Development and Application of Global Optimization Methods

PhD Thesis

Dániel Zombori

Supervisor:
Dr. Balázs Bánhelyi

Doctoral School of Computer Science
Department of Computational Optimization
Faculty of Science and Informatics
University of Szeged

Szeged
2025

1 Introduction

Optimization and verification are closely related research fields that help in finding and validating desired properties in a system. These ubiquitous tools are being utilized in some way by practically every profession and research field.

In the field of optimization black-box problems are the most generic, where properties of the optimization problem are unknown. When solving a black-box problem, the algorithm cannot know whether optimality is reached. To acquire a good candidate for an optimal solution, we aim to invest the available best effort. This requires the careful balancing of algorithm efficiency and performance. While an efficient algorithm wastes minimal resources during execution, a performant algorithm tries to utilize all available resources with only a secondary focus on minimizing wasted effort.

In the field of verification reliability is key, besides the secondary goals of efficiency and performance. If algorithm answers are unreliable for validating the target properties or infeasible to obtain, verification becomes impossible.

The PhD thesis presents advances in both topics. New versions of the `GLOBAL` optimizer are presented, increasing the algorithm performance in multi-threaded environments. The results open the possibility to use `GLOBAL` at different scales of optimization problems and computing resources. In neural network verification serious flaws are uncovered that showcase problems in reliable MILP model based verification. Based on an exploit a practical attack demonstrates the ease of fooling state-of-the-art verifiers. Further problems are uncovered regarding the naive application of interval arithmetic. Limitations are found in computing the reachable output space of floating point operations. Possible ways of strengthening MILP and interval arithmetic based neural network verification is discussed.

2 Parallel global optimization

Unsurprisingly, those multi-thread algorithms perform the best, which are originally designed to utilize a parallel execution environment. Developing and reasoning about these algorithms can be much harder than their linearly executed counterparts. A less ideal, but still valid approach is the development and research of single-thread algorithms, where results can be extended to a multi-thread implementation.

The obstacles in development of a multi-threaded optimizer range from simple correctness issues where we try to avoid deadlocks, to differences in performance that are introduced by subtle design choices. There are established strategies to parallelize an algorithm, for example *Geometric Parallelism* or *Task Parallelism*.

Creating a multi-threaded version of the Global optimizer has all the usual benefits in increasing the algorithm performance. With larger available computing capacity new problems can be solved, and the currently typical computing environment is better utilized.

Depending on the objective function characteristics, - like the required computing capacity and the amount of evaluations required for finding good candidates for the global optimum - different parallelization strategies can be necessary. A low-cost function with a large number of required evaluations will encounter different performance bottlenecks compared to a high-cost function with low number of required evaluations.

Chapter 2. of the PhD thesis presents two largely different approaches for a multi-thread Global implementation, *SynchronizedGlobal* and *ParallelGlobal*. A third approach, *DistributedGlobal* is discussed, that is a more general version of *ParallelGlobal*.

2.1 SynchronizedGlobal

SynchronizedGlobal is a multi-threaded implementation of Global that is designed to balance performance gain and algorithmic efficiency. The single-thread Global algorithm can maximize efficiency, SynchronizedGlobal closely resembles it to benefit from the efficient use of available information. SynchronizedGlobal is based on low inter-thread latency and short critical sections to avoid blocking both reading and writing actions on shared containers. A typical computing environment with these characteristics is the modern computer with multi-threaded CPUs, and high-performance computing servers. The low cost of inter-thread communication is imperative, as idle threads reduce the algorithm performance.

To efficiently utilize the available threads, SynchronizedGlobal implements a task-queue based system. Worker threads have a priority for each task type. They check each type in priority order, to find the work with the highest priority. This task distribution is based on the “pull principle”, where each worker finds the most important work to do, resulting in the most value for performing the overall task.

SynchronizedGlobal deconstructs a Global iteration into five tasks, most of which can be independently executed. Tasks are prioritized in reverse order, a worker thread evaluates the task guards one-by-one to find an available unit of work. When a unit of work is done, the worker thread restarts the search from the highest priority.

Clustering is a special task that can be paused and resumed. The latest version of the Global algorithm alternates the clustering and local search processes to reduce the number of local searches as much as possible. SynchronizedGlobal replicates this task alternation by pausing the clustering process if clustering failed

for all samples and resuming it if new cluster data is available.

Workers are free to enter any task at any point, given that the guard condition is satisfied. While a long running local search is still in progress, other workers can start sampling, clustering, or even a new round of local searches.

