Safe Start Condition Generation for Neural Policies

Bachelor's Thesis

Daniel Sherbakov

Universität des Saarlandes
Faculty of Mathematics and Computer Science
Department of Computer Science

17.06.2025

Reviewers

Prof. Jörg Hoffmann
Prof. Verena Wolf

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, 17.06.2025 _____
(Datum/Date)

_____
(Unterschrift/Signature)

Acknowledgements

I would like to thank Chaahat Jain for for their continuous guidance and support that helped shape this thesis. I wish to also express my gratitude to the rest of the Foundations of Artificial Intelligence team for their assistance and feedback. Finally, I am deeply thankful to my wife, Jasmine – without her support this work would not have come to fruition. Thank you.

Abstract

Ensuring the safety of neural action policies in planning domains is a critical challenge, especially in safety critical applications. This thesis addresses the problem of automatically generating safe start conditions – sets of initial states from which no unsafe policy execution can occur. We formalize this problem and introduce a general algorithmic framework for generating sound and complete safe start conditions. We present two variants of the algorithm: Invariant Strengthening, which derives a safe start condition from an unsafety condition, and Start Condition Strengthening, which refines an existing safe start condition. In addition, we introduce several enhancements to improve the efficiency and scalability of the approach, in the form of testing-based methods that quickly identify unsafe policy executions, and approximation methods that manage the growth of the start condition during refinement. We evaluate our approach on a set of benchmarks, demonstrating its effectiveness in generating safe start conditions and its potential to salvage otherwise unsafe policies by restricting their deployment to verified-safe contexts.

Contents

1 Introduction **2**

2 Background **4**
2.1 Planning Tasks and State Spaces 4
2.2 Action Policies 5
2.3 Policy Safety Verification 6
2.4 Encoding NN into SMT 7
2.5 Benchmarks 8

3 Safe Start Condition Generation Algorithms **10**
3.1 General Approach 10
3.1.1 Soundness and Completeness Guarantees 11
3.2 Invariant Strengthening 11
3.2.1 Experiments and Results 13
3.3 Start Condition Strengthening 14
3.3.1 Experiments and Results 15

4 Safety Testing **16**
4.1 Uniform Sampling 16
4.2 Biased Sampling 17
4.2.1 Distance based heuristic 19
4.3 Experiments and Results 19

5 Compactly Representing the Set of Unsafe States **24**
5.1 Bounded Box 25
5.1.1 Experiments and Results 26
5.2 Bounding Box 27
5.2.1 Experiments and Results 29

6 Conclusion **32**
6.1 Future Work 33

7 Appendix **35**

Chapter 1

Introduction

AI planning is concerned with designing agents capable of achieving specific goals through a decision-making process. In this context, an agent is an autonomous entity that performs actions based on its observations of the environment in order to solve a given problem. These problems are posed as **planning tasks**, and finding an agent that can successfully navigate such tasks, typically involves specifying a simulator that emulates the environment and identifying a strategy, called **action policy**, that dictates the sequence of actions the agent must perform to successfully complete the task. With the increasing complexity of planning tasks machine learning models such as Neural Networks (NNs) have been proven to be successful representation of action policies with exceptional decision-making capabilities in complex environments [1]. While these learned policies offer flexibility and scalability, their black-box nature introduces risks when deployed in safety-critical domains such as robotics, autonomous driving, and healthcare. In such settings a single unsafe action may have catastrophic repercussions. Therefore, it becomes essential to ensure that a learned policy avoids unsafe behavior under all relevant circumstances. This requirement has given rise to the nascent field of **action policy safety verification** – the application of formal methods to prove the safety of learned action policy. One important verification problem in this field is **closed-loop (or multi-step) safety verification**, where a policy is considered safe if, starting from any allowed initial situation, it cannot eventually result in an undesirable outcome. Formally, this corresponds to checking whether any policy execution can lead from an initial condition ϕ_0 to an unsafe condition ϕ_U.

Recent studies [2, 3], have shown that many learned policies are either provably unsafe or cannot be verified due to computational issues such as timeouts or excessive memory consumption during the verification process. In such cases, policies must be conservatively treated as unsafe, as formal safety guarantees cannot be established. Discarding such policies entirely, however, may be wasteful – particularly if they exhibit high performance in large, meaningful regions of the state space. This necessitates the exploration of alternative approaches to traditional verification that can provide formal guarantees regarding the safety of machine-learning-model representations of action policies. A critical and under-explored aspect of action policy safety is **automatic generation of safe start conditions**. It may be both desirable and sufficient to deploy a policy only within the subset of states where it can be guaranteed to behave safely – particularly when a constrained but safe deployment is preferable to no deployment at all. This motivates the goal of identifying a (maximal) set of initial states from which the policy is provably

safe. Automatically generating such safe start conditions allows us to salvage and utilize otherwise unsafe policies by restricting their deployment to verified-safe contexts.

This thesis investigates the problem of generating safe start conditions for neural action policies in both **discrete and continuous deterministic state spaces** as well as **discrete and non-deterministic state spaces**. We introduce a general algorithm, Safe Start Generator (SSG), that combines formal verification and testing based methods to identify a subset of the state space from which the policy execution is guaranteed not to reach an unsafe state. The general approach is instantiated in two variants: **Invariant Strengthening**, which derives a safe start condition from the unsafety condition, and **Start Condition Strengthening**, which derives a safe start condition from an existing one. These two variants are complemented by testing methods that employ uniform and biased sampling to identify unsafe paths more efficiently than verification. To further enhance the scalability and performance of the overall approach, we introduce approximation methods that compute compact symbolic representations of the set of identified unsafe paths to manage the growth of the start condition during refinement.

The remainder of this thesis is structured as follows. In Chapter 2, we provide the necessary background on action policy safety verification and discuss the challenges it presents. Chapter 3 introduces our approach for generating maximal safe start conditions, detailing the underlying methodologies and algorithms, along with an empirical evaluation of their performance. In Chapter 4, we present two testing-based methods that serve as inexpensive and efficient prechecks for identifying unsafe policy executions. This chapter also explores the challenges posed by the growth of the start condition during iterative refinement. Chapter 5 addresses the scalability limitations by introducing approximation strategies that balance computational efficiency with formal safety guarantees. Finally, Chapter 6 summarizes our findings and outlines promising directions for future work in the area of safe start condition generation.

The key contributions of this thesis are: (1) The development and implementation of two safe start generation techniques – Invariant Strengthening and Start Condition Strengthening, (2) introduction of a testing-based approach to identify unsafe paths, and (3) the exploration of approximation strategies to manage the growth of the start condition during refinement, (4) an empirical evaluation of the proposed methods on a set of benchmarks, demonstrating their effectiveness, scalability, and limitations.

Chapter 2

Background

2.1 Planning Tasks and State Spaces

A planning task $\Pi := \langle \mathcal{V}, \mathcal{L}, \mathcal{O} \rangle$ formally represents the problem at hand. I.e., it specifies the set of **action labels** $\mathcal{L}$, their associated **operators** $\mathcal{O}$, and the parameters of interest as a finite set of **state variables** $\mathcal{V}$, where each $v \in \mathcal{V}$ has a corresponding bounded integer domain $D_v \neq \emptyset$. We define Exp as the set of **linear integer expressions** over $\mathcal{V}$ of the form $z_1 \cdot v_1 + ... + z_r \cdot v_r + c$ where $z_i, c \in \mathbb{Z}$. $\mathbb{B}_{\text{lin}}$ denotes the set of linear constraints of the form $\sum_{v \in \mathcal{V}} z_v \cdot v \bowtie c$ where $\bowtie \in \{=, \leq, \geq\}$, and the quantifier free **boolean combinations** of thereof. The operators $o \in \mathcal{O}$ define the semantics of actions. Specifically, each l-labeled operator $o \in \mathcal{O}_l$ is defined as a tuple $o := (g, l, u)$ that specifies a **guard** $g \in \mathbb{B}_{\text{lin}}$, which determines the applicability of the action, and an **update function** $u : \mathcal{V} \to Exp$, which specifies how the variables are modified when the action is applied. The separation between action labels and operators allows for an easier representation of non-deterministic actions by specifying several different update rules for a single action.

The **state space** depicts the semantic relation between actions and variables of the planning task Π as a labeled transition system (LTS), a model that describes the evolution of states in a system, where transitions between states are labeled with actions.

The LTS describing the state space, denoted as the tuple $\Theta := \langle \mathcal{S}, \mathcal{L}, \mathcal{T} \rangle$, consists of a finite set of states $\mathcal{S}$, where each state $s \in \mathcal{S}$ represents a complete variable assignment. Formally, s is a function with the domain $dom(s) = \mathcal{V}$ and $s(v) \in D_v$. Transitions between states are defined by the relation $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$, also written as $s \xrightarrow{l} s'$. A transition $(s, l, s') \in \mathcal{T}$ exists if and only if a corresponding $o \in \mathcal{O}_l$ with g satisfied in s and applying u over the variables of s, denoted as $s[o] = \{(v, u(v)) \mid v \in dom(s)\}$, results in the successor state s'. As syntactic sugar, we also write $s[l]$ to denote all the possible resulting states from applying the action l in state s, i.e., $s[l] := \bigcup_{o \in \mathcal{O}_l} \{s[o]\}$.

In addition, we define the denotation function $[\![\cdot]\!] : \mathbb{B}_{\text{lin}} \to 2^{\mathcal{S}}$. E.g., $[\![\phi]\!]$ denotes the set of states in $\mathcal{S}$ satisfying ϕ i.e. $[\![\phi]\!] := \{s \in \mathcal{S} \mid s \models \phi\}$, where $s \models \phi$ denotes the evaluation $\phi(s)$ s.t. $\phi(s) \Leftrightarrow \top$. Moreover, we define $\phi_s \in \mathbb{B}_{\text{lin}}$ as the symbolic representation of the state s in the form of a linear constraint, i.e., $\phi_s := \bigwedge_{v \in \mathcal{V}} v = s(v)$.

2.2 Action Policies

Definition 2.2.1 (Action Policy)**.** An **action policy** is a function $\pi : \mathcal{S} \to \mathcal{L}$, mapping states to action labels.

An action policy π specifies an agents strategy of achieving a certain goal within a planning task by dictating the choice of the action for the current state of the agent. **Neural action policies** are action policies that are typically represented by machine learning (ML) models such as Feed-Forward Neural Networks (FFNNs), which are trained via reinforcement or supervised learning. FFNNs consist of multiple layers of neurons $L_1, ..., L_n$, where each layer L_i is parameterized with a weight matrix $W^i \in \mathbb{Q}^{d_i \times d_{i-1}}$ and bias vector $B^i \in \mathbb{Q}^{d_i}$, and the forwarding function of each layer defined as $f_i : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$, $X \mapsto W^i \cdot X + B^i$. In addition, an activation function is applied to the output of each layer, and in this thesis we consider the Rectified Linear Unit activation function, $ReLU(x) = max(0, x)$. We also introduce the input layer L_0 that represents the input interface, $f_0 : \mathcal{S} \to \mathbb{R}^{d_1}, s \mapsto (s(v_1), ..., s(v_{d_1}))$, that maps state variable values to the corresponding neuron in layer L_1. The overall forwarding behavior of the network can be represented as the function:

$$\mathrm{NN}(s) = (f_n \circ A \circ f_{n-1} \circ \cdots \circ A \circ f_1 \circ f_0)(s) \tag{2.1}$$

where A is the element-wise application of the activation function to the output of each layer.

$\mathrm{NN}(s)$ maps a state s to a real-valued score vector over actions in $\mathcal{L}$, rather than selecting a single action. To obtain a deterministic action policy π, we compose the network with an arg max operator:

$$\pi(s) = \arg\max_{l \in \mathcal{L}} \mathrm{NN}(s), \tag{2.2}$$

The current definition allows for the selection of inapplicable actions for a given state, i.e., $\pi(s) = l \in \mathcal{L}$ where $\nexists o = (g, l, u) \in \mathcal{O}_l : s \models g$. To avoid this issue, an applicability filter is applied, ensuring that only applicable actions are selected. This can be achieved by constricting the output of the policy to the set of applicable actions in the state s, denoted as $\mathcal{L}(s)$, i.e.,

$$\pi_{App}(s) = \arg\max_{l \in \mathcal{L}(s)} \mathrm{NN}(s). \tag{2.3}$$

Example 2.2.1 (Neural Action Policy)**.** An example of a neural action policy is illustrated in Figure 2.1, which shows a simple feedforward neural network (FFNN) with a single hidden layer. The input layer contains two neurons, each corresponding to a coordinate on a 2D grid. Given the input $(0, 0)$, the network selects the action *up*, as the corresponding output neuron has the highest activation value, which is 1. This can be verified by computing the output value for the *up* neuron:

$$o_{up} = w^2_{1,1} \cdot (W^1_{1,1}x + W^1_{1,2}y + B^1_1 + W^1_{2,1}x + W^1_{2,2}y + B^1_2) + B^2_{1,1}$$

Substituting $x = 0$ and $y = 0$, and the given weights and biases:

$$-1 \cdot (1 \cdot 0 + 0 + 0 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 + 0) + 1 = 1,$$

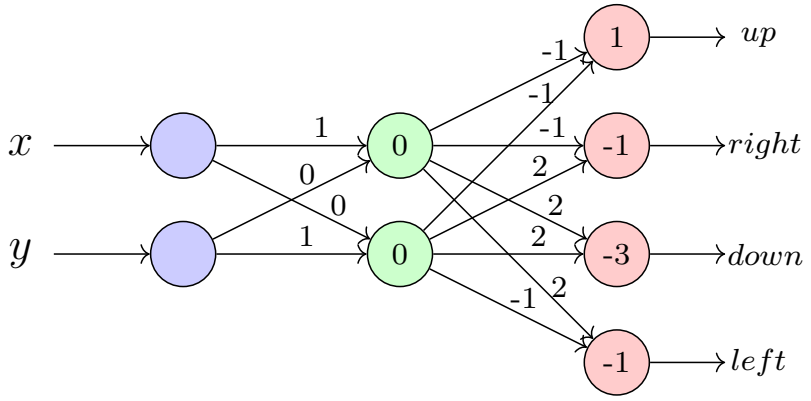


Figure 2.1: An example of an action policy that moves the agent clockwise in a 1x1 grid represented by a neural network. Edges are labelled with weights, and nodes with bias values.

where

$$W^1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad B^1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} -1 & -1 \\ -1 & 2 \\ 2 & 2 \\ 2 & -1 \end{bmatrix}, \quad B^2 = \begin{bmatrix} 1 \\ -1 \\ -3 \\ -1 \end{bmatrix}$$

We can see that the output values for the other actions will be negative, as the only contributing factor is the bias of the output layer, which is negative for all other actions. Thus, the action *up* is selected by the policy.

2.3 Policy Safety Verification

Safety verification in AI planning aims to determine whether a given policy can lead to undesirable outcomes when executed from a set of initial states. In this context, we define a safety property that must be satisfied to ensure the policy avoids unsafe states throughout its execution. Let ϕ_0 denote the start condition, defining the set of initial states in Θ, and let ϕ_U denote the unsafety condition, specifying the set of states that must not be reached.

Definition 2.3.1 (Safety Property)**.** For a planning task Π, the **safety property** is defined as the tuple $\rho := (\phi_0, \phi_U)$, where $\phi_0, \phi_U \in \mathbb{B}_{\text{lin}}$.

If there exists any policy-induced execution path from a state satisfying the start condition ϕ_0 to a state satisfying the unsafety condition ϕ_U, the safety property is violated and the policy is considered unsafe. Otherwise, the policy is safe with respect to the safety property.

To determine whether an action policy violates the safety property, safety verification is performed in the form of multistep reachability analysis. One approach is **Bounded Model Checking (BMC)** [4], where the entire policy execution path of length k is encoded into a linear formula and checked against a Satisfiability Modulo Theories (SMT)

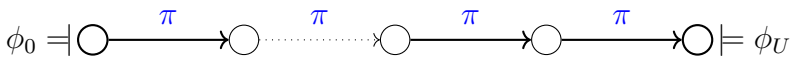


Figure 2.2: illustration of unsafe path induced by a policy π from an initial state satisfying ϕ_0 to an unsafe state satisfying ϕ_U

solver for its satisfiability. Not only this approach does not guarantee termination (in case the policy is safe), but it requires iterative checking of all path lengths which gets expensive quickly and therefore only feasible for short unsafe paths. Another approach is **Policy Predicate Abstraction (PPA)** [5] that performs reachability analysis on the policy-restricted abstract interpretation of the state space. This significantly reduces the size of the states space making it feasible to enumerate the abstract state space and determine an existence of an unsafe policy path. However, PPA has a key limitation, as it requires a manually provided abstraction mapping in the form of a set of predicates that define the abstraction, which is a non-trivial task that requires significant effort, and hinders scalability. To address this, a **Counterexample-Guided Abstraction Refinement (CEGAR)** algorithm is applied to the PPA framework [2]. This approach automates the process of selecting suitable predicates through an iterative refinement mechanism, eliminating the need for manual abstraction selection. The system dynamically refines predicates based on spurious counterexamples, unsafe paths that arise due to overly coarse abstractions, encountered during verification. While promising, this method is limited by the size of the neural network, and its scalability becomes a challenge as the model complexity increases.

2.4 Encoding NN into SMT

It is often necessary to encode the NN into a SMT formula to perform verification. Specifically, we need to encode (1) the forwarding function 2.1, (2) policy action selection, and (3) the applicability filter. We first define the encoding of the forwarding function, which we will define as ϕ_{NN}. Each neuron in the NN is represented by a variable $x_{i,j} \in \mathbb{R}$ where i is the layer index and j is the neuron index. The encoding of the forwarding function is defined as follows:

$$\begin{aligned}\phi_{NN}(s) = & \bigwedge_{1\le j\le d_0} x_{0,j} = s(v_j) \quad \text{(input layer)} \\ & \wedge \bigwedge_{1\le i\le n-1} \bigwedge_{0\le j\le d_i} x_{i,j} = \sigma(\sum_{k=1}^{d_{i-1}} W^i_{j,k} \cdot x_{i-1,k} + B^i_j) \quad \text{(hidden layers)} \\ & \wedge \bigwedge_{1\le j\le d_n} x_{n,j} = \sum_{k=1}^{d_{n-1}} W^n_{j,k} \cdot x_{n-1,k} + B^n_j \quad \text{(output layer)}\end{aligned} \tag{2.4}$$

where σ is the ReLU activation function. The expression $y = \sigma(x)$ expands into the formula $(y = x \wedge x > 0) \vee (y = 0 \wedge x \le 0)$, allowing us to express the piecewise-linear function as $\mathbb{B}_{\text{lin}}$ formula.

Next we define the encoding for (2) as ϕ_{SEL}. The policy action selection requires that the action with the highest output value is selected. This can be encoded as follows:

$$\phi_{SEL}(l) = \bigwedge_{1 \leq j \leq d_n} (x_{n,k} \geq x_{n,j}) \quad \forall j \neq k \tag{2.5}$$

where l is the action selected action label corresponding to the neuron k in the output layer.

Lastly, we define the encoding for (3) as $\phi_{APP}(l)$. The applicability filter can be encoded as follows:

$$\phi_{APP}(l) = \phi_{SEL}(l) \vee \bigwedge_{(g,l',u) \in \mathcal{O}_{l' \in \mathcal{L} \setminus \{l\}}} \neg g \tag{2.6}$$

This encoding ensures that either the selected action is the one with the highest output value or that all other actions are not applicable. Combining all three encodings, we can define the overall encoding of the NN as follows:

$$\phi_\pi(s, l) := \phi_{NN}(s) \wedge \phi_{APP}(l) \tag{2.7}$$

2.5 Benchmarks

For the evaluation of our approach, provided in [3, 5], in the form of **planning domains** and **action policies**, trained with Q-learning [6] for these domains, were used. The planning domains were modelled with the automata language JANI, and the neural action policies for the respective planning task consisting of 2 hidden layers with 16-128 neurons with ReLU as the activation function. The additional benchmarks, Beluga and Two-way line Transport, were trained by larger models and provide more complex environments.

In **Blocksworld (BW)** the agent must stack n blocks in a certain order without placing all blocks on the table. The Agents controls a hand that can pick up a block and put it down either on another block or on the table. When performing any action the agent can non-deterministically fail which, in case of a cost-aware variant, increases the cost associated with the block that was attempted to be manipulated.

One-way line (1way) specifies the task of delivering packages via a truck across a bridge while avoiding carrying more than one package when crossing the bridge. In this task, 15 packages are distributed among 9 different locations in a straight line. In addition, several other variants of the task are used.

N-Puzzle is a sliding puzzle where the agent must arrange the tiles in a certain order. The agent can slide tiles into the empty space, and the goal is to reach a specific configuration of the tiles. The agent can non-deterministically fail to slide a tile, which results increased cost of moving the tile.

Beluga in this task a beluga flight carries jigs with aircraft parts that need to be delivered to a factory in a certain order. The jigs can be unloaded from the flight in LIFO manner, and loaded onto a trailer which loads or unloads jigs from and onto a rack or onto a truck that delivers the jig to the factory. The rack acts a queue where jigs located at the edges can be loaded and unloaded. While unloading the jigs from the flight and handling the racks, with each performed action the next required jig by the factory can non-deterministically change.

Two-way line (2way) extends the Transport domain with actions such as acceleration and deceleration that modify the velocity of the truck. In addition, these actions can non-deterministically fail on certain road segments which causes the velocity to remain unchanged or drop a package. Moreover, the unsafety condition here is to avoid crashing on either end of the two-way line segment by failing to adjust the truck's velocity to stop at the ends.

The following planning domains are of **continuous** nature, i.e., the state space is defined over real-valued variables resulting in an infinite number of states.

Bouncing Ball in this task, a ball is dropped from a certain height and bounces off the ground. The balls velocity diminishes with each bounce, and eventually the ball stops bouncing. The agent can push downwards on the ball to increase its velocity, or do nothing, with the goal of keeping the ball bouncing indefinitely.

Inverted Pendulum a pole is initially balanced in the upright position and can be influenced by applying leftward or rightward force. The task is to keep the pole balanced. The system becomes unsafe if the pole tilts beyond a certain angle.

Cart-Pole a variation of the inverted pendulum where a pole is mounted on a cart. Instead of directly applying force to the pole, the cart itself is moved left or right to maintain balance.

Stopping Car here, a vehicle must follow a lead car moving at a fixed velocity, using only two actions: accelerate or decelerate. The goal is to maintain a safe distance without crashing. The system is unsafe if the follower car collides with the lead car.

Chapter 3

Safe Start Condition Generation Algorithms

In this chapter, we present our general approach for generating a maximal safe start condition for a given neural policy. At its core, the algorithm iteratively identifies unsafe policy executions and refines the search space by removing unsafe start states – until no further unsafe executions can be found. We then present two concrete approaches for generating safe start conditions. Both rely on formal verification for identifying unsafe policy transitions, but differ in their underlying goals and their method for verifying existence of unsafe policy transitions. These differences yield complementary strengths, which we explore in the subsequent sections.

3.1 General Approach

Algorithm 1 Generate a safe start condition

Input: π, ϕ_0, ϕ_U
Output: safe ϕ_0

repeat
  $U \leftarrow$ `FindUnsafeTransitions`(ϕ_0, ϕ_U, π)
  $\phi_0, \phi_U \leftarrow$ `UpdateConditions`(ϕ_0, ϕ_U, U)
until $U = \emptyset$
return ϕ_0
procedure UpdateConditions(ϕ_0,ϕ_U,U)
  for all $s_u \in U$ and $s_u \models \phi_0$ **do**
    $\phi_0 \leftarrow \phi_0 \wedge \neg\phi_{s_u}$
    $\phi_U \leftarrow \phi_U \vee \phi_{s_u}$
  return ϕ_0, ϕ_U

Algorithm. 1 shows a high level overview of the safe start condition generation (SSG) algorithm. The algorithm starts with a provided policy π, a start condition ϕ_0, and an unsafety condition ϕ_U. It then iteratively finds unsafe transitions and updates the start and unsafety conditions until no more unsafe transitions can be found. The update is performed according to the **update rule** which is defined by the `UpdateConditions`

procedure. Conceptually, this is done by *removing* the unsafe start states (policy executions starting in ϕ_0 and ending in ϕ_U) from the start condition and *adding* them to the unsafety condition. As a result, the start condition shrinks over time, while the unsafety condition expands to account for newly discovered unsafe behaviors. The procedure `FindUnsafeTransitions` captures the mechanism for discovering unsafe executions. Its implementation differs depending on the concrete approach used, which we describe in the following sections of this chapter.

3.1.1 Soundness and Completeness Guarantees

We now state the formal guarantees of the Safe Start Generation (SSG) algorithm. Soundness ensures that any start condition returned by the algorithm excludes all executions that lead to unsafe states. Completeness ensures that if a maximal safe start condition exists, the algorithm will find it and terminate in finite time.

Theorem 1 (Soundness)**.** If the algorithm returns a start condition ϕ_0, then it is safe. That is, no policy execution starting from a state satisfying ϕ_0 leads to ϕ_U.

Theorem 2 (Completeness)**.** For any policy π and safety property ρ, the SSG algorithm terminates in finite time, returning a maximal start condition ϕ_0.

Proof Sketch of Theorem 1. The soundness of the general algorithm follows from the use of a sound verification backend, which ensures that no unsafe executions exist within the returned start condition. Each update removes unsafe executions from the current candidate start condition, and the algorithm terminates only when no violations remain. □

Proof Sketch of Theorem 2. Assuming a finite state space, the algorithm eliminates at least one unsafe execution in each update of the conditions. Since the number of executions is finite, the algorithm must terminate in a finite number of steps, either returning a empty or non-empty start condition. □

3.2 Invariant Strengthening

In this approach, we aim to generate a safe start condition from a given policy π and an unsafety condition ϕ_U. To do so, we invoke the SSG algorithm with the start condition $\neg\phi_U$, and unsafety condition ϕ_U. This partitions the state space into two disjoint sets:

- The invariant ($[\![\phi_0]\!]$), representing candidate safe states (initially $\phi_0 := \neg\phi_U$), and
- The non-invariant ($[\![\phi_U]\!]$), containing provably unsafe states (initially ϕ_U).

We refer to this method as Invariant Strengthening (INV) and will use this abbreviation throughout the remainder of the thesis.

With each iteration, the algorithm refines this partition by removing from the $[\![\phi_0]\!]$ any state with a policy-induced transition to $[\![\phi_U]\!]$ and adding it to the $[\![\phi_U]\!]$, thereby preserving disjointness ($[\![\phi_0]\!] \cap [\![\phi_U]\!] = \emptyset$) and joint exhaustiveness ($\forall s \in S : s \in [\![\phi_0]\!] \cup [\![\phi_U]\!]$), as formalized in Lemma 7.0.1 and Lemma 7.0.2.

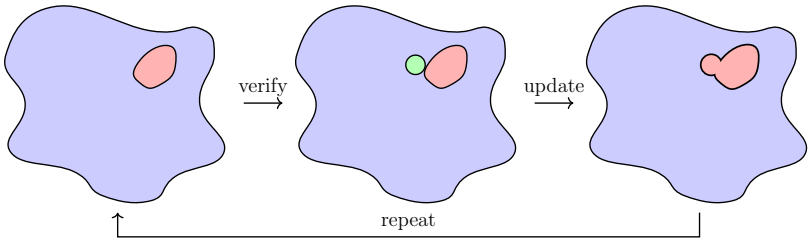


Figure 3.1: Illustration of the invariant strengthening process. The leftmost figure shows the initial partitioning of the state space into the invariant (blue) and the non-invariant (red). In the middle figure, the green region represents an source state in the invariant with a transition into the non-invariant that was determined through verification. The rightmost figure shows the final partitioning of the state space after updating the the sets.

By construction, this iterative refinement guarantees that, once the algorithm terminates, the invariant reached a fixedpoint and is closed under the transition relation induced by the policy – meaning, no state in $[\![\phi_0]\!]$ has a successor in $[\![\phi_U]\!]$, as required by the guarantees in Section 3.1.1. As a result, the final $[\![\phi_0]\!]$ forms a maximal inductive invariant within $\neg\phi_U$ (Lemma 7.0.3), ensuring that every included state is provably safe under one-step transitions.

To check for existence of violating transitions, such transition is encoded into a linear constraint and an SMT solver is used as a verifier to prove its existence by finding a satisfying assignment. Since reachability to unsafe states is captured by this fixedpoint refinement process, multistep reachability analysis is not required. Instead, multistep safety is ensured compositionally via repeated single-step refinements – a key efficiency advantage of the approach. Formally, the **safety violating abstract policy transition** $\phi_0 \xrightarrow[\pi]{l} \phi_U$ encoding is defined as:

$$\bigwedge_{v \in \mathcal{V}} s(v) \in D_v \wedge s \models \phi_0 \wedge s \models g \wedge s(u) \models \phi_U \wedge \phi_\pi(s, l), \tag{3.1}$$

This encoding involves the following constraints: domain bounds for each variable, the invariant and non-invariant, existence of a valid transition between two concrete states, and the selection of the action label l by the policy π corresponding to the valid transition. The latter constraint requires to encode the entire NN architecture into the SMT encoding (cf. 2.4), which is exponential with the number of variables. If a satisfying assignment is found, the sets are updated by removing the source state of the transition from invariant and adding it to the non-invariant, as described in `UpdateConditions` in algorithm 1.

The invariant and non-invariant sets are updated iteratively until no satisfying assignment is found. In which case the invariant is either empty, $\phi_0 \Leftrightarrow \bot$, meaning there exists no safe start condition under which the policy is safe, or a safe start condition was successfully generated. The entire process is depicted in figure 3.1.

The underlying verification procedure, `FindUnsafeTransitions`, for invariant strengthening is shown in Algorithm 2. The algorithm performs a single-step reachability analysis by iterating over all actions and operators, and for each operator first performs a precheck whether there exists an abstract transition, $\phi_0 \xrightarrow{l} \phi_U$, which is not necessarily induced by the policy. This transition is check is done via Z3 [7] and is significantly cheaper as it does not include the NN encoding. By performing this precheck we avoid a more expensive

check that is guaranteed to return UNSAT if the precheck is unsatisfiable. Otherwise, we perform the policy induced abstract transition check, $\phi_0 \xrightarrow[\pi]{l} \phi_U$, which is done with a NN specialized solver called Marabou [8]. If this check returns SAT, we extract the source state from the satisfying assignment (line 7) and add it to the set of unsafe start states (USS) U, i.e., states satisfying ϕ_0 that have a policy transition to the non-invariant. The algorithm terminates when all actions and operators have been checked.

Algorithm 2 Verify the existence of unsafe single-step policy transitions

```
1: function FindUnsafeTransitions(π,φ_0,φ_U)
2:     U ← ∅
3:     for all l ∈ L do
4:         for all o ∈ O_l do
5:             if exists(φ_0 -l-> φ_U) then
6:                 if exists(φ_0 -l,π-> φ_U) then
7:                     s ← getSolution(solver)
8:                     U ← U ∪ {s}
9: return U
```

3.2.1 Experiments and Results

We now evaluate the effectiveness of the Invariant Strengthening approach as an instantiation of the Safe Start Condition Generation (SSG) algorithm. The goal of this evaluation is to assess *To what extent can formal verification methods identify provably safe subsets of the state space for a given neural action policy?*

Experimental Setup. The method was implemented within the C++ codebase of the verification framework PlaJA[1] introduced by [5]. Transition verification is performed using the Z3 SMT solver [7] and, where necessary, a branch-and-bound strategy on top of the neural network solver Marabou [8]. All experiments were run on machines with Intel(R) Xeon(R) Gold 5418Y CPU at 2.2GHz, with a memory limit of 4 GB and a time limit of 24 hours. The evaluation was conducted on the set of benchmarks described in Section 2.5, covering a diverse range of planning tasks with varying degrees of policy complexity and state space size.

Findings. Across all evaluated benchmarks, the invariant strengthening approach failed to generate a non-empty safe start condition within the resource limits. In most cases, the procedure timed out, while in others it exceeded the memory limit before termination.

In addition, Figure 3.2 shows that the verification time increases exponentially with the number of iterations, while the number of newly identified unsafe start states (USS) grows only linearly. This pattern was consistent in nearly all benchmarks, except for a few where each verification was quite expensive which quickly exhausted the allocated

[1]PlaJA is publicly available under https://gitlab.cs.uni-saarland.de/vinzent/PlaJAPublic/-/tree/icaps24Public

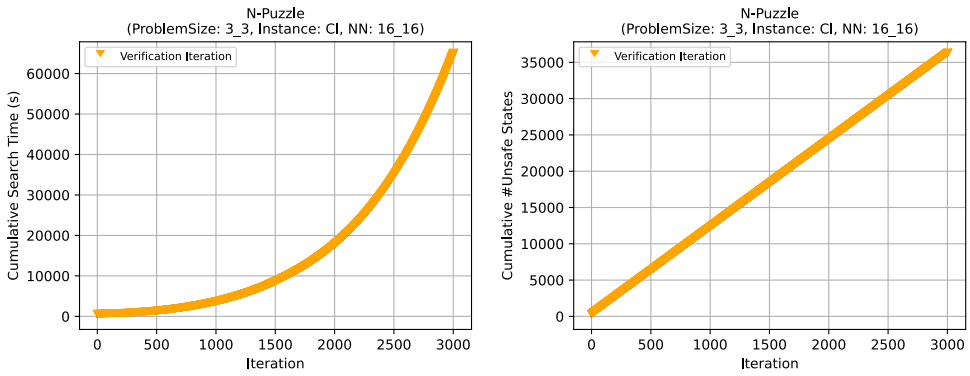


Figure 3.2: Plots for the **N-Puzzle** domain showing non-linear growth of verification time and linear growth of unsafe states identified. left: verification time vs iterations, right: unsafe states vs iterations.

time and resulted in too few iterations for meaningful trends to emerge[2]. Moreover, no significant difference was observed between benchmarks in continuous versus discrete state spaces. These observations motivate the need to explore alternative or complementary strategies – either by focusing verification on a more restricted region of the state space or by employing faster methods to identify unsafe behavior.

3.3 Start Condition Strengthening

After presenting the invariant strengthening algorithm, and showing its inability of generating a safe start condition, we now look at different approach in an effort to overcome the limitation of invariant strengthening by generating a maximal safe start condition within a subset of the state space rather than the entire state space. Unlike invariant strengthening that derives a completely new start condition, the goal of Start Condition Strengthening (SCS) is to find a safe subset of an already manually provided start condition. Generally, in AI Planning, there already exists a predefined notion of a start condition from which the policy is intended to be executed, often based on domain knowledge or task specifications. Making it a practical and natural choice for the purpose of generating safe start conditions.

In this scenario, the start and unsafety conditions do not form a jointly exhaustive partition of the state space, and thus multistep reachability analysis is required to identify unsafe policy executions. This kind of verification is done by performing an incremental search over an abstract interpretation of the state space called **Policy predicate abstraction (PPA)** [5]. The idea of PPA is to abstract the state space by a set of predicates $\mathcal{P}$ and perform a reachability analysis over the abstracted state space. When a unsafe path is found, various checks are performed to determine whether the path is spurious or not. In the case of the former, the predicate set is refined by adding new predicates that remove the spurious transition that occur by the over-approximation nature of the abstraction. However, if the path is not spurious, the policy is not safe under the provided start condition. The refinement is done by a CEGAR procedure to the PPA[2],

[2]The full set of plots was omitted for brevity but can be found in Appendix 7

which finds predicates that capture only the spurious unsafe transitions and refines the set of predicates resulting in a non-spurious path.

3.3.1 Experiments and Results

The experimental setup for evaluating Start Condition Strengthening is similar to that described in the previous section for Invariant Strengthening, including benchmark domains, hardware, resource limits (cf. Section 3.2.1), and start conditions constructed by using domain knowledge for each of the benchmarks. Therefore, we directly present the results and analysis.

Findings. Similar to Invariant Strengthening, this method was unable to generate safe start conditions except for the cases where the provided start condition was safe and therefore already the maximal safe start condition[3]. In addition, SCS proved to be significantly more computationally demanding as opposed to INV. In all benchmarks, the algorithm was only able to identify between **1 to 5 unsafe start states** before terminating early due to memory or time constraints. In contrast to the INV setting, where trends such as the exponential growth in runtime could still be observed over many iterations, SCS typically completed only a handful of iterations – too few to extract meaningful patterns. The low number of unsafe states stem from the fact that unlike INV where the number of USSs identified in a single iteration is bound by number of single step transitions possible, in SCS verification is performed on a single multistep policy path which typically only yields a single USS per path. We also note that real-valued benchmarks experienced modeling issues making them incompatible with the PPA CEGAR procedure, and as a result produced no data that we could analyze.

Despite these limitations, it is not immediately clear whether Start Condition Strengthening should be dismissed entirely. On the one hand, its focus on a restricted region of the state space offers a potentially more practical starting point in real-world applications, where deployment environments often come with predefined constraints. On the other hand, the ability of SCS to find and verify unsafe transitions appears to be a fundamental scalability bottleneck. The question of whether this bottleneck can be addressed justifies further investigation of this method alongside alternative strategies.

[3] All results can be found in Appendix 7.

Chapter 4

Safety Testing

In section 3 we have described formal verification approaches for determining unsafe policy transitions, and based on them generate safe start conditions. While these approaches guarantee to find all such transitions, they were shown to be computationally infeasible in practice. In this section we will introduce two testing-based methods that serve as inexpensive prechecks for the existence and identification of unsafe transitions. Specifically, by sampling concrete executions of the policy, these prechecks can quickly uncover many unsafe policy executions. If no unsafe executions are found, we fall back to verification based methods. The goal is to accelerate the the overall SSG algorithm by avoiding incurring the computational burden of full formal verification for the purpose of unsafe policy transition identification.

Algorithm 3 SSG with Safety Testing

Input: π, ϕ_0, ϕ_U
Output: safe ϕ_0

1: **repeat**
2: $\quad U \leftarrow \texttt{SampleUnsafePaths}(\pi, \phi_0, \phi_U, T)$
3: $\quad$ **if** $U = \emptyset$ **then**
4: $\quad\quad U \leftarrow \texttt{FindUnsafeTransitions}(\pi, \phi_0, \phi_U)$
5: $\quad \phi_0, \phi_U \leftarrow \texttt{UpdateConditions}(\phi_0, \phi_U, U)$
6: **until** $U = \emptyset$
7: **return** ϕ_0

In Algorithm 6, we present the testing adapted version of SSG algorithm. The algorithm is similar to the one presented in Chapter 3, but it includes an additional step that samples unsafe paths before attempting to find unsafe transitions using verification (line 2). The sampling is performed by the function $\texttt{SampleUnsafePaths}(\pi, \phi_0, \phi_U)$, which is described in detail in the following sections. If no unsafe paths are found, the algorithm falls back to the verification-based approach to identify unsafe transitions.

4.1 Uniform Sampling

In the context of this thesis we define safety testing as the time-bounded execution of the action policy for the purpose of identifying unsafe paths. Specifically, the policy

is executed on uniform randomly sampled start states $s_0 \in [\![startCond]\!]$. During the policy run if any state $s' \models \phi_U$ is encountered, the entire path is stored and the policy is executed on another start state. If no unsafe state is encountered within a time frame T, the path is discarded, and the testing procedure is terminated. We also include a check for cycles to avoid infinite loops that will only increase overhead without contributing to the identification of unsafe paths. The motivation behind this, is that identifying unsafe paths, by other means such as verification is expensive. With testing we can quickly find many unsafe paths which are easier to identify and only use verification to identify the more difficult ones that testing was not able to. Therefore, reducing the computational overhead which would have been resulted from identifying the easier unsafe path instances.

Algorithm 4 Uniform Policy Path Sampling

Input: π, ϕ_0, ϕ_U, T
Output: U *// Unsafe states*

```
1:  U, σ ← ∅
2:  t ← 0
3:  while t < T do
4:      s ← sample_state(φ_0)
5:      while True do
6:          l ← π(s)
7:          s ← s[l]
8:          if s ⊨ φ_U then break
9:          if L(s) = ∅ or |σ| > 1000 then
10:             σ ← ∅
11:             break
12:         σ ← σ ∪ s
13:     U ← U ∪ σ
14: return U
```

Algorithm 4 describes the uniform sampling procedure. The algorithm takes as input the policy π, start condition ϕ_0, the unsafe condition ϕ_U and a time limit T. In each iteration of the outer loop, a state s is sampled uniformly random from the start condition ϕ_0 (line 4). The inner loop executes the action policy π on the sampled state s until either an unsafe state is encountered, a dead end reached, or the policy has entered a loop. If an unsafe state is encountered, the path σ is stored in U (line 13), and the policy is rerun on a newly sample start state. In case of dead-ends and cycles no progress can be made so the path is discarded and a new policy run begins (lines 9-11). To avoid cycles we simply bound the policy execution to 1000 steps. If none of the aforementioned conditions are met, the current state is appended to the path (line 12) and the policy run continues. The algorithm terminates when the time limit T is reached.

4.2 Biased Sampling

When dealing with non-deterministic state spaces, policy execution might miss unsafe transitions with low probability. Therefore, in this approach the policy execution is biased towards states that are more likely to result in an unsafe execution according to a distance

based heuristic h_U that approximates a distance for a state towards the unsafety region. However, the policy execution can only be biased on non-deterministic steps without contradicting the policy itself. For example, for the state s the policy outputs action l whose execution in s results in one of the following possible target states $t_1, \ldots, t_n$ we can select t_i that minimizes the evaluation of a heuristic function over the target states. Choosing any other target state $t_j \notin s[l]$ will violate the policy transition.

This notion of biased sampling is extended to a lookahead search algorithm. Instead of simply evaluating successor states with a heuristic function, and selecting the target state with the lowest heuristic value, we can also evaluate the next k states of every successor. The lookahead search algorithm is more computationally expensive than the simple biased sampling approach, but by evaluating states deeper in the search tree we obtain a more informative decision when selecting the policy trajectory. The lookahead search algorithm is a bread-first search algorithm that explores the search space until the heuristic function can discriminate a single state at the current depth d, i.e. $h_U(s_i) < h_U(s_j) \in \{T_d \mid i \neq j\}$ where T_d denotes the current search tree with depth d.

Algorithm 5 Perform BFS exploration of the policy envelope until h_U discriminates successor.

Input: $S \subseteq \mathcal{S}$
Output: $\sigma \subseteq \mathcal{S}$

$SearchTree, Q \leftarrow S$
while $t < T$ **do**
  $D \leftarrow \emptyset$
  // Evaluate all states in Q
  for all $s \in Q$ **do**
    $D \leftarrow D \cup h_U(s)$
  // Check if h_U discriminated
  if $D_i < D_j \quad \forall j \neq i \in [0, \ldots, |D| - 1]$ **then**
    return `get_path`(Q_i)
  // Check if all states in Q are terminal
  if $\forall s \in Q\ \mathcal{L}(s) = \emptyset$ **then break**
  // Expand all states in Q
  $Q' \leftarrow \emptyset$
  for all $s \in Q$ **do**
    if $s \models \phi_U$ **then return** `get_path`(s)
    $S' \leftarrow s[\pi(s)]$
    for all $s' \in S'$ **do**
      if $s' \notin SearchTree \cup Q$ **then**
        $Q' \leftarrow Q' \cup \{s'\}$
        $SearchTree \leftarrow SearchTree \cup \{s'\}$
  if $Q' \neq \emptyset$ **then break**
  $Q \leftarrow Q'$
$s \leftarrow \arg\min_{s \in Q} h_U(s)$
return `get_path`(s)

Algorithm 5 receives as input a set of states that result from applying a non-deterministic action. These states are used as root nodes that initiate the exploration procedure. These are inserted into a the frontier set Q and the set $SearchTree$ which is used to avoid exploring already seen states and also to reconstruct paths. In each iteration of the while loop, the algorithm first evaluates the heuristic function h_U over the frontier set Q (lines 5-6). Then it proceeds to check if the heuristic function was able to find a state for which obtains a unique minimum value (line 9), in which case the algorithm terminates and returns the path to the state with the lowest heuristic value (line 9). If (a) no unique minimum has been found, (b) all states in the frontier set are terminal (line 11), (c) new frontier after expansion is empty (line 20), or (d) the time limit T is reached, multiple minimal contenders are present in Q. This necessitates tie-breaking, which is performed by selecting uniformly at random a state from the a subset of states in Q that have the lowest heuristic value (line 22). When expanding the frontier set Q, it is necessary to check if the state has been already seen as cycles might otherwise occur. In addition, if an unsafe state is encountered we immediately terminate and return the path to it (line 15). If Q' is empty after expansion, we still want to return states from the previous iteration so we require Q' to be non-empty for the update to occur (line 21). The function `get_path`(s) is used to reconstruct the path from the root node to the state s in the search tree. If s is unsafe it will be excluded from the path.

