17 ms. Formulating the min-cost flow problem and calling the `network_simplex` implementation from the `networkx` python library, however, used about 1.9 ms per call, about 11x slower. Further, while Gurobi always obtained a result, numerical issues caused network simplex to fail about 0.65% of the time.

In summary, while octatopes effectively reduce the bottleneck step of input bounds computation, further improvements must be made to octatope domain contraction algorithms as well as to implementation optimizations of min-cost flow solvers, before an overall speedup can be achieved. Nonetheless, it is an encouraging result for DNN verification as developing more efficient domain contraction algorithms and improving min-cost flow implementations is likely easier than coming up with new ways to speed up LP solving.

6 Conclusion

The advent of deep neural networks and their inevitable widespread adoption necessitates tools by which we can reason about their robustness. The verification community has made great strides on this front in recent years through the development of neural network verification solutions based on search, optimization, and reachability. While search and optimization can often be used to yield sound and complete solutions, such techniques pay the cost of scalability. Methods based on reachability analysis, on the other, can often scale better at the cost of completeness. These methods typically employ an abstract domain representation of the input-output behavior of nodes in the neural network for a given set of inputs. These abstract domains range from zonotopes to star sets that differ in their trade-off between scalability and precision.

We proposed octatopes as a new abstract domain which corresponds to affine transformations of unit two-variable per inequality (UTVPI) constraints. Octatopes provide tighter abstractions than zonotopes while optimization can be formulated as a min-cost flow problem that is theoretically more efficient than linear programming. Our experiments using octatope abstract domains for exact range computation of neural networks confirmed their accuracy, as we were able to reduce the bottleneck step of using LP to compute each neuron's input bounds. However, engineering improvements must still be made to min-cost flow libraries. In our application of UTVPI optimization, it was faster to use the highly-optimized commercial LP solver Gurobi instead of the theoretically faster min-cost flow formulation. In future work, we plan to examine ways to improve domain contraction, as well as investigating other application areas of octatopes such as neural network verification with over-approximations, software analysis, and hybrid systems reachability.

Acknowledgment. This material is based upon work supported by the Air Force Office of Scientific Research and the Office of Naval Research under award numbers FA9550-19-1-0288, FA9550-21-1-0121, FA9550-22-1-0450, FA9550-22-1-0029 and N00014-22-1-2156. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or the United States Navy. This research was supported in part by the Air Force Research Laboratory Information Directorate, through the Air Force Office of Scientific Research Summer Faculty Fellowship Program, Contract Numbers FA8750-15-3-6003, FA9550-15-0001 and FA9550-20-F-0005.

This work is also supported by the National Science Foundation (NSF) grant CCF-2009022 and by NSF CAREER award CCF-2146563.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms and Applications. Prentice Hall (1993)
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993)

3. Akintunde, M., Lomuscio, A., Maganti, L., Pirovano, E.: Reachability analysis for neural agent-environment systems. In: Sixteenth International Conference on Principles of Knowledge Representation and Reasoning (2018)
4. Aws Albarghouthi: Introduction to Neural Network Verification (2021). http://verifieddeeplearning.com
5. Bak, S.: nnenum: verification of ReLU neural networks with optimized abstraction refinement. In: Dutle, A., Moscato, M.M., Titolo, L., Muñoz, C.A., Perez, I. (eds.) NFM 2021. LNCS, vol. 12673, pp. 19–36. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-76384-8_2
6. Bak, S., Liu, C., Johnson, T.: The second international verification of neural networks competition (VNN-COMP 2021): summary and results. arXiv preprint arXiv:2109.00498 (2021)
7. Bak, S., Tran, H.-D., Hobbs, K., Johnson, T.T.: Improved geometric path enumeration for verifying ReLU neural networks. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 66–96. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_4
8. Bazaraa, M.S., Jarvis, J.J., Sherali, H.D.: Linear Programming and Network Flows. Wiley, Hoboken (2008)
9. Behrmann, G., et al.: UPPAAL 4.0. In: Third International Conference on the Quantitative Evaluation of Systems (QEST 2006), 11–14 September 2006, Riverside, California, USA, pp. 125–126. IEEE Computer Society (2006)
10. Cohen, M.B., Lee, Y.T., Song, Z.: Solving linear programs in the current matrix multiplication time. J. ACM **68**(1), 1–39 (2021)
11. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, pp. 238–252 (1977)
12. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
13. Duggirala, P.S., Viswanathan, M.: Parsimonious, simulation based verification of linear systems. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 477–494. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_26
14. Friedmann, O., Hansen, T.D., Zwick, U.: Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In: Symposium on Theory of Computing (STOC 2011), pp. 283–292, ACM, New York (2011)
15. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: AI2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 3–18. IEEE (2018)
16. Ghorbal, K., Goubault, E., Putot, S.: The zonotope abstract domain Taylor1+. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 627–633. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_47
17. Goldberg, A.V., Tarjan, R.E.: Finding minimum-cost circulations by canceling negative cycles. J. ACM **36**(4), 873–886 (1989)
18. Henriksen, P., Lomuscio, A.: Efficient neural network verification via adaptive refinement and adversarial search. In: ECAI 2020, pp. 2513–2520. IOS Press (2020)
19. Henriksen, P., Lomuscio, A.: DEEPSPLIT: an efficient splitting method for neural network verification via indirect effect analysis. In: Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI21) (2021). To appear

20. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
21. Katz, G., et al.: The Marabou framework for verification and analysis of deep neural networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 443–452. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_26
22. Khachiyan, L.G.: A polynomial time algorithm for linear programming. Dokl. Akad. Nauk SSSR **244**(5), 1093–1096 (1979). English translation in Soviet Math. Dokl. 20, 191–194
23. Klee, F., Minty, G.J.: How good is the simplex algorithm? Inequalities **III**, 159–175 (1972)
24. Lahiri, S.K., Musuvathi, M.: An efficient decision procedure for UTVPI constraints. In: Gramlich, B. (ed.) FroCoS 2005. LNCS (LNAI), vol. 3717, pp. 168–183. Springer, Heidelberg (2005). https://doi.org/10.1007/11559306_9
25. Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M.J.: Algorithms for verifying deep neural networks. Found. Trends Optim. **4**(3–4), 244–404 (2021)
26. Manzanas Lopez, D., Johnson, T., Tran, H.D., Bak, S., Chen, X., Hobbs, K.L.: Verification of neural network compression of ACAS Xu lookup tables with star set reachability. In: AIAA Scitech 2021 Forum, p. 0995 (2021)
27. Miné, A.: The octagon abstract domain. High.-Order Symb. Comput. **19**(1), 31–100 (2006)
28. Orlin, J.B.: A polynomial time primal network simplex algorithm for minimum cost flows. Math. Program. **78**, 109–129 (1997). https://doi.org/10.1007/BF02614365
29. Singh, G., Gehr, T., Mirman, M., Püschel, M., Vechev, M.: Fast and effective robustness certification. NeurIPS **1**(4), 6 (2018)
30. Singh, G., Gehr, T., Püschel, M., Vechev, M.: An abstract domain for certifying neural networks. Proc. ACM Program. Lang. **3**(POPL), 1–30 (2019)
31. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, 2nd edn. MIT Press (2018)
32. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: International Conference on Learning Representations (2018)
33. Tran, H.-D., Bak, S., Xiang, W., Johnson, T.T.: Verification of deep convolutional neural networks using ImageStars. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 18–42. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_2
34. Tran, H.D., Cai, F., Diego, M.L., Musau, P., Johnson, T.T., Koutsoukos, X.: Safety verification of cyber-physical systems with reinforcement learning control. ACM Trans. Embed. Comput. Syst. **18**(5s), 1–22 (2019)
35. Tran, H.-D., et al.: Star-based reachability analysis of deep neural networks. In: ter Beek, M.H., McIver, A., Oliveira, J.N. (eds.) FM 2019. LNCS, vol. 11800, pp. 670–686. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30942-8_39
36. Tran, H.-D., et al.: Robustness verification of semantic segmentation neural networks using relaxed reachability. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12759, pp. 263–286. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81685-8_12
37. Tran, H.-D., et al.: NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. In: Lahiri, S.K., Wang, C.

(eds.) CAV 2020. LNCS, vol. 12224, pp. 3–17. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_1
38. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Efficient formal safety analysis of neural networks. Adv. Neural Inf. Process. Syst. **31** (2018)
39. Wang, S., et al.: Beta-crown: efficient bound propagation with per-neuron split constraints for neural network robustness verification. Adv. Neural. Inf. Process. Syst. **34**, 29909–29921 (2021)