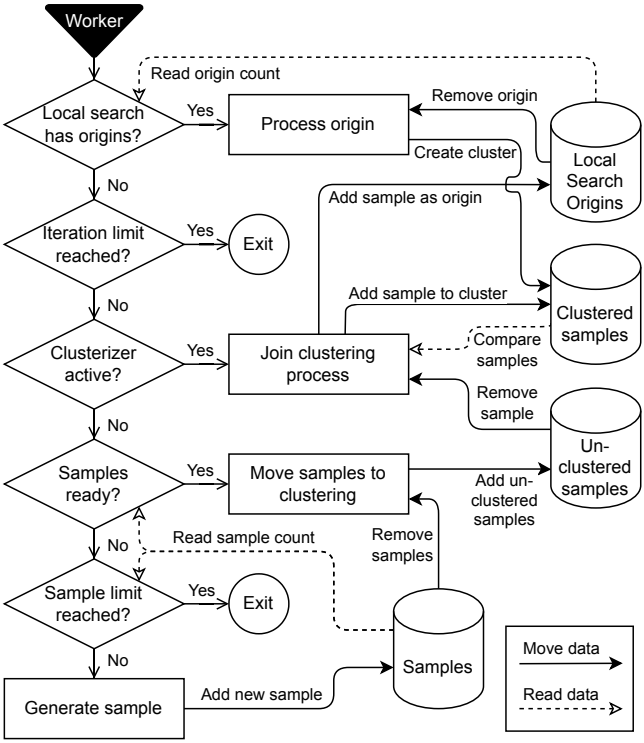

Figure 1: *SynchronizedGlobal worker algorithm*

The implementation of SynchronizedGlobal was studied from the aspect of speedup in finding a known global optimum and from the aspect of similarity to Global in a single worker configuration. A test was also conducted on performance of the multi-threaded clustering solution.

Figure 2 shows that Global and SynchronizedGlobal are very similar in terms of the number of function evaluations needed in a single-thread configuration.

Table 1 shows the effects of different number of worker threads, different objective functions, and different evaluation cost of the objective function (Hardness). The findings include a large dependence of NFEV on the objective function

combined with the number of worker threads. Depending on the hardness a decreased runtime with additional worker threads, and saturation effects where additional threads only increase runtime after a point.

Clustering results show similar effects, where the parallel clustering algorithm is tested using increasing amounts of cluster data and increasing amounts of worker threads.

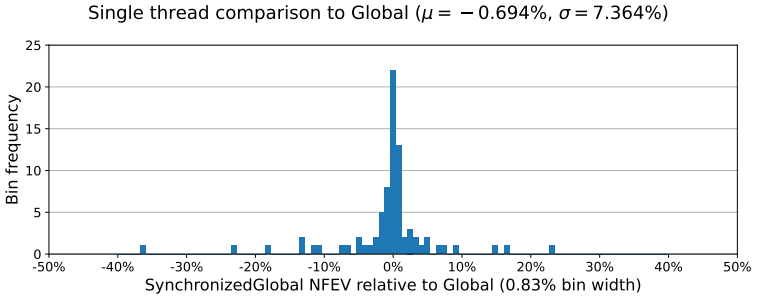

Figure 2: *Histogram on the number of function evaluations (NFEV) needed by SynchronizedGlobal relative to Global, in a single thread configuration.*

Hardness	Threads	Ackley		Easom		Levy 3	
		NFEV	Runtime	NFEV	Runtime	NFEV	Runtime
1x	1	100,447	3,553.7	10,120.7	122.7	101,742	3,245.8
	2	101,544	2,216.0	10,246.4	118.4	104,827	2,062.1
	4	102,881	1,515.3	10,506.2	112.0	112,351	1,473.0
	8	102,908	1,145.9	11,078.7	145.0	129,056	1,218.9
	16	110,010	1,319.3	12,335.9	149.0	156,907	1,548.4
10x	1	100,553	5,838.5	10,141.9	165.0	102,412	5,414.6
	2	101,510	3,370.6	10,273.7	132.0	106,339	3,057.4
	4	103,495	2,014.7	10,524.9	111.6	114,325	2,100.8
	8	105,977	1,480.0	11,096.6	135.7	126,848	1,592.3
	16	112,008	1,623.1	12,308.2	157.6	155,884	1,714.4
100x	1	100,516	27,868.7	10,117.7	413.2	102,227	25,364.9
	2	101,585	14,544.5	10,256.9	352.5	106,423	13,788.8
	4	103,420	7,806.3	10,546.1	323.9	115,158	8,030.5
	8	107,657	4,544.7	11,083.6	296.3	130,109	5,368.2
	16	115,264	3,648.3	12,313.7	257.8	167,983	5,205.0
1000x	1	100,567	258,690.0	10,198.5	1,722.4	102,028	236,066.5
	2	101,561	126,315.0	10,249.7	865.1	106,792	123,220.0
	4	103,616	68,691.5	10,521.7	508.7	114,847	70,399.6
	8	107,718	39,430.4	11,116.0	441.6	128,450	45,762.8
	16	115,875	27,021.5	12,358.5	389.3	160,364	37,713.4

Table 1: *Results obtained by running SynchronizedGlobal*

2.2 ParallelGlobal

ParallelGlobal is a multi-threaded implementation of Global. It has a simpler approach that maximizes resource utilization and simple implementation instead of algorithmic efficiency. ParallelGlobal workers share much less information than SynchronizedGlobal, to decrease inter-thread communication overhead. Depending on the optimization task, this can be an advantage or a disadvantage.