4.2.1 Distance based heuristic

The heuristic h_U is a non admissible heuristics that approximates the distance of a state s to the unsafe region of the state space described by ϕ_U. The underlying computation is based on $d(s, \phi)$, which is a manhattan distance function adapted for $\mathbb{B}_{\text{lin}}$, i.e., $h_U(s) := d(s, \phi_U)$. The distance function is formally defined as follows:

$$d(s, v \bowtie c) := \begin{cases} 0 & \text{if } s \models v \bowtie c \\ |s[v] - c| & \text{otherwise} \end{cases}$$

$$d(s, \phi_1 \wedge \phi_2) := d(s, \phi_1) + d(s, \phi_2)$$

$$d(s, \phi_1 \vee \phi_2) := \min(d(s, \phi_1), d(s, \phi_2))$$

Here $d(s, \phi)$ uses a recursive definition to handle the decomposition of conjunctions and disjunction of the formula $\phi \in \mathbb{B}_{\text{lin}}$. The function is broken down until it can evaluate the distance of a state s to a linear constraint bounding a single variable in s. In this case the function returns 0 if the state satisfies the constraint, and otherwise the absolute difference between the state and the bound c is returned.

4.3 Experiments and Results

This section evaluates the impact of integrating testing-based prechecks into the Safe Start Generation (SSG) process. Specifically, we want to answer *How does the integration of testing-based methods influence the effectiveness of SSG?* We compare this adaptation of the SSG algorithm using either testing method against the pure verification-based

approach, presented in Chapter 3 and against each other, in terms of the number of unsafe start states identified. The goal is to assess the extent to which testing method accelerates the overall SSG progress and which method is the most efficient for the generation of safe start conditions. The experiments were conducted using a testing time limit of 15 minutes with the same setup, benchmarks, and computational limits described in Section 3.2.1, ensuring comparability with the verification-only approaches. For the full list of results, see Appendix 7.

Findings. During experimentation, we encountered issues relating to the simulator in the case of Invariant Strengthening, deeming many benchmarks unsuitable for this method. By using the negation of the unsafety condition as the start condition, we encounter many semantically unsound states that the transition semantics do not account for, crashing our program. For example, in the Blocksworld domain, the unsafety conditions is simply `blocks_on_table` $> x$, which allows for states where the manipulator hand is not empty, indicating that the agent is holding a block, while simultaneously no block is actually being held. Many actions do not account for such states, making it possible to apply an action resulting in an invalid state, instead of treating such states as dead-ends. In the case of the example, strengthening the guards of the actions is sufficient to avoid such issues, however, other domains may require more complex and non-trivial handling. Such problems did not arise in the SCS setting, as the search begins from a semantically sound region of the state space dictated by the provided start condition. This ensures that only valid, semantically well-formed states are explored during policy execution, thereby avoiding simulator crashes and inconsistent behavior.

From Figure 4.1, it is evident that testing based methods are more efficient than verification with uniform sampling identifying a higher absolute number of unsafe start states compared to biased sampling. This is primarily because biased sampling explores a broader set of possible paths taken by the policy, resulting in longer runtimes per episode and, consequently, a smaller overall sampling pool. However, as shown in Figure 4.2, biased sampling achieves a higher fraction of unsafe paths per episode, indicating that its focused exploration is more effective at uncovering unsafe behavior within individual policy rollouts.

To further analyze the behavior of testing over time, we examine how the number of newly identified unsafe start states evolves across iterations. As shown in Figure 4.3, the number of unsafe states found per iteration decreases rapidly after the initial few iterations. This trend is primarily observable in the case of SCS, where sufficient iterations allowed meaningful analysis. In contrast, the INV method had too few iterations to reveal any clear trends. When comparing uniform and biased sampling, both sampling strategies exhibit similar behavior overall, however, in the case of biased sampling, low iteration counts in some benchmarks prevented the identification of clear trends. To understand whether this is due to a diminishing size of the unsafe region or a slowdown in sampling, we consider the discovery rate, i.e., the number of unsafe start states found per path sampled. Figure 4.4 shows the different trends observed across the benchmarks. We see that in some benchmarks the discovery rate also decreases over time, suggesting a converging behavior: as the safe start condition becomes more restrictive and unsafe regions are ruled out, fewer unsafe states remain reachable. This indicates that the testing method is progressively depleting the space of unsafe behaviors. Additionally, we note that the the evaluation of the unsafety condition over the states – which is performed frequently during the testing

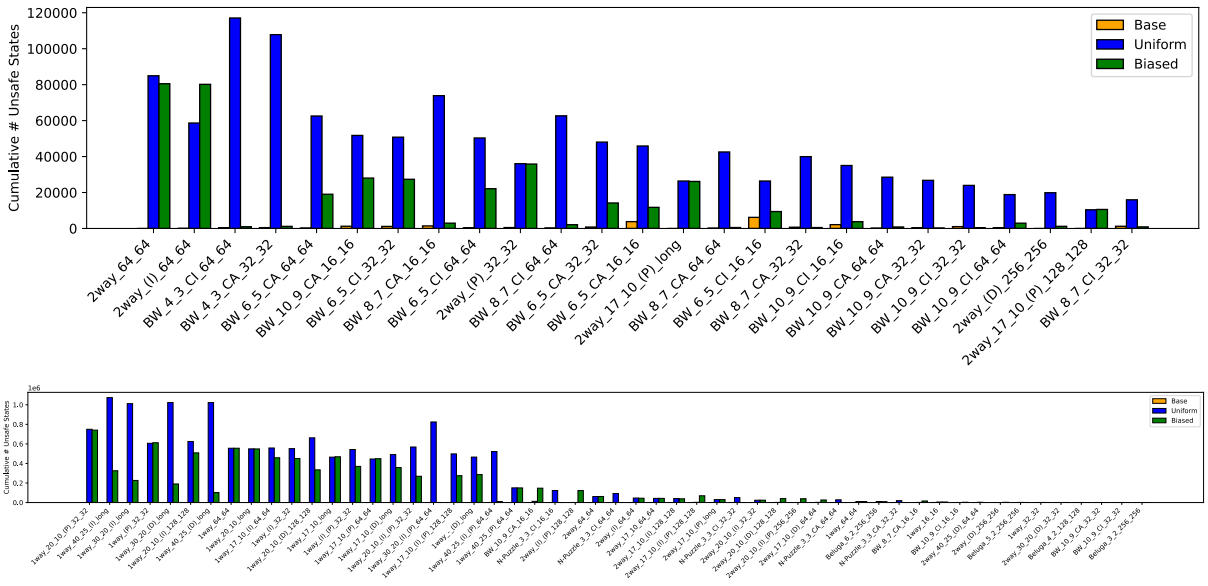


Figure 4.1: Comparison of the number of unsafe states identified with testing methods (Uniform, Biased) and with verification (Base) per 900 seconds of runtime. Top diagram shows the results for INV, bottom diagram shows the results for SCS (number unsafe states in SCS are represented as factors of 10^6). Benchmarks on which memory limit was reached or caused the program to crash were excluded from the results for brevity. This shows that testing is more effective than verification in identifying unsafe start states.

process – does not significantly impact the overall performance of the testing methods. Per testing iteration of 15 minutes, the evaluation of the unsafety condition although increases with each iteration, accounts for less than a microsecond per iteration. This further strengthens the converging behavior observed. However, not all benchmarks show this trend. In some cases, the discovery rate fluctuates irregularly with no clear trend, suggesting either a more uniform distribution of unsafe start states or a larger unsafe region that has not yet been sufficiently explored. This was particularly evident in the 1-Way and 2-Way line domains. A notable exception is observed in one real-valued domain (Bouncing Ball), where we see an initial increase in the discovery rate. This behavior can be attributed to the relaxation of the unsafety specification, which makes it easier to identify unsafe behaviors that were previously excluded, leading to a temporary increase in unsafe discoveries.

Furthermore, the experiments reveal an important limitation encountered during the condition update process: in nearly all benchmarks where the number of USS was high (20k+) memory was exhausted within just 1-10 iterations. This is primarily due to the exponential growth of the conditions as more unsafe start states are identified and removed from the start condition and added to the unsafety condition. This issue is especially pronounced during the update of the unsafety condition. When an unsafe start states, represented as a conjunction of variable assignments,is added to the unsafety condition it introduces a disjunction, thereby breaking its Conjunctive Normal Form (CNF). As such when the condition is transformed back to a CNF formula it grows exponentially in the number of clauses. This results in a rapid increase in memory usage, and in some cases crashing the program.

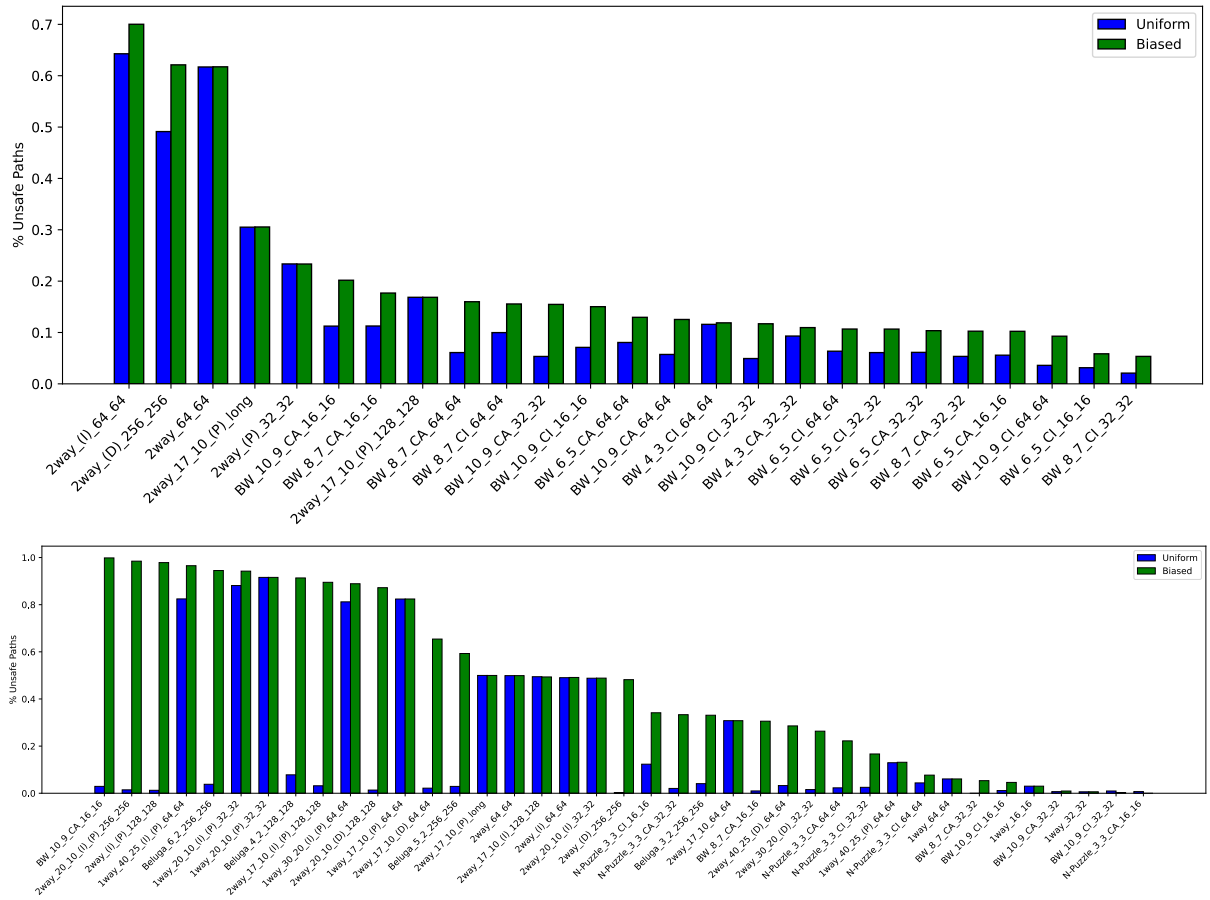


Figure 4.2: Comparison of the percentage of unsafe paths out of total number sampled paths. Top diagram shows the results for invariant strengthening, bottom diagram shows the results for start condition strengthening. Benchmarks on which memory limit was reached or caused the program to crash were excluded from the results for brevity.

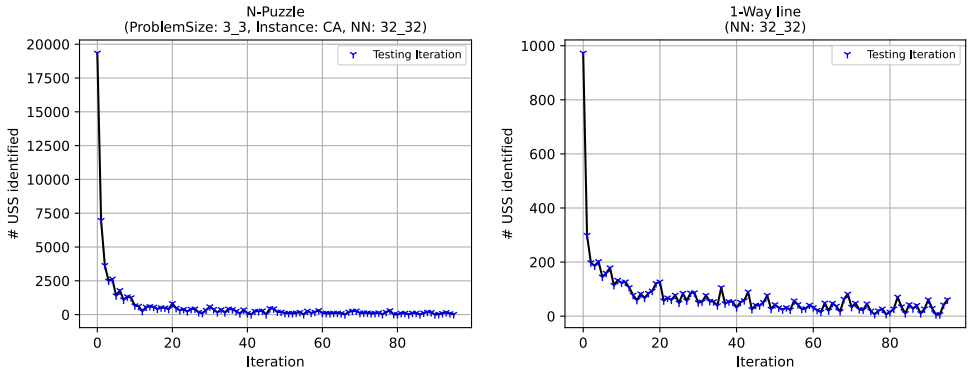


Figure 4.3: Unsafe start states (USS) identified per iteration for two representative benchmarks. Left: Uniform sampling. Right: Biased sampling. Shown to illustrate typical trends where testing finds many USS initially but then the number quickly decreases.

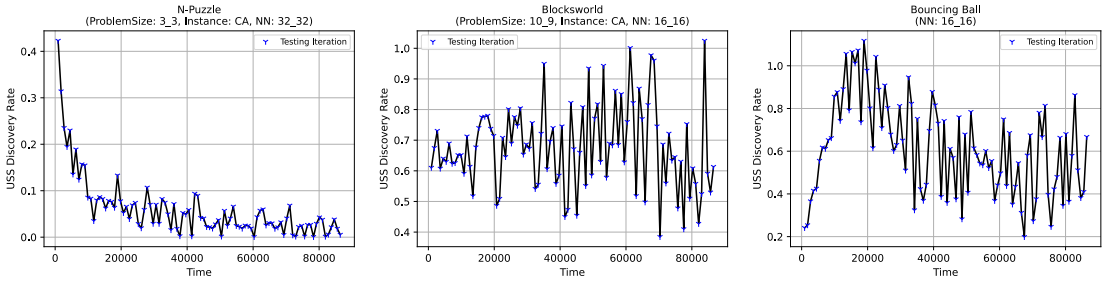


Figure 4.4: Discovery rate of unsafe start states per iteration for three representative benchmarks showing 3 patterns that can be observed in the benchmarks. The discovery rate is defined as the number of unsafe start states found per path sampled. Convergence (Left), no convergence (Middle), and an initial increase (Right). This shows that testing can on occasion exhaust the search space of USS.

While this limitation more apparent with testing, it is in fact inherent to the condition update process itself. In the base configuration, the same issue exists but is often masked by timeouts occurring before the conditions grow prohibitively large. In contrast, the use of testing allows the algorithm to identify unsafe states more efficiently, reaching the same scalability bottleneck in fewer iterations. From this perspective, testing accelerates progress, bringing the underlying limitations of the approach to light more quickly. This highlights a critical need for improving the scalability of the approach, not just by accelerating the identification of unsafe states, but also by managing the growth of the updated conditions. Therefore, in the next chapter we will explore techniques for finding compact representations of the start and unsafety conditions with the use of approximation.

Chapter 5

Compactly Representing the Set of Unsafe States

In this chapter, we introduce the approximation techniques used in this thesis to enhance the scalability of the SSG algorithm. A major challenge in the SSG process is the growing complexity of the start- and unsafety conditions as they are refined over time by removing and adding individual unsafe states. This growth can lead to high memory consumption and increasingly expensive verification steps.

To address this issue, we propose to approximate the set of unsafe states – rather than removing each individual unsafe start state, we remove a compact approximation of them. Specifically, we compute box constraints (axis-aligned hyperrectangles) that generalize from multiple unsafe start states and efficiently represent larger regions of the state space. These box constraints are then used to update the conditions reducing the growth in size.

Algorithm 6 SSG with Safety Testing

Input: π, ϕ_0, ϕ_U
Output: safe ϕ_0

1: **repeat**
2: $\quad U \leftarrow$ `SampleUnsafePaths`(π, ϕ_0, ϕ_U, T)
3: $\quad$ **if** $U = \emptyset$ **then**
4: $\quad\quad U \leftarrow$ `FindUnsafeTransitions`(π, ϕ_0, ϕ_U)
5: $\quad U \leftarrow$ `Approximate`(U)
6: $\quad \phi_0, \phi_U \leftarrow$ `UpdateConditions`(ϕ_0, ϕ_U, U)
7: **until** $U = \emptyset$
8: **return** ϕ_0

We extend the SSG algorithm with a new step that computes an approximation of the unsafe start states, which is then used to update the conditions. The approximation is computed within the `Approximate` method (line 5) whose concrete implementation is described in the subsequent sections. This method returns a single box constraint that represents the set U of unsafe states identified by either testing or verification. To allow for updating the conditions based on box constraints rather than individual states we adapt the `UpdateConditions` method (line 6) to handle box constraints as well.

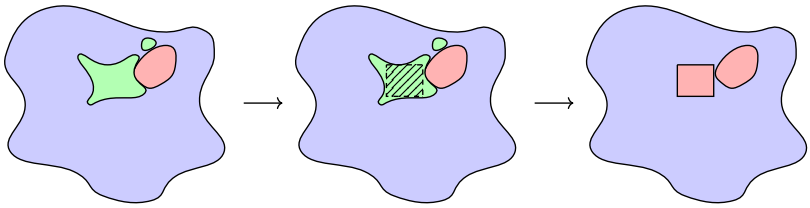


Figure 5.1: Illustration of the under-approximation process. The leftmost figure shows the initial partitioning of the state space into the invariant (blue) and the non-invariant (red), and the set of safety violating states (greed). The middle figure shows the bounded box that is used to approximate the green region. The rightmost figure shows the final partitioning of the state space after updating the conditions.

5.1 Bounded Box

The **Bounded Box Approximation** is an **under-approximation** technique that aims to find a conservative but compact representation of a set of states. Here we are interested in finding a **maximal** box B that is fully contained within the set of states that we aim to approximate. Formally, $\forall B' \subseteq S,\ B' \neq B \Rightarrow B > B'$. Using this approach we do not exclude any safe states from the start condition, however, the main limitation of this approach is that the set of states might be of an irregular non-continuous shape, which will lead the box representation to be ineffective. In the worst case, the box will represent a single state, which regresses the approximation approach to the trivial update of the conditions where we remove and add individual states as formulated in Chaper 3, with the additional cost of computing the box. Simple example of this behavior can be observed in the in Figure 5.1 where the bounded box approximation leaves out many unsafe states due to the structure of the set of unsafe start states.

The idea of Algorithm 7 can be generalized to expanding a box in every direction for every point in a given point cloud, and comparing the volumes of the expanded boxes. Here a state can be viewed a a point in an n-dimensional space, where number of dimensions is equal to the size of $\mathcal{V}$. The algorithm first initializes two vectors (u, ℓ), that correspond to the furthest away points along the positive and negative direction of each axis respectively. These points represent the upper and lower corners of the current maximal box which is represented by $\mathcal{B}(u, \ell)$. Then the algorithm proceeds to iterate over all states s in the set S. Each state is used as the center of the box we aim to expand whose corners are initialized to the center point s (line 3). Inside of the while-loop (lines 5-19) the current box $\mathcal{B}(u', \ell')$ is expanded until a fixed point is reached indicated by the flag `fixedpoint`. The expansion is done by moving the lower corner ℓ' further along the negative direction of each direction one step at the time (line 9), and checking if the box with the updated corner exceeds the the set S (line 10). When the the move of the corner results in violating $\mathcal{B}(u', \ell') \subset S$ we revert the move of the corner (line 11). Otherwise we set the flag `fixedpoint` to false to indicate the loop should continue. The check of $\mathcal{B}(u', \ell') \subset S$ involves checking whether the model bounds were violated by the box, and performing a membership check $s' \in S\ \forall s' \in \mathcal{B}(u', \ell')$ which takes $\mathcal{O}(|\mathcal{B}|)$ time. The upper corner undergoes the same process however we move it in the positive direction (lines 15-19). After the expansion we check whether the current box $\mathcal{B}(u', \ell')$ is larger than the current maximal box $\mathcal{B}(u, \ell)$ (line 20). If it is, we update the current maximal box with

the new box. The algorithm terminates when all states have been checked.

Algorithm 7 Compute maximal bounded box

Input: $S \subseteq \mathcal{S}$
Output: $B \in \mathbb{B}_{\text{lin}}$

$u, \ell \leftarrow []$ *// Initialize upper and lower corners of the maximal box*
for all $s \in S$ **do**
  $u', \ell' \leftarrow s$ *// corners of current box*
  `fixedpoint` $\leftarrow$ False
  while `!fixedpoint` **do**
    `fixedpoint` $\leftarrow$ True
    for all $v \in dom(s)$ **do**
      // attempt to expand lower corner
      $\ell'_v \leftarrow \ell'_v - 1$
      if $\mathcal{B}(u', \ell') \not\subset S$ **then**
        $\ell'_v \leftarrow \ell'_v + 1$
      else
        `fixedpoint` $\leftarrow False$
      // attempt to expand upper corner
      $u'_v \leftarrow u'_v + 1$
      if $\mathcal{B}(u', \ell') \not\subset S$ **then**
        $u'_v \leftarrow u'_v - 1$
      else
        `fixedpoint` $\leftarrow False$
  if $|\mathcal{B}(u', \ell')| > |\mathcal{B}(u, \ell)|$ **then**
    $\ell \leftarrow \ell'$
    $u \leftarrow u'$
$B \leftarrow \bigwedge_{i=1}^{|s|} (v_v \geq \ell_v) \wedge (v_v \leq u_v)$
return B

5.1.1 Experiments and Results

To evaluate the effectiveness of the bounded box under-approximation method, we apply it within the SSG algorithm using only the uniform sampling precheck. As shown in the previous chapter, uniform sampling consistently identified a higher number of unsafe states compared to biased sampling, leading to higher rates of memory exhaustion caused by the growth of conditions. This makes it a particularly suitable setting for studying the impact of approximation. Additionally, we do not consider benchmarks with real-valued domains here, as the bounded box approximation is not applicable in such cases. An extensive list of results can be found in Appendix 7

Findings. We were again unable to generate a safe start condition; however, we observe that in the setting of INV a single bounded box constraint can sometimes exclude hundreds of unsafe start states. Since one box constraint is roughly equivalent in size to a single state constraint, this translates to a 100x reduction in the number of constraints needed. This

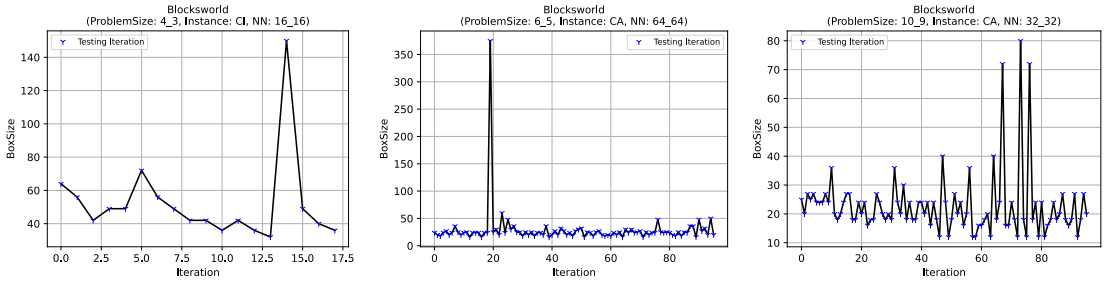


Figure 5.2: Plots for various benchmarks of the **Blocksworld** domain showing sudden increase in the number of states captured by a single box constraint (BoxSize), suggesting structured regions of the USSs.

represents a substantial improvement over previous approaches, which required removing each unsafe state individually.

Despite this advantage in compactness, the total number of unsafe states eliminated in this configuration remains significantly lower than in earlier experiments. This is because a large portion of the runtime per iteration is spent computing the bounded box itself – a computationally expensive task – while the resulting box often covers only a small portion of the unsafe states ($\ll 1\%$). Even when compared to the base SSG algorithm, the bounded box approximation processes far fewer unsafe states overall.

Nevertheless, it is worth noting that some benchmarks exhibit a spiking behavior, where the bounded box approximation captures significantly larger regions of unsafe start states. This results in highly compact and efficient representations. Such behavior was particularly evident in benchmarks from the Blocksworld domain, as shown in Figure 5.2. These cases suggest that although many unsafe regions are highly irregular and thus difficult to approximate compactly, others display more structure and regularity, making them a better fit to the bounded box method. Therefore, while bounded box approximation is not a universally effective solution, it may still prove valuable in structured domains, or in specific regions of the state space.

In the setting of SCS, the results are even less favorable. Across all benchmarks, the bounded boxes were able to only represent single digit of unsafe start states, and more often than not only a single state. This suggests that the region of the state space that is constrained to the given start condition is highly irregular.

We therefore conclude that, due to high computational overhead paired with low yield, the bounded box approximation does not offer sufficient scalability improvements for the SSG algorithm.

5.2 Bounding Box

Given the limitations identified in the previous section, we shift our focus from finding a maximal safe start condition to identifying any safe start condition. In other words, we now prioritize scalability and feasibility over completeness. This relaxation allows us to explore over-approximation techniques. Specifically, we introduce the **Bounding Box Approximation** method, which aims to efficiently represent a set of unsafe states by computing a single box constraint that contains all unsafe states.

In this approach we compute a **minimal** box B that contains the set of states that

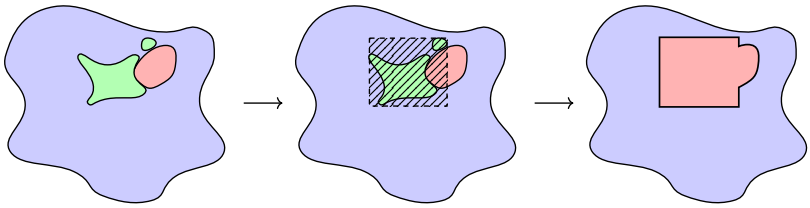


Figure 5.3: Illustration of the over-approximation process. The leftmost figure shows the initial partitioning of the state space into the invariant (blue) and the non-invariant (red), and the set of safety violating states (green). The middle figure shows the bounding box that is used to approximate the green region. The rightmost figure shows the final partitioning of the state space after updating the conditions.

we aim to approximate. Formally, the bounding box is a boolean formula $B \in \mathbb{B}_{\text{lin}}$ of the form $\bigwedge_{i=1}^{n}(v_i \geq l_i) \wedge (v_i \leq u_i)$, where l_i and u_i are the lower and upper bounds of the box for each variable v_i. The bounding box approximation is a simple and efficient way to represent a set of states, but it may not be very precise, as it can include states that are not part of the original set. Therefore, when used in the context of the Invariant- and Start condition Strengthening, the algorithms may falsely produce an empty start condition indicating that the policy is not safe for any state. The reason for this is that bounding box approximation may remove safe states from the start and add them to the set of unsafe states. This can be observer in the example in Figure 5.3 where the bounding box approximation includes large areas of states that are not necessarily unsafe.

Algorithm 8 Compute minimal bounding box

Input: $S \subseteq \mathcal{S}$
Output: $B \in \mathbb{B}_{\text{lin}}$

$Bounds \leftarrow \{(\infty, -\infty), \ldots, (\infty, -\infty)\}$
for all $s \in S$ **do**
    for all $v \in dom(s)$ **do**
        if $s(v) < Bounds_{v,0}$ **then**
            $Bounds_{v,0} \leftarrow s(v)$
        if $s(v) > Bounds_{v,1}$ **then**
            $Bounds_{v,1} \leftarrow s(v)$
$B \leftarrow \bigwedge_{v \in \mathcal{V}}(v \geq Bounds_{v,0}) \wedge (v \leq Bounds_{v,1})$
return B

The computation of the bounding box, as depicted in Algorithm 8, is done by iterating over all states s in the set S and updating the bounds for each variable $v \in dom(s)$ based on the minimum and maximum values found in the states. The algorithm returns a boolean formula that represents the bounding box. While this method is, in principle, compatible with real-valued domains – unlike the bounded box approximation – support for real-valued variables was not implemented at the time of writing this thesis.

5.2.1 Experiments and Results

Following the same experimental setup as in previous section, run the SSG algorithm with the bounding box approximation method.

Findings. In contrast to all previous experiments, we observe that for some benchmarks *over-approximation was able to generate a safe non-empty start condition.* When paired with Invariant Strengthening, the algorithm generated a non-trivial safe start condition for 13 policies, however it also falsely concluded that 11 policies are unsafe for any state in the state space. SCS with the bounding box approximation was able to generate, in comparison to the base SCS algorithm, additional 8 safe start conditions, and producing one false negative. The results of the approximation methods are summarized in Table 5.1.

An additional observation is that verification times in the INV setting are very fast where the start condition determined safe (between 1-2 seconds). The reason for this is that we never perform verification on policy induced transitions that include the NN encoding. Instead the Z3 solver is able to determine that there are generally no unsafe transitions. This suggests that this approach restricted the start condition to an isolated region of the state space. While these start conditions guarantee safety, it is unclear whether they reflect any meaningful regions of the state space. This highlights a potential limitation of the bounding box approximation, by prioritizing scalability it may lead to overly conservative start conditions limiting the capabilities of the policy.

Table 5.1: Summary of bounding box over-approximation results, showing total runtime (Time) in seconds and whether the over-approximation successfully generated a safe start condition (Safe) for each benchmark. Columns INV and SCS report results for invariant- and start condition strengthening, respectively. For brevity benchmarks where both INV and SCS timed out or ran out of memory were omitted.

Benchmark	NN	INV		SCS	
		Safe	Time	Safe	Time
Beluga 3	256	×	900.0	?	-
Beluga 4	128	×	901.0	?	-
Beluga 5	256	×	901.0	?	-
Beluga 6	256	×	901.0	?	-
10Blocks (CA)	16	×	1800.0	×	901.0
	32	×	2701.0	?	-
	64	×	2700.0	?	-
10Blocks (CI)	16	×	1800.0	✓	2717.0
	32	×	2701.0	?	-
	64	×	2701.0	?	-
4Blocks (CA)	16	×	1801.0	✓	946.0
	32	×	1800.0	✓	1154.0
	64	×	1801.0	✓	3454.0

Benchmark	NN	INV		SCS	
		Safe	Time	Safe	Time
4Blocks (CI)	16	×	1800.0	✓	918.0
	32	×	1800.0	✓	942.0
	64	×	1800.0	✓	1042.0
6Blocks (CA)	16	×	1800.0	✓	28798.0
	32	×	1800.0	✓	43142.0
	64	×	1800.0	?	-
6Blocks (CI)	16	×	1800.0	✓	33248.0
	32	×	1800.0	✓	8867.0
	64	×	1800.0	✓	12893.0
8Blocks (CA)	16	×	1800.0	?	-
	32	×	1800.0	?	-
	64	×	1800.0	?	-
8Blocks (CI)	32	×	1800.0	?	-
	64	×	1800.0	?	-
8Puzzle (CA)	16	✓	2700.0	?	-
	32	✓	1800.0	✓	2700.0
	64	✓	2700.0	?	-
8Puzzle (CI)	16	✓	1800.0	?	-
	32	✓	2700.0	✓	2701.0
	64	✓	1800.0	×	900.0
1-way 15-10 (det)	64	×	900.0	?	-
1-way 15-10	64	×	901.0	?	-
1-way 15-10 (fail-dec)	long	×	901.0	?	-
1-way 15-10 (icy-drop)	32	×	902.0	?	-
1-way 15-10 (icy-drop-park)	32	✓	1801.0	?	-
1-way 15-10 (park)	32	✓	1801.0	?	-
1-way 17-10	long	×	901.0	?	-
1-way 17-10 (fail-dec)	long	×	902.0	?	-
1-way 17-10 (icy-drop)	64	×	901.0	?	-
1-way 17-10 (icy-drop-park)	128	✓	1800.0	?	-
1-way 17-10 (park)	64	✓	1800.0	?	-
1-way 20-10	long	×	901.0	?	-
1-way 20-10(fail-dec)	128	×	901.0	?	-
1-way 20-10 (icy-drop)	128	×	901.0	?	-
1-way 20-10 (icy-drop-park)	32	✓	1801.0	✓	1801.0
1-way 20-10 (park)	32	✓	1801.0	?	-
1-way 30-20	64	×	901.0	?	-
1-way 30-20 (park)	64	✓	1800.0	✓	1801.0
1-way 40-25	16	×	902.0	?	-
1-way 40-25 (fail-dec)	long	×	902.0	?	-
1-way 40-25 (icy-drop)	long	×	902.0	?	-
1-way 40-25 (icy-drop-park)	64	✓	1801.0	?	-
1-way 40-25 (park)	64	✓	1801.0	?	-

Benchmark	NN	INV		SCS	
		Safe	Time	Safe	Time
2-way 17-10	64	×	900.0	✓	1800.0
2-way 17-10 (fail-dec)	64	×	900.0	?	-
2-way 17-10 (park)	128	✓	1800.0	?	-
	long	✓	1800.0	?	-
2-way 20-10 (icy-drop)	32	?	-	✓	1823.0
2-way 40-25 (fail-dec)	64	×	901.0	?	-
2-way 15-10	64	×	900.0	✓	1850.0
2-way 15-10 (fail-dec)	256	×	900.0	?	-
2-way 15-10 (icy-drop)	64	×	900.0	✓	6535.0
2-way 15-10 (icy-drop-park)	128	✓	1801.0	?	-
2-way 15-10(park)	32	✓	1800.0	?	-

Chapter 6

Conclusion

The central problem addressed in this thesis is the automatic generation of safe start conditions for neural action policies in planning domains. We developed a sound and complete[1] method that iteratively derives start conditions that are guaranteed to be safe with respect to a given unsafety condition. We enhanced it by leveraging testing to quickly identify unsafe states and improved scalability by computing compact representations of the identified unsafe states. Specifically, the contributions of this thesis are:

- Formalization of safe start condition generation.
- Development of two verification-based methods: Invariant Strengthening (INV) and Start Condition Strengthening (SCS).
- Integration of testing and approximation to improve scalability.
- Experimental evaluation on multiple benchmarks.

The base methods, INV and SCS, form the theoretical core of this thesis. INV offers a lightweight and conceptually simple approach, allowing for quick refinement of the start condition by iteratively excluding unsafe start states. One of its key advantages is that it requires only an unsafety condition as input and can automatically construct a safe start condition from it, thus providing a more general safe policy execution context. SCS extends this idea by performing multistep reasoning, enabling it to focus its effort on a smaller, predefined region of the state space. Both methods are formally sound for discrete and continuous state spaces as well as complete for discrete state spaces, serving as foundational components of the safe start condition generation framework. However, despite their strong formal guarantees, the experimental results reveal that they do not scale well in practice. For INV, the verification effort grows exponentially with each additional unsafe state, quickly becoming infeasible. SCS, while more expressive, incurs even greater overhead due to the added complexity of multistep verification. These limitations underscore the need for more scalable techniques and motivate the integration of testing and approximation methods explored later in this work.

To address the scalability challenges of the base methods, we introduced testing-based approaches that provide an inexpensive precheck for identifying unsafe policy paths. These methods proved highly effective in practice, quickly uncovering large numbers of unsafe

[1] in the case of infinite state spaces the SSG algorithm is not complete cf. 3.1.1

start states with minimal computational overhead. We showed that biased sampling achieves a higher fraction of unsafe paths, making it efficient at finding unsafe trajectories, while uniform sampling yields a larger overall number of unique unsafe start states. However, a key limitation was identified in the Start State Generation (SSG) algorithm: as the start conditions are iteratively updated to exclude unsafe start states, the complexity of these conditions grows substantially. This growth eventually leads to resource exhaustion, limiting the scalability of the approach and motivating further work on condition approximation techniques.

To tackle this limitation, we proposed an approximation-based approach that replaces individual unsafe start states during condition updates with a more compact geometric representation of the entire unsafe set. Our first method employs an under-approximation technique that computes a maximal bounded box (axis-aligned hyperrectangle) for a given set of unsafe start states. This strategy effectively manages the growth of the conditions by substituting the repeated removal and addition of individual states with a single box constraint, which was shown to encompass up to hundreds of unsafe states. However, due to the often irregular and non-convex shape of unsafe regions, the bounded box covers only a small fraction of the unsafe start states identified in practice. Combined with the computational overhead required to compute the box, this under-approximation method proved impractical for large or complex sets of unsafe start states. Nevertheless, it represents an important step toward scalable geometric approximations of unsafe state spaces.

We then shift our focus to an over-approximation of the set of unsafe start states, relaxing the completeness requirement and the need to find a maximal safe start condition. In this approach, we compute a bounding box that fully encapsulates the set of unsafe start states identified. This results in highly compact box constraints that, while potentially including some safe states, provide a scalable and efficient representation of unsafe regions. Our experiments demonstrate that this method scales well in practice and, unlike previous approaches, is capable of generating non-trivial safe start conditions. Although this relaxation sacrifices completeness and optimality, and can yield start conditions covering less useful areas of the state space for policy deployment, it represents a practical trade-off that enables safe policy execution in complex domains where exact methods fail.

6.1 Future Work

Several promising directions remain for future research that could further improve the scalability issues that arise from the growth of the conditions:

- **Improved Under-Approximation:** Developing and using more efficient algorithms for computing bounded boxes or other methods that can under-approximate non-convex geometric sets, may reduce the computational overhead and improve applicability.
- **Restarting Schemes:** A strategy that periodically recomputes the bounded box after a certain number of iterations based on the sofar accumulated unsafe states could help improve the size of the computed boxes and thus mitigate the growth of the conditions even further.

- **Tighter Over-Approximations:** Future work could investigate more expressive geometric representations e.g., computation of convex hulls or unions of boxes, to approximate unsafe regions more accurately while retaining scalability.

Concluding Remarks

This thesis demonstrates the feasibility of formally generating safe start conditions for neural policies, bridging symbolic verification with testing- and approximation methods. While exact methods struggle with scalability, approximation-based techniques open a practical path forward. Ensuring safety in neural planning remains a critical and open challenge, and this work provides foundational steps into an under-explored area of this field.

Chapter 7

Appendix

Lemma 7.0.1 (Disjointness)**.** The invariant and non-invariant sets remain disjoint for all iterations i, i.e.,

$$\left[\!\left[\phi_0^{(i)}\right]\!\right] \cap \left[\!\left[\phi_U^{(i)}\right]\!\right] = \emptyset.$$

Proof. We proceed by induction.

Base case ($i = 0$): By initialization, we have

$$\phi_0^{(0)} := \neg\phi_U \quad \text{and} \quad \phi_U^{(0)} := \phi_U,$$

thus

$$\left[\!\left[\phi_0^{(0)} \wedge \phi_U^{(0)}\right]\!\right] \Leftrightarrow [\![\neg\phi_U \wedge \phi_U]\!] \Leftrightarrow [\![\bot]\!] \Leftrightarrow \emptyset.$$

Induction hypothesis: Assume that for some n, the sets are disjoint:

$$\left[\!\left[\phi_0^{(n)}\right]\!\right] \cap \left[\!\left[\phi_U^{(n)}\right]\!\right] = \emptyset.$$

Induction step ($i = n + 1$): The update rule is:

$$\phi_0^{(n+1)} := \phi_0^{(n)} \wedge \neg\phi_s, \quad \phi_U^{(n+1)} := \phi_U^{(n)} \vee \phi_s.$$

We compute:

$$\begin{aligned}
\left[\!\left[\phi_0^{(n+1)} \wedge \phi_U^{(n+1)}\right]\!\right] &\Leftrightarrow \left[\!\left[(\phi_0^{(n)} \wedge \neg\phi_s) \wedge (\phi_U^{(n)} \vee \phi_s)\right]\!\right] \\
&\Leftrightarrow \left[\!\left[(\phi_0^{(n)} \wedge \neg\phi_s \wedge \phi_U^{(n)}) \vee (\phi_0^{(n)} \wedge \neg\phi_s \wedge \phi_s)\right]\!\right] \\
&\Leftrightarrow (\left[\!\left[\phi_0^{(n)} \wedge \phi_U^{(n)}\right]\!\right] \cap [\![\neg\phi_s]\!]) \cup \left[\!\left[(\phi_0^{(n)} \wedge \neg\phi_s \wedge \phi_s)\right]\!\right] \\
&\Leftrightarrow (\emptyset \cap [\![\neg\phi_s]\!]) \cup \left[\!\left[\phi_0^{(n)} \wedge \bot\right]\!\right] \quad \text{(by I.H.)} \\
&\Leftrightarrow [\![\bot]\!] \\
&\Leftrightarrow \emptyset.
\end{aligned}$$

□

Lemma 7.0.2 (Joint exhaustiveness)**.** The invariant and non-invariant sets cover the entire state space for all iterations i, i.e.,

$$\left[\!\left[\phi_0^{(i)}\right]\!\right] \cup \left[\!\left[\phi_U^{(i)}\right]\!\right] = \mathcal{S}.$$

Proof. We proceed by induction.

Base case ($i = 0$): By initialization, we have

$$\phi_0^{(0)} := \neg\phi_U \quad \text{and} \quad \phi_U^{(0)} := \phi_U,$$

thus

$$\left[\!\left[\phi_0^{(0)} \vee \phi_U^{(0)}\right]\!\right] \Leftrightarrow [\![\neg\phi_U \vee \phi_U]\!] \Leftrightarrow [\![\top]\!] \Leftrightarrow \mathcal{S}.$$

Induction hypothesis: Assume that for some n, the sets are disjoint:

$$\left[\!\left[\phi_0^{(n)}\right]\!\right] \cup \left[\!\left[\phi_U^{(n)}\right]\!\right] = \mathcal{S}.$$

Induction step ($i = n + 1$):

$$\begin{aligned}
\left[\!\left[\phi_0^{(n+1)} \vee \phi_U^{(n+1)}\right]\!\right] &\Leftrightarrow \left[\!\left[(\phi_0^{(n)} \wedge \neg\phi_s) \vee (\phi_U^{(n)} \vee \phi_s)\right]\!\right] \\
&\Leftrightarrow \left[\!\left[(\phi_0^{(n)} \vee \phi_U^{(n)} \vee \phi_s) \wedge (\neg\phi_s \vee \phi_U \vee \phi_s)\right]\!\right] \\
&\Leftrightarrow [\![(\top \vee \phi_s) \wedge (\phi_U \vee \top)]\!] \quad \text{(by I.H.)} \\
&\Leftrightarrow [\![\top]\!] \\
&\Leftrightarrow \mathcal{S}.
\end{aligned}$$

□

Lemma 7.0.3 (Inductive Invariant Closure)**.** After any iteration $i \in \mathbb{N}_0$ it holds that,

$$\forall s \in \left[\!\left[\phi_0^{(i)}\right]\!\right], \forall s' \in s[\pi(s)] : s' \notin \left[\!\left[\phi_U^{(i)}\right]\!\right],$$

i.e., $\phi_0^{(i)}$ is closed under policy transitions w.r.t $\phi_U^{(i)}$.