The algorithm keeps the main steps of a Global iteration, however there are no discrete iterations that could be counted. The worker threads are independent, without a global state governing the algorithm steps taken. ParallelGlobal is still meant to run in a system where access to shared containers is cheap, read and write operations on clustering data still requires synchronization.

The ParallelGlobal worker algorithm encapsulates a Global iteration, while only concentrating on one sample. The one sample needs to be piped through clustering, local search and clustering of the local optimum.

Sample generation creates the amount of samples that results in a reduced sample set containing one sample, or simply $\lceil 1/\gamma \rceil$, where γ is the sample reducing factor. The `Samples` queue is thread local, which means that each worker thread owns a separate container, no synchronization is required to access it, and no sample sharing happens between threads.

After sampling, the single sample is handled similarly to the Global algorithm. Clustering happens on the shared cluster data, optimized for independent access by multiple threads. Some efficiency is sacrificed, recently clustered samples may be ignored in the search.

If the sample was not clustered, a local search is started with the sample as the origin, and the resulting local optimum is considered for creating a new cluster.

The stopping criteria are evaluated by the workers at the start of each iteration, during an iteration the worker will not stop if a criterion is met.

ParallelGlobal was evaluated with differing thread counts and hardness settings to gain an understanding on how the algorithmic efficiency and runtime behaves. Figure 4 shows the runtime performance of ParallelGlobal compared to Global on a set of 14 objective functions, when finding a known optimum value. Results vary a lot depending on the objective function. On three of the functions ParallelGlobal produced a worse result than the single thread Global, due to the large loss in algorithmic efficiency. On these functions a single local search is enough to find the optimal value, therefore a single Global iteration does suffice. ParallelGlobal however cannot limit the evaluations to a single local search, on every worker thread a local search will be started. Even worse, if the optimal value is missed by the first worker thread to finish local search, it will start a new iteration. The runtime can even double or triple due to the poor algorithmic efficiency.

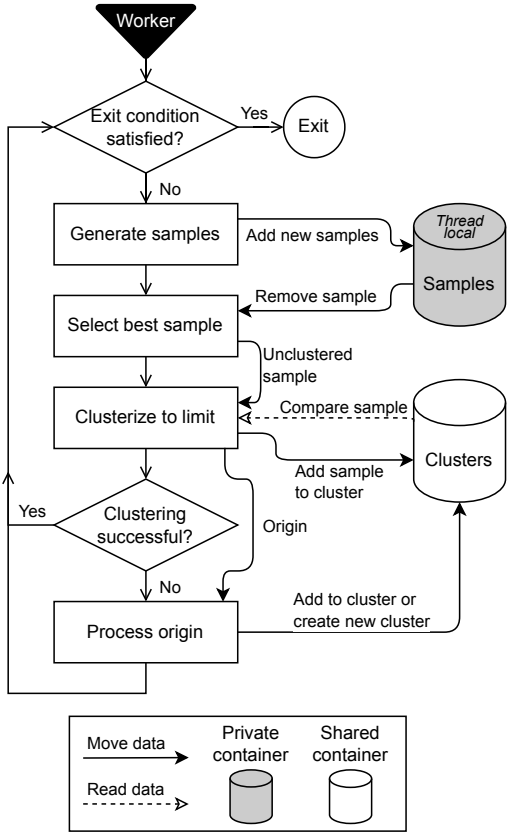

Figure 3: *ParallelGlobal worker algorithm*

On the Spikes test function the algorithm has a result near the theoretical speedup limit, finding the optimal value about 16 times faster than the single threaded Global, using 16 threads. This is not a coincidence, Spikes is crafted such that local searches are always very short and many iterations and local searches are needed to find the optimal value. This enables ParallelGlobal to quit the optimization process almost instantly after the optimal value was found, while still requiring a substantial amount of computing effort to utilize the worker threads.

Figure 5 shows the results for all tested configurations on the Shubert test function, where ParallelGlobal is compared to Global. As the NFEV plot shows, ParallelGlobal has a large increase in the number of function evaluations. Even