Proof. We proceed by induction.

Base case ($i = 0$): By initialization, we have

$$\phi_0^0 := \neg\phi_U,$$

together with lemma 7.0.1, and lemma 7.0.2, the update rule removes the states satisfying the encoding of the transition $\phi_0 \xrightarrow[\pi]{l} \phi_U$. Thus, by the update rule, we have that:

$$\forall s \in [\![\phi_0^0]\!], \forall s' \in s[\pi(s)] : s' \notin [\![\phi_U^0]\!].$$

Induction hypothesis: Assume that for some n, the invariant is closed under policy transitions:

$$\forall s \in \left[\!\left[\phi_0^{(n)}\right]\!\right], \forall s' \in s[\pi(s)] : s' \notin \left[\!\left[\phi_U^{(n)}\right]\!\right]$$

Induction step ($i = n + 1$): According to the update rule is the invariant and non-invariant are constructed as follows:

$$\left[\!\left[\phi_0^{(n+1)}\right]\!\right] := \left[\!\left[\phi_0^{(n)} \wedge \neg\phi_{s_U}\right]\!\right] = \left[\!\left[\phi_0^{(n)}\right]\!\right] \cap [\![\neg\phi_{s_U}]\!]$$

$$\left[\!\left[\phi_U^{(n+1)}\right]\!\right] := \left[\!\left[\phi_0^{(n)} \vee \phi_{s_U}\right]\!\right] = \left[\!\left[\phi_U^{(n)}\right]\!\right] \cup [\![\phi_{s_U}]\!],$$

where $\phi_{s_U} = \bigwedge_{s \models \phi_0 \xrightarrow[\pi]{l} \phi_U} \phi_s$.

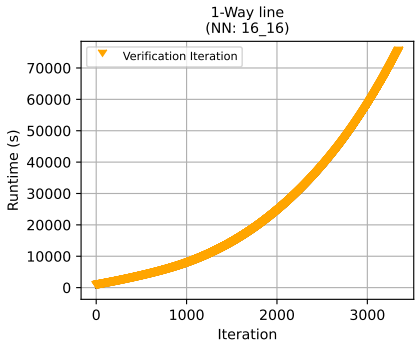
Consider any state $s \in \left[\!\left[\phi_0^{(n+1)}\right]\!\right]$, any policy induced successor s' of s is $s' \notin \left[\!\left[\phi_U^{(n)}\right]\!\right]$ by the induction hypothesis and $s' \notin [\![\phi_{s_U}]\!]$ by the update rule. Thus, we have:

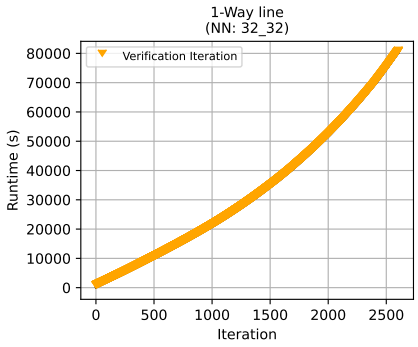
$$\forall s \in \left[\!\left[\phi_0^{(n+1)}\right]\!\right], \forall s' \in s[\pi(s)] : s' \notin \left[\!\left[\phi_U^{(n+1)}\right]\!\right]$$

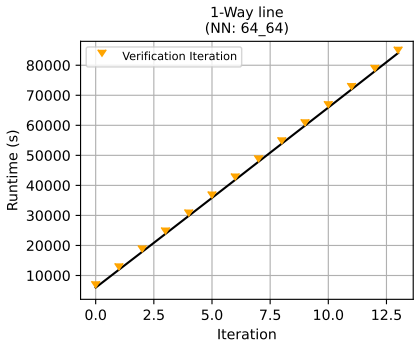
□

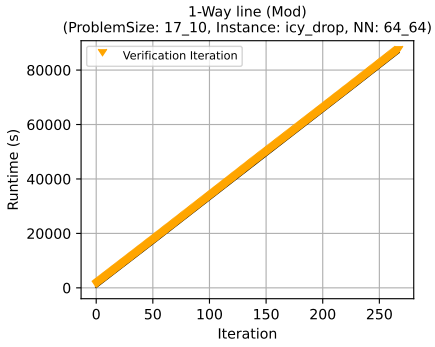
- Results for chapter 3 -

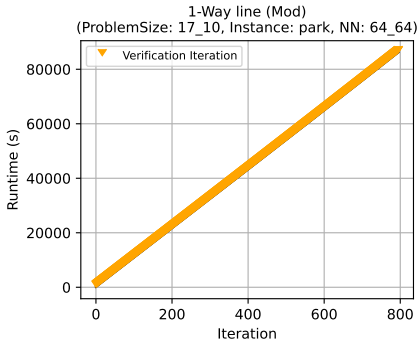
Invariant Strengthening

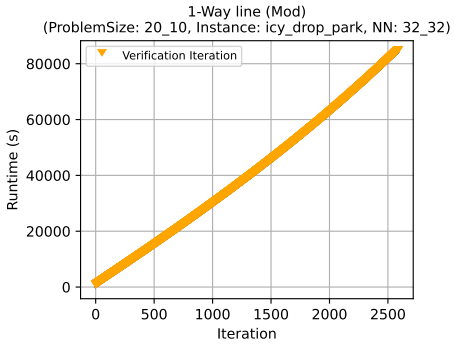

1-Way line
(NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

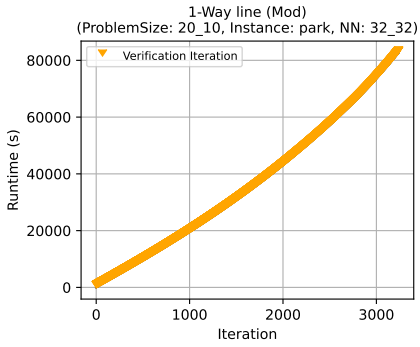

1-Way line
(NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

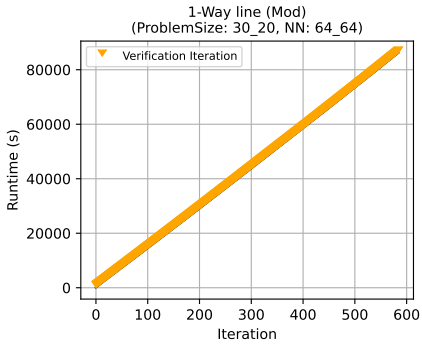

1-Way line
(NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

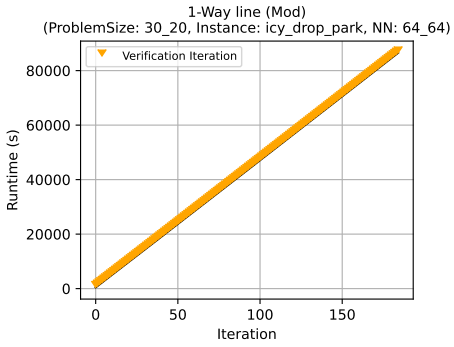

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

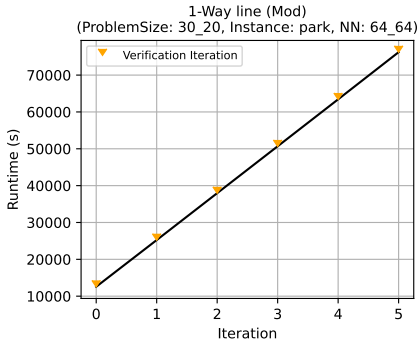

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

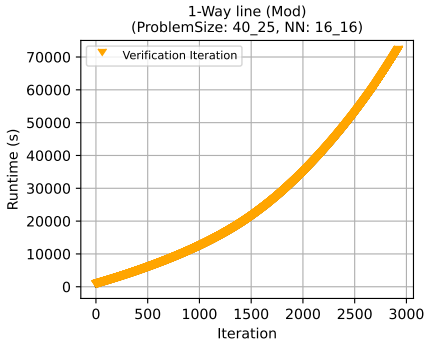

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

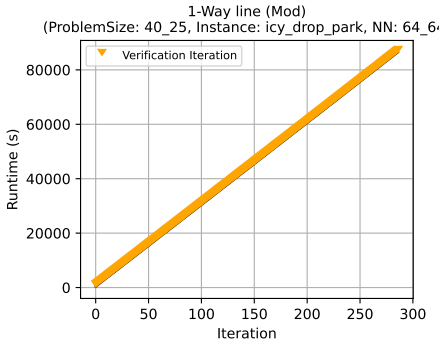

1-Way line (Mod)
(ProblemSize: 20_10, Instance: park, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

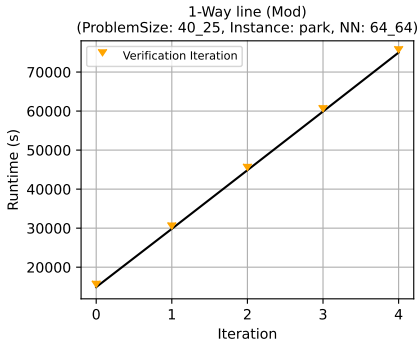

1-Way line (Mod)
(ProblemSize: 30_20, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

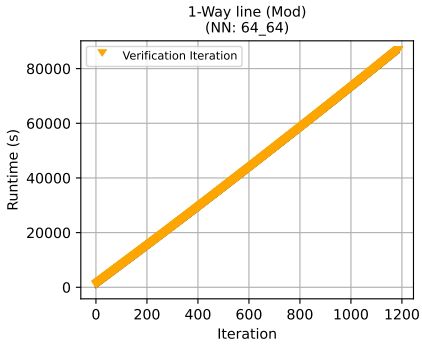

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop_park, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

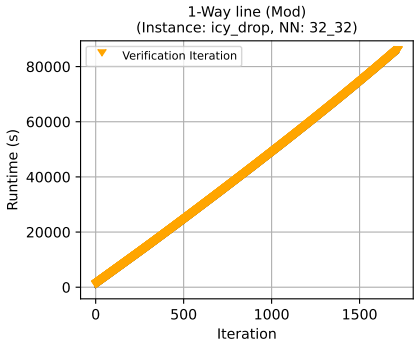

1-Way line (Mod)
(ProblemSize: 30_20, Instance: park, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

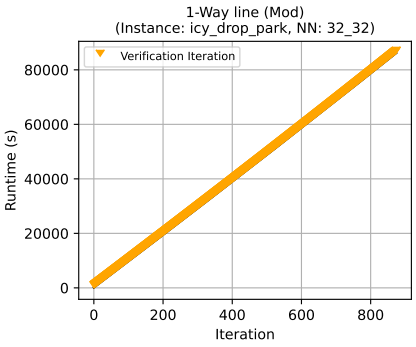

1-Way line (Mod)
(ProblemSize: 40_25, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

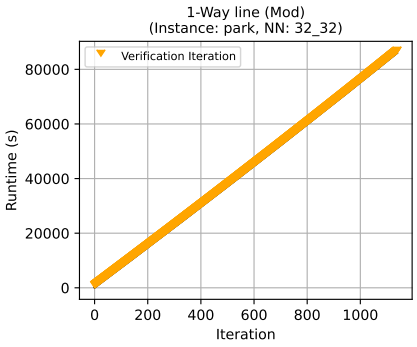

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

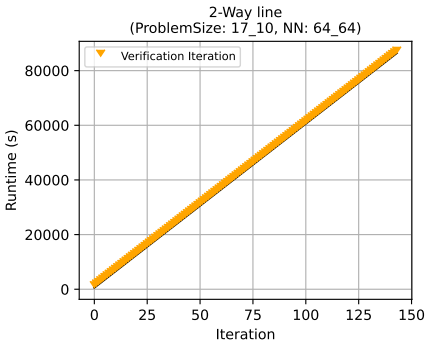

1-Way line (Mod)
(ProblemSize: 40_25, Instance: park, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

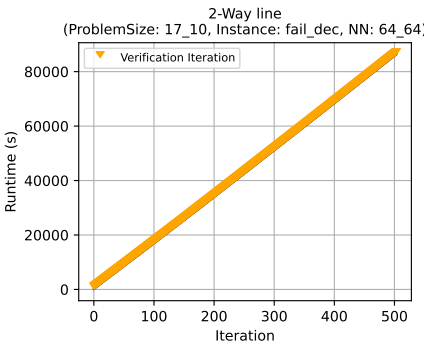

1-Way line (Mod)
(NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

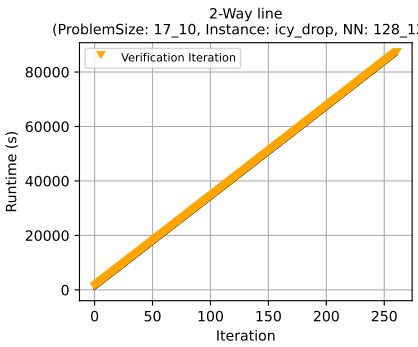

1-Way line (Mod)
(Instance: icy_drop, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

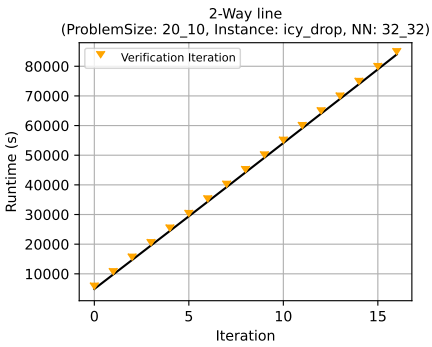

1-Way line (Mod)
(Instance: icy_drop_park, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

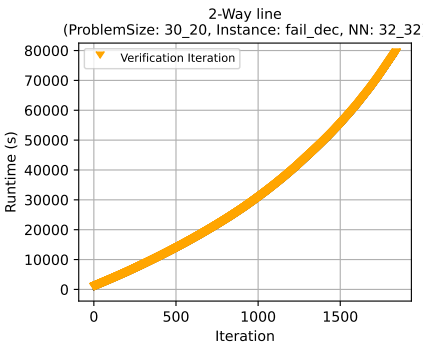

1-Way line (Mod)
(Instance: park, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

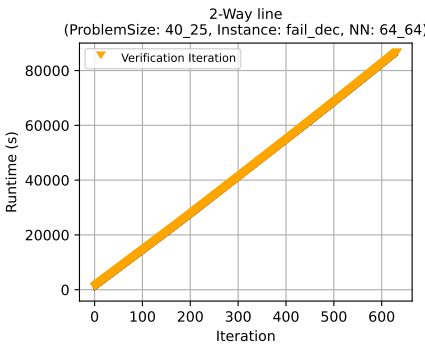

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

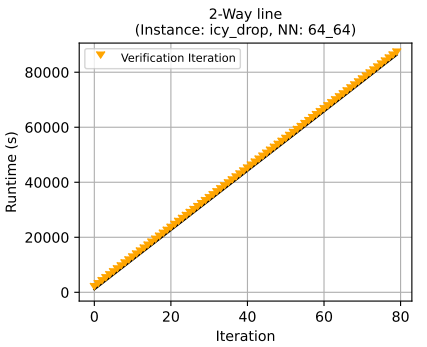

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

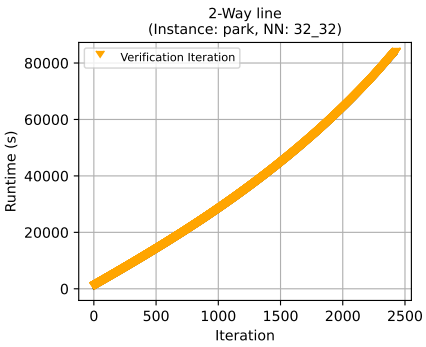

2-Way line
(ProblemSize: 17_10, Instance: icy_drop, NN: 128_128)
Verification Iteration
Runtime (s)
Iteration

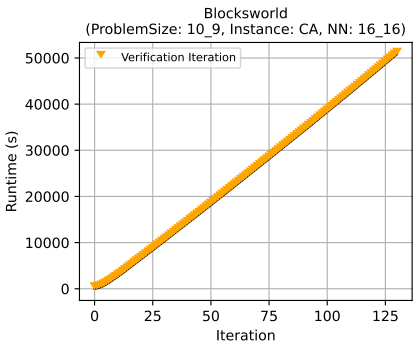

2-Way line
(ProblemSize: 20_10, Instance: icy_drop, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

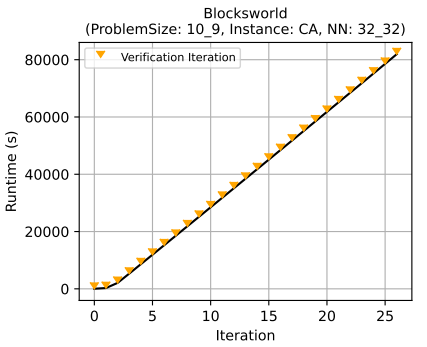

2-Way line
(ProblemSize: 30_20, Instance: fail_dec, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

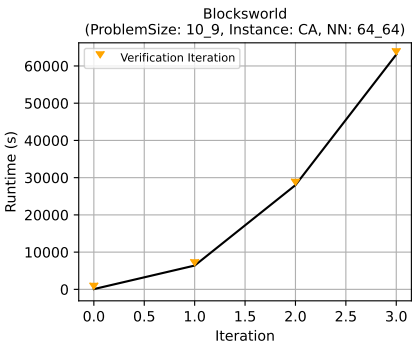

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

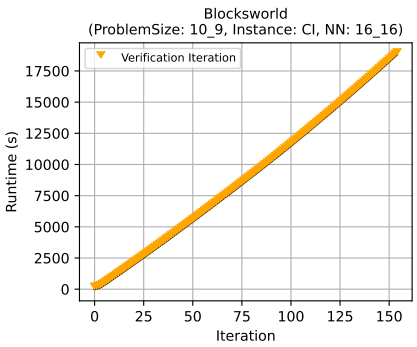

2-Way line
(Instance: icy_drop, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

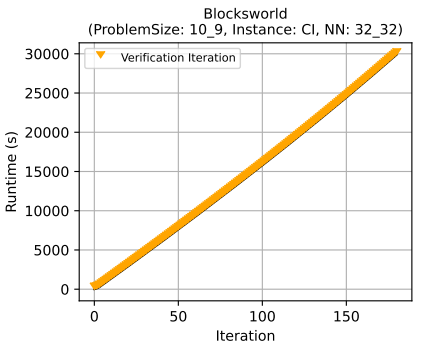

2-Way line
(Instance: park, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

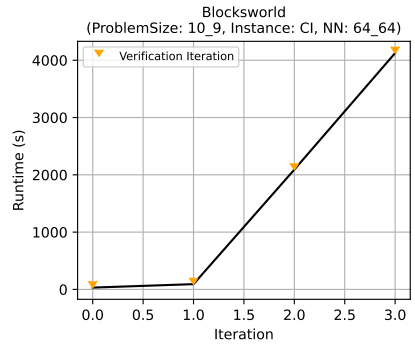

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

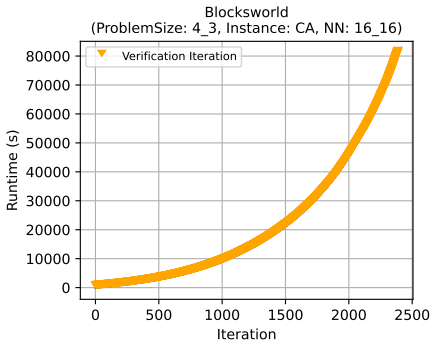

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

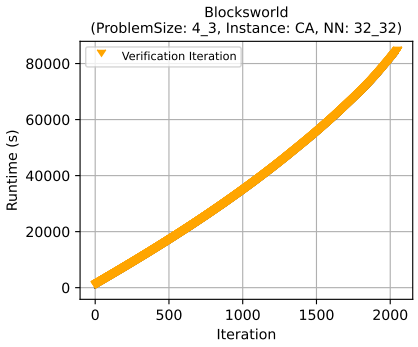

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

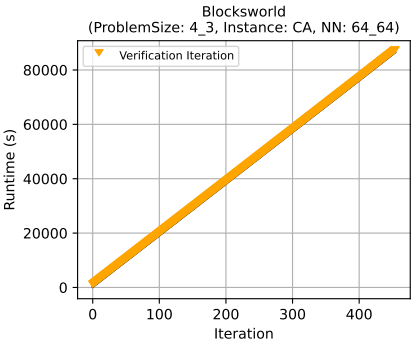

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

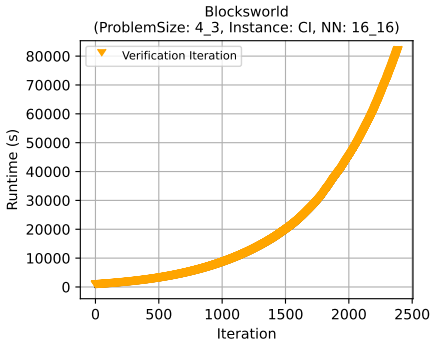

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

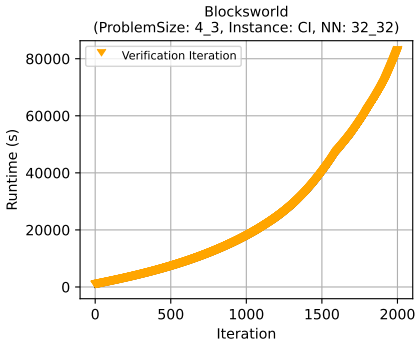

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

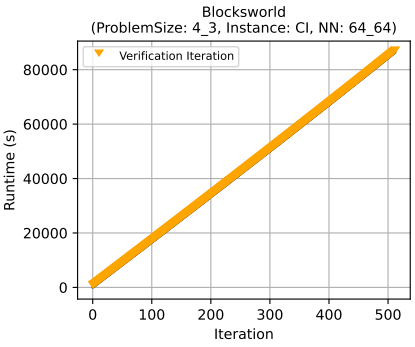

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

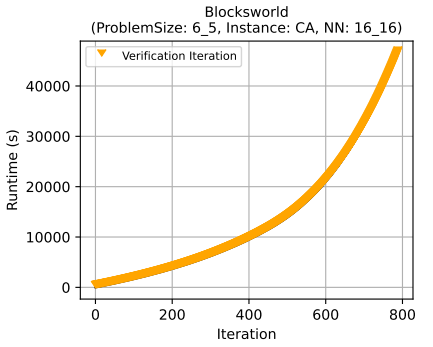

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

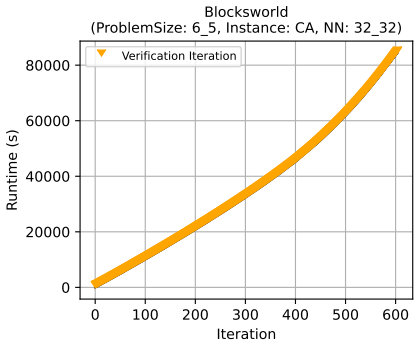

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

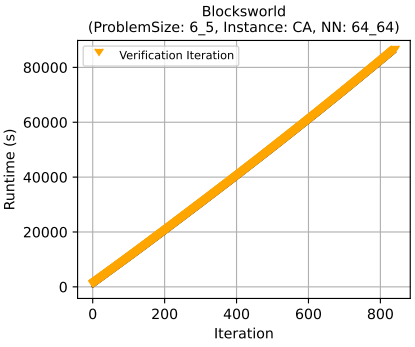

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

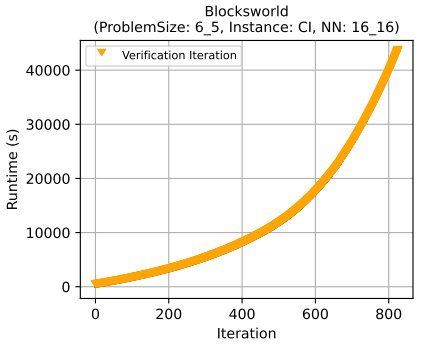

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

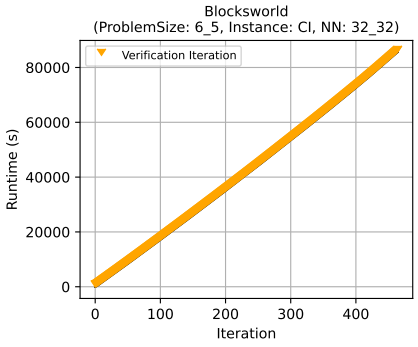

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

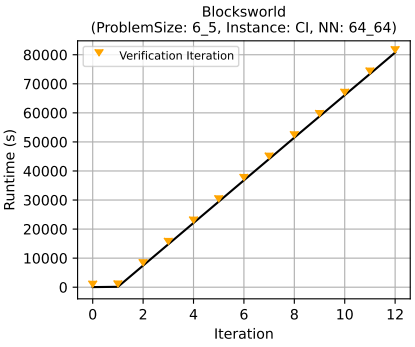

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

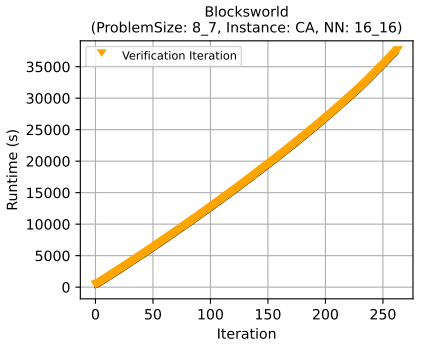

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

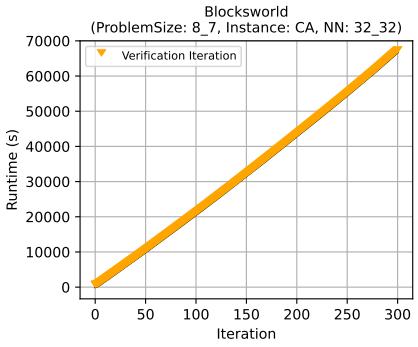

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

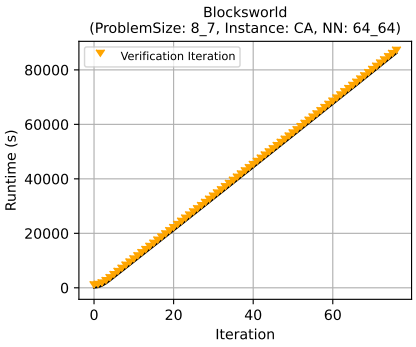

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

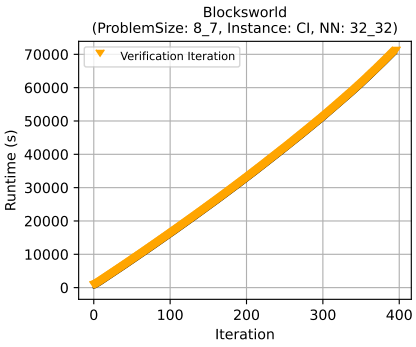

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

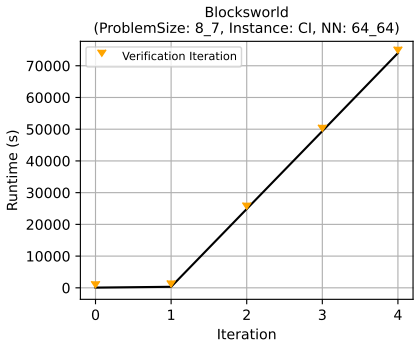

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

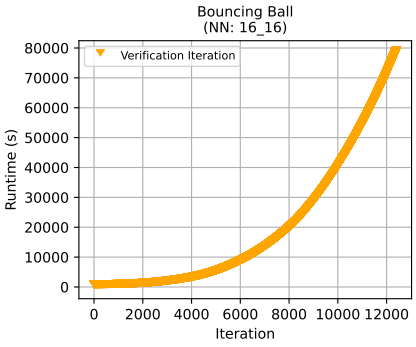

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

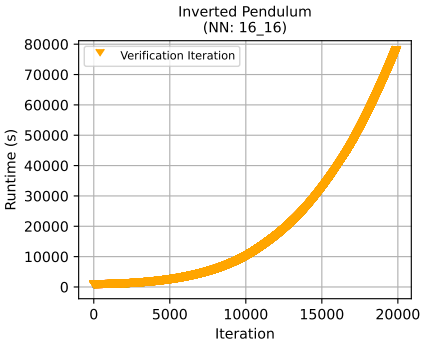

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

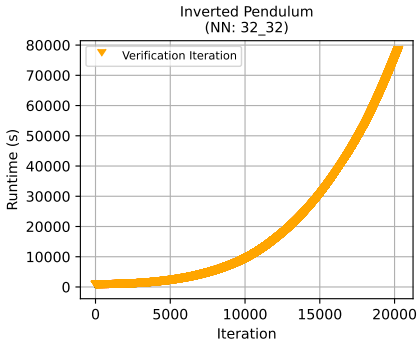

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

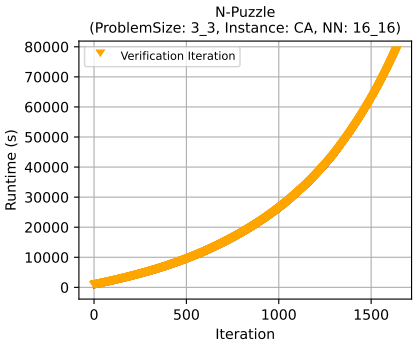

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

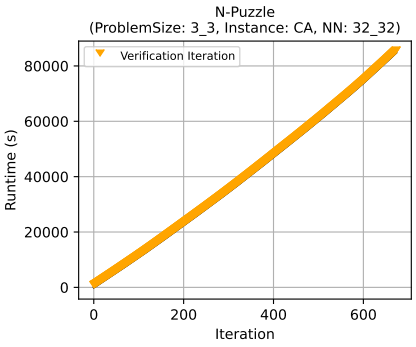

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

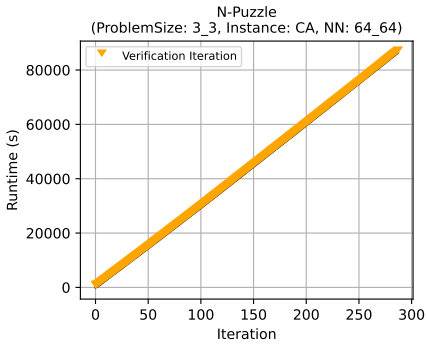

Bouncing Ball
(NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

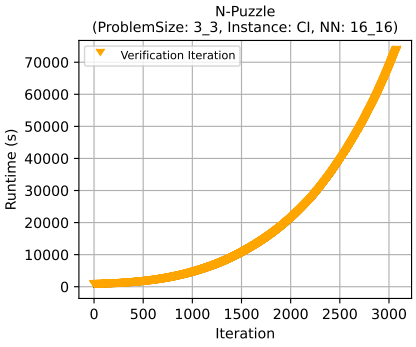

Inverted Pendulum
(NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

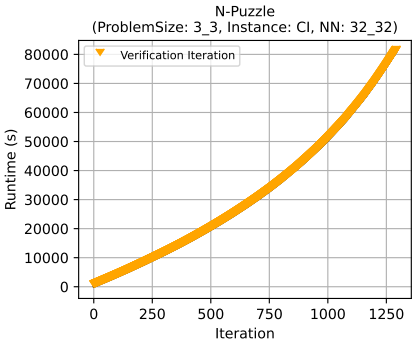

Inverted Pendulum
(NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

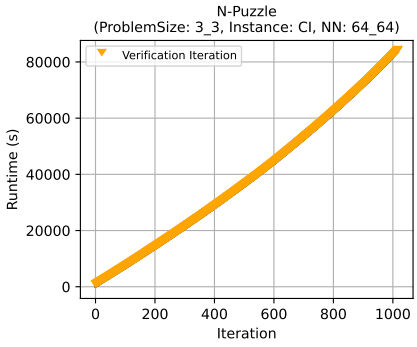

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

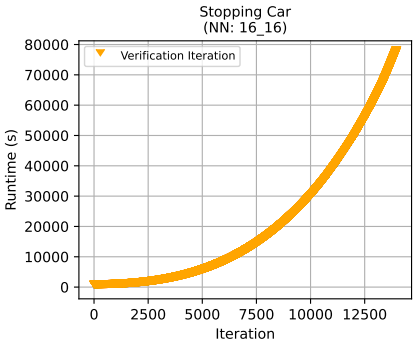

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

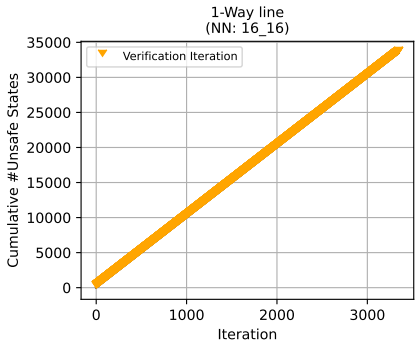

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

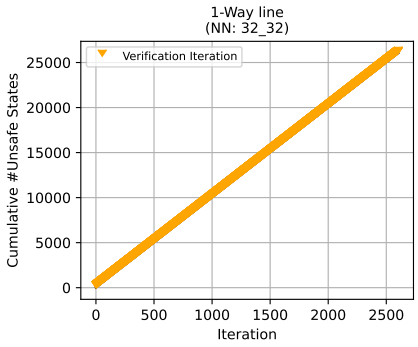

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

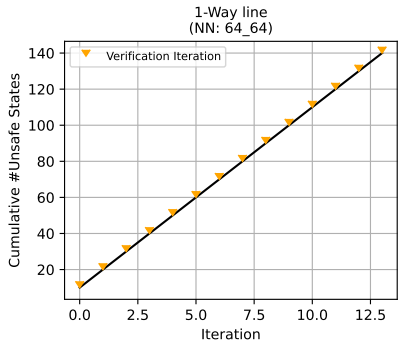

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

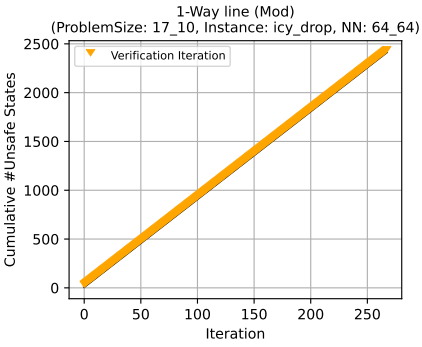

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

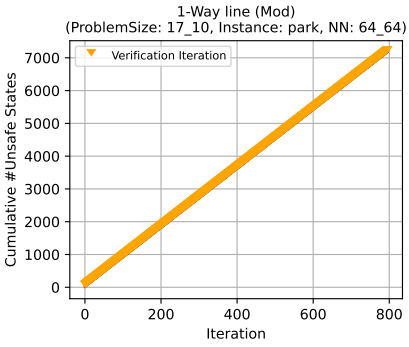

Stopping Car
(NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

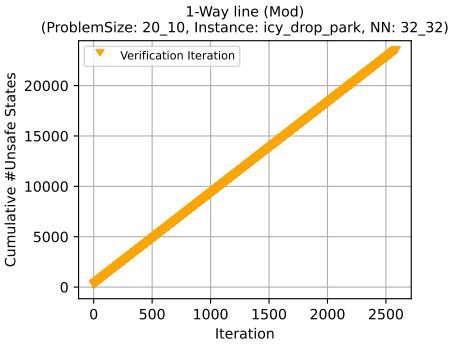

1-Way line
(NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

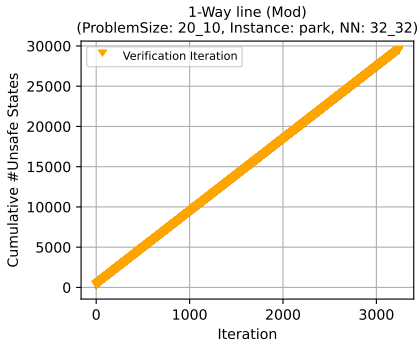

1-Way line
(NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

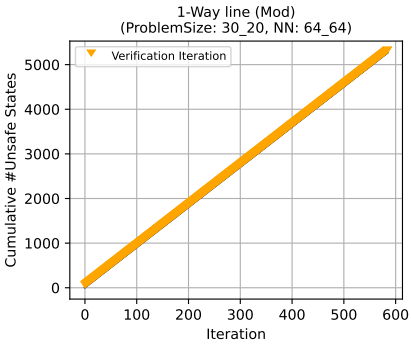

1-Way line
(NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

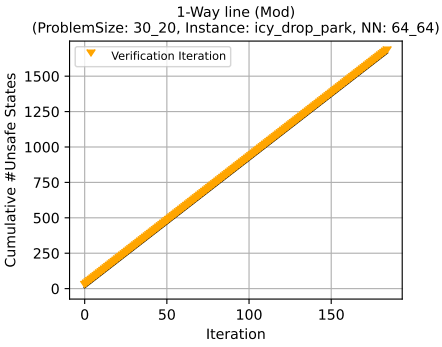

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

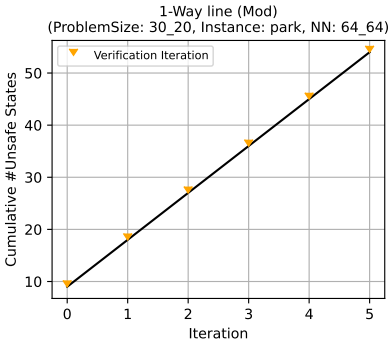

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

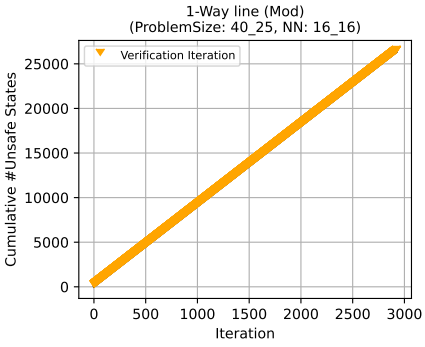

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

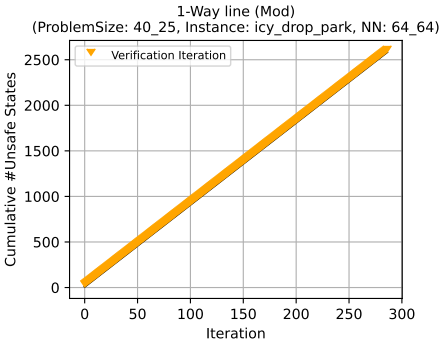

1-Way line (Mod)
(ProblemSize: 20_10, Instance: park, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

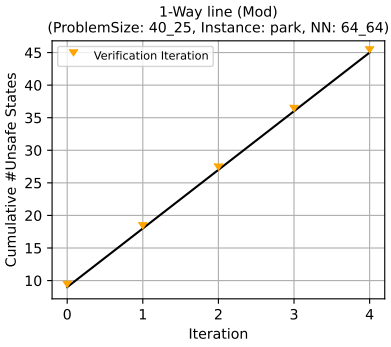

1-Way line (Mod)
(ProblemSize: 30_20, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

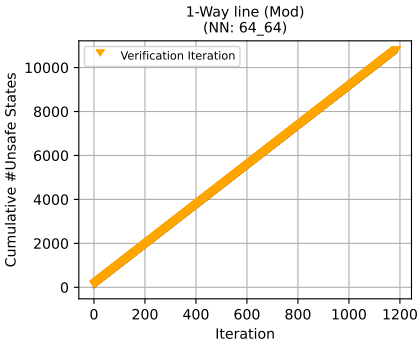

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop_park, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

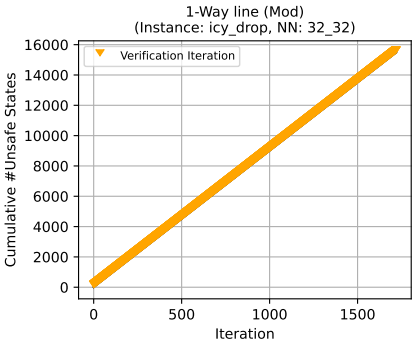

1-Way line (Mod)
(ProblemSize: 30_20, Instance: park, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

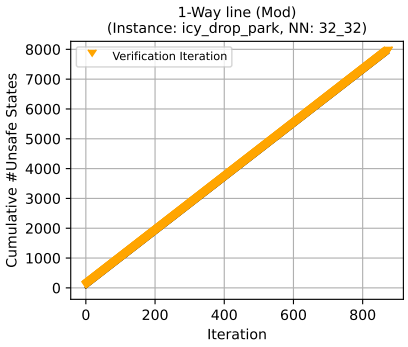

1-Way line (Mod)
(ProblemSize: 40_25, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

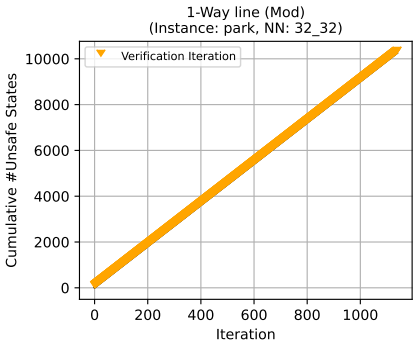

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

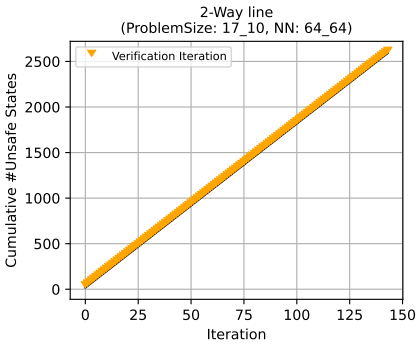

1-Way line (Mod)
(ProblemSize: 40_25, Instance: park, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

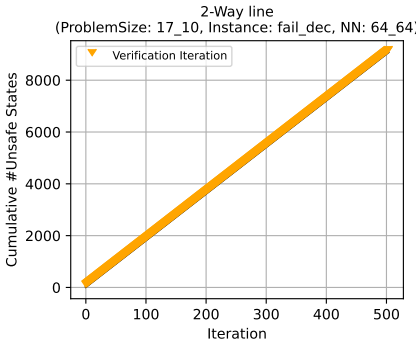

1-Way line (Mod)
(NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

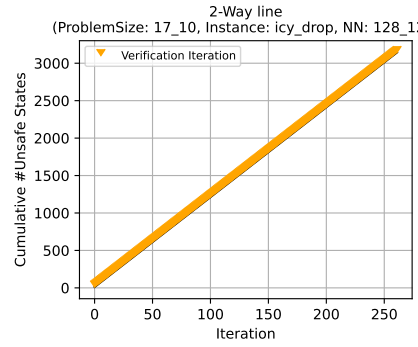

1-Way line (Mod)
(Instance: icy_drop, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

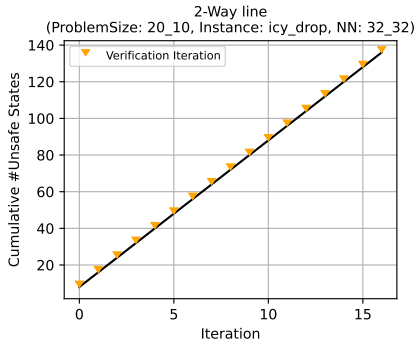

1-Way line (Mod)
(Instance: icy_drop_park, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

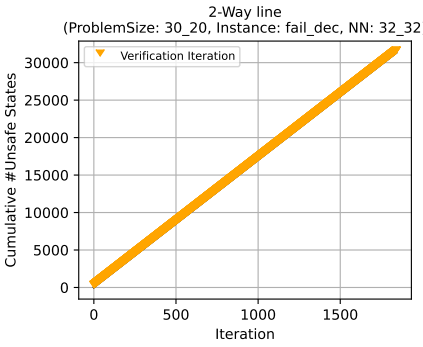

1-Way line (Mod)
(Instance: park, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

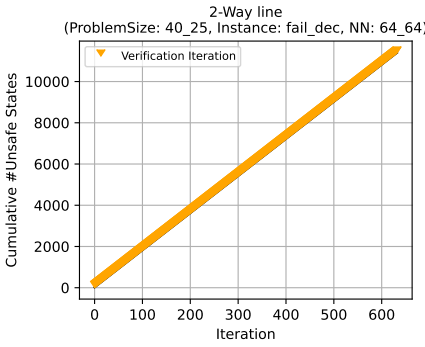

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

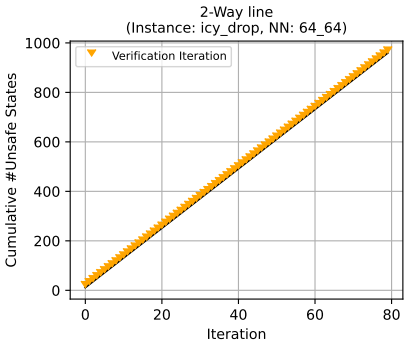

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

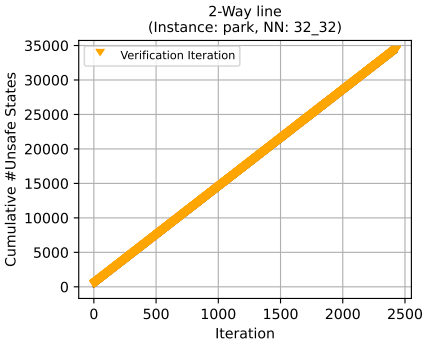

2-Way line
(ProblemSize: 17_10, Instance: icy_drop, NN: 128_128)
Verification Iteration
Cumulative #Unsafe States
Iteration

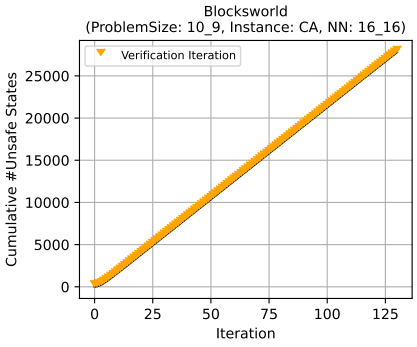

2-Way line
(ProblemSize: 20_10, Instance: icy_drop, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

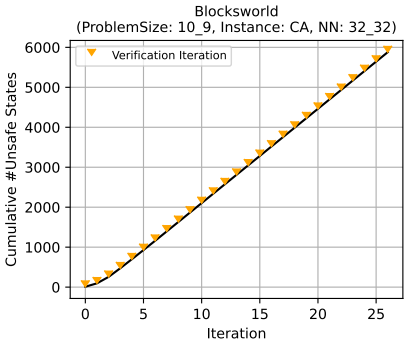

2-Way line
(ProblemSize: 30_20, Instance: fail_dec, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

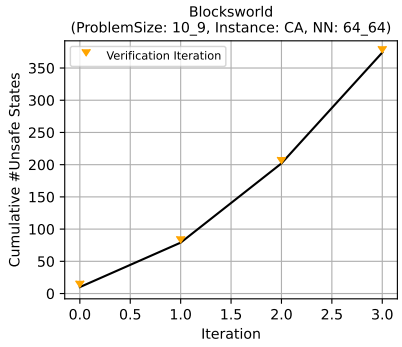

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

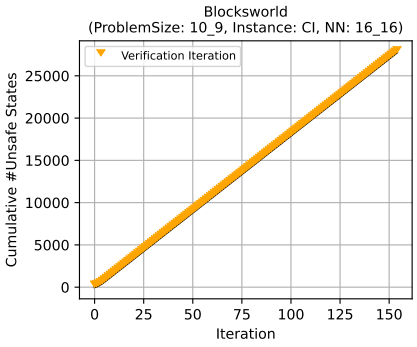

2-Way line
(Instance: icy_drop, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

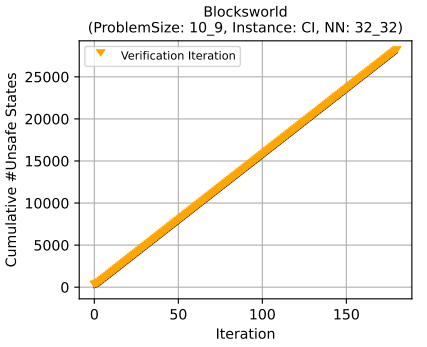

2-Way line
(Instance: park, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

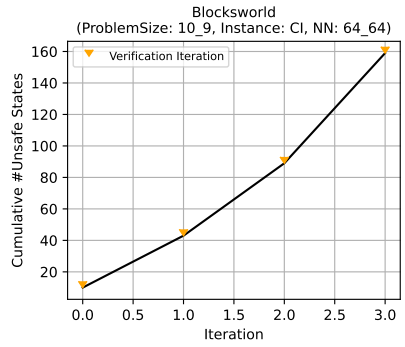

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

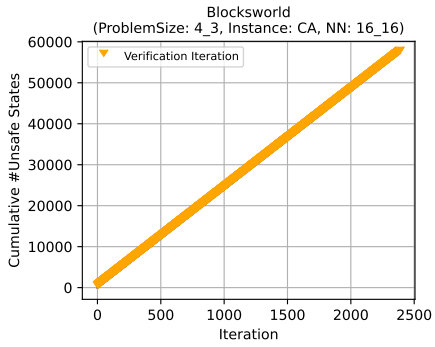

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

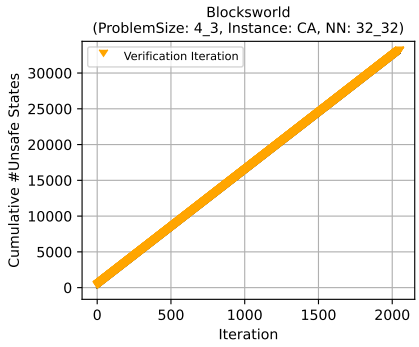

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

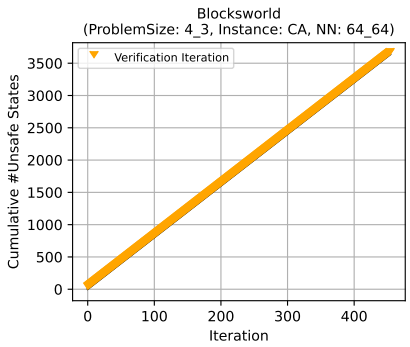

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

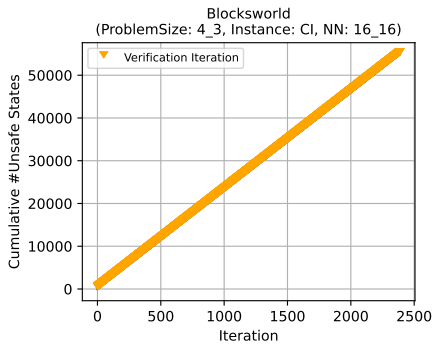

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

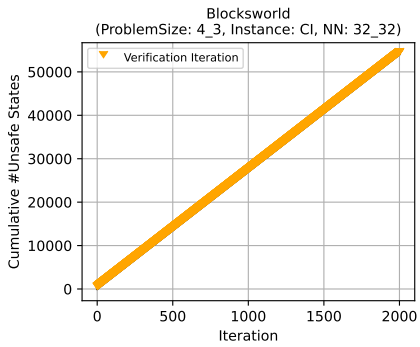

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

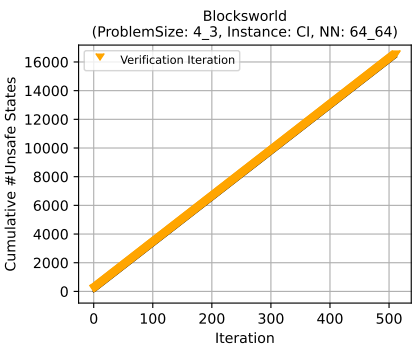

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

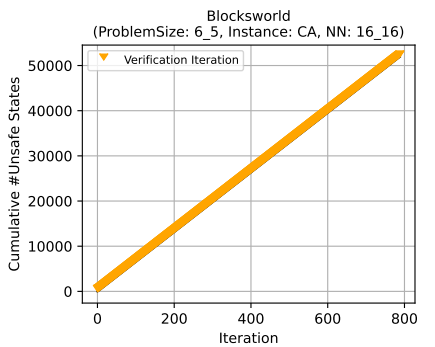

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

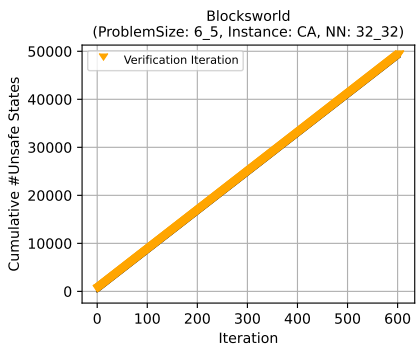

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

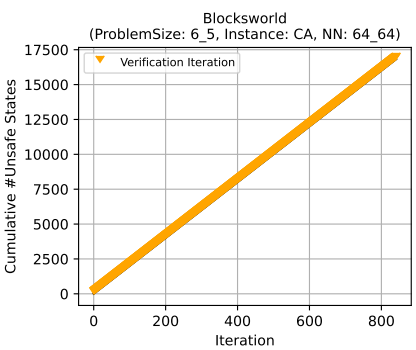

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

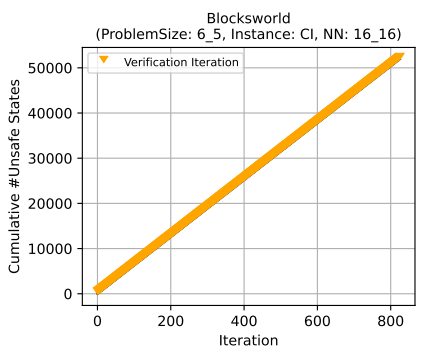

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

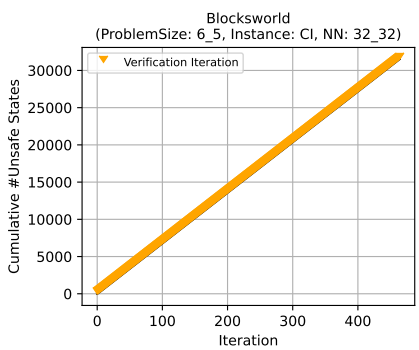

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

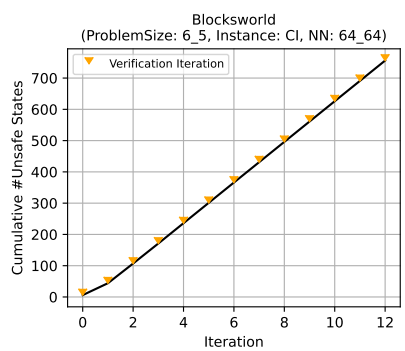

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

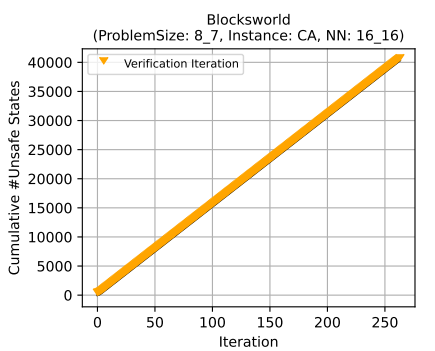

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

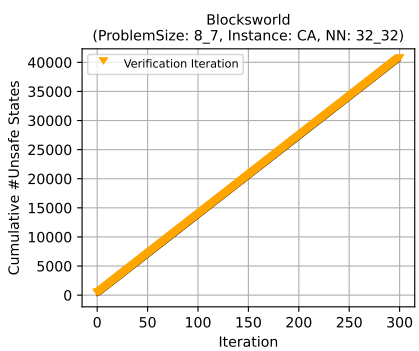

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

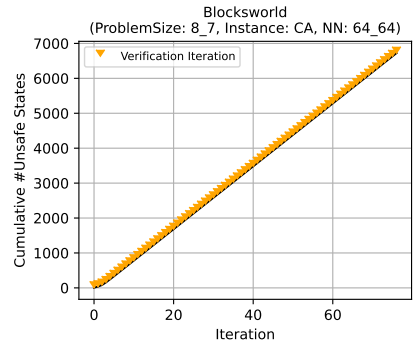

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

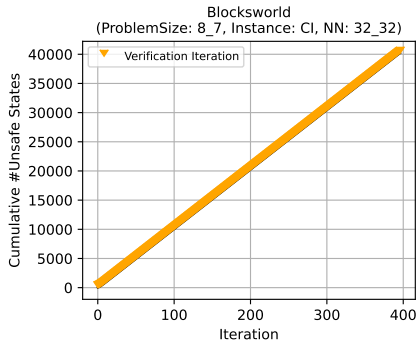

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

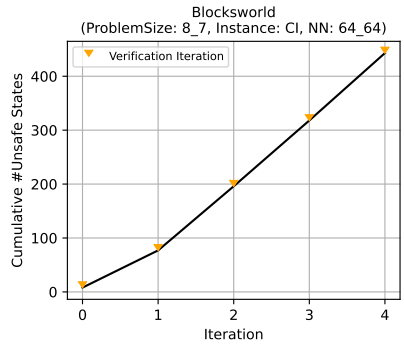

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

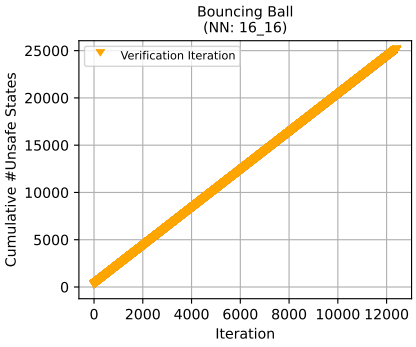

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

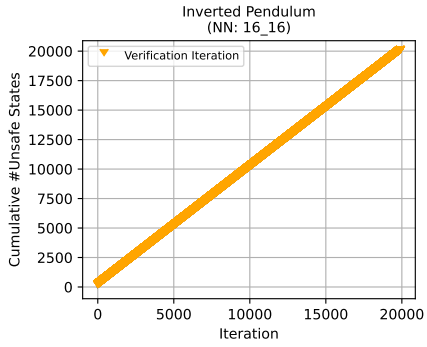

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

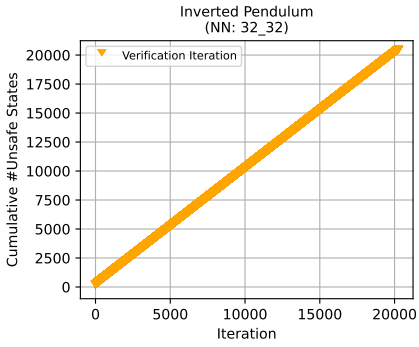

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

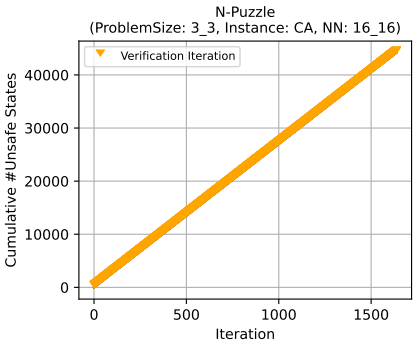

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

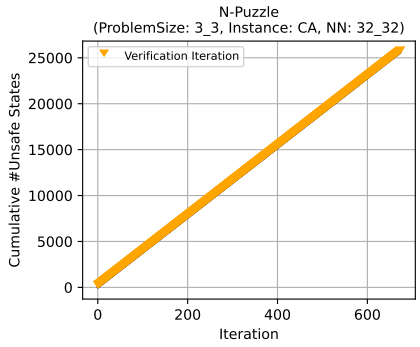

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

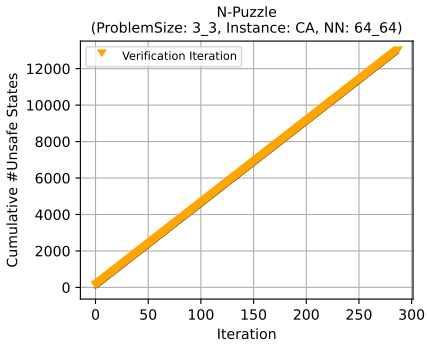

Bouncing Ball
(NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

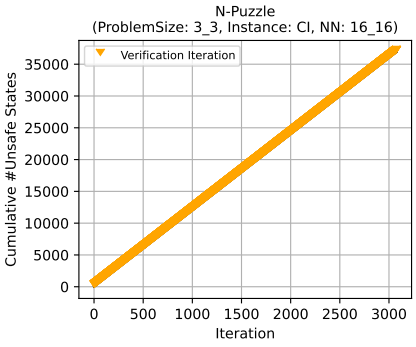

Inverted Pendulum
(NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

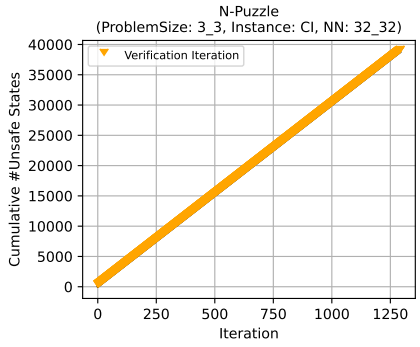

Inverted Pendulum
(NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

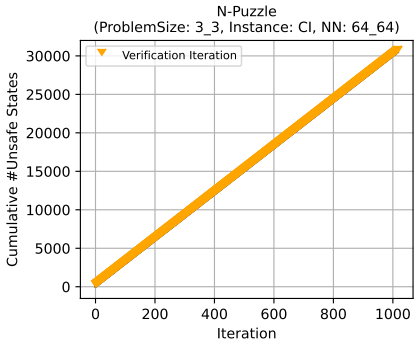

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

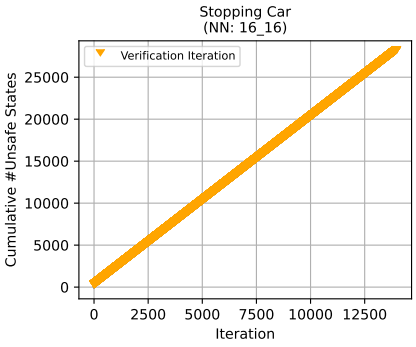

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

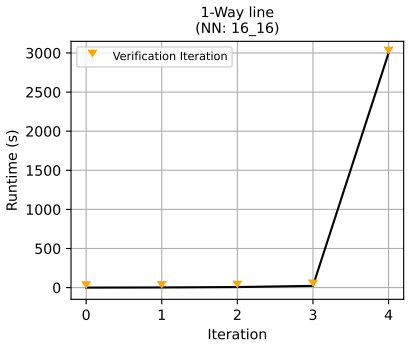

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

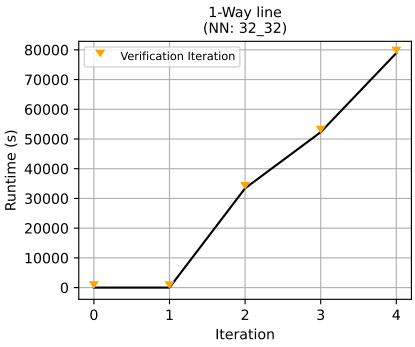

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

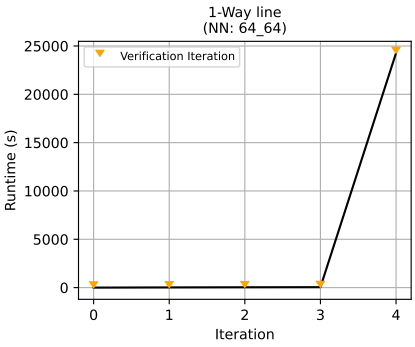

Stopping Car
(NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

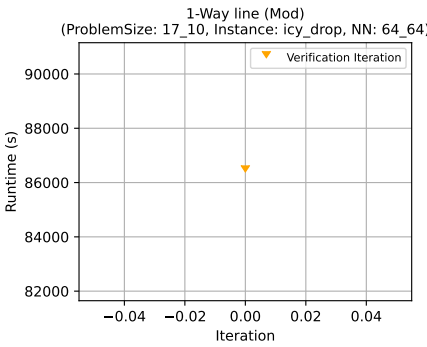
- Results for chapter 3 -

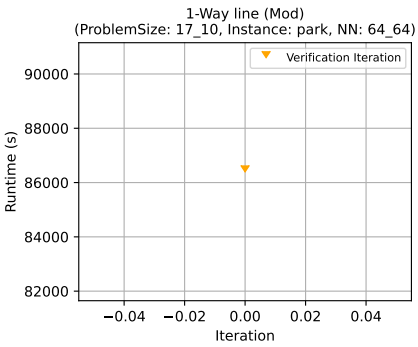
Start Condition Strengthening

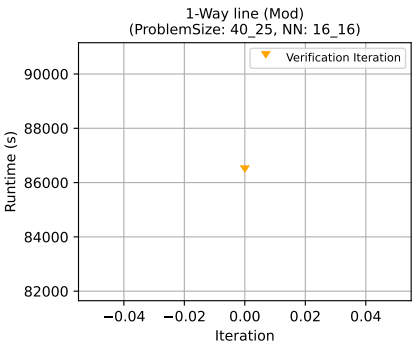

1-Way line
(NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

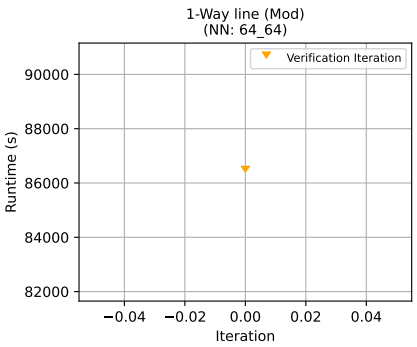

1-Way line
(NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

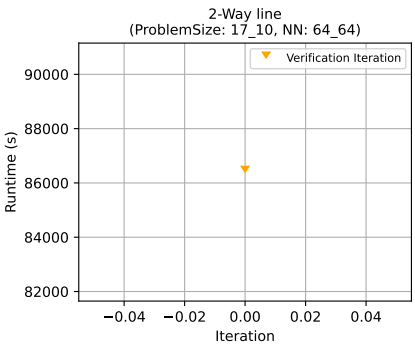

1-Way line
(NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

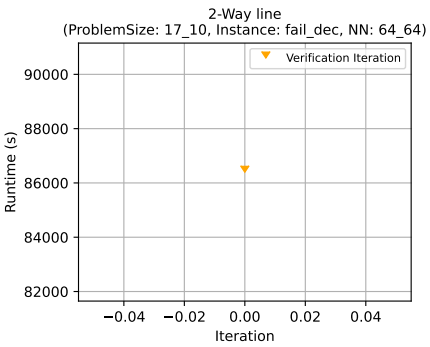

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

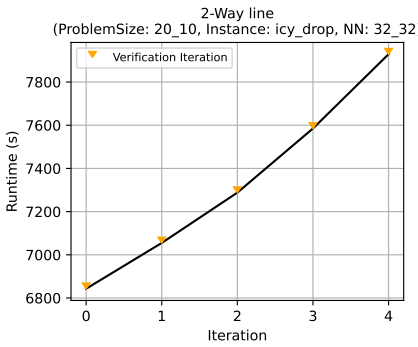

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

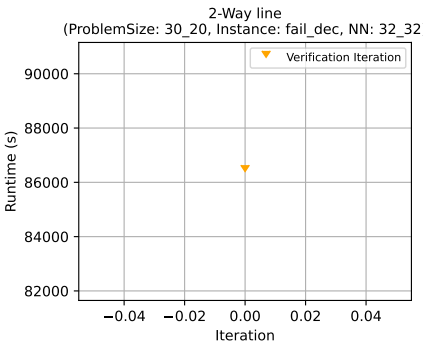

1-Way line (Mod)
(ProblemSize: 40_25, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

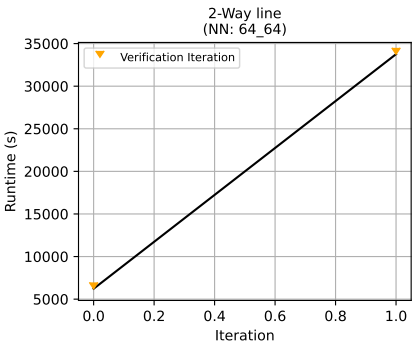

1-Way line (Mod)
(NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

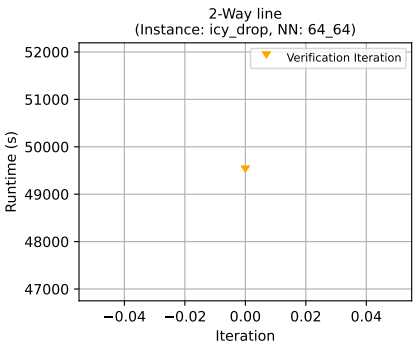

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

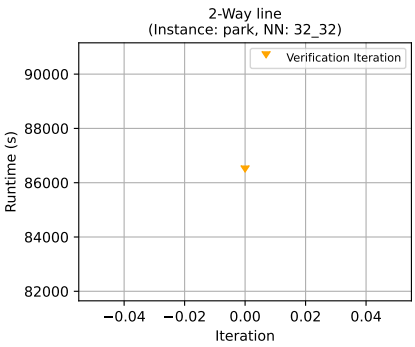

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

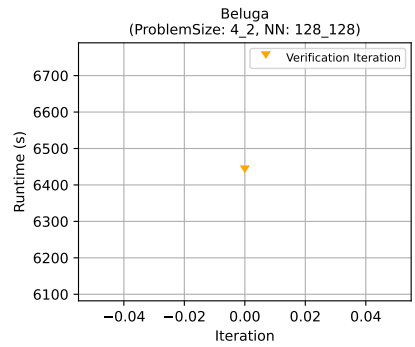

2-Way line
(ProblemSize: 20_10, Instance: icy_drop, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

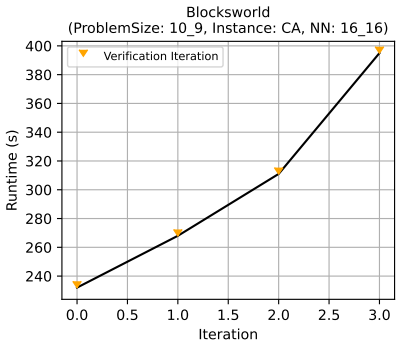

2-Way line
(ProblemSize: 30_20, Instance: fail_dec, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

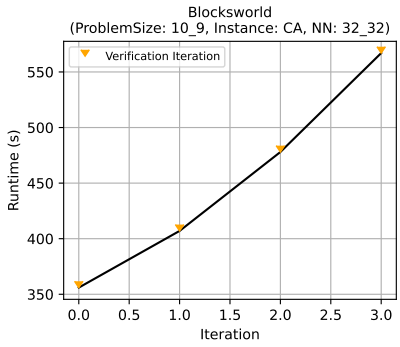

2-Way line
(NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

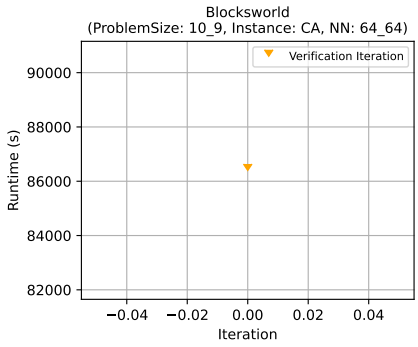

2-Way line
(Instance: icy_drop, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

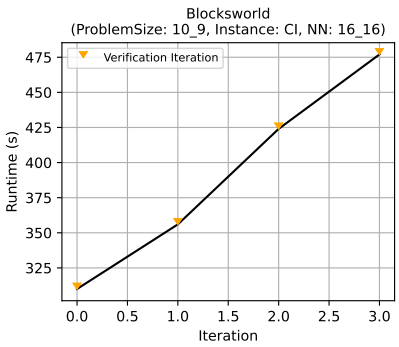

2-Way line
(Instance: park, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

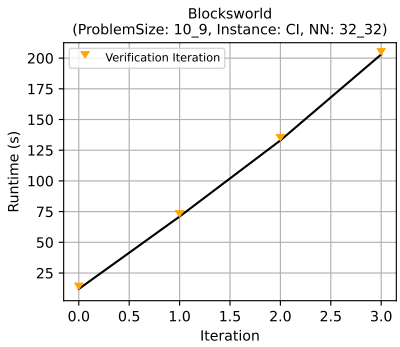

Beluga
(ProblemSize: 4_2, NN: 128_128)
Verification Iteration
Runtime (s)
Iteration

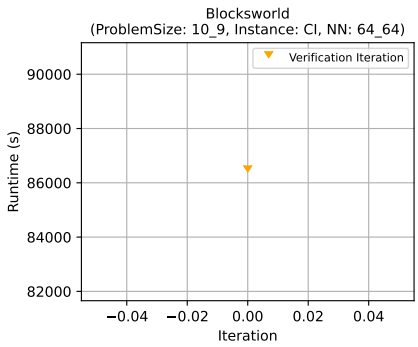

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

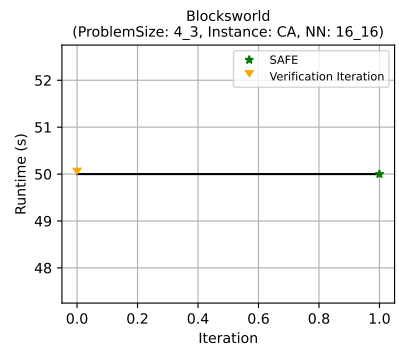

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

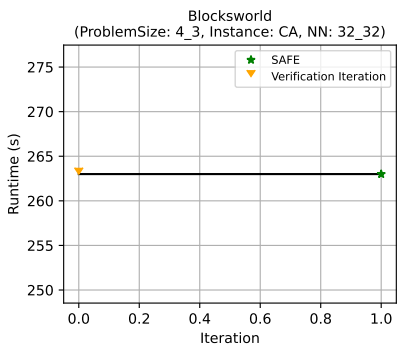

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

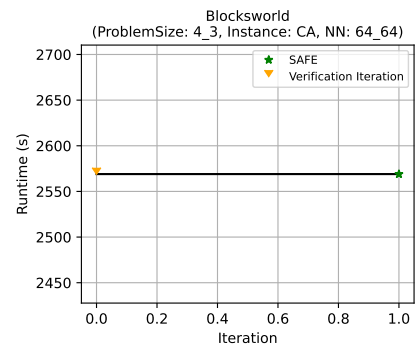

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

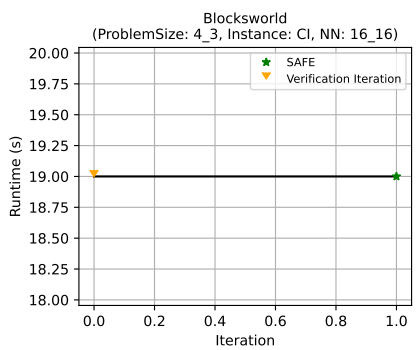

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

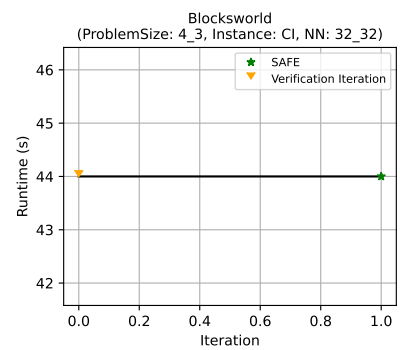

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

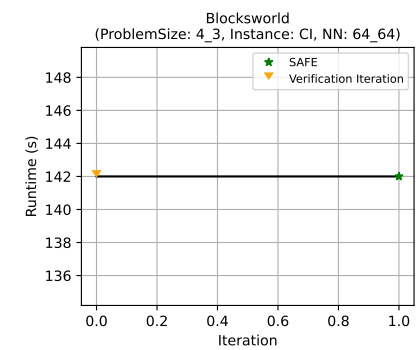

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
SAFE
Verification Iteration
Runtime (s)
Iteration

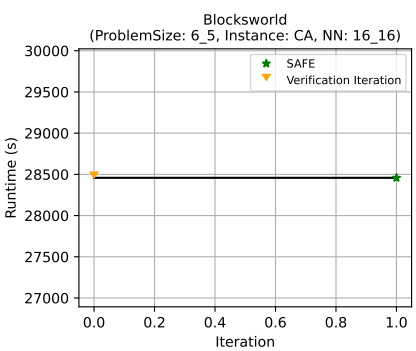

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
SAFE
Verification Iteration
Runtime (s)
Iteration

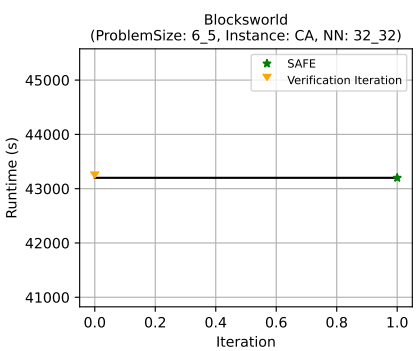

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
SAFE
Verification Iteration
Runtime (s)
Iteration

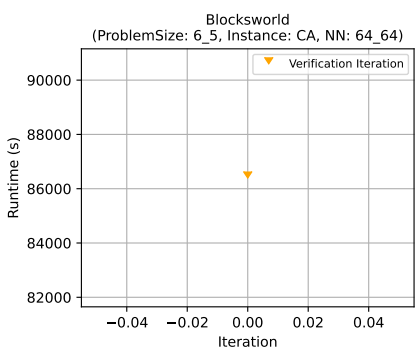

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
SAFE
Verification Iteration
Runtime (s)
Iteration

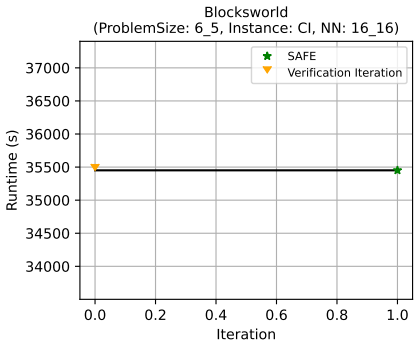

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
SAFE
Verification Iteration
Runtime (s)
Iteration

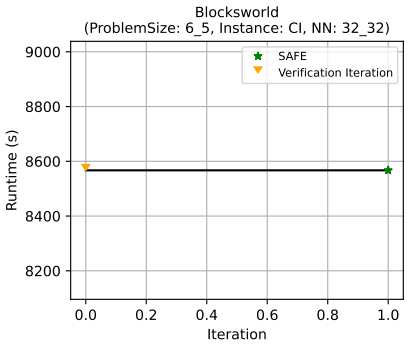

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
SAFE
Verification Iteration
Runtime (s)
Iteration

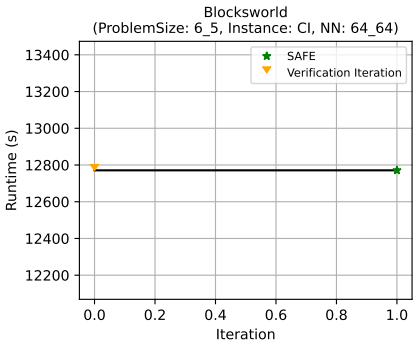

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
SAFE
Verification Iteration
Runtime (s)
Iteration

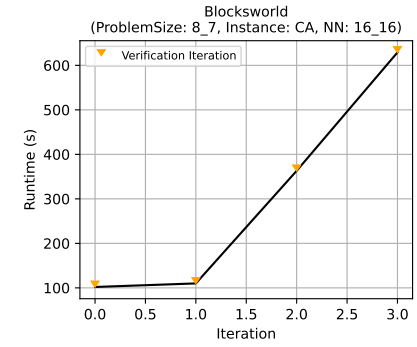

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
SAFE
Verification Iteration
Runtime (s)
Iteration

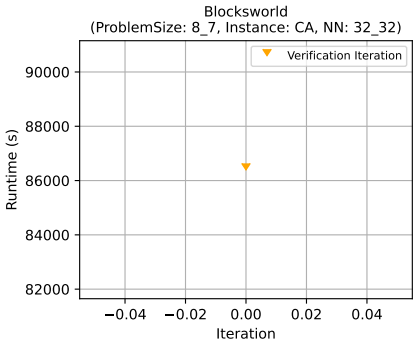

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

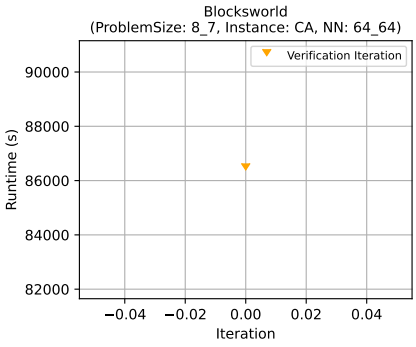

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
SAFE
Verification Iteration
Runtime (s)
Iteration

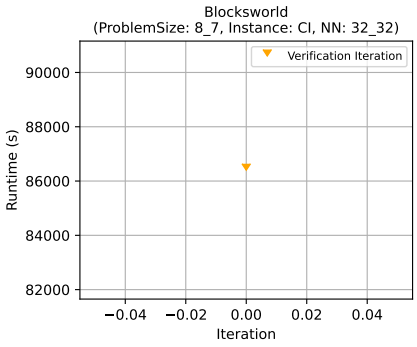

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
SAFE
Verification Iteration
Runtime (s)
Iteration

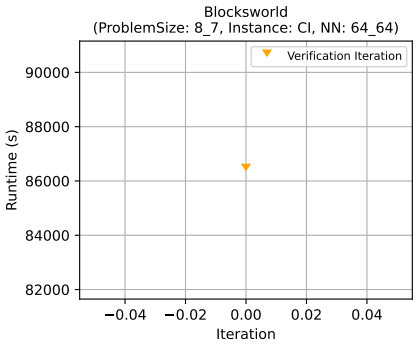

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
SAFE
Verification Iteration
Runtime (s)
Iteration

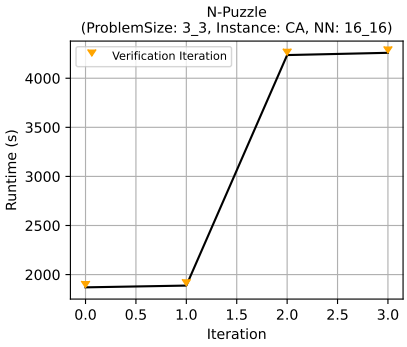

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

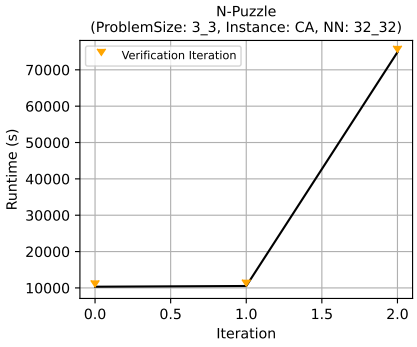

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

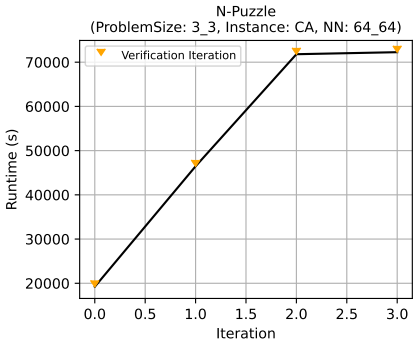

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

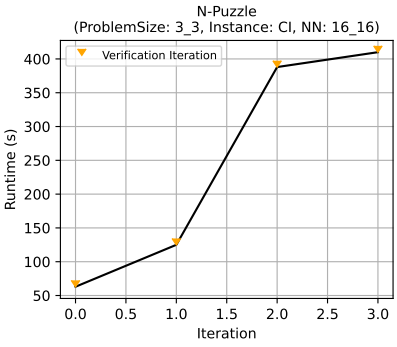

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

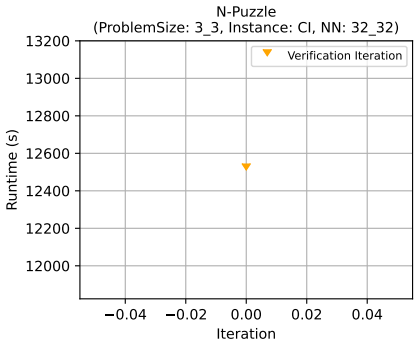

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

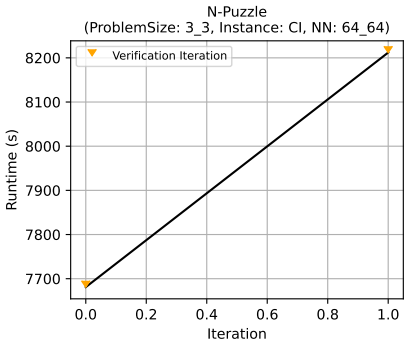

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

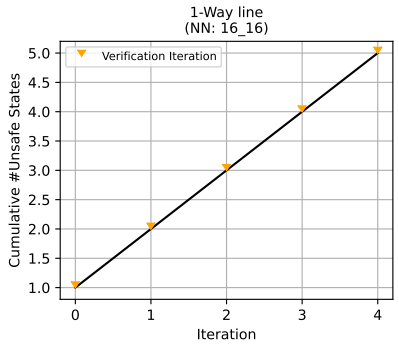

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

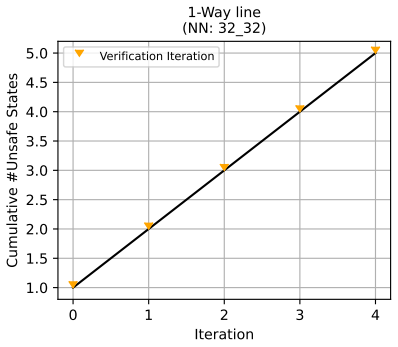

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

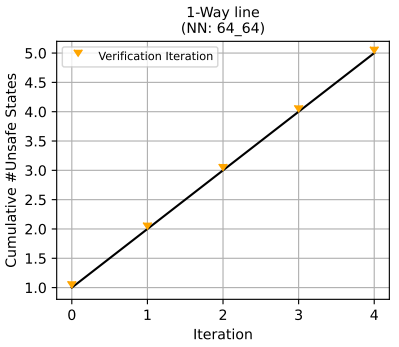

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Verification Iteration
Runtime (s)
Iteration

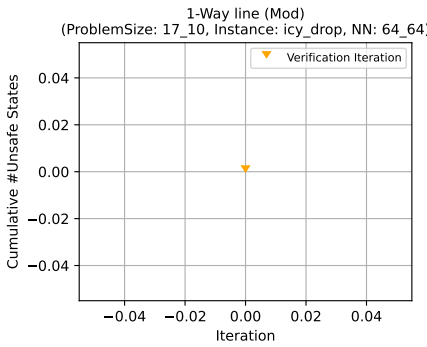

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Verification Iteration
Runtime (s)
Iteration