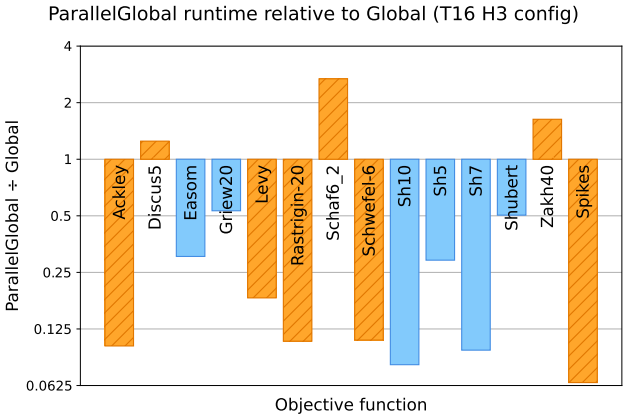

Figure 4: *ParallelGlobal runtimes compared to Global*

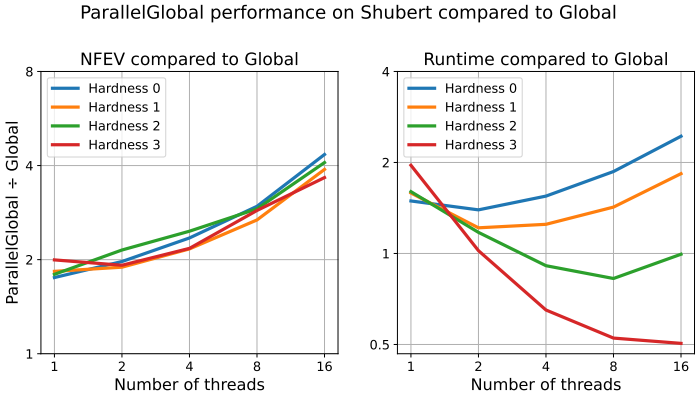

Figure 5: *ParallelGlobal results compared to Global on the Shubert function*

in the single thread configuration it requires twice the amount as Global does, the loss of algorithmic efficiency is intrinsic to ParallelGlobal. The efficiency loss grows with the number of threads, in the case of Shubert the loss can be overpowered by the additional computing resources, given a high enough computing cost. On other objective functions ParallelGlobal can have much better or much worse characteristics, it depends on what the solving process requires most, efficiency or performance.

2.3 DistributedGlobal

DistributedGlobal is a multi-threaded version of Global that can be executed in a distributed computing environment. The need for shared containers is eliminated so that workers have the most autonomy. DistributedGlobal retains the main characteristics of Global, a worker iteration contains the steps of a Global iteration. In addition, a data exchange step provides the opportunity to share information with other workers in a network. To improve clustering efficiency, shared information is incorporated in the cluster database, enabling the more effective reduction of candidate samples. Similarly to ParallelGlobal, DistributedGlobal workers are independent without a coordinated global state that would force workers to wait. With no shared container in the algorithm, read and write operations are non-blocking, the external information is incorporated asynchronously without disturbance.

DistributedGlobal eliminates the need for a central authority. All discussed

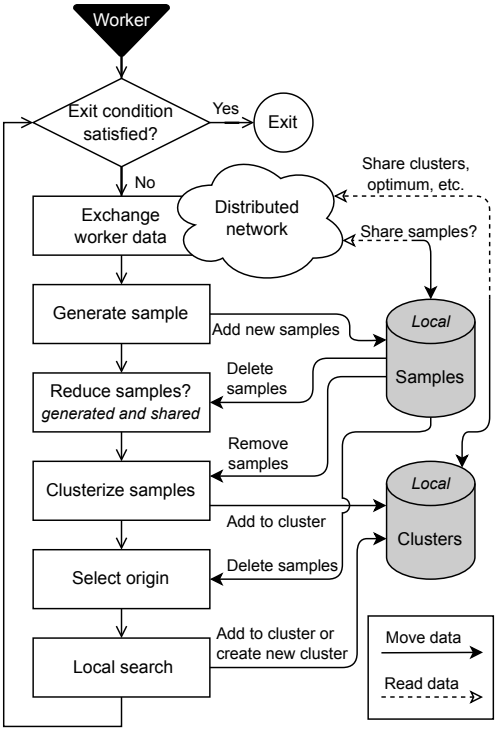

Figure 6: *DistributedGlobal worker algorithm*

versions so far can utilize the help of a distributed network to perform expensive function evaluations remotely, while the main algorithm has to run on a single machine. Given a central authority, there is always a possibility of failure, where the central node becomes unresponsive and the optimization process halts.

In a distributed algorithm every node can perform tasks on its own, while a central bookkeeping node can conveniently store results. Even if the central bookkeeper becomes unreachable, the new information is stored on nodes and can be extracted at a later point.

To aggregate different kinds of data, well known distributed information dissemination algorithms can be used. For example, at any given time the global optimum candidate with the lower function value is the true candidate, any incoming candidates with a higher value can be rejected.

An implementation of DistributedGlobal was not reached, therefore the algorithm is simply stated as-is, evaluations are not available. Figure 6 shows the proposed high-level algorithm structure.

3 Problems and solutions in neural network verification

Neural networks are well known to produce peculiar behavior on seemingly normal inputs. Neural network verification is the task of recognizing undesired artifacts in the network function. Large models are not feasible to quantitatively verify, the network complexity is far larger than a verifier could possibly explore. On small to medium size networks verification becomes feasible. A verifier should be able to determine whether the network satisfies the required properties. Small to medium sized networks are useful in tasks, where the solution to a problem is not exact. Typical examples of such tasks are character recognition, simple classification tasks, control system implementations like ACAS Xu.

Several verifier implementations exist that aim to prove or disprove the safety of neural networks. The mathematical models provide a solid theory for executing a verification task, but implementations lack the rigorousness necessary for strong statements. In average scenarios the existing verifiers do function correctly, however many edge cases exist that can be exploited by an adversary, or just bad luck with network structure. The currently known implementations do not differentiate sufficiently between rigorous and heuristic results, answers seem to always suggest rigorousness.

Chapter 3. of the PhD thesis presents findings on the vulnerability of neural network verifiers to adversarial attacks, and discussion on how to achieve more robust algorithms. A simple viable attack is presented, then a practical network is extended with adversarial functionality, both of which are capable of exploiting verifiers. A successful defense is presented against attacks based on perfectly

canceling float values. A logical flaw is uncovered in the application of interval arithmetic for estimating reachable sets of expressions with unspecified evaluation order. Some possibilities for strengthening the models against floating point errors are discussed. An algorithm improvement for MIPVerify is presented, that reduces the number of evaluated MILP models.

3.1 Neural network verification is not solved

Knowing that MILP solvers are used for verification, we are intrigued about how these models handle with reliability issues of the calculated MILP optimum. As we found, there are no satisfying solutions to this problem. Even mentions of the reliability issues are hard to come by, our goal is to improve the situation by highlighting the topic. We focus on the MIPVerify algorithm to showcase potential issues, which is an arbitrary choice from the available options. The findings can apply to any verifier that uses a MILP solver, or otherwise ignores floating point rounding errors.

To show that verification algorithms are lacking the capability for mathematical proofs we identify aspects of the algorithm that can permit adversarial behavior. MILP solvers are known to have limits in their robustness against floating point errors, a verifier needs explicit mechanisms to mitigate them. Floating point rounding errors and under-modeled reachable sets are identified as potential issues.

Based on the findings we crafted the trivial adversarial network shown on Figure 7. It causes a floating point computation with a large rounding error to confuse unsound verifiers. The network performs the classification task of deciding whether the $x \in [0, 1]$ input is above or below the 0.5 threshold. The ω parameter controls the rounding error in the $\omega + 1 - \omega$ computation. Given a sufficiently large ω for the used floating point precision, the value of the `C` neuron becomes binary, either 1.0, or 0.0.

The trivial adversarial network is shown to reliably exploit verifiers, where entire network sections are made invisible. MIPVerify tasked with finding an input that results in the y_1 output being maximal reported no such regions. Of course, it is easy to see that any $x < 0.5$ input is a valid example for such an output, MIPVerify missed very distinct behavior of the network. On the given task, the $C(x) = 1.0$ neuron output is not modeled, which in turn disregards the D_1 neuron and its effect in the model. On a sample of 500 combinations of σ and ω values MIPVerify missed the adversarial region in all cases.

Based on positive results of the trivial adversarial network, it is integrated in the `WK17a` MNIST classifier network to express a toy exploit. The adversarial network still correctly classifies the handwritten digits, but conditionally an off-by-one answer can be activated by a trigger on the input.

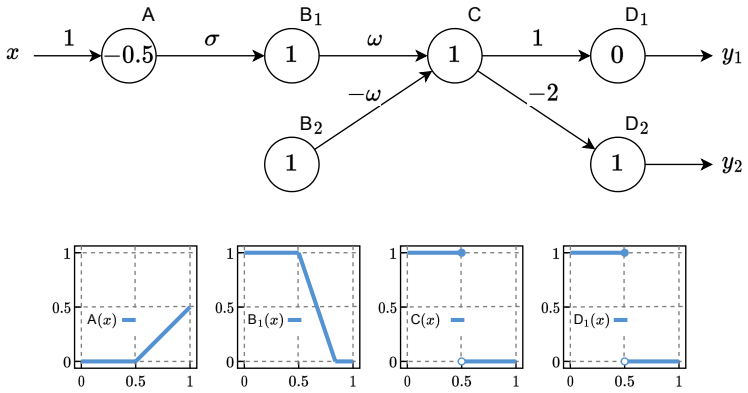

Figure 7: *Trivial adversarial network exploiting large floating point rounding errors*

The exploit is shown to still effectively mislead MIPVerify on the complete MNIST test set of 10,000 samples. Heuristic local search based methods are able to detect the adversarial region. When the adversarial region is shrunk from half to 1/512th of the search space, the heuristic method is no longer effective.

The trivial adversarial network has a very characteristic structure that clearly stands out even when integrated in the `WK17a` network. A large contributing part to its visibility is the large ω weight and neurons without an input. An obfuscated version of the trivial adversarial network is presented, that has no weights larger than 10 and all neurons have non-zero weight input edges. The obfuscated network is shown to be easily created, and over 95% of the created networks is marked as safe by MIPVerify configured with three different MILP solvers.

Finally, a defense is presented that can defuse attacks based on the perfectly canceling ω value method. Simply perturbing the network weights and biases with white noise, the non-zero the perfectly canceling $\omega + 1 - \omega$ calculation is disrupted. In 50% of cases the switch is completely deactivated, the network reverts to its original function. In the other cases the switch output is amplified. The trivial adversarial network and the WK17a adversarial network functions are relatively unaffected, only the output of the C neuron is larger. This widening of the network activation allows MILP solvers to detect the adversarial behavior due to the lessened scaling factor between model parameters. The effectiveness of this can be seen on Table 2. As Table 3 shows, the necessary noise levels do not alter the original network functionality, below 10^{-6} relative noise the test accuracy of WK17a does not change.