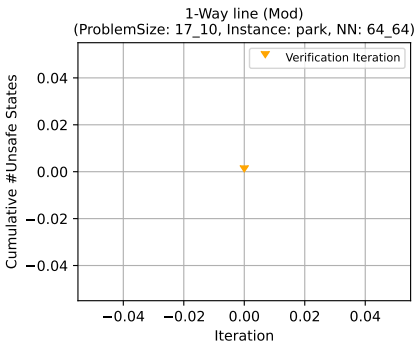

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Verification Iteration
Runtime (s)
Iteration

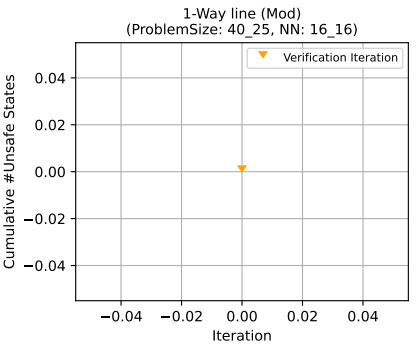

1-Way line
(NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

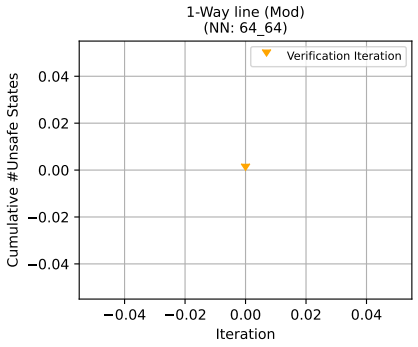

1-Way line
(NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

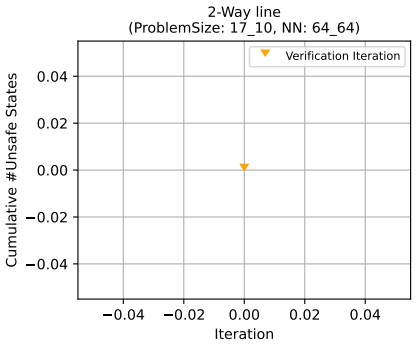

1-Way line
(NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

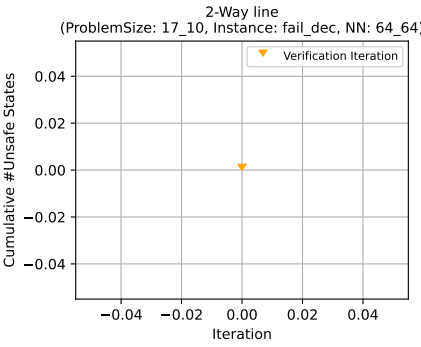

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

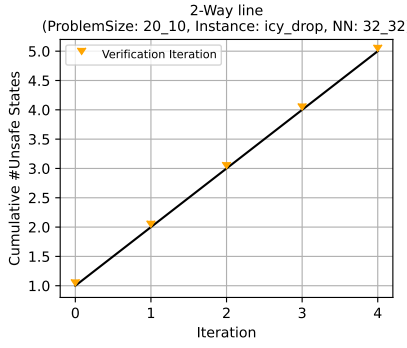

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

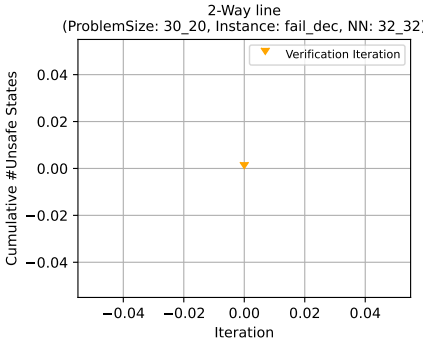

1-Way line (Mod)
(ProblemSize: 40_25, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

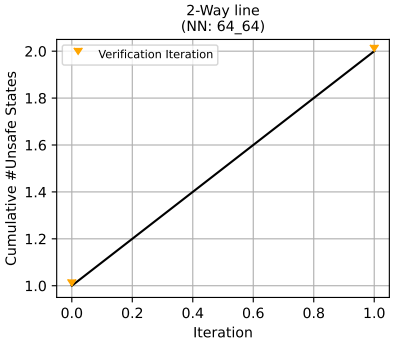

1-Way line (Mod)
(NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

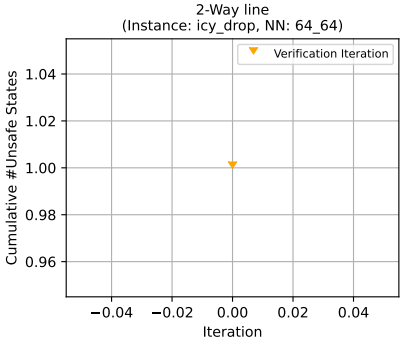

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

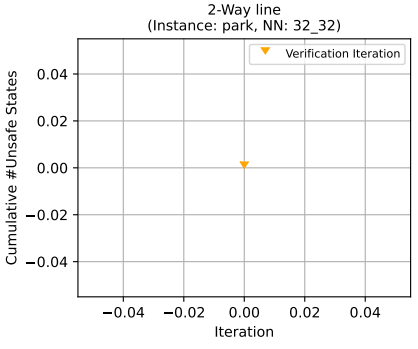

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

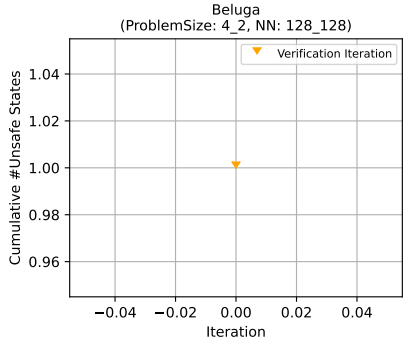

2-Way line
(ProblemSize: 20_10, Instance: icy_drop, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

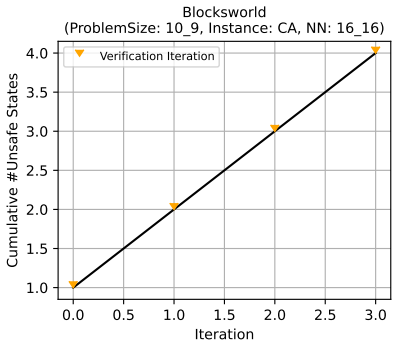

2-Way line
(ProblemSize: 30_20, Instance: fail_dec, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

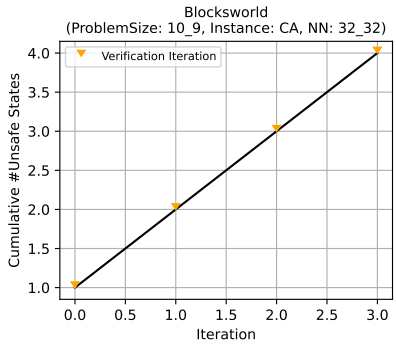

2-Way line
(NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

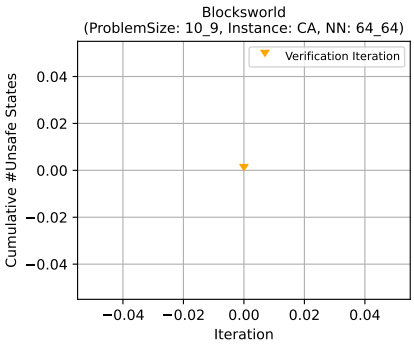

2-Way line
(Instance: icy_drop, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

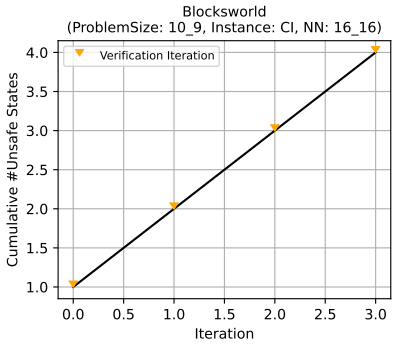

2-Way line
(Instance: park, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

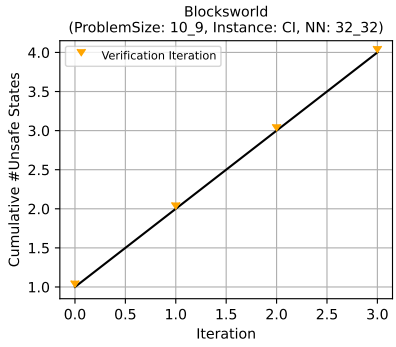

Beluga
(ProblemSize: 4_2, NN: 128_128)
Verification Iteration
Cumulative #Unsafe States
Iteration

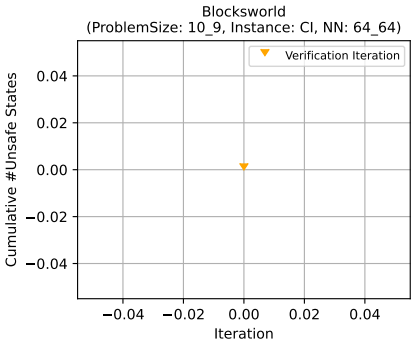

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

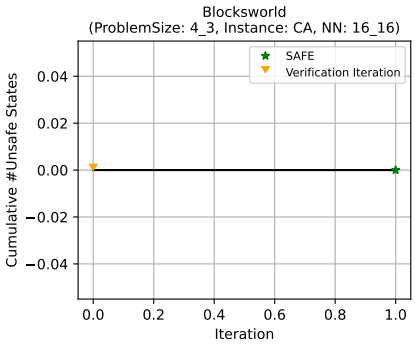

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

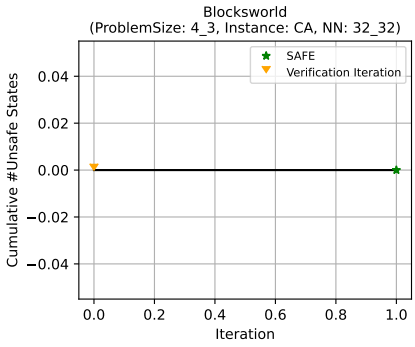

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

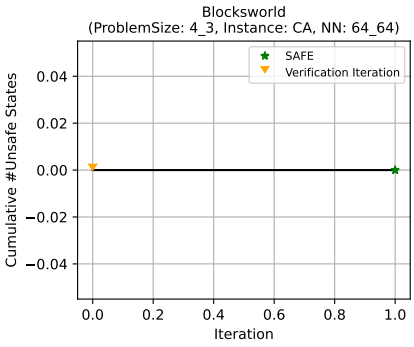

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

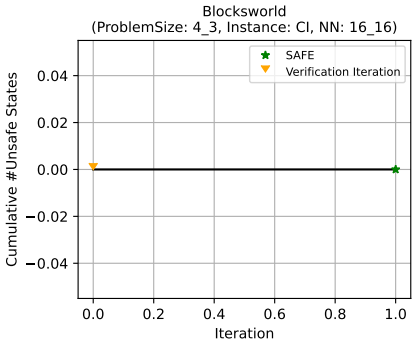

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

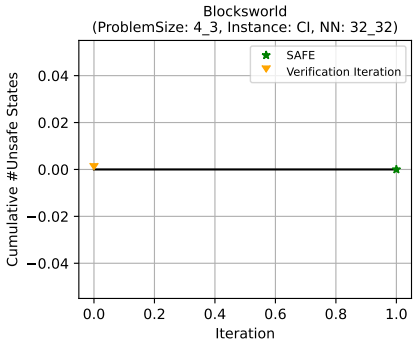

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

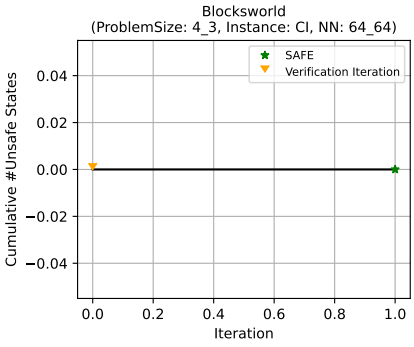

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

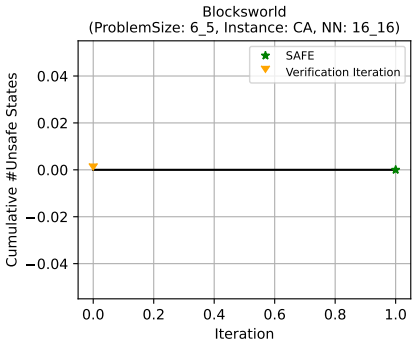

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

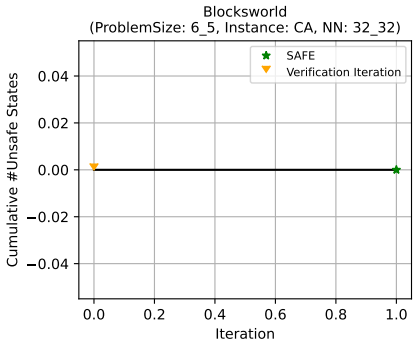

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

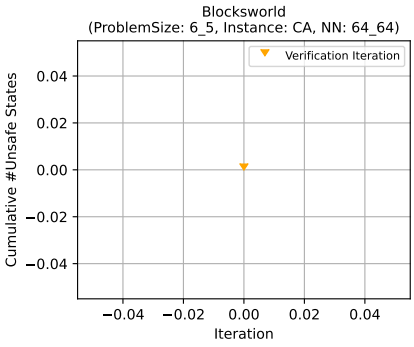

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

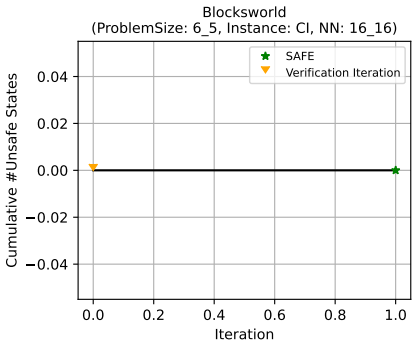

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

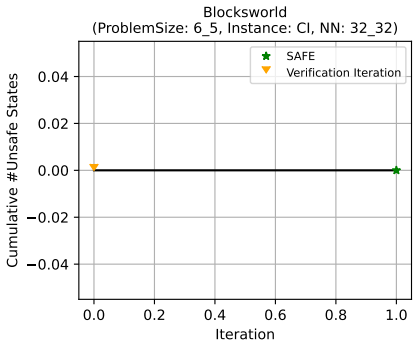

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

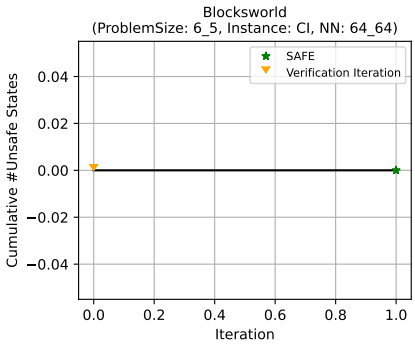

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

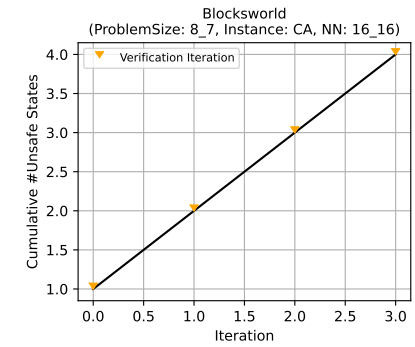

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

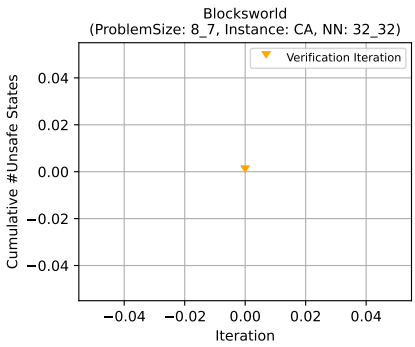

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

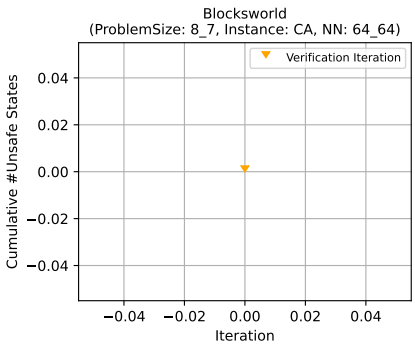

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

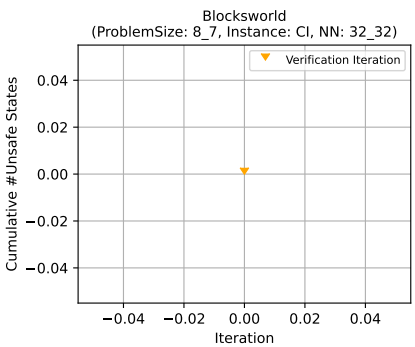

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

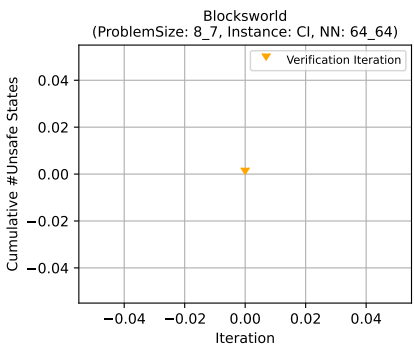

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
SAFE
Verification Iteration
Cumulative #Unsafe States
Iteration

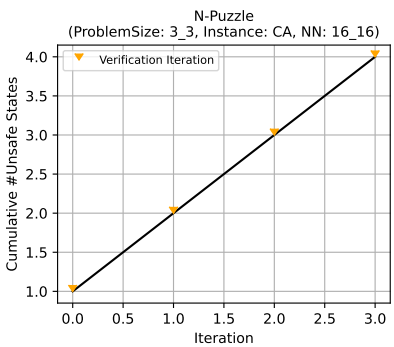

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

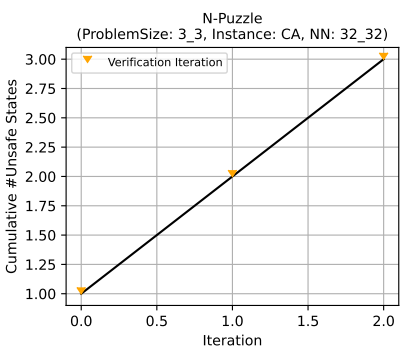

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

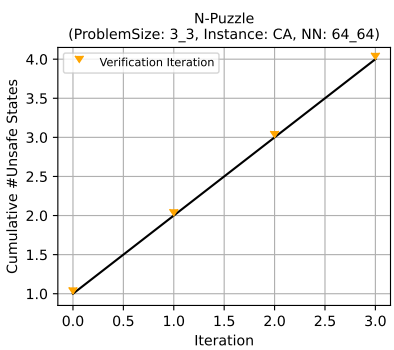

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

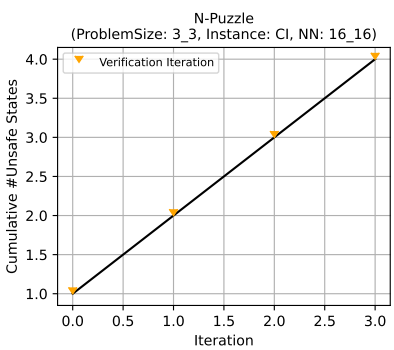

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

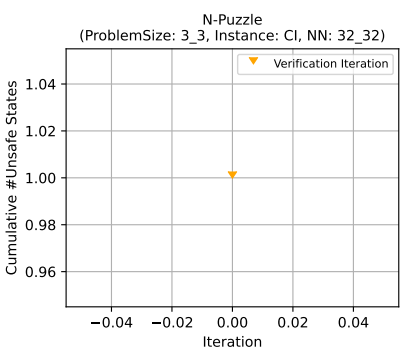

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

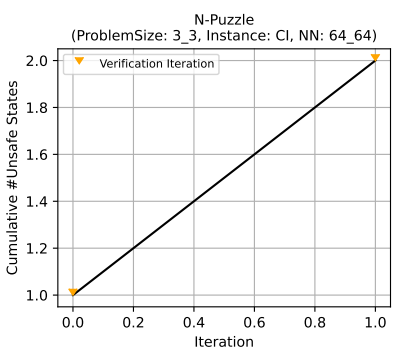

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

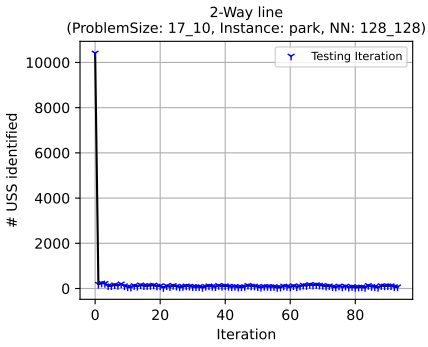

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Verification Iteration
Cumulative #Unsafe States
Iteration

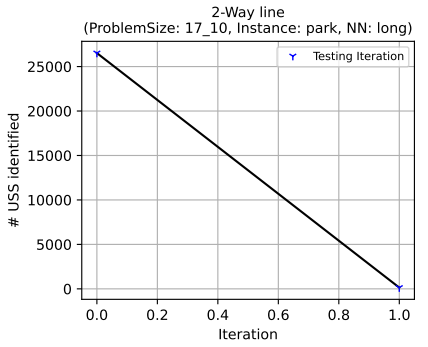

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Verification Iteration
Cumulative #Unsafe States
Iteration

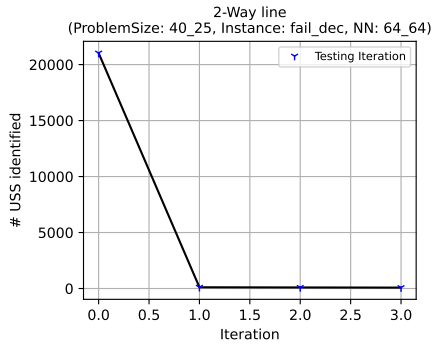

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Verification Iteration
Cumulative #Unsafe States
Iteration

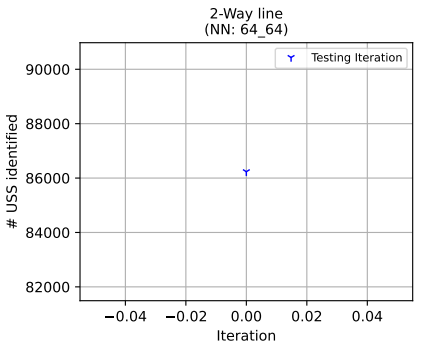
- Results for chapter 4 -

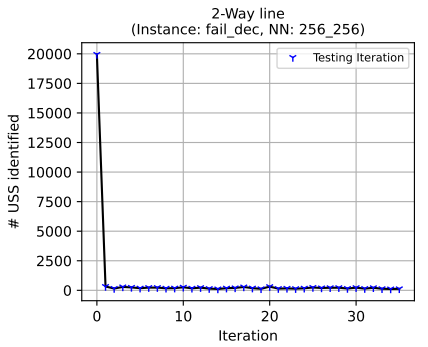
Invariant Strengthening (Uniform Sampling)

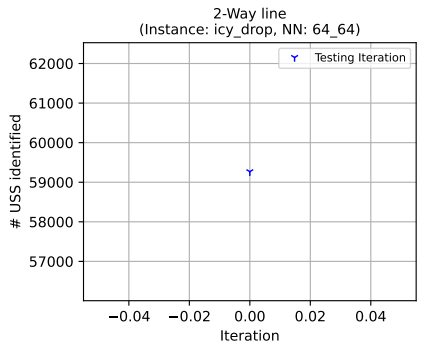

2-Way line
(ProblemSize: 17_10, Instance: park, NN: 128_128)
Testing Iteration
USS identified
Iteration

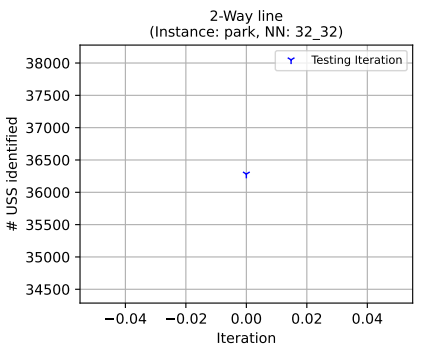

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
USS identified
Iteration

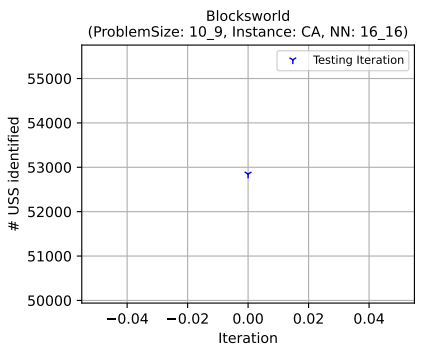

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS identified
Iteration

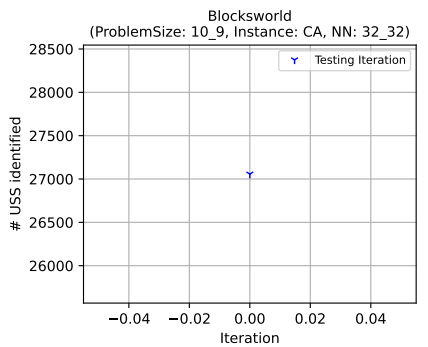

2-Way line
(NN: 64_64)
Testing Iteration
USS identified
Iteration

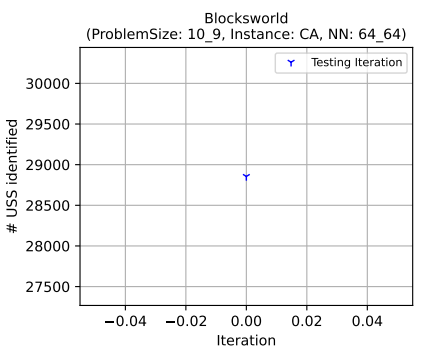

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
USS identified
Iteration

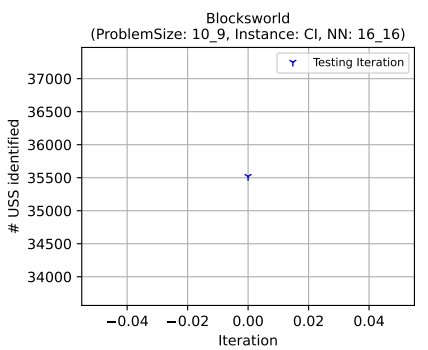

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
USS identified
Iteration

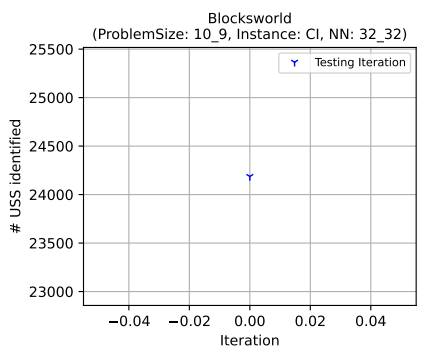

2-Way line
(Instance: park, NN: 32_32)
Testing Iteration
USS identified
Iteration

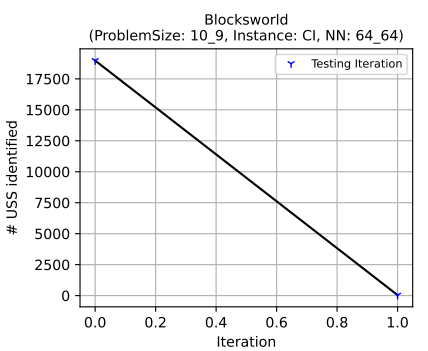

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

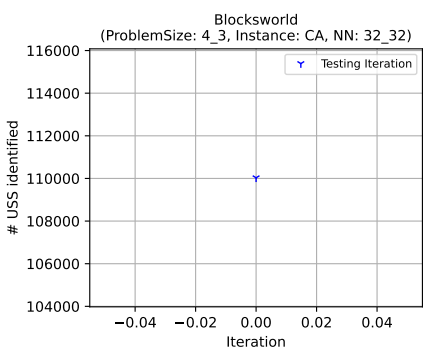

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

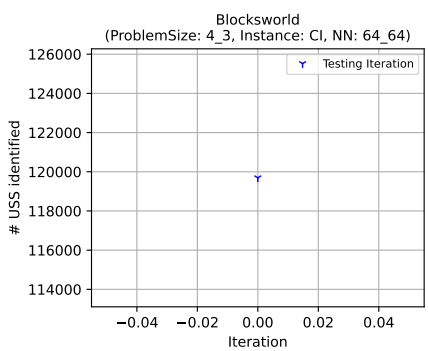

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

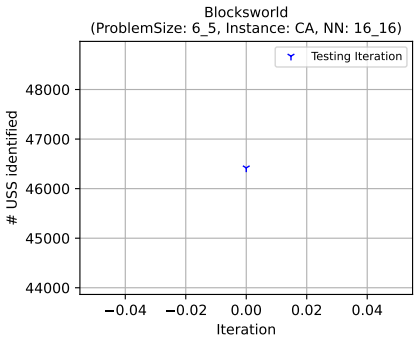

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

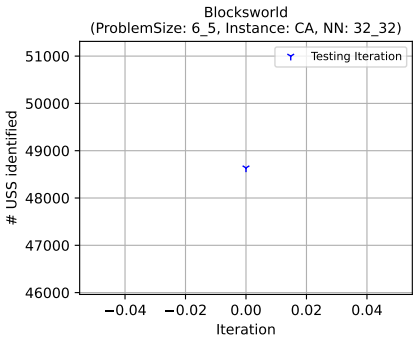

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

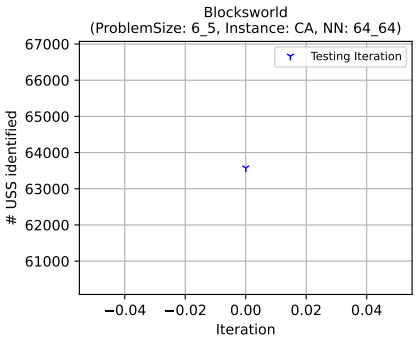

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

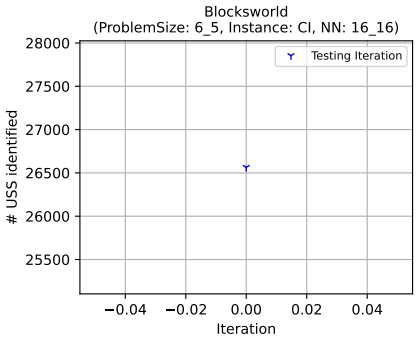

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

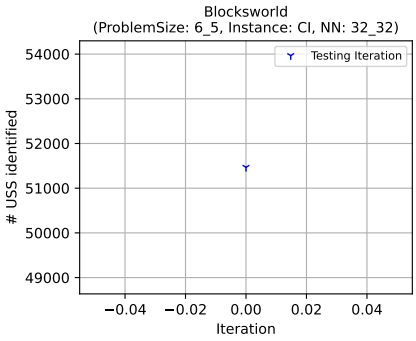

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

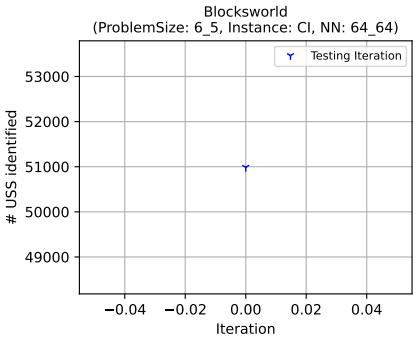

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

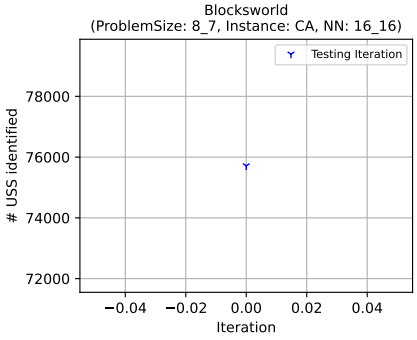

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

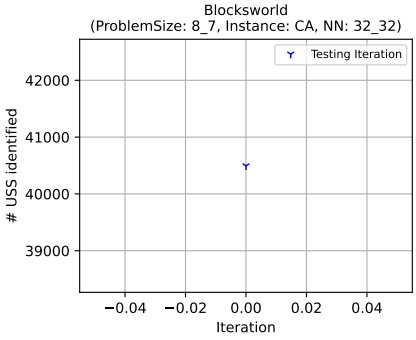

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

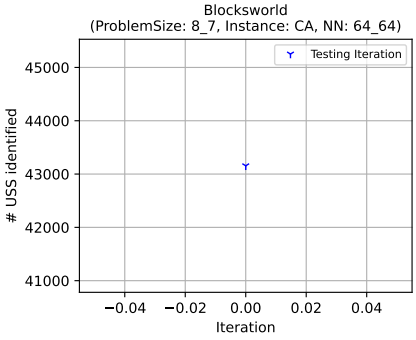

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

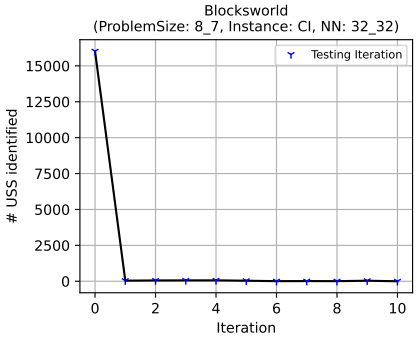

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

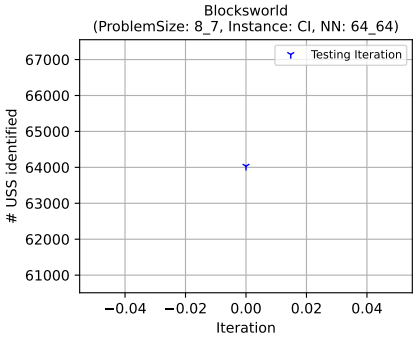

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

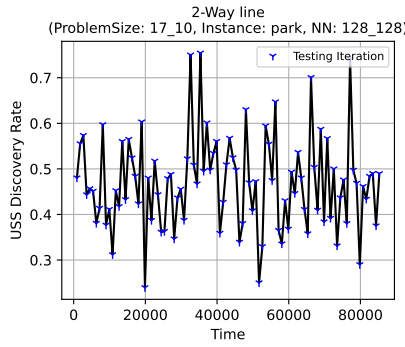

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

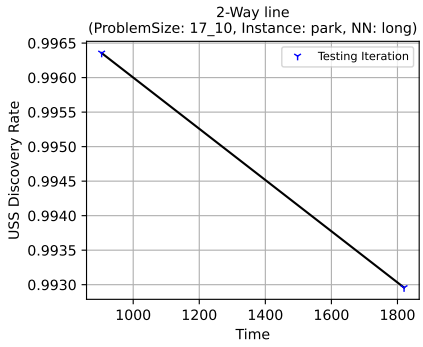

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

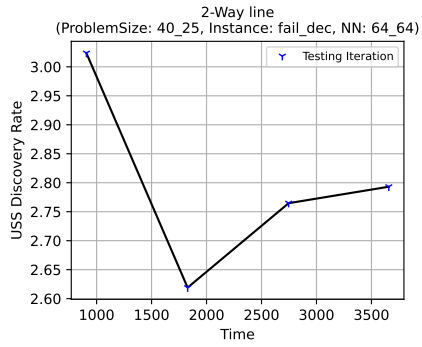

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

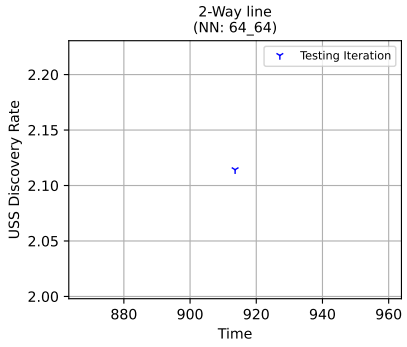

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

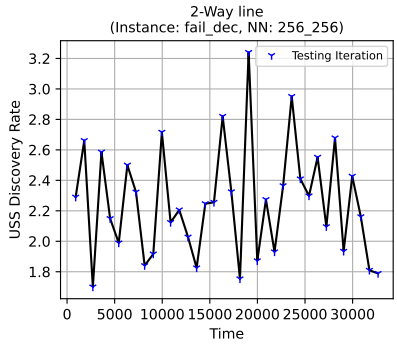

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

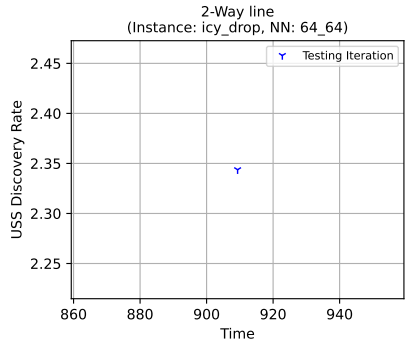

2-Way line
(ProblemSize: 17_10, Instance: park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

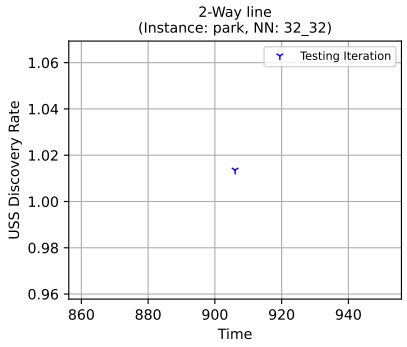

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
USS Discovery Rate
Time

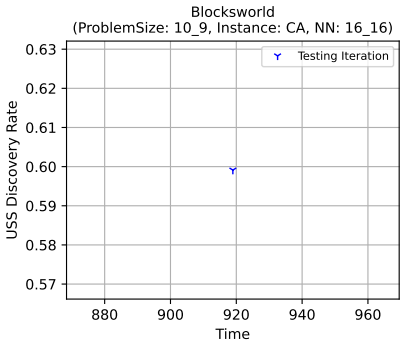

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

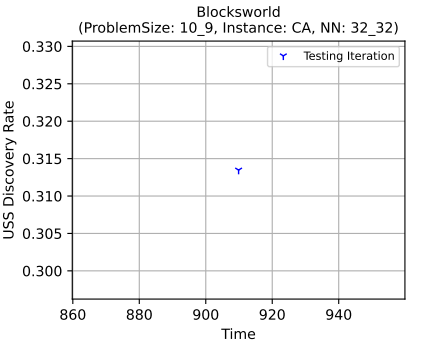

2-Way line
(NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

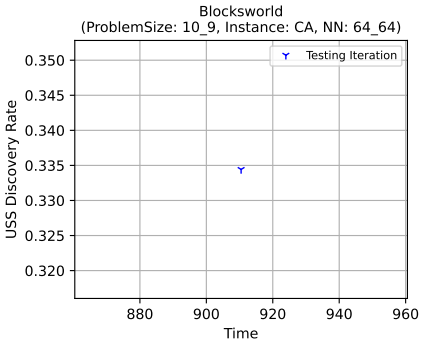

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

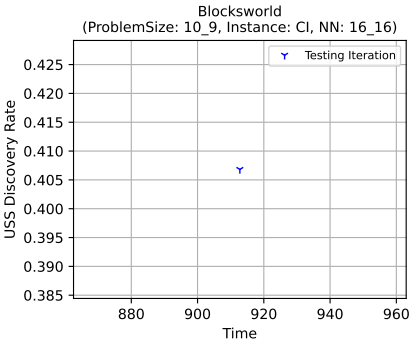

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

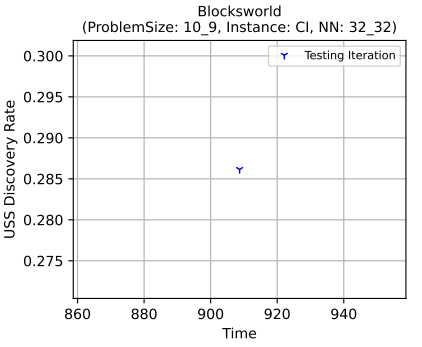

2-Way line
(Instance: park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

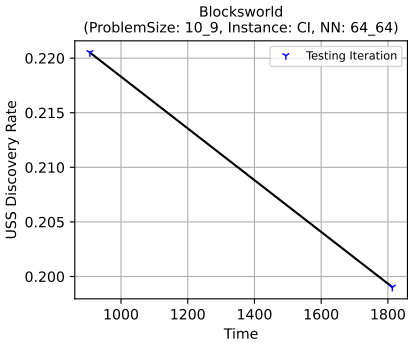

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

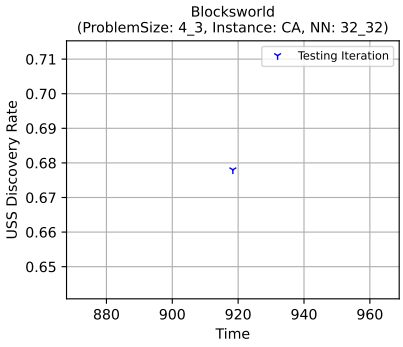

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

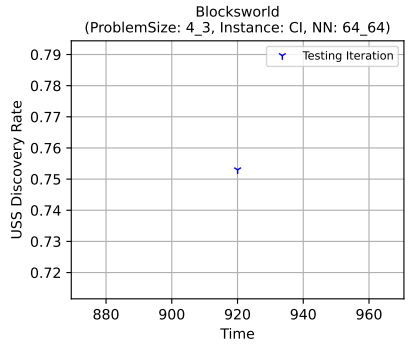

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

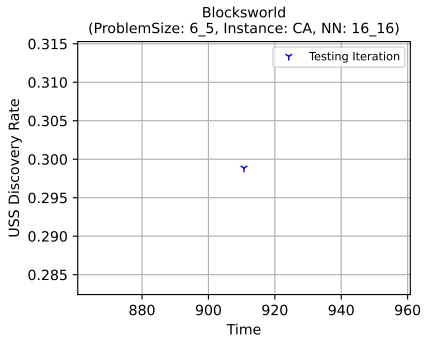

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

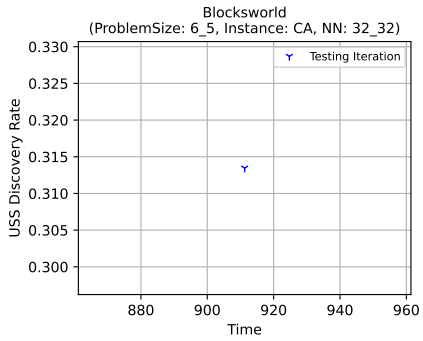

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

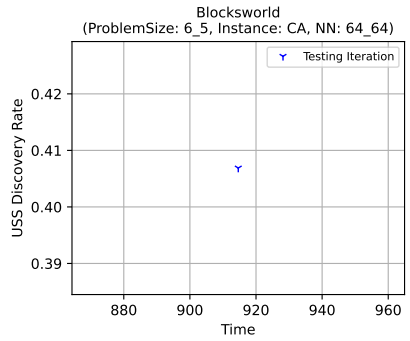

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

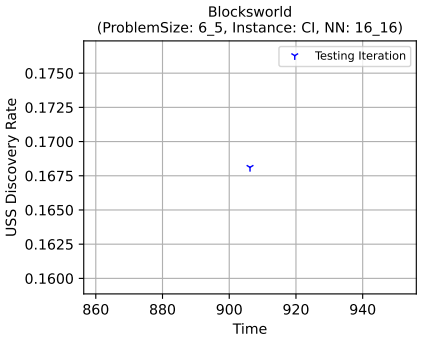

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

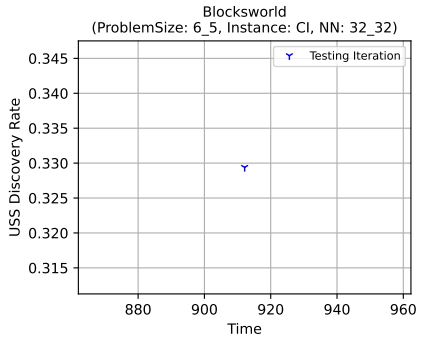

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

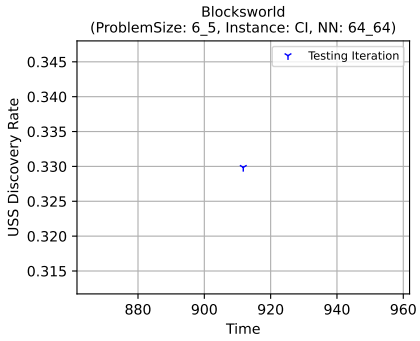

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

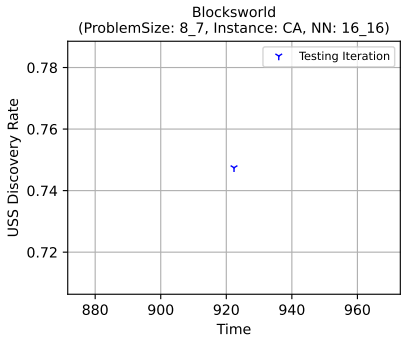

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

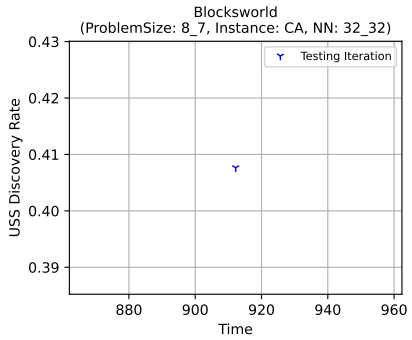

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

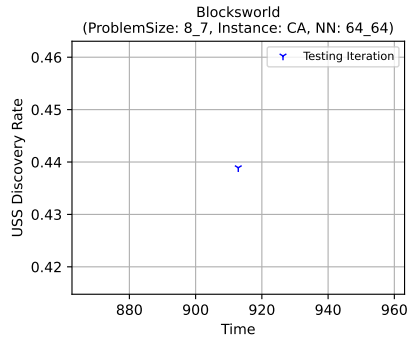

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

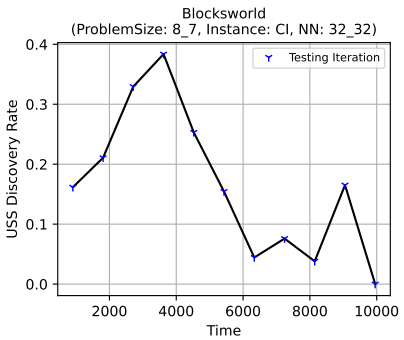

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

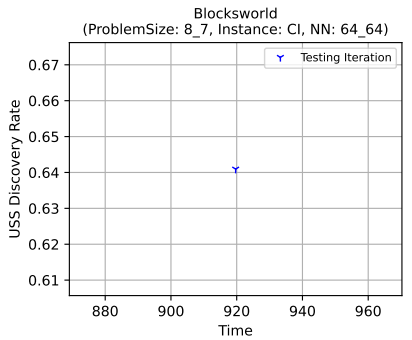

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time


Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time


Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

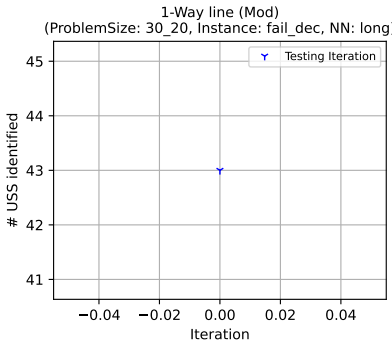

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

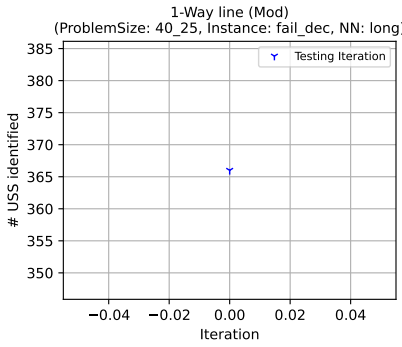

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

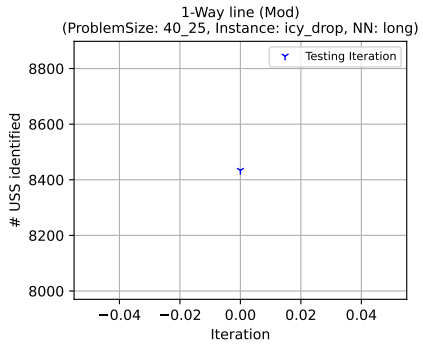

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

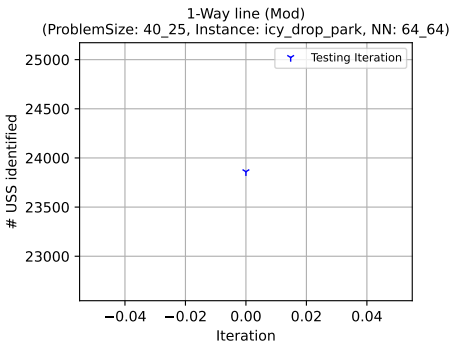
- Results for chapter 4 -

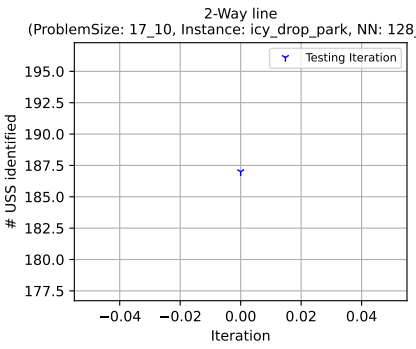
Invariant Strengthening (Biased Sampling)

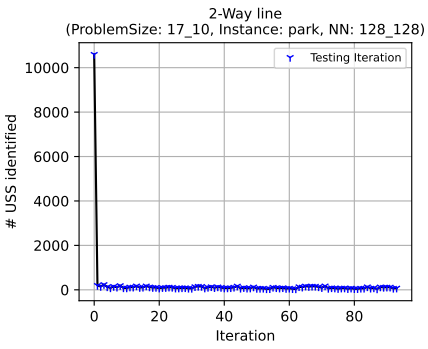

1-Way line (Mod)
(ProblemSize: 30_20, Instance: fail_dec, NN: long)
Testing Iteration
USS identified
Iteration

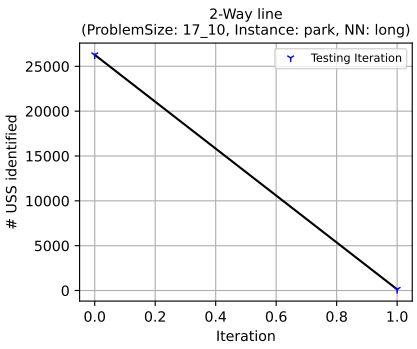

1-Way line (Mod)
(ProblemSize: 40_25, Instance: fail_dec, NN: long)
Testing Iteration
USS identified
Iteration

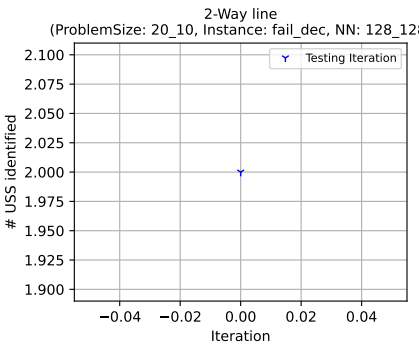

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop, NN: long)
Testing Iteration
USS identified
Iteration

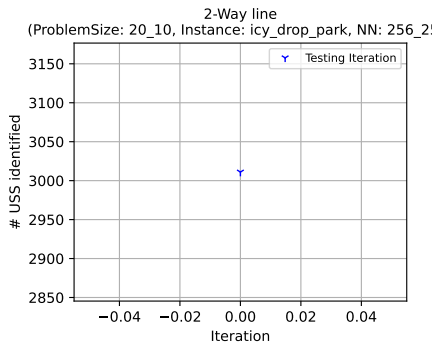

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS identified
Iteration

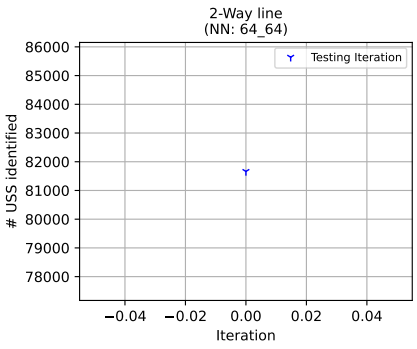

2-Way line
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS identified
Iteration

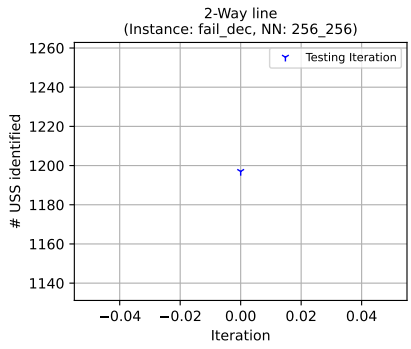

2-Way line
(ProblemSize: 17_10, Instance: park, NN: 128_128)
Testing Iteration
USS identified
Iteration

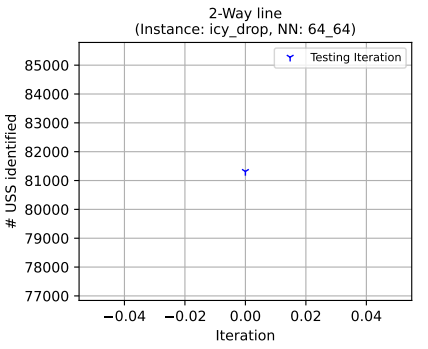

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
USS identified
Iteration

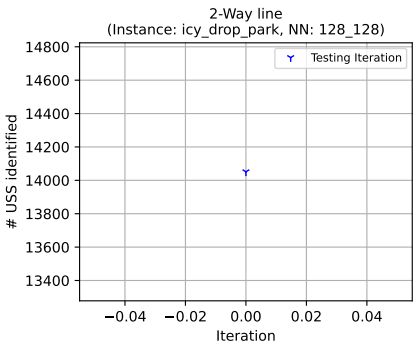

2-Way line
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS identified
Iteration

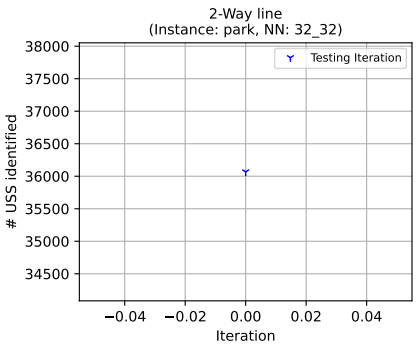

2-Way line
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 256_256)
Testing Iteration
USS identified
Iteration

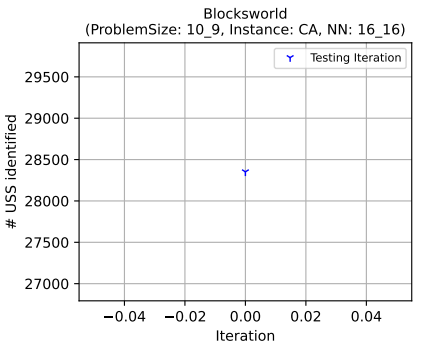

2-Way line
(NN: 64_64)
Testing Iteration
USS identified
Iteration

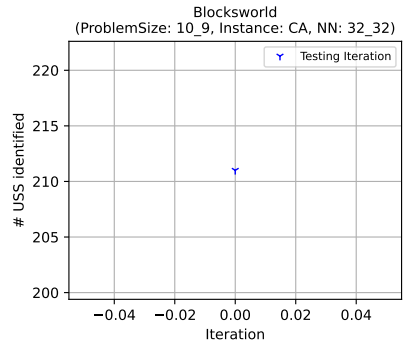

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
USS identified
Iteration

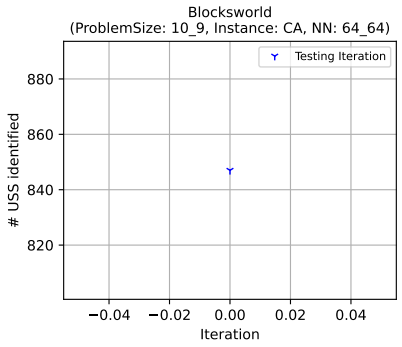

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
USS identified
Iteration

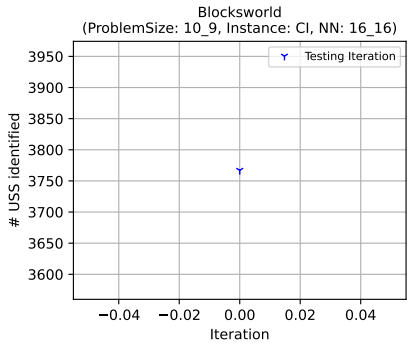

2-Way line
(Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS identified
Iteration

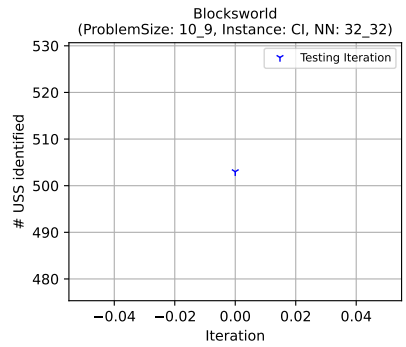

2-Way line
(Instance: park, NN: 32_32)
Testing Iteration
USS identified
Iteration

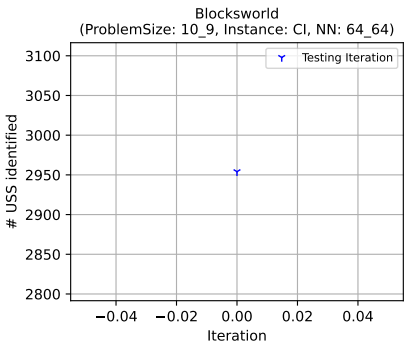

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

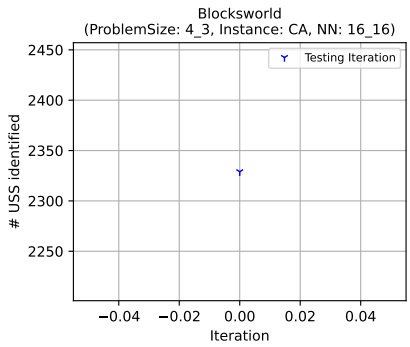

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

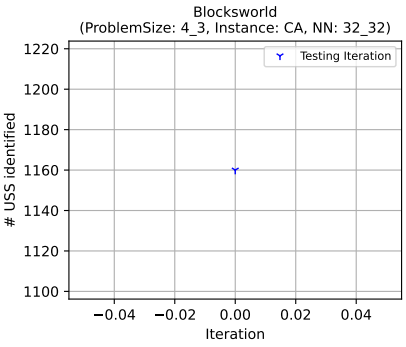

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

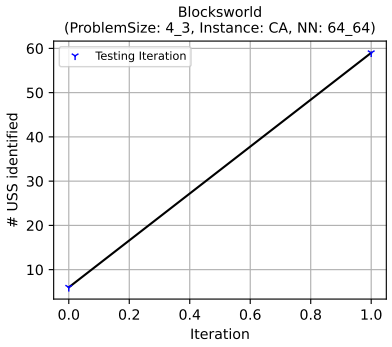

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

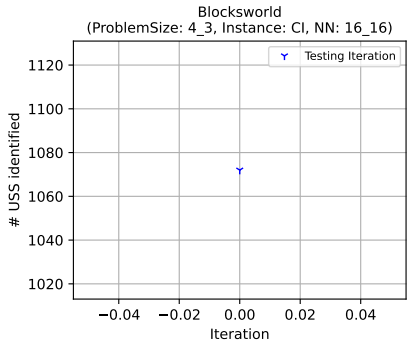

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

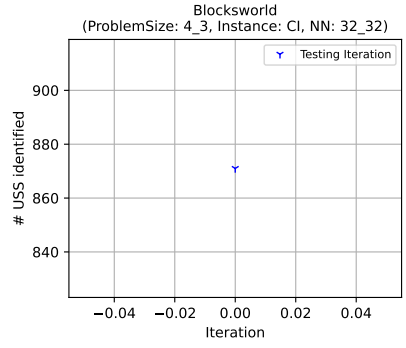

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

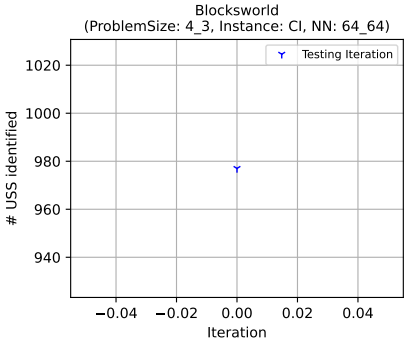

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

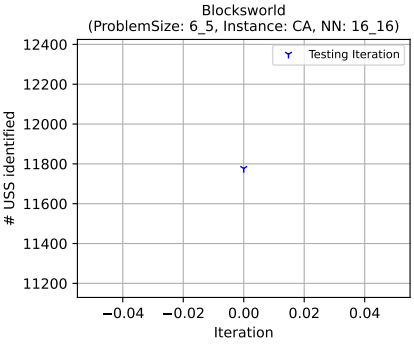

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

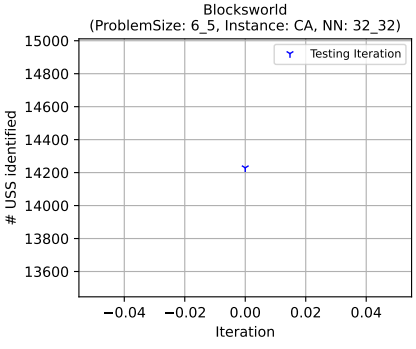

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

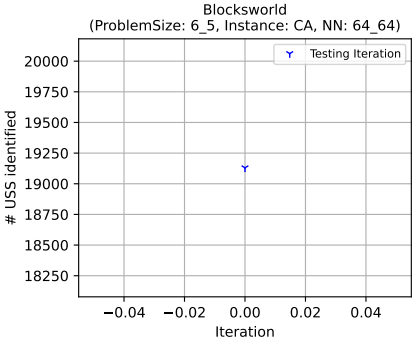

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

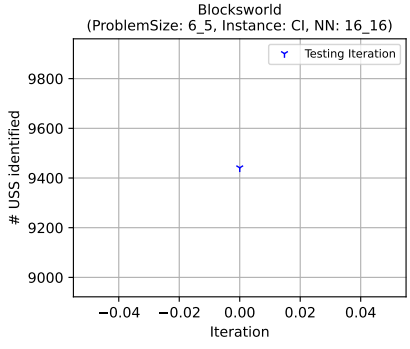

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

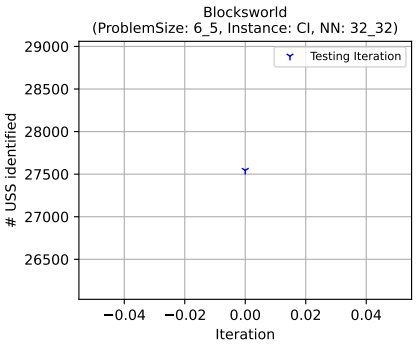

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

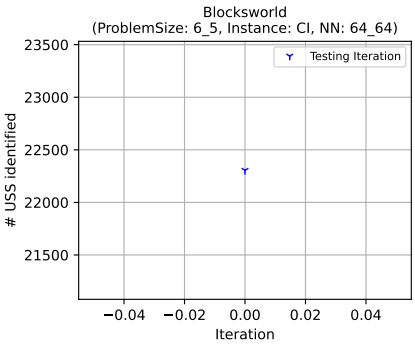

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

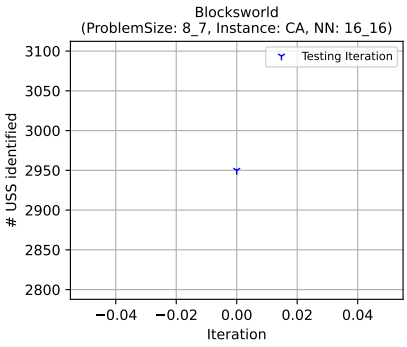

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

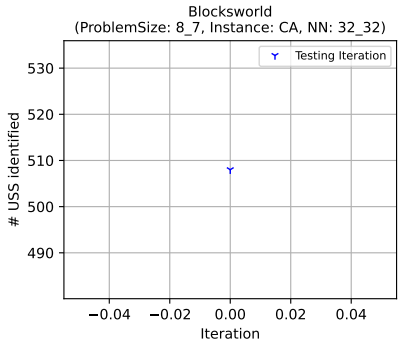

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

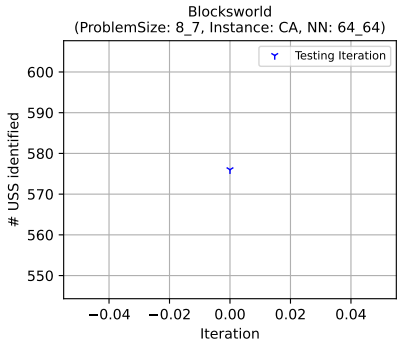

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

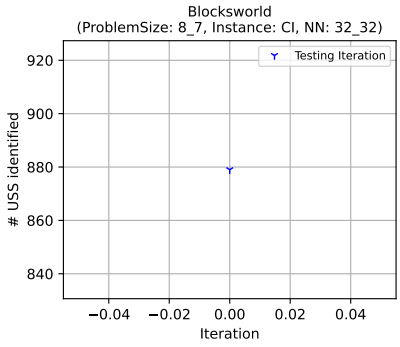

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

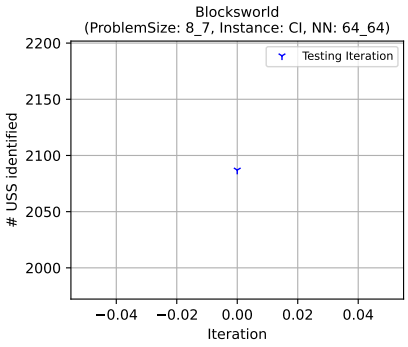

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

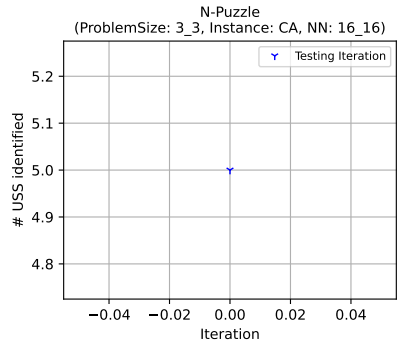

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

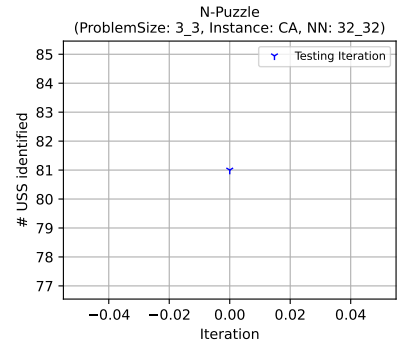

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

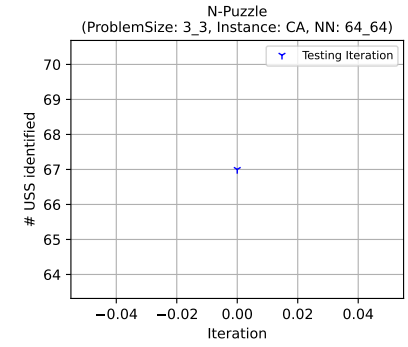

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

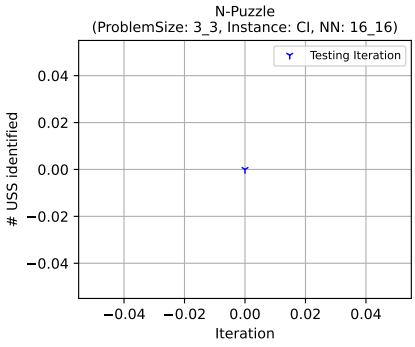

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

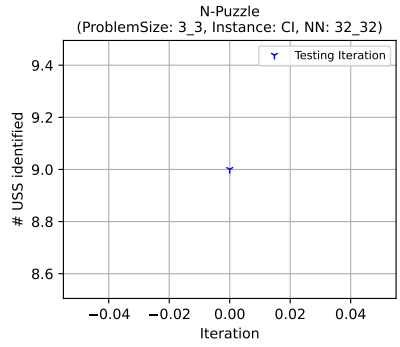

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

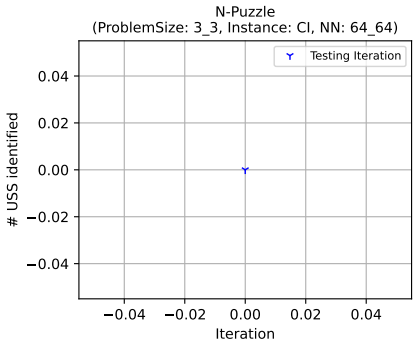

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

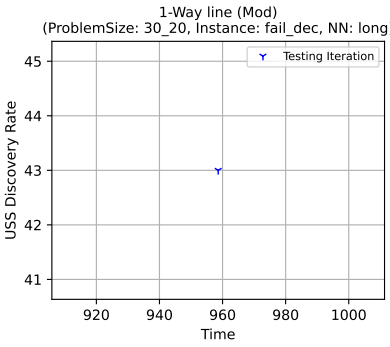

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

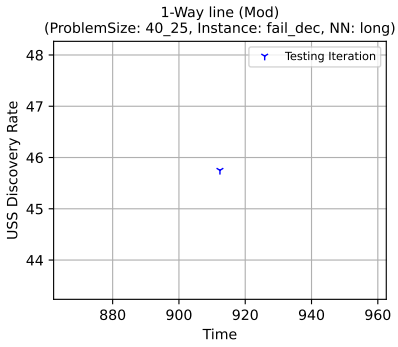

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

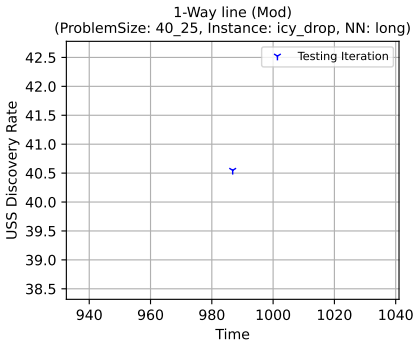

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

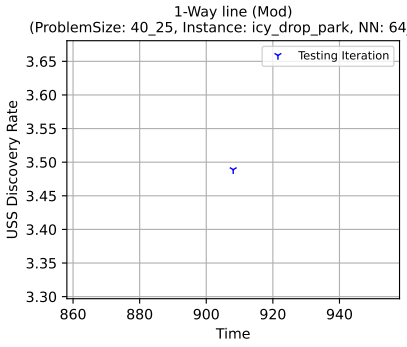

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

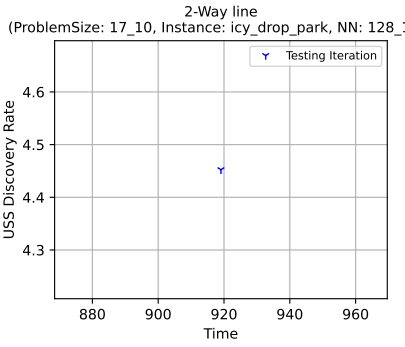

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

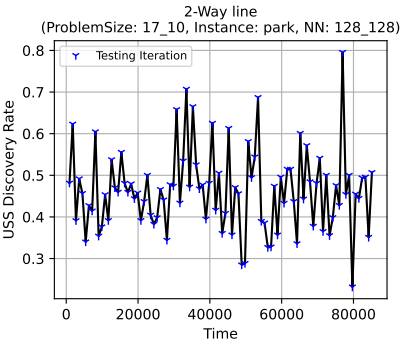

1-Way line (Mod)
(ProblemSize: 30_20, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

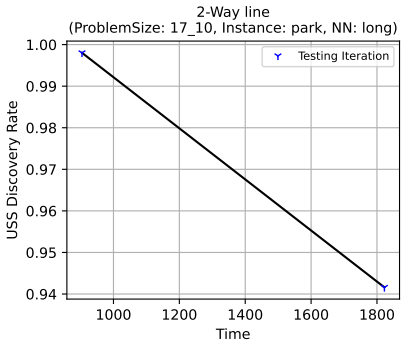

1-Way line (Mod)
(ProblemSize: 40_25, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

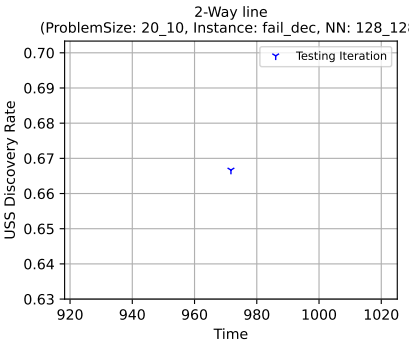

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop, NN: long)
Testing Iteration
USS Discovery Rate
Time

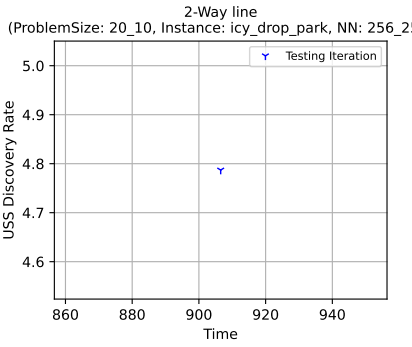

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

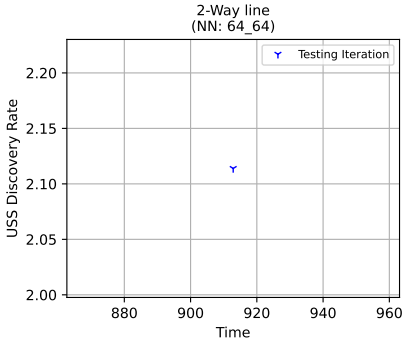

2-Way line
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

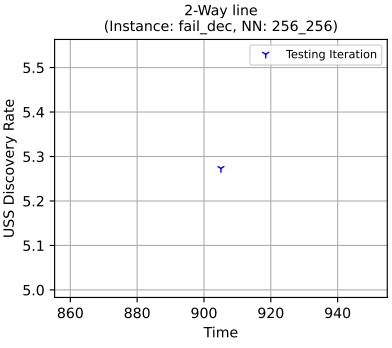

2-Way line
(ProblemSize: 17_10, Instance: park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

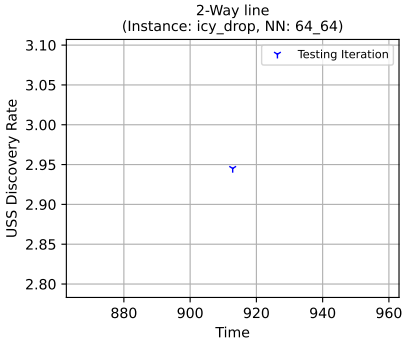

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
USS Discovery Rate
Time

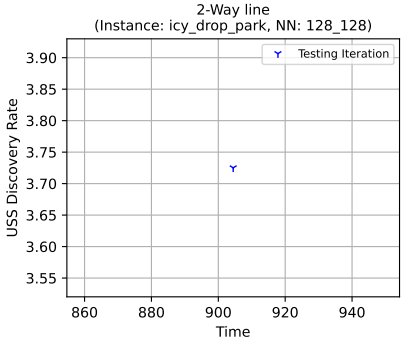

2-Way line
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

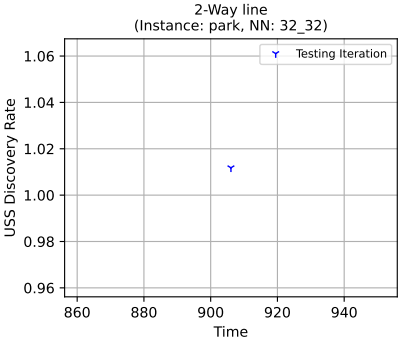

2-Way line
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

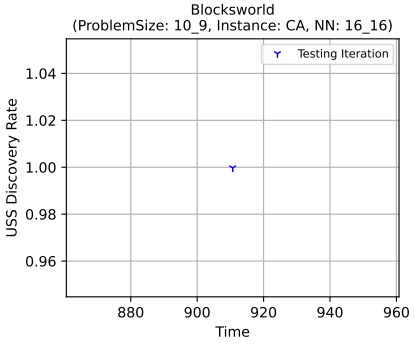

2-Way line
(NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

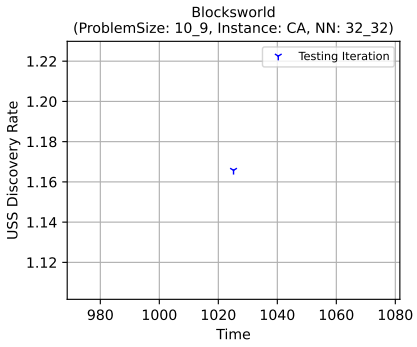

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

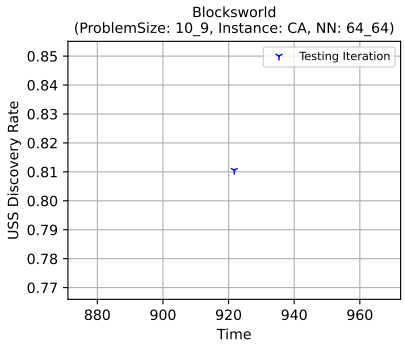

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

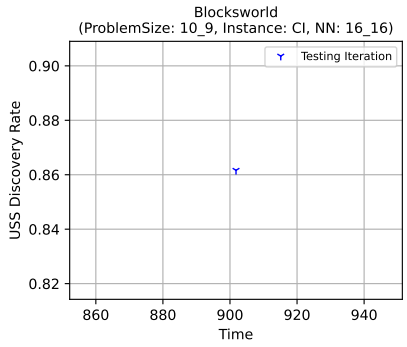

2-Way line
(Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

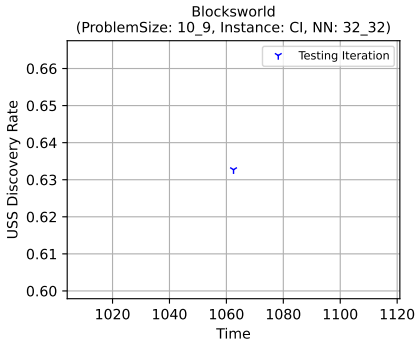

2-Way line
(Instance: park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

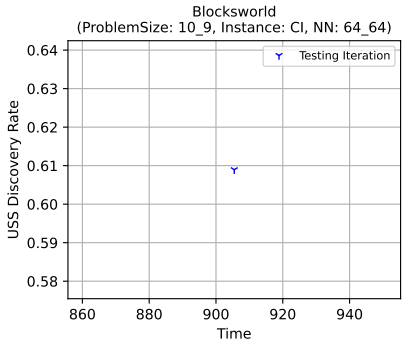

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

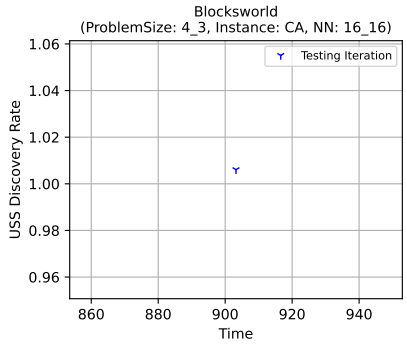

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

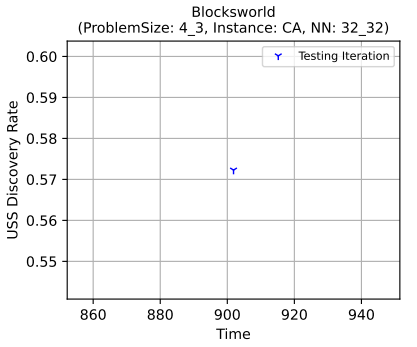

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

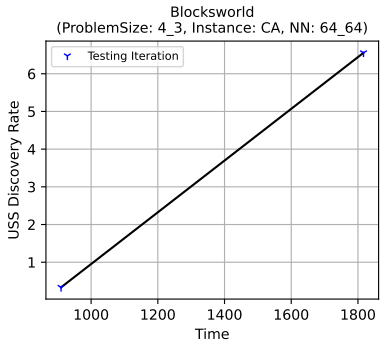

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

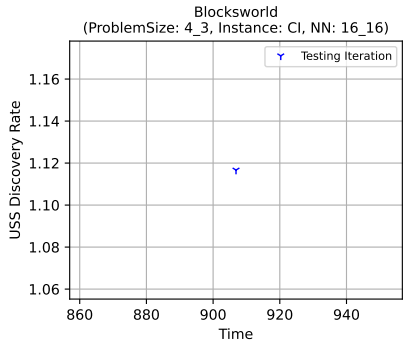

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

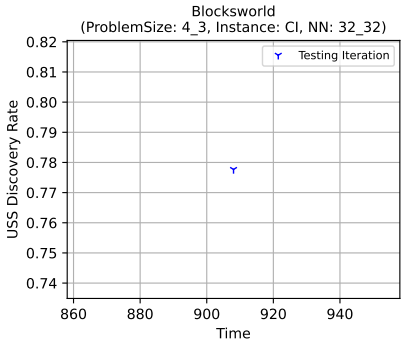

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

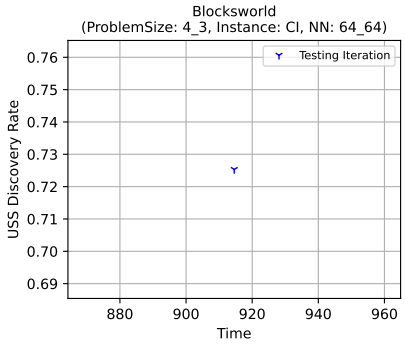

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

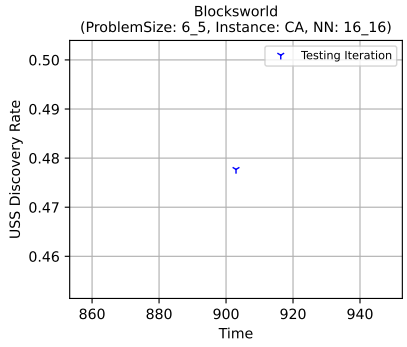

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

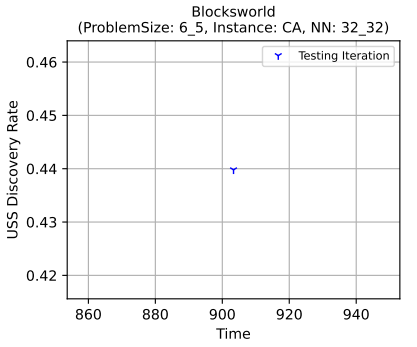

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

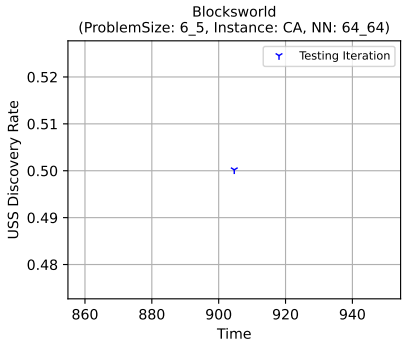

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

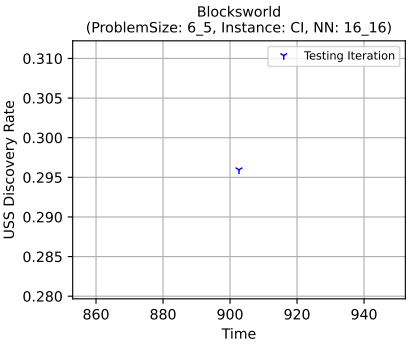

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

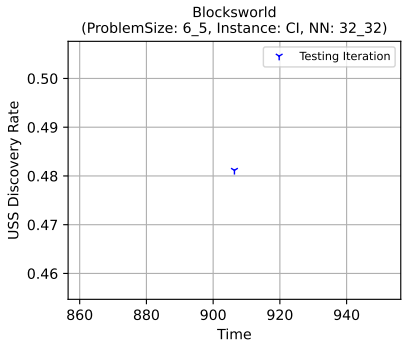

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

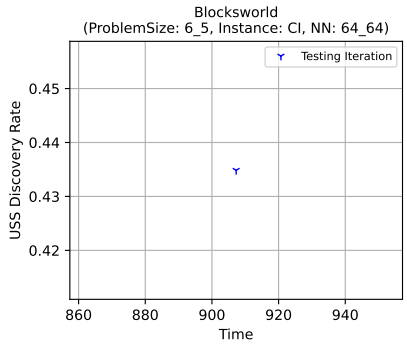

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

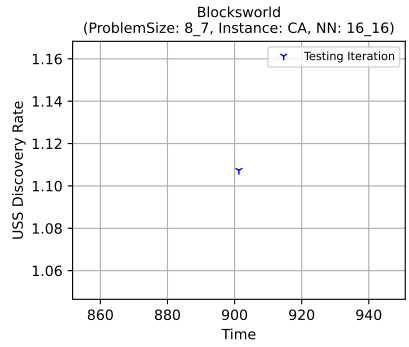

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

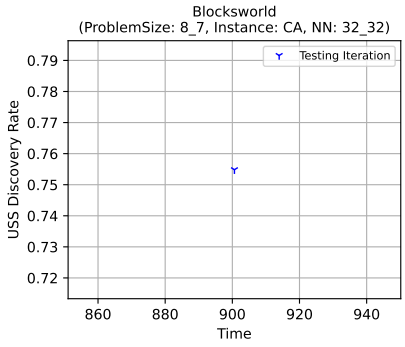

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

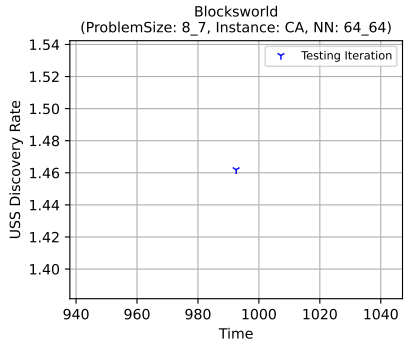

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

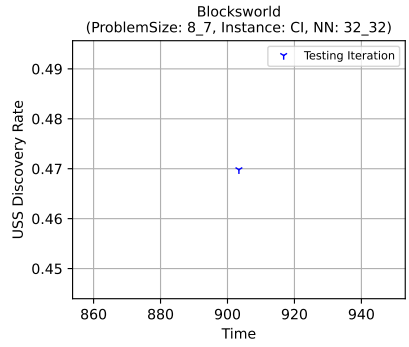

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

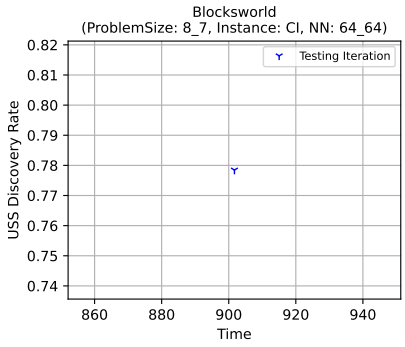

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

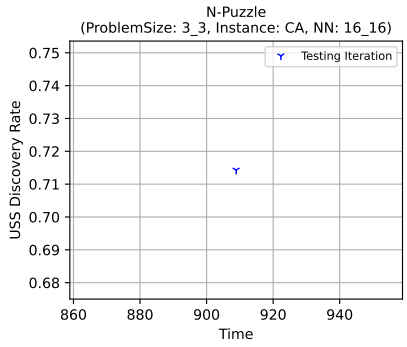

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

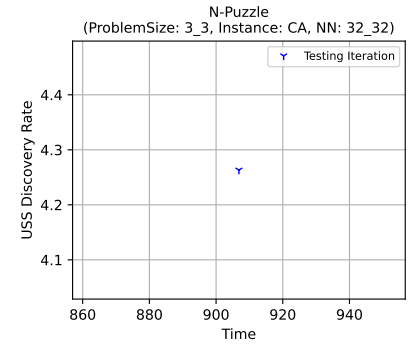

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

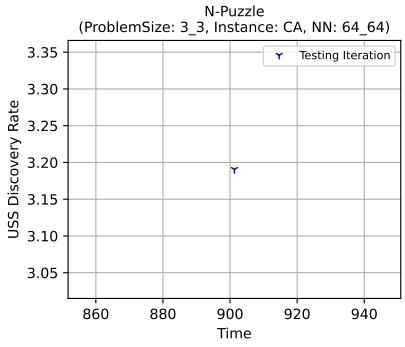

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

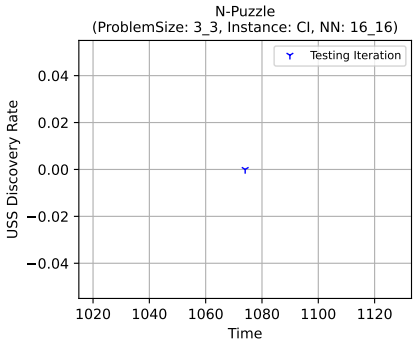

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

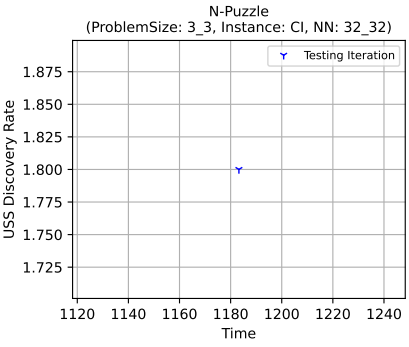

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

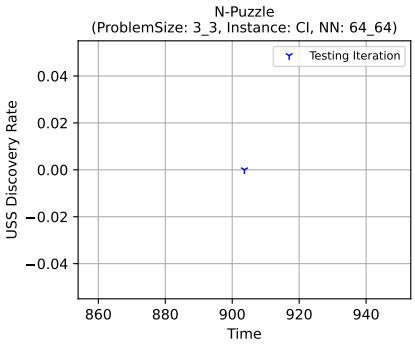

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

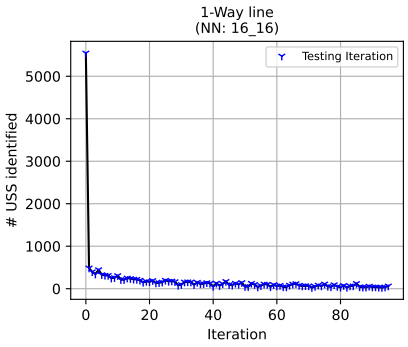

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

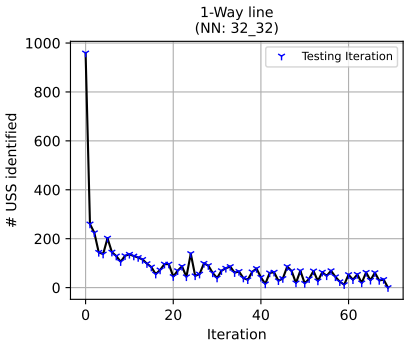

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

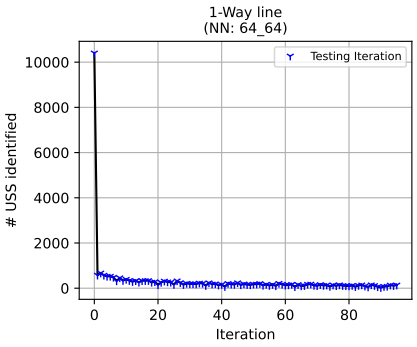

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

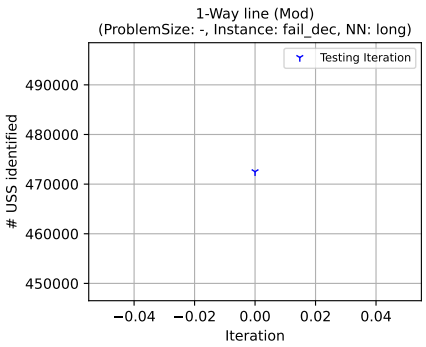
- Results for chapter 4 -

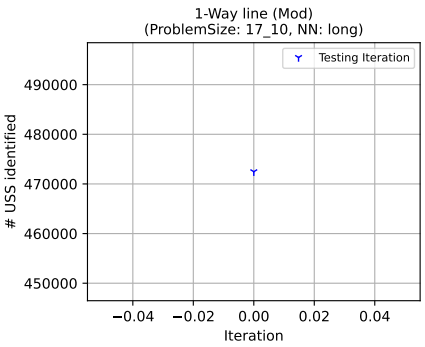
Start Condition Strengthening (Uniform Sampling)