δ	10^{-3}	10^{-6}	10^{-9}	0
WK17a	4.37%	4.38%	4.38%	4.38%
WK17a-adv	75.85%	91.03%	98.3%	4.38%

Table 2: *Adversarial sensitivity of the WK17a MNIST networks with perturbed parameters, measured by MIPVerify+Gurobi.*

δ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-9}	0
WK17a	0.9788	0.9811	0.9810	0.9811	0.9811	0.9811	0.9811	0.9811
WK17a-adv	0.1118	0.3744	0.9725	0.98105	0.98105	0.9811	0.9811	0.9811

Table 3: *Test accuracy of the WK17a MNIST network with perturbed parameters (average of 10 independent perturbed networks).*

3.2 Thoughts on modeling floating-point computations

It is well known, and was demonstrated by the trivial adversarial network, that MILP solvers are sensitive to floating point rounding errors. In this application of the solvers the accuracy of an optimum value is only a secondary goal after a reliable bound, additional slack in the model can alleviate issues. This course of action could yield verifiers that are heuristic in nature, but very hard to exploit without the verifier noticing the attempt. More rigorous thought is needed on the reliability of verification toolchains. A non-technical issue is that verifier implementations usually do not make it clear whether a verification result should be regarded as robust with strong guarantees. Answers mainly fall into the `yes/no` categories without nuance. This leads to misunderstandings about results that can be easily prevented by crafting a better result classification system.

Limitations in the applicability of interval arithmetic for verification is shown in a simple scenario. Given an expression with a fixed order of operations, the interval arithmetic result will contain the floating point result. However, if the order of operations is not fixed, the interval arithmetic result is not guaranteed to contain the floating point result. Given a sufficiently large ω number and an order of operations where the produced interval is not maximal, the rounding errors can cause a significant deviation between the interval and floating point results. Verifiers implicitly assume that interval arithmetic provides a valid bounding on the model, as seen on Table 4 this is not necessarily the case.

MILP solvers are known to be sensitive to model scaling, Gurobi for example applies fixed slack when computing constraints. Scaling of the reachable activations in a neural network directly drives the scaling of the built MILP models. An adversary can directly affect the applied slack in the MILP solving process, therefore control the ease of finding the real optimum and affect the accuracy of a found optimum value.

Expression	Result with f_{64}	Result with IA
$(1-\epsilon)+\omega-\omega$	0	$[0, 2]$
$\omega-\omega+(1-\epsilon)$	$1-\epsilon$	$[1-\epsilon, 1]$
$\omega-\omega+(1+\epsilon)$	$1+\epsilon$	$[1, 1+\epsilon]$
$(1+\epsilon)+\omega-\omega$	2	$[0, 2]$

Table 4: *Differing results achieved by different ordering of an expression using floating point arithmetic and interval arithmetic. The ϵ removes rounding ambiguity.*

A heuristic defense is presented that widens model constraints with an estimated error term. The added slack amounts to an estimate on the maximum committed rounding error when computing the constraint on the modeled activation interval. The defense should be effective against scaling attacks, and help with attacks that force large rounding errors compared to the activation interval.

The "epsilon widening" heuristic strategy is presented for defending against numeric errors when the MILP optimum value is computed. Widening of input values helps interval arithmetic to incorporate all rounding errors in the result, lowering the attack surface. The defense is capable of handling attacks based on perfectly canceling float values. Previously the $0+\omega-\omega$ expression produced the $[0, 0]$ interval, the ϵ-widened version $[-\epsilon, \epsilon]+0+\omega-\omega$ produces $[eps(\omega), eps(\omega)]$, where $eps(\omega)$ is the rounding error of ω.

An algorithmic improvement is presented, that reduces the number of MILP solver runs. MILP solves dominate the cost of verification, reducing the necessary amount highly affects the execution time. Efficient modeling of the ReLU activations require knowing whether a given ReLU is stable. The MIPVerify implementation uses one or two evaluations per stable ReLU. By saving the activations of a single neural network evaluation ReLU stability can be decided solving a single MILP model. For tight modeling of an unstable ReLU we still need two MILP solves. Tight modeling of ReLU constraints might not be required, however the efficiency of model solving might decrease leading to an overall more resource intensive verifier.

A better classification for verification answers is provided, allowing the user to understand the implications of a specific result. In practice, it enables the assessment of the replicated MIPVerify algorithm results, where information sources disagree on the final answer. The decision can be `SAFE`, `ADVERSARIAL`, or `UNKNOWN`, with an additional prefix to indicate the result strength. If the result is considered rigorous the prefix is `PROVEN`, if there is a level uncertainty the prefix is `POSSIBLY`. The `UNKNOWN` answer should indicate a complete lack of confidence in the result.

On Figure 8 five versions of MIPVerify are evaluated, the effects of a different base implementation, a primitive rounding error heuristic, and the improved ReLU

modeling strategy. Runtime of the baseline re-implemented MIPVerify version is significantly increased, the runtime with improved ReLU modeling decreased. The cost of applying an interval arithmetic based heuristic defense against rounding errors has no appreciable effect on the overall runtime.