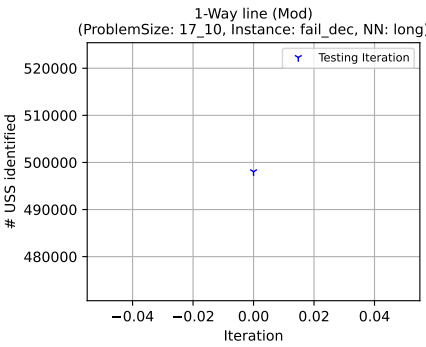

1-Way line
(NN: 16_16)
Testing Iteration
USS identified
Iteration

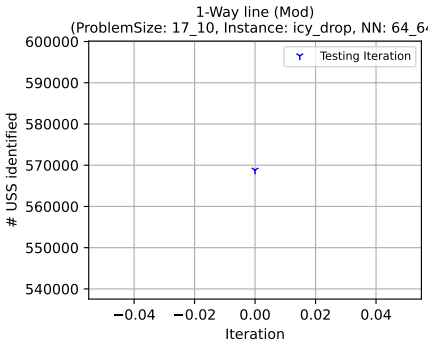

1-Way line
(NN: 32_32)
Testing Iteration
USS identified
Iteration

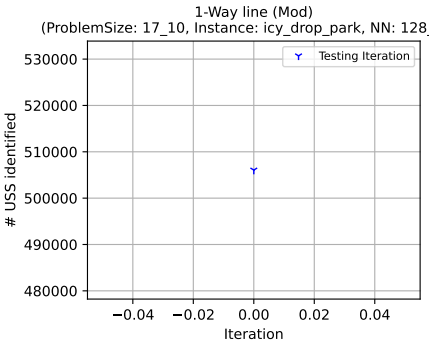

1-Way line
(NN: 64_64)
Testing Iteration
USS identified
Iteration

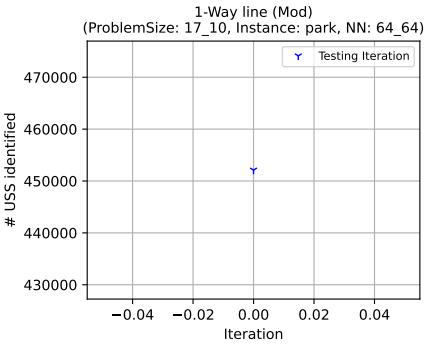

1-Way line (Mod)
(ProblemSize: -, Instance: fail_dec, NN: long)
Testing Iteration
USS identified
Iteration

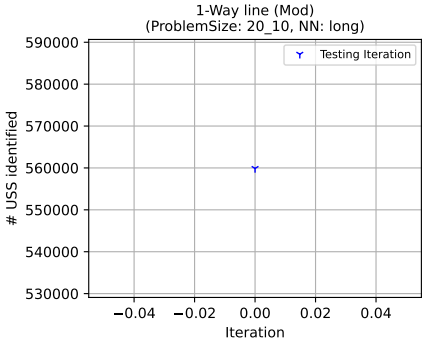

1-Way line (Mod)
(ProblemSize: 17_10, NN: long)
Testing Iteration
USS identified
Iteration

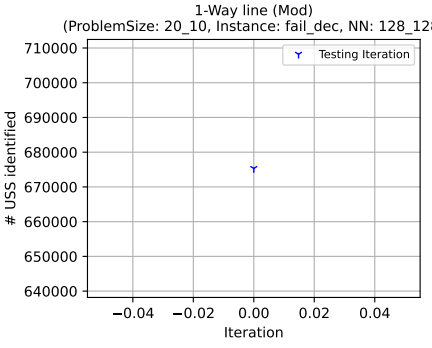

1-Way line (Mod)
(ProblemSize: 17_10, Instance: fail_dec, NN: long)
Testing Iteration
USS identified
Iteration

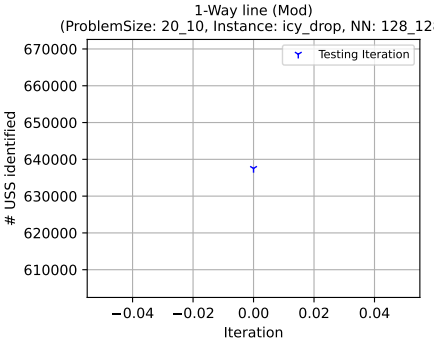

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Testing Iteration
USS identified
Iteration

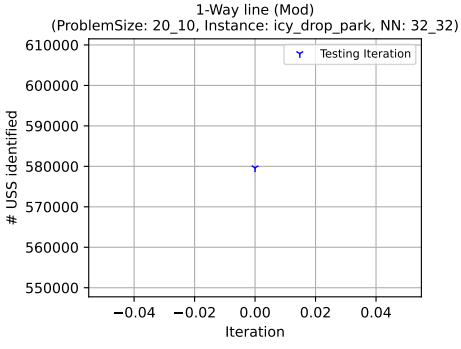

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS identified
Iteration

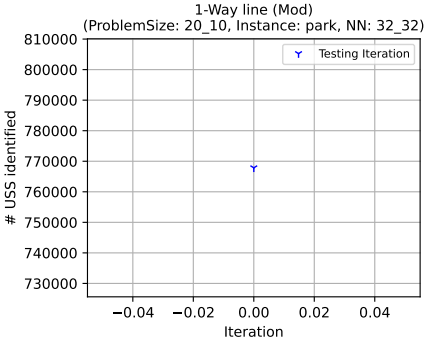

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Testing Iteration
USS identified
Iteration

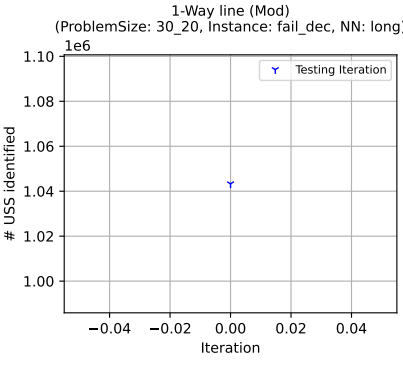

1-Way line (Mod)
(ProblemSize: 20_10, NN: long)
Testing Iteration
USS identified
Iteration

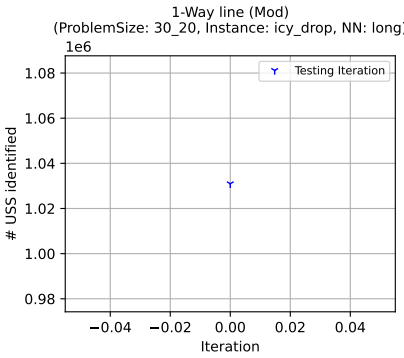

1-Way line (Mod)
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS identified
Iteration

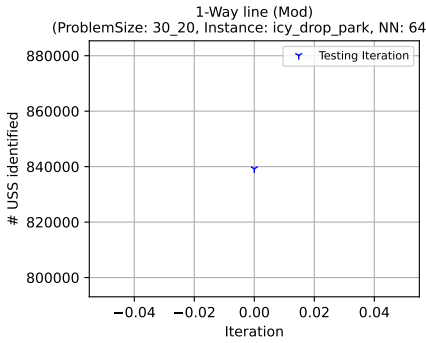

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
USS identified
Iteration

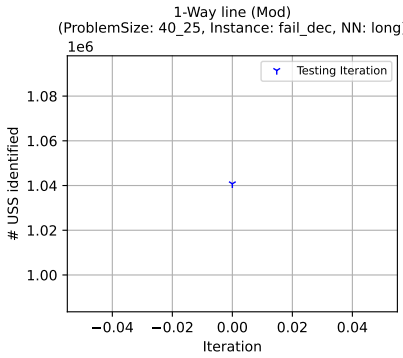

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 32_32)
Testing Iteration
USS identified
Iteration

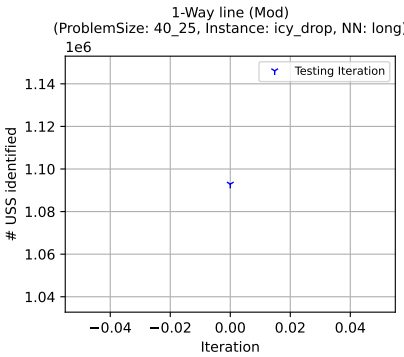

1-Way line (Mod)
(ProblemSize: 20_10, Instance: park, NN: 32_32)
Testing Iteration
USS identified
Iteration

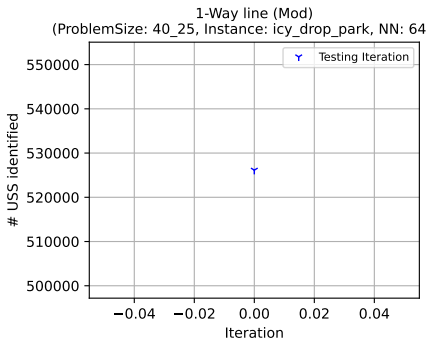

1-Way line (Mod)
(ProblemSize: 30_20, Instance: fail_dec, NN: long)
1e6
Testing Iteration
USS identified
Iteration

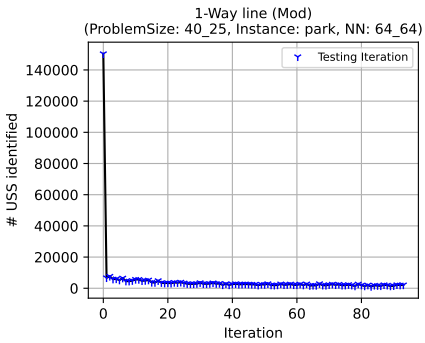

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop, NN: long)
1e6
Testing Iteration
USS identified
Iteration

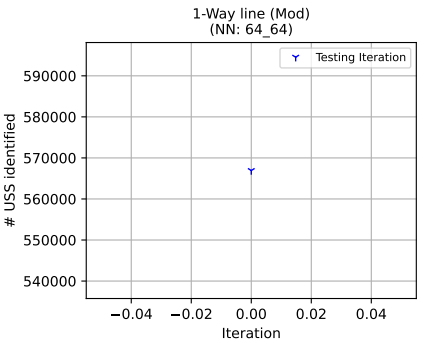

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS identified
Iteration

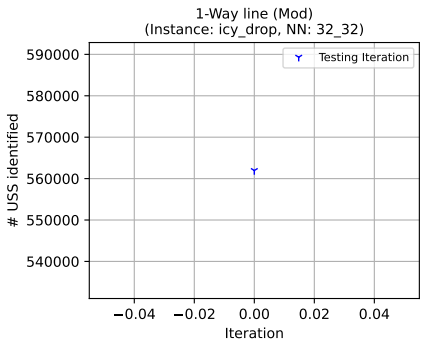

1-Way line (Mod)
(ProblemSize: 40_25, Instance: fail_dec, NN: long)
1e6
Testing Iteration
USS identified
Iteration

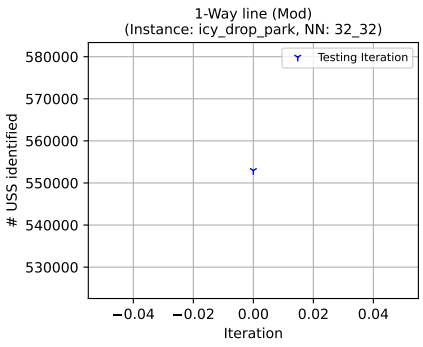

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop, NN: long)
1e6
Testing Iteration
USS identified
Iteration

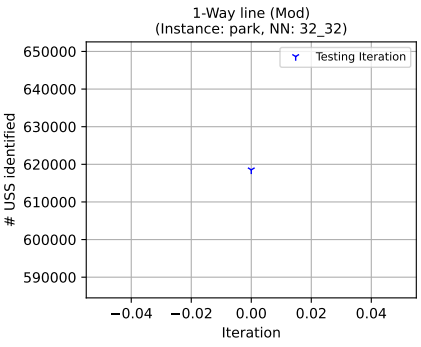

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS identified
Iteration

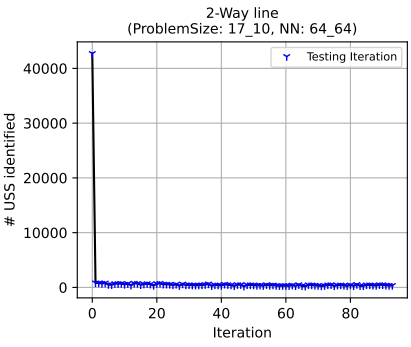

1-Way line (Mod)
(ProblemSize: 40_25, Instance: park, NN: 64_64)
Testing Iteration
USS identified
Iteration

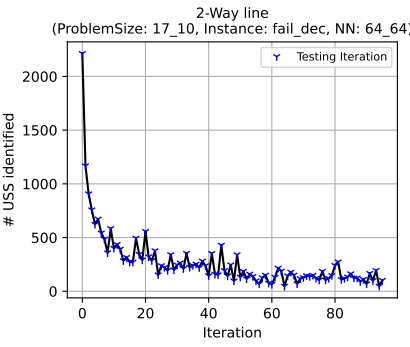

1-Way line (Mod)
(NN: 64_64)
Testing Iteration
USS identified
Iteration

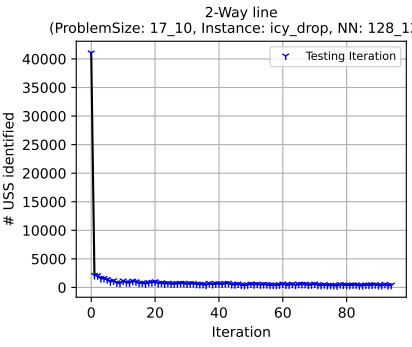

1-Way line (Mod)
(Instance: icy_drop, NN: 32_32)
Testing Iteration
USS identified
Iteration

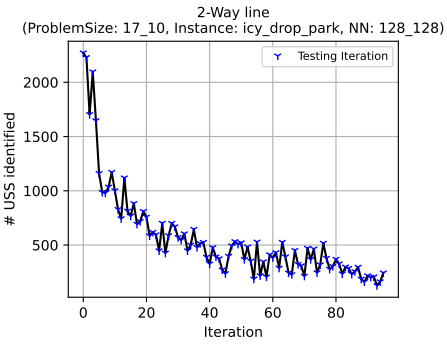

1-Way line (Mod)
(Instance: icy_drop_park, NN: 32_32)
Testing Iteration
USS identified
Iteration

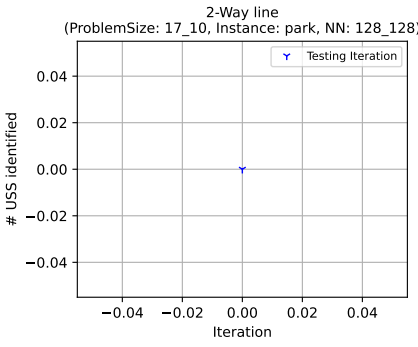

1-Way line (Mod)
(Instance: park, NN: 32_32)
Testing Iteration
USS identified
Iteration

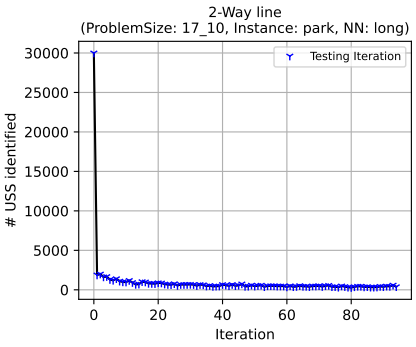

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Testing Iteration
USS identified
Iteration

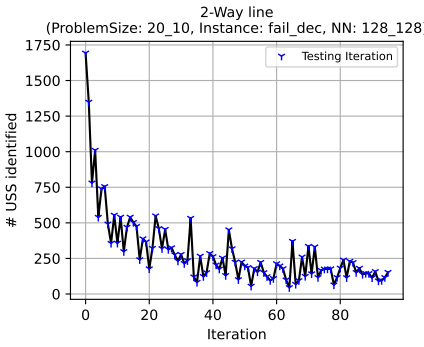

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS identified
Iteration

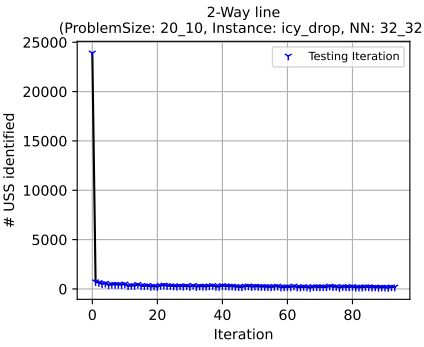

2-Way line
(ProblemSize: 17_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
USS identified
Iteration

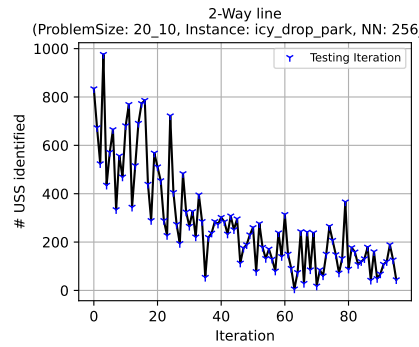

2-Way line
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS identified
Iteration

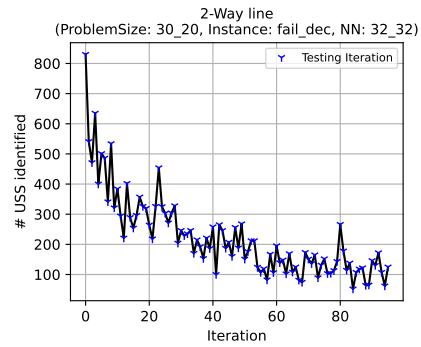

2-Way line
(ProblemSize: 17_10, Instance: park, NN: 128_128)
Testing Iteration
USS identified
Iteration

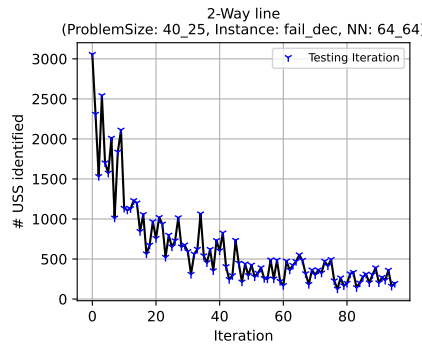

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
USS identified
Iteration

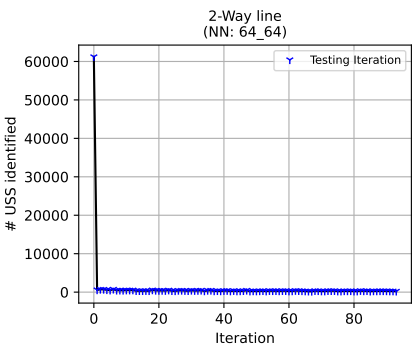

2-Way line
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS identified
Iteration

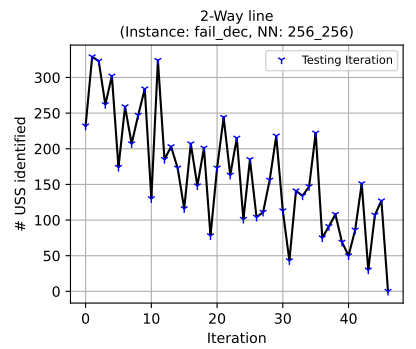

2-Way line
(ProblemSize: 20_10, Instance: icy_drop, NN: 32_32)
Testing Iteration
USS identified
Iteration

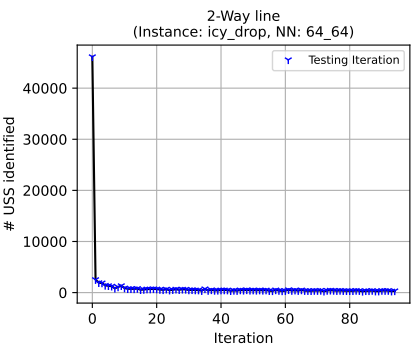

2-Way line
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 256_256)
Testing Iteration
USS identified
Iteration

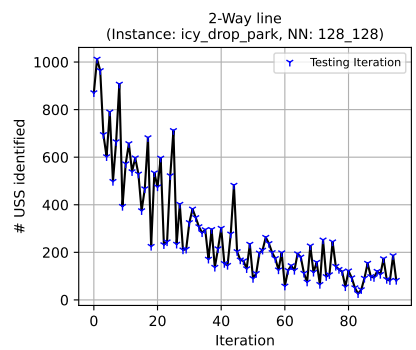

2-Way line
(ProblemSize: 30_20, Instance: fail_dec, NN: 32_32)
Testing Iteration
USS identified
Iteration

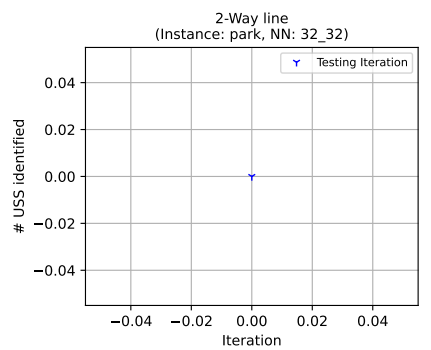

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS identified
Iteration

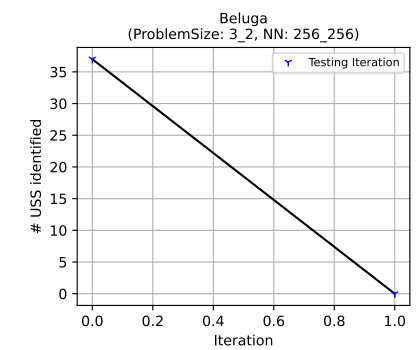

2-Way line
(NN: 64_64)
Testing Iteration
USS identified
Iteration

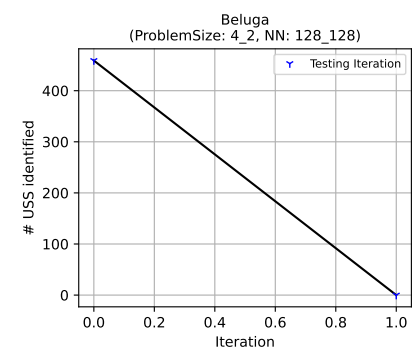

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
USS identified
Iteration

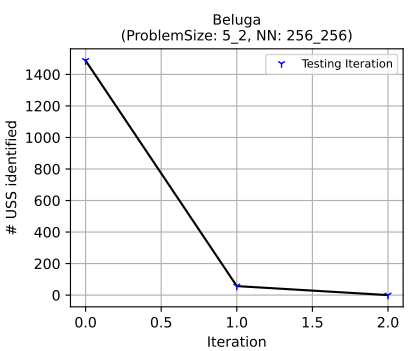

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
USS identified
Iteration

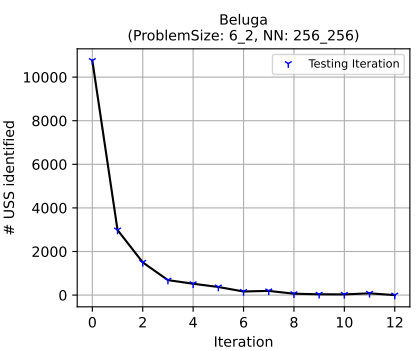

2-Way line
(Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS identified
Iteration

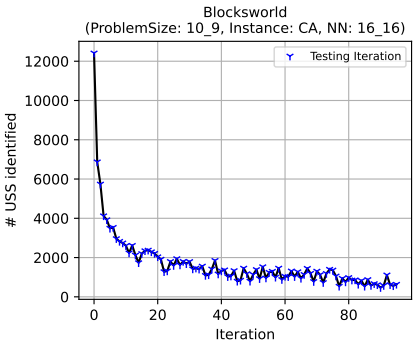

2-Way line
(Instance: park, NN: 32_32)
Testing Iteration
USS identified
Iteration

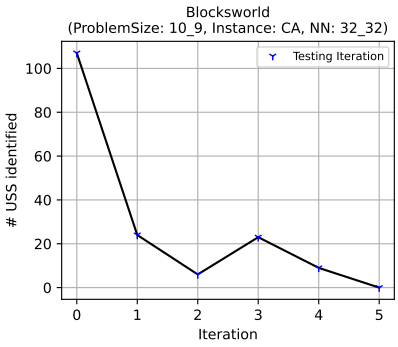

Beluga
(ProblemSize: 3_2, NN: 256_256)
Testing Iteration
USS identified
Iteration

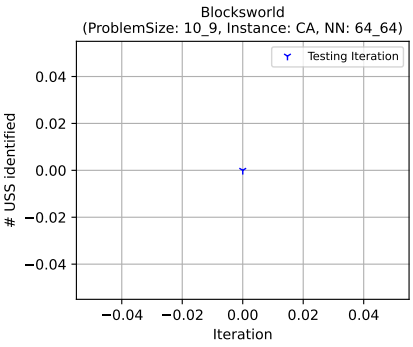

Beluga
(ProblemSize: 4_2, NN: 128_128)
Testing Iteration
USS identified
Iteration

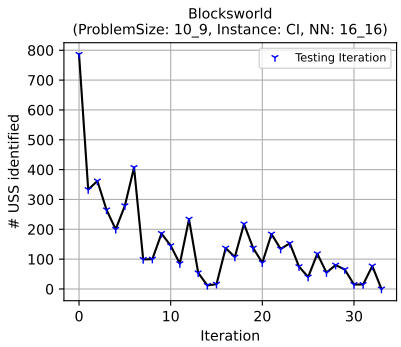

Beluga
(ProblemSize: 5_2, NN: 256_256)
Testing Iteration
USS identified
Iteration

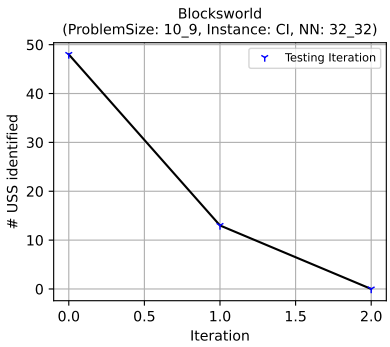

Beluga
(ProblemSize: 6_2, NN: 256_256)
Testing Iteration
USS identified
Iteration

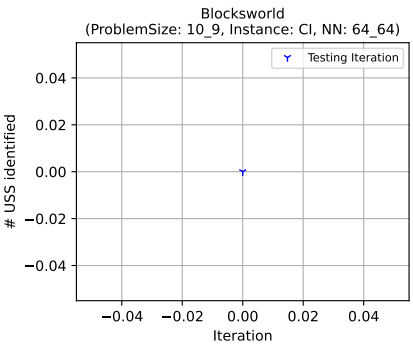

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

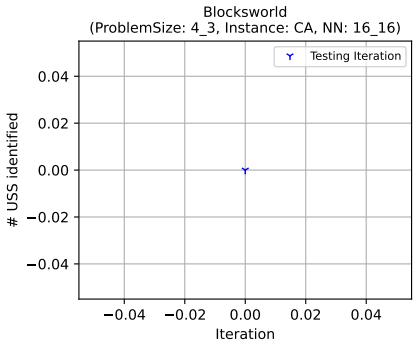

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

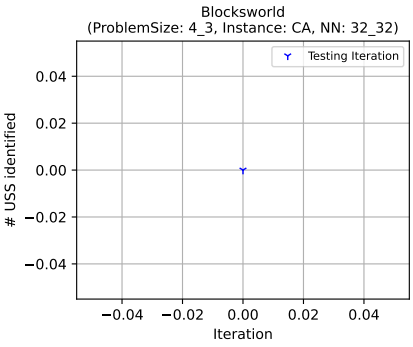

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

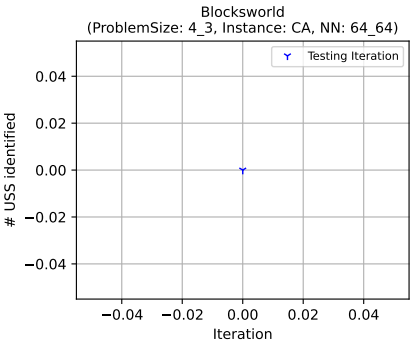

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

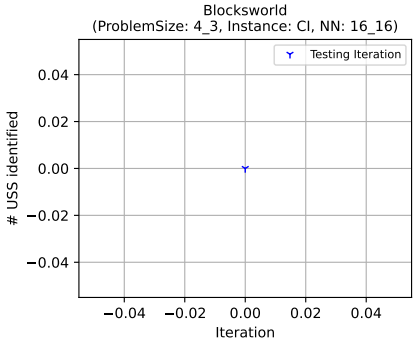

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

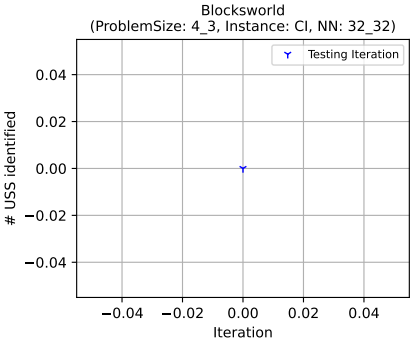

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

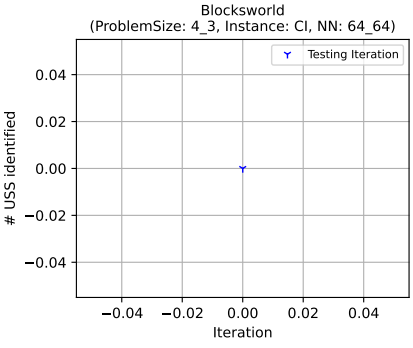

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

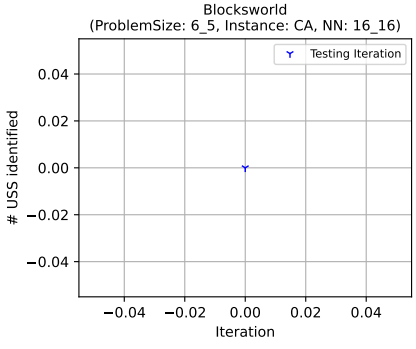

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

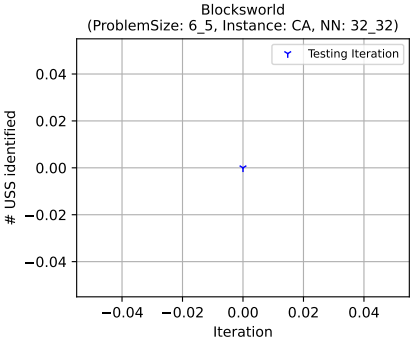

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

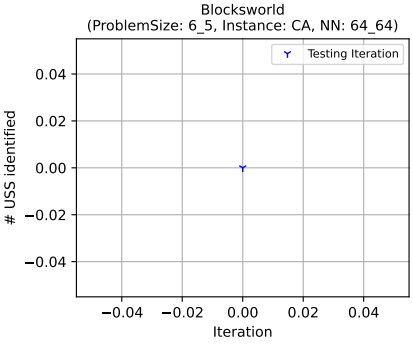

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

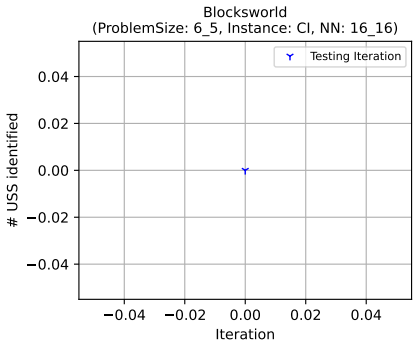

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

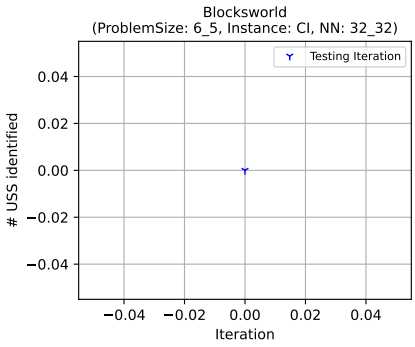

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

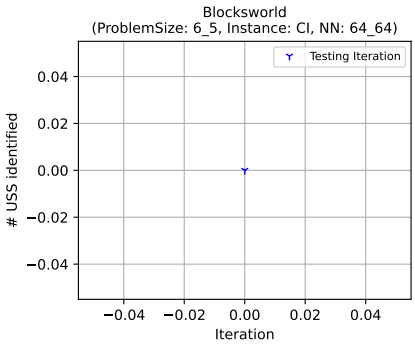

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

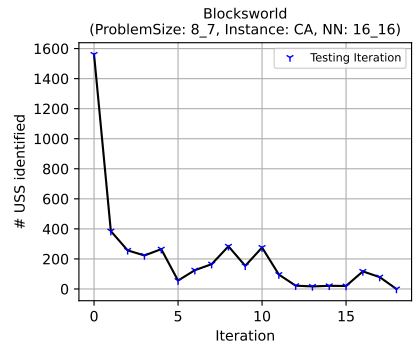

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

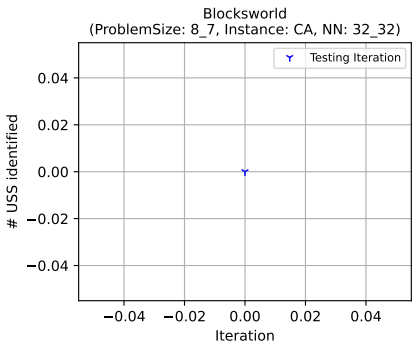

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

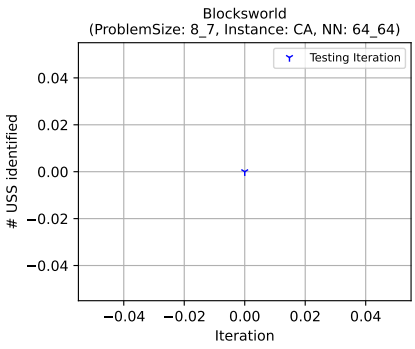

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

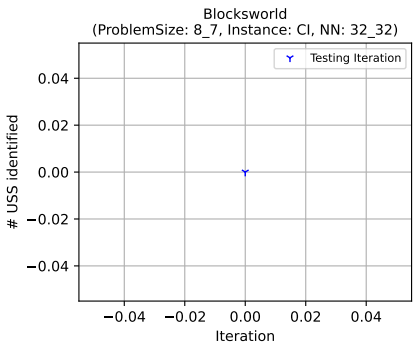

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

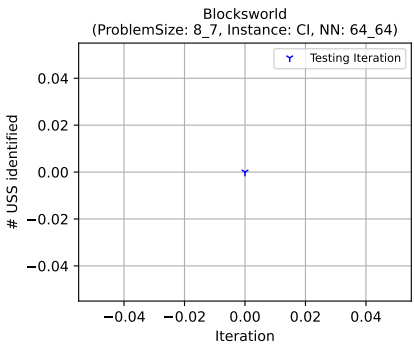

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

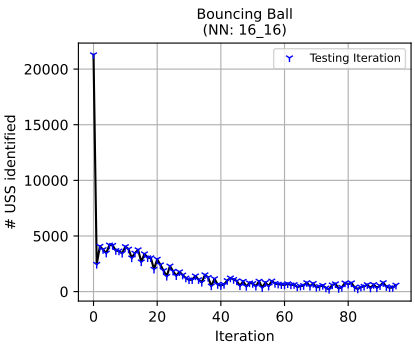

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

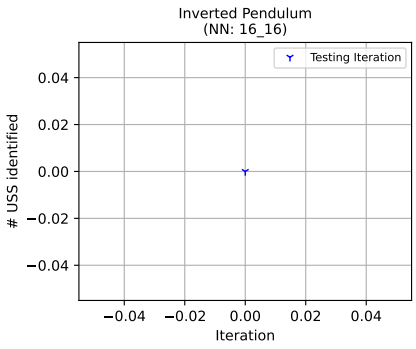

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

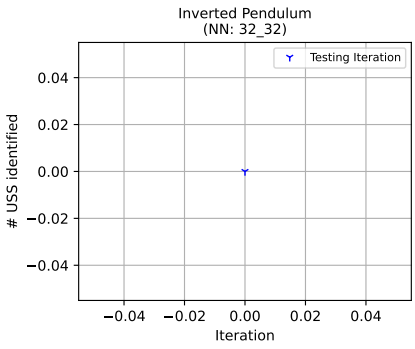

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

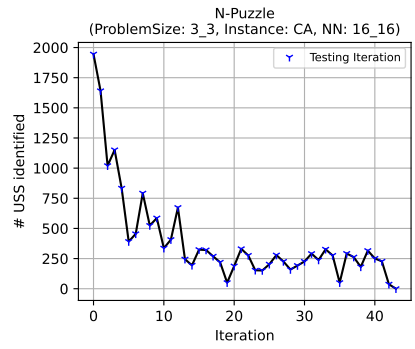

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

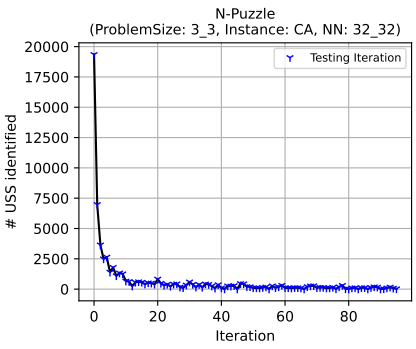

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

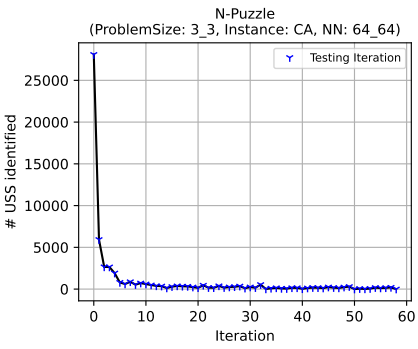

Bouncing Ball
(NN: 16_16)
Testing Iteration
USS identified
Iteration

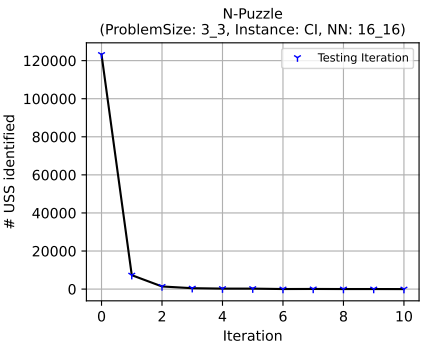

Inverted Pendulum
(NN: 16_16)
Testing Iteration
USS identified
Iteration

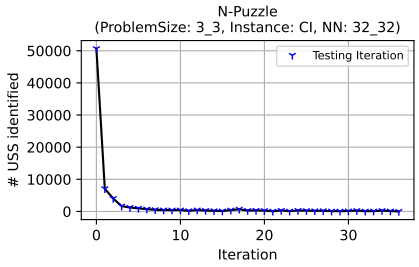

Inverted Pendulum
(NN: 32_32)
Testing Iteration
USS identified
Iteration

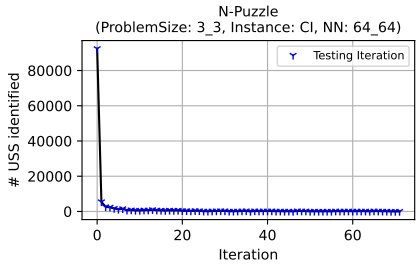

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

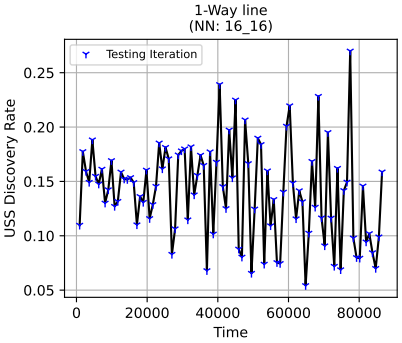

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

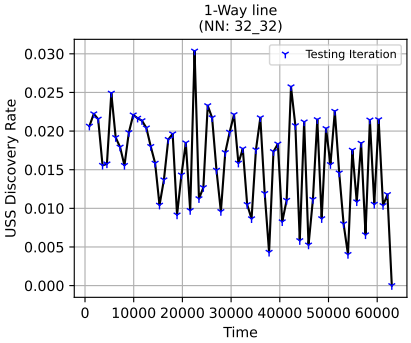

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

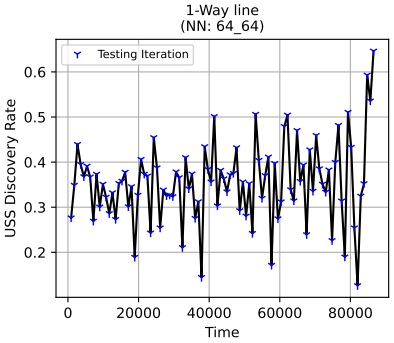

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

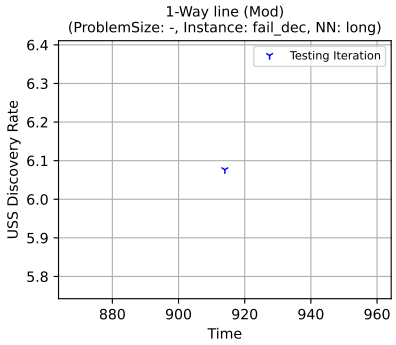

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration
0
10000
20000
30000
40000
50000
0
10
20
30

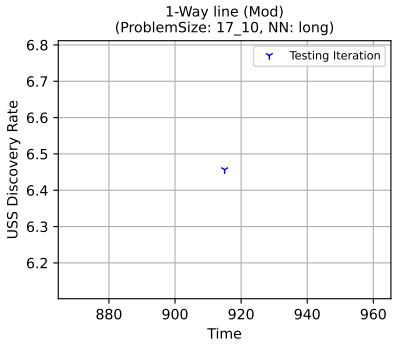

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration
0
20000
40000
60000
80000
0
20
40
60

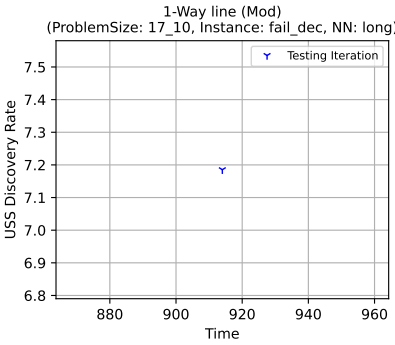

1-Way line
(NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

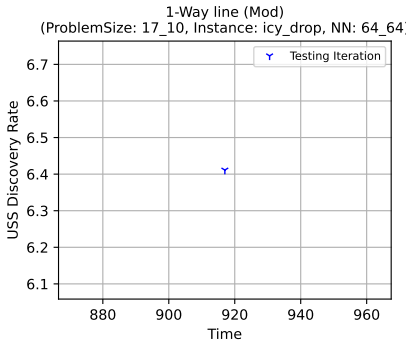

1-Way line
(NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

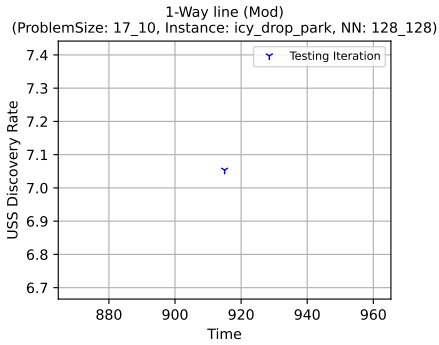

1-Way line
(NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

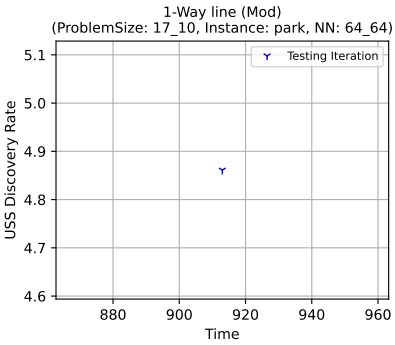

1-Way line (Mod)
(ProblemSize: -, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

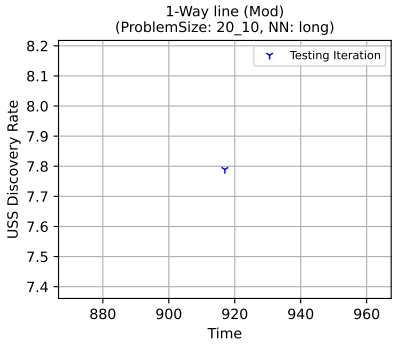

1-Way line (Mod)
(ProblemSize: 17_10, NN: long)
Testing Iteration
USS Discovery Rate
Time

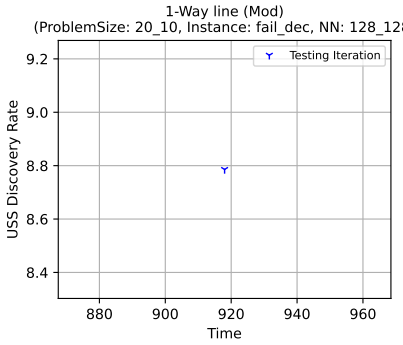

1-Way line (Mod)
(ProblemSize: 17_10, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

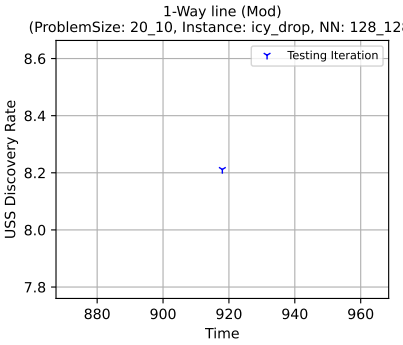

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

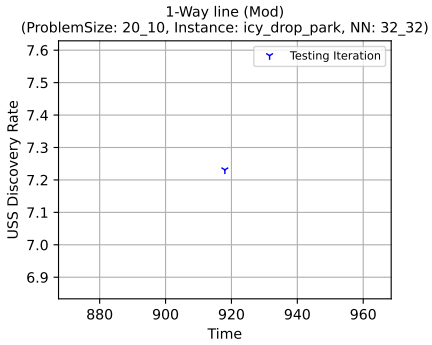

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

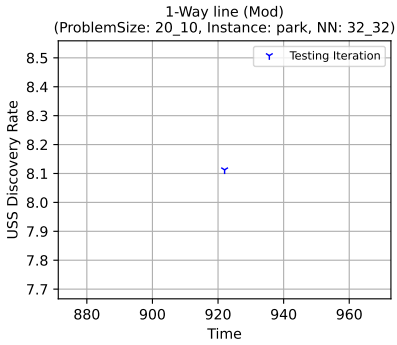

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

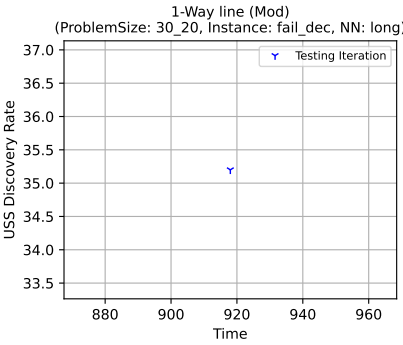

1-Way line (Mod)
(ProblemSize: 20_10, NN: long)
Testing Iteration
USS Discovery Rate
Time

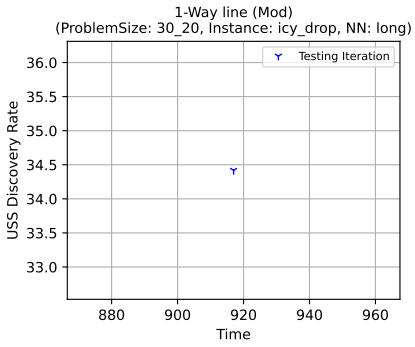

1-Way line (Mod)
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

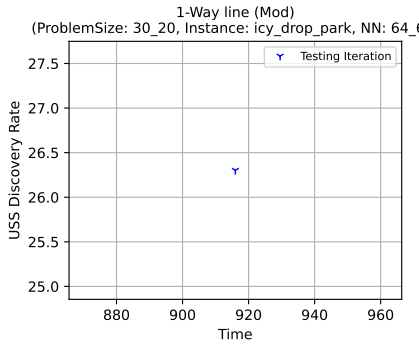

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

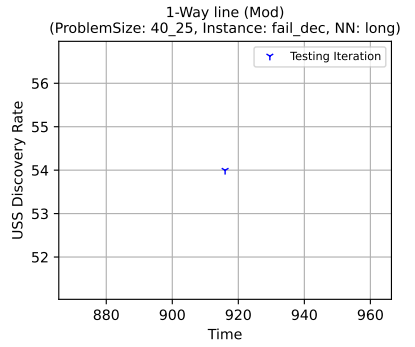

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

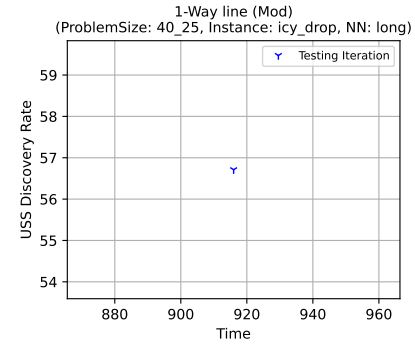

1-Way line (Mod)
(ProblemSize: 20_10, Instance: park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

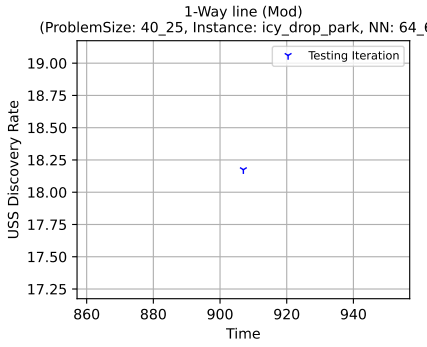

1-Way line (Mod)
(ProblemSize: 30_20, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

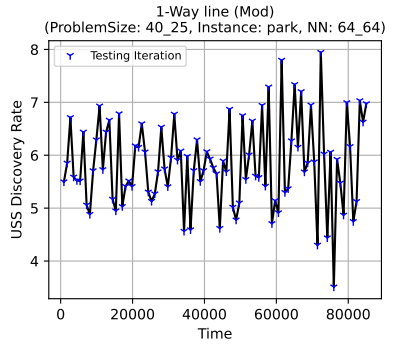

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop, NN: long)
Testing Iteration
USS Discovery Rate
Time

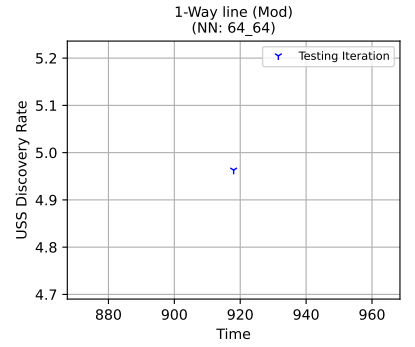

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

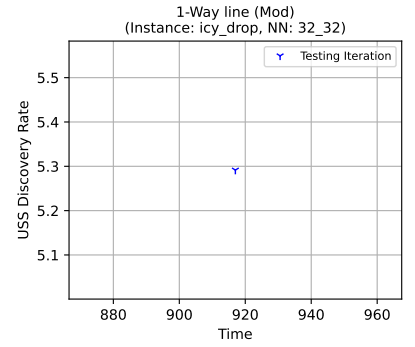

1-Way line (Mod)
(ProblemSize: 40_25, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

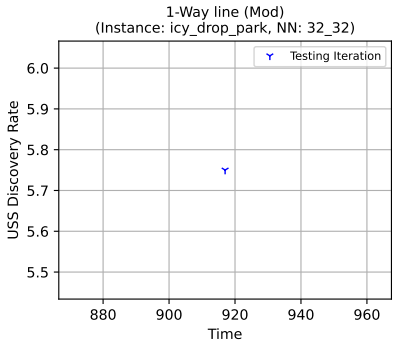

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop, NN: long)
Testing Iteration
USS Discovery Rate
Time

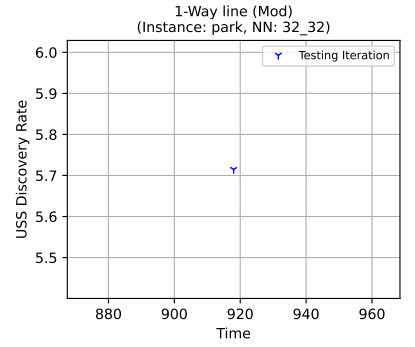

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

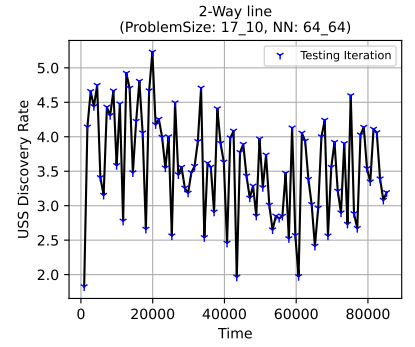

1-Way line (Mod)
(ProblemSize: 40_25, Instance: park, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

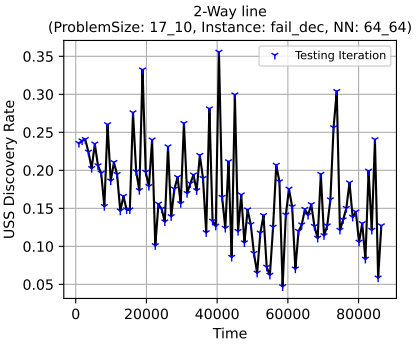

1-Way line (Mod)
(NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

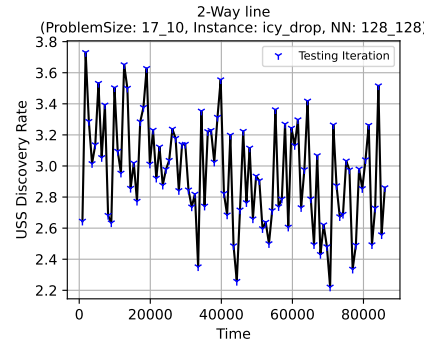

1-Way line (Mod)
(Instance: icy_drop, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

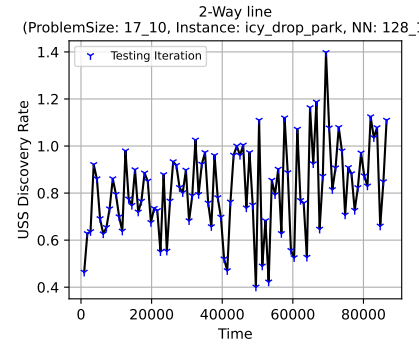

1-Way line (Mod)
(Instance: icy_drop_park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

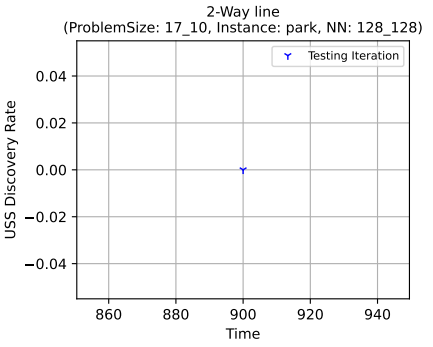

1-Way line (Mod)
(Instance: park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

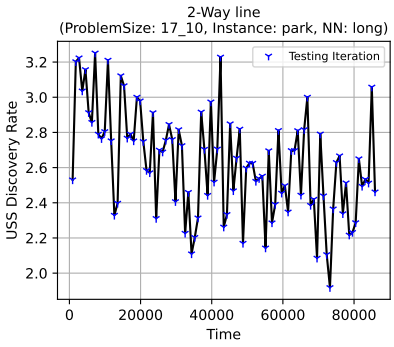

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

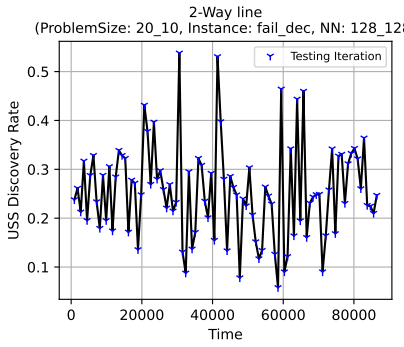

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

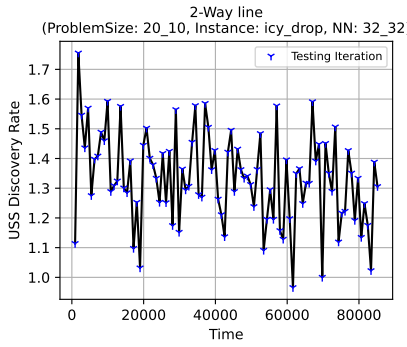

2-Way line
(ProblemSize: 17_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

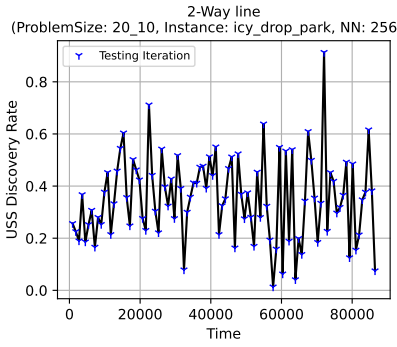

2-Way line
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

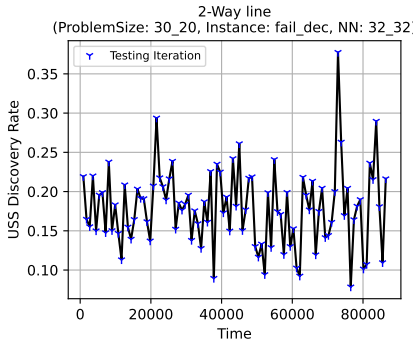

2-Way line
(ProblemSize: 17_10, Instance: park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

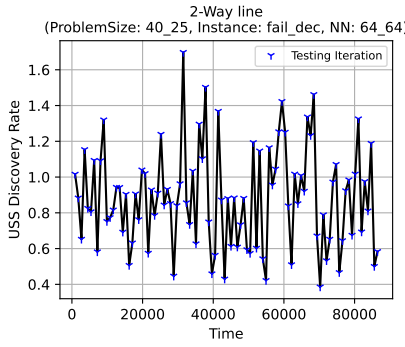

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
USS Discovery Rate
Time

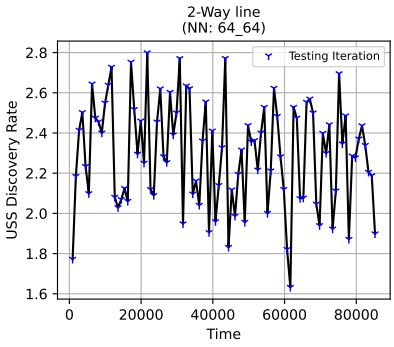

2-Way line
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

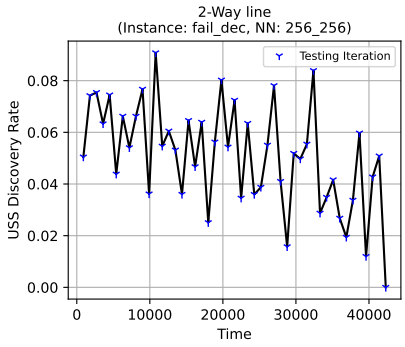

2-Way line
(ProblemSize: 20_10, Instance: icy_drop, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

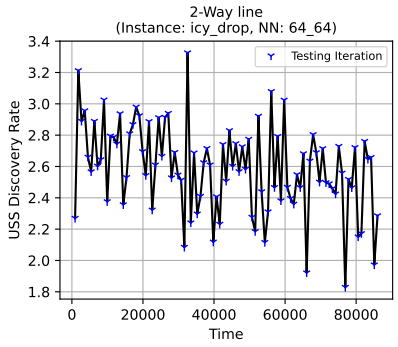

2-Way line
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

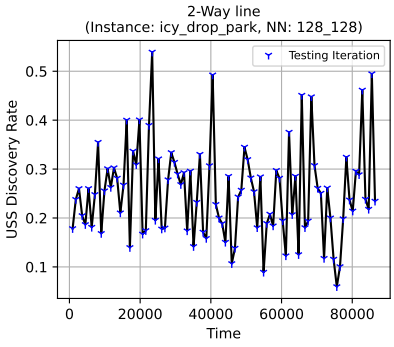

2-Way line
(ProblemSize: 30_20, Instance: fail_dec, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

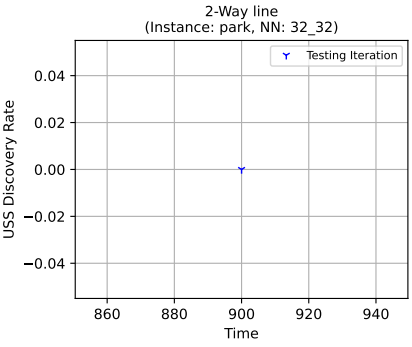

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

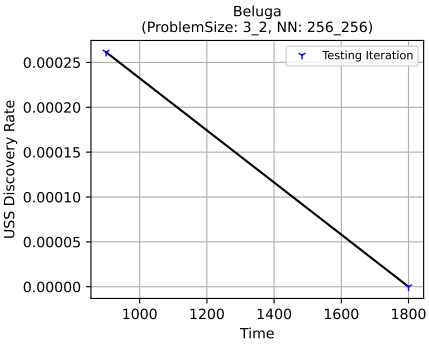

2-Way line
(NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

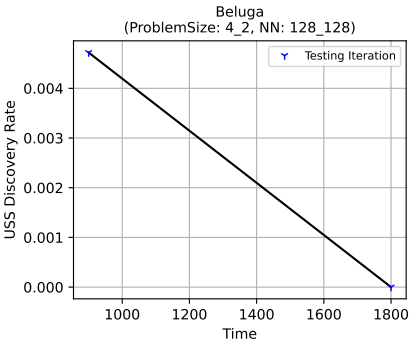

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

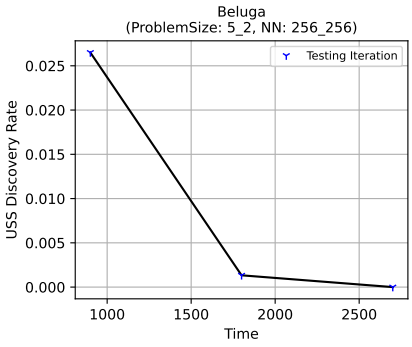

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

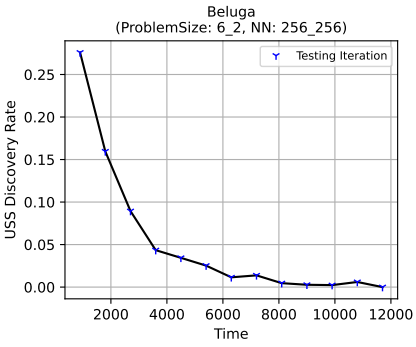

2-Way line
(Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

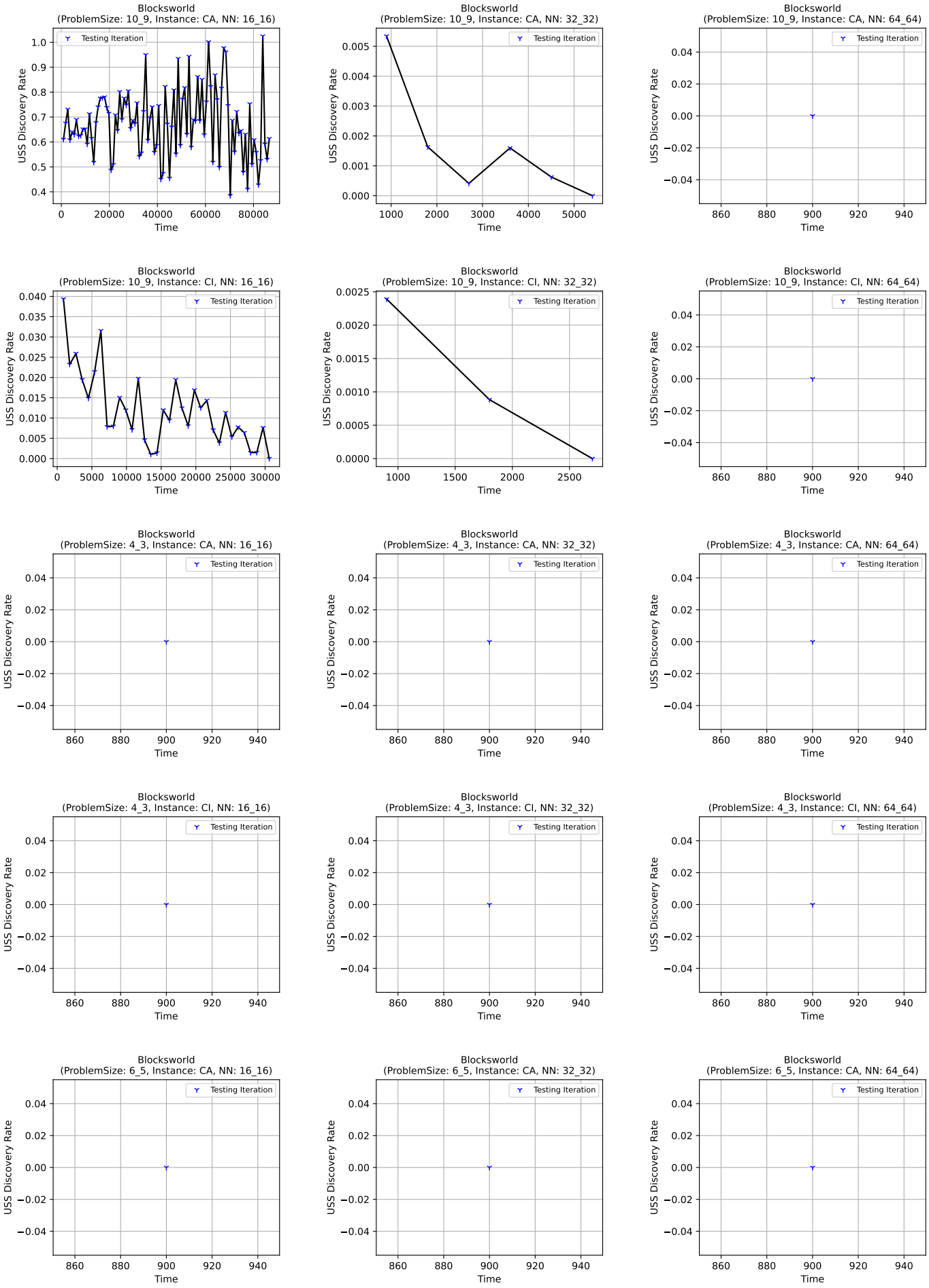

2-Way line
(Instance: park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

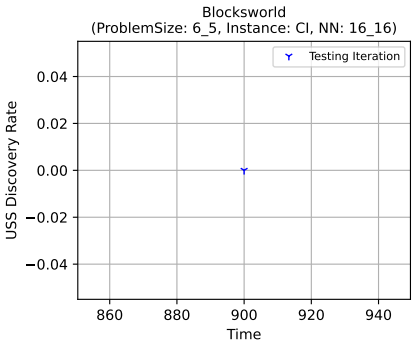

Beluga
(ProblemSize: 3_2, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

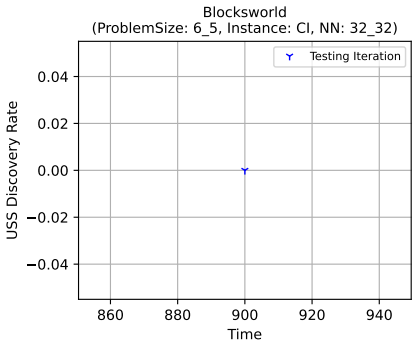

Beluga
(ProblemSize: 4_2, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

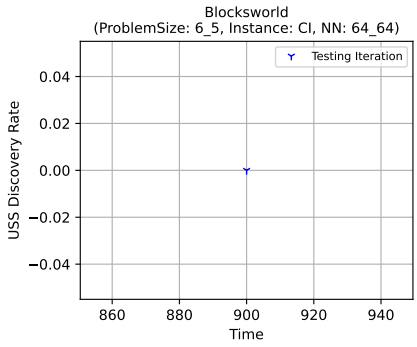

Beluga
(ProblemSize: 5_2, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

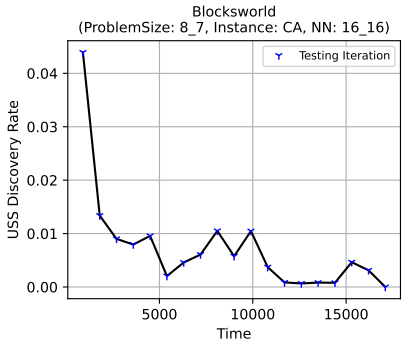

Beluga
(ProblemSize: 6_2, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

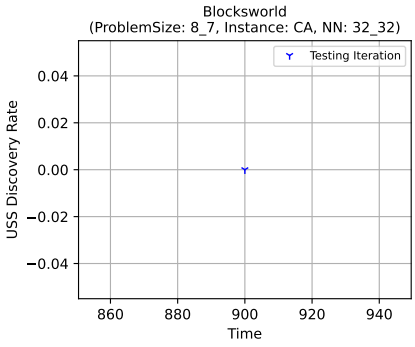

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

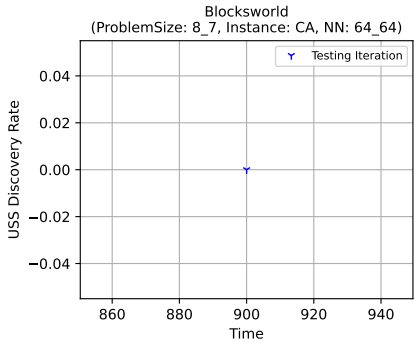

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

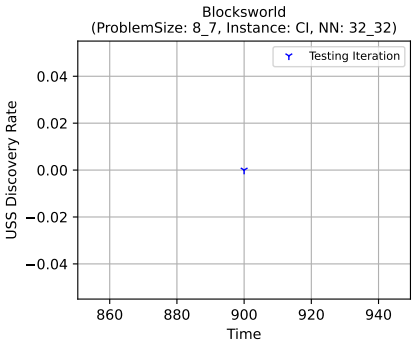

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

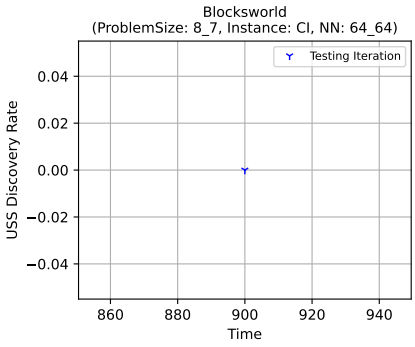

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

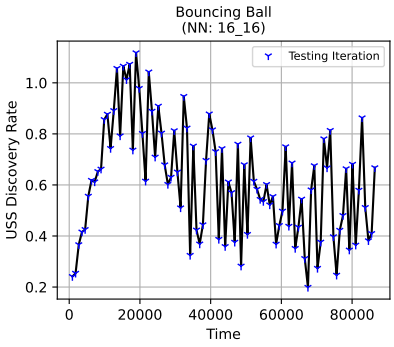

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

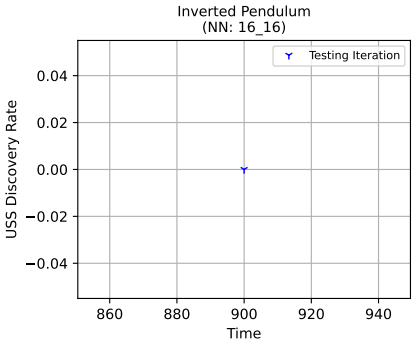

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

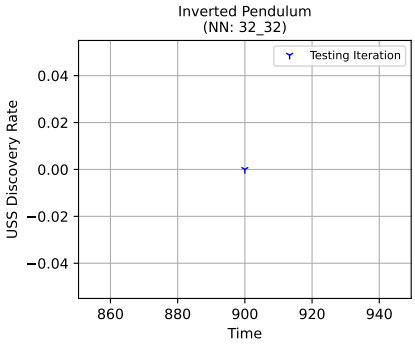

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

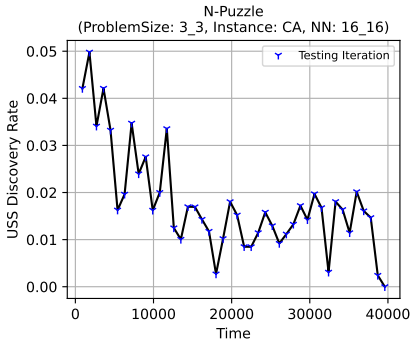

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

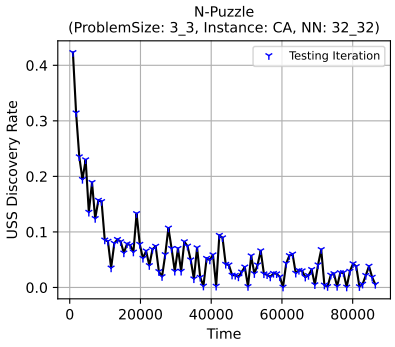

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

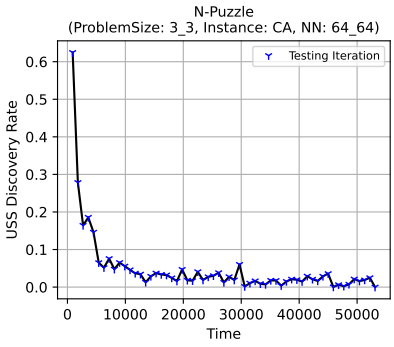

Bouncing Ball
(NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

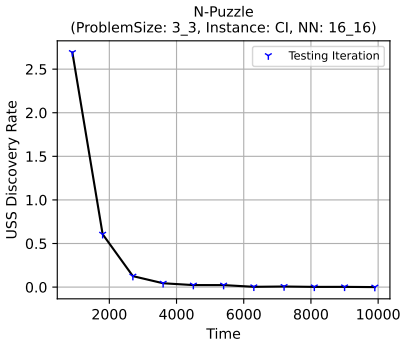

Inverted Pendulum
(NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

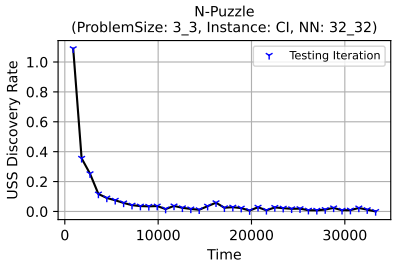

Inverted Pendulum
(NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

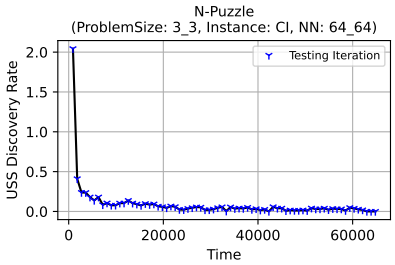

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

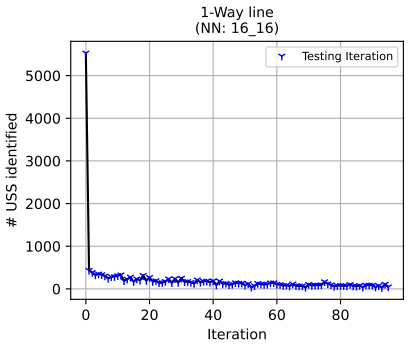

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

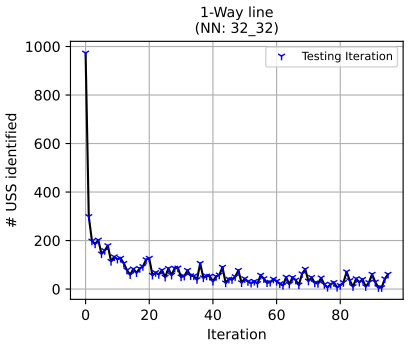

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
1.0
0.8
0.6
0.4
0.2
0.0
0
10000
20000
30000
Time

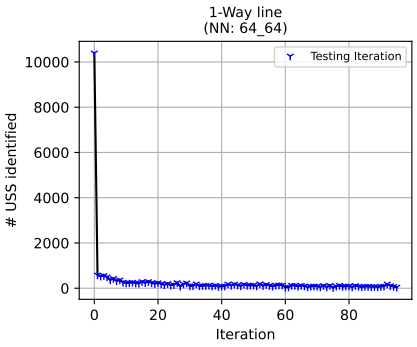

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
2.0
1.5
1.0
0.5
0.0
0
20000
40000
60000
Time

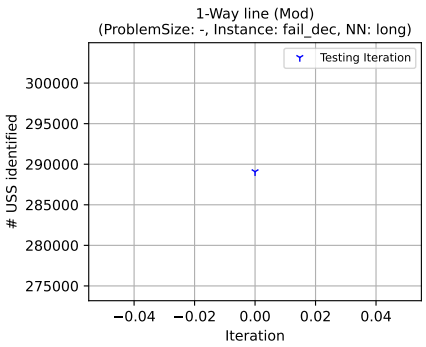
- Results for chapter 4 -

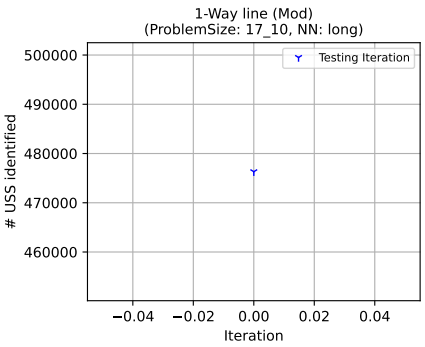
Start Condition Strengthening (Biased Sampling)

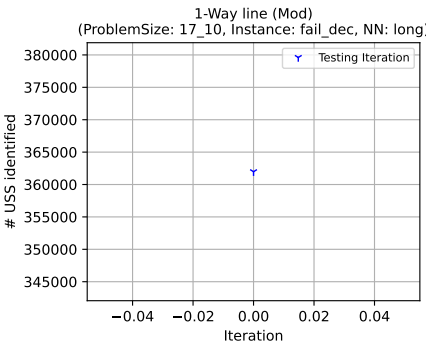

1-Way line
(NN: 16_16)
Testing Iteration
USS identified
Iteration

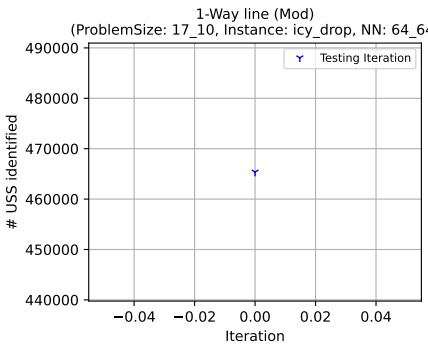

1-Way line
(NN: 32_32)
Testing Iteration
USS identified
Iteration

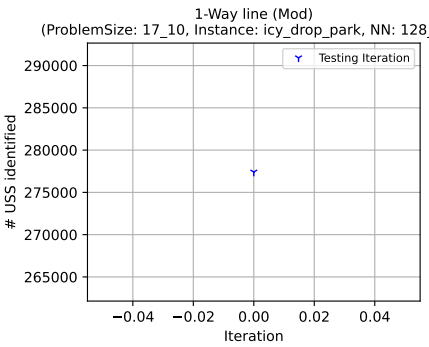

1-Way line
(NN: 64_64)
Testing Iteration
USS identified
Iteration

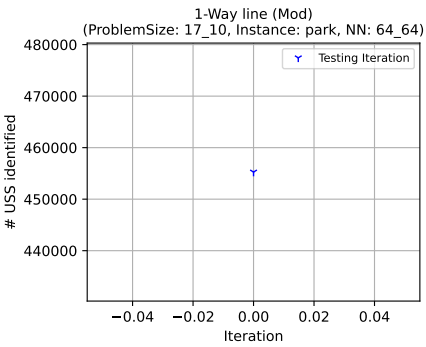

1-Way line (Mod)
(ProblemSize: -, Instance: fail_dec, NN: long)
Testing Iteration
USS identified
Iteration

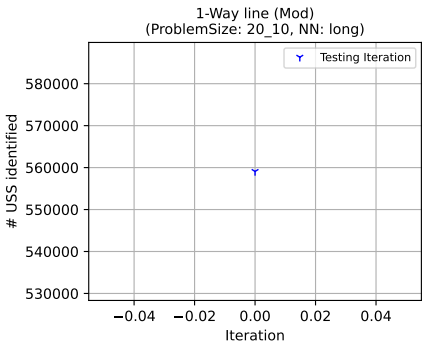

1-Way line (Mod)
(ProblemSize: 17_10, NN: long)
Testing Iteration
USS identified
Iteration

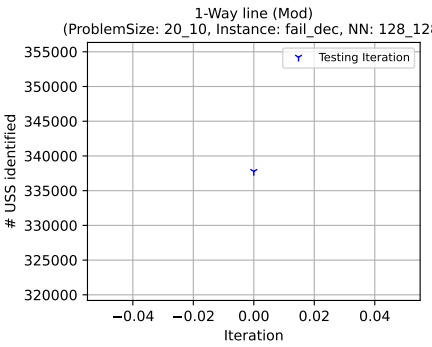

1-Way line (Mod)
(ProblemSize: 17_10, Instance: fail_dec, NN: long)
Testing Iteration
USS identified
Iteration

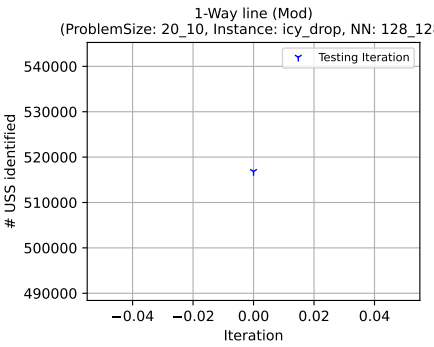

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Testing Iteration
USS identified
Iteration

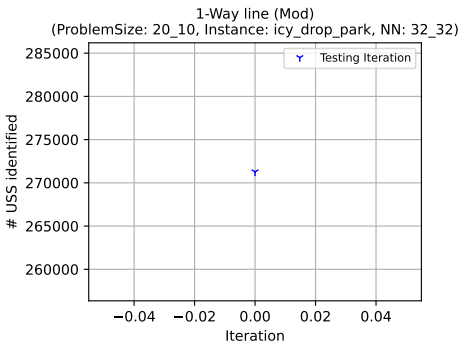

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS identified
Iteration

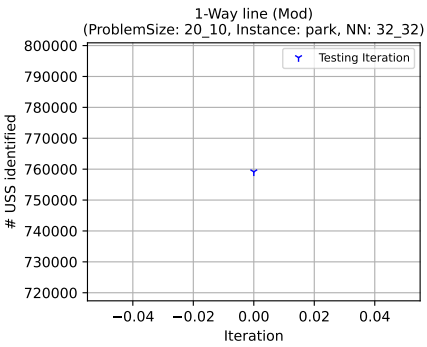

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Testing Iteration
USS identified
Iteration