The algorithm versions with improved reliability have promising results on the *WK17a adversarial* network, with `POSSIBLY_ADVERSARIAL` answers. Rounding error compensation is capable of detecting the feasibility of an adversarial point, however the model is still too ill-defined to locate a corresponding input.

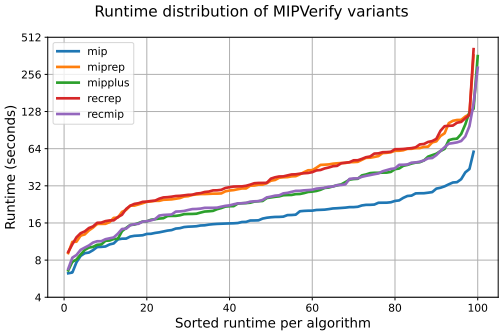

Figure 8: *Runtime distribution of different verifier implementations evaluating the first 100 MNIST test set samples on the WK17a neural network. Distributions are shown for each algorithm. Each verifier correctly found the adversarial examples.*

4 Contributions of the thesis

In the **first thesis group**, my contributions are related to parallel global optimization and improved versions of the Global optimizer algorithm. Detailed discussion can be found in Chapter 2.

I / 1. I developed the SynchronizedGlobal and SynchronizedGlobal Clustering algorithm, and showed their effectiveness in a multi-threaded execution environment. I demonstrated that the Global and SynchronizedGlobal algorithms have identical performance in a single thread configuration. I further improved the GlobalJ toolbox functionality. [3] [2] [1]

I / 2. I developed the ParallelGlobal algorithm and showed its effectiveness and weakness on objective functions. I demonstrated that the objective function structure greatly affects the effectiveness of parallelization. [5] [4]

I / 3. I designed the DistributedGlobal algorithm.

In the **second thesis group**, my contributions are related to MILP solver based neural network verification, with the focus on discovering potential attack vectors and improvements in defenses. Detailed discussion can be found in Chapter 3.

II / 1. I demonstrated the existence of an erroneously verified neural network, and refined it to the trivial adversarial network. I conducted tests and uncovered the exploit mechanism leading to an arbitrarily large difference between modeled and real network output. I showed that the exploit can be applied on a host neural network to create a backdoor that is invisible to MIPVerify. [6]

II / 2. I demonstrated that interval arithmetic is not sufficient for modeling output regions of arbitrary evaluation order linear expressions. I proposed several heuristic improvements to reduce verification sensitivity to rounding errors. I created an improved MILP solver based ReLU modeling algorithm and demonstrated its effectiveness. I demonstrated that a simple interval arithmetic based heuristic defense has little to no effect on verification runtime.

The author's publications on the subjects of the thesis

Journal publications

[1] A. Mester, D. Zombori, L. Pál, and B. Bánhelyi. Efficiency improvement of the GLOBAL optimization method by local search changes. *Acta Polytechnica Hungarica*, 19(2):29–42, 2022

[2] D. Zombori and B. Bánhelyi. Effects of pooling in ParallelGlobal with low thread interactions. *Informatica*, 45(2):191–196, 2021

Full papers in conference proceedings

[3] B. Bánhelyi, T. Csendes, B. Lévai, D. Zombori, and L. Pál. Improved versions of the GLOBAL optimization algorithm and the GlobalJ modularized toolbox. In *AIP Conference Proceedings*, volume 2070, no. 020022, 2019

[4] D. Zombori and B. Bánhelyi. ParallelGlobal with low thread interactions. In *Proceedings of the 22nd International Multiconference Information Society*, volume I, pages 83–86, 2019

[5] D. Zombori, B. Bánhelyi, T. Csendes, I. Megyeri, and M. Jelasity. Fooling a complete neural network verifier. In *9th International Conference on Learning Representations*. `https://openreview.net/pdf?id=4IwieFS44l`, 2021

Further related publications

[6] B. Bánhelyi, T. Csendes, B. Lévai, L. Pál, and D. Zombori. *The GLOBAL Optimization Algorithm*. Springer Briefs in Optimization, 2018

5 Összefoglalás

Az értekezés ismerteti egy optimalizáló algoritmus több szálas implementációját, ahol főbb megoldandó problémák az algoritmus hatékonysága és teljesítménye, valamint az algoritmus és a megoldandó célfüggvény tulajdonságai közt potenciálisan fennálló negatív kölcsönhatás. Az értekezés továbbá vegyes egészértékű lineáris programozási problémára viszavezetett neurális háló verifikáció témakörében ismerteti, illetve demonstrálja az optimalizált modell és a valós rendszer közötti különbségekből fakadó sebezhetőséget. Tárgyalja az intervallum aritmetika felhasználhatóságát, a támadhatóságon túl különböző lehetséges védekezéseket ismertet.

Parallel global optimization címmel a 2. fejezet három különböző több szálas algoritmus verziót ismertet, melyek a *Global* optimalizáló alapján készültek. A *SynchronizedGlobal* algoritmus szorosan követi a Global struktúráját és csak ott tér el tőle, ahol ezt a több szálas futás feltétlenül megköveteli. A SynchronizedGlobal algoritmus célja a hatékonyság. Minél nagyobb teljesítmény növekedést próbál elérni, de a célfüggvény kiértékelések számának csökkentését helyezi előbbre. A *ParallelGlobal* algoritmus a rendelkezésre álló számítási kapacitás kihasználására fókuszál, a célfüggvény kiértékelések számát csak másodlagos célként csökkenti. A szálak közötti interakciókat igyekszik csökkenteni, elkerülve a járulékosan kieső számítási kapacitást. A *DistributedGlobal* algoritmus ötlete a több számítógépen futtatott optimalizálási problémák megoldására koncentrál. Teljesen független szálakkal operál, amelyek különböző számítógépen is futhatnak egy elosztott rendszerben. Elsődlegesen a számítási teljesítmény kihasználására koncentrál, miközben lehetőséget biztosít a célfüggvény kiértékelések csökkentésére is a végrehajtást nem blokkoló szálak közötti kommunikációval. Az eredmények a több szálas algoritmusok teljesítménynövekedését és a célfüggvények karakterisztikájára való érzékenységét mutatják.

Problems and solutions in neural network verification címmel a 3. fejezet neuronháló verifikáló algoritmusok modell kiértékelési és modellezési hibákon alapuló sebezhetőségét ismerteti, valamint különböző heurisztikus stratégiákat a sebezhetőségek csökkentésére. Demonstrálja a *MIPVerify* érzékenységét a lebegőpontos számítások során elkövetett numerikus hibákra és a támadás kiterjeszthetőségét gyakorlatban előforduló neuronhálókra. A vegyes egész értékű lineáris programozási modellek jól ismert kiértékelési hibái mellett az intervallum aritmetikán alapuló eredményhalmazok felhasználhatóságában is fontos limitációt tár fel. Tört-lineáris kifejezések modellezésére új stratégiát mutat be, amely csökkenti *MIPVerify* algoritmus által felhasznált modellkiértékelések számát, ami jelentős futásidő csökkenést jelent.

Társszerzői nyilatkozat

Zombori Dániel *„Development and Application of Global Optimization Methods”* című PhD disszertációjában a következő eredményekben Zombori Dániel hozzájárulása volt a meghatározó:

1. **GlobalJ keretrendszer fejlesztése (2. fejezet)**
   - Algoritmus és célfüggvény konfigurációs modul [1]
   - Vonalmenti keresés modul, Unirandi algoritmus szeparált implementációja [1]
2. **Párhuzamos Global algoritmusok kifejlesztése (2. fejezet)**
   - SynchronizedGlobal algoritmus [2][3]
   - ParallelGlobal algoritmus [4]
   - DistributedGlobal algoritmus koncepció és prototípus implementáció [5]
3. **Neuronháló verifikáló eljárás sérülékenységeinek feltárása (3. fejezet)**
   - Minimális ellenséges neuronháló [6]
   - Obfuszkálás koncepció, minimális obfuszkált neuronháló [6]
   - Integrált ellenséges neuronháló [6]
   - Perturbáción alapuló védekezés [6]

Ezek az eredmények Zombori Dániel PhD disszertációján kívül más tudományos fokozat megszerzésére nem használhatók fel.

Szeged, 2024.12.03.

Zombori Dániel

Dr. Bánhelyi Balázs

Az Informatika Doktori Iskola vezetője kijelenti, hogy jelen nyilatkozatot minden társszerzőhöz eljuttatta, és azzal szemben egyetlen társszerző sem emelt kifogást.

Szeged, 2024.12.02.

Dr. Jelasity Márk

Hivatkozott publikációk

[1] A. Mester, **D. Zombori**, L. Pál and B. Bánhelyi. *Efficiency Improvement of the GLOBAL Optimization Method by Local Search Changes,* Acta Polytechnica Hungarica, Volume 19, Issue 2, 2022.

[2] B. Bánhelyi, T. Csendes, B. Lévai, L. Pál and **D. Zombori**. *The GLOBAL Optimization Algorithm,* Springer Briefs in Optimization, 2018.

[3] B. Bánhelyi, T. Csendes, B. Lévai, **D. Zombori** and L. Pál. *Improved versions of the GLOBAL optimization algorithm and the GlobalJ modularized toolbox,* AIP Conference Proceedings 2070, 020022, 2019.

[4] **D. Zombori** and B. Bánhelyi. *ParallelGlobal with Low Thread Interactions,* Proceedings of the 22nd International Multiconference Information Society, Volume I 83-86, 2019.

[5] **D. Zombori** and B. Bánhelyi. *Effects of Pooling in ParallelGlobal with Low Thread Interactions,* Informatica, Vol. 45, Issue 2, 191-196, 2021.

[6] **D. Zombori**, B. Bánhelyi, T. Csendes, I. Megyeri and M. Jelasity. *Fooling a Complete Neural Network Verifier,* 9th International Conference on Learning Representations, 2021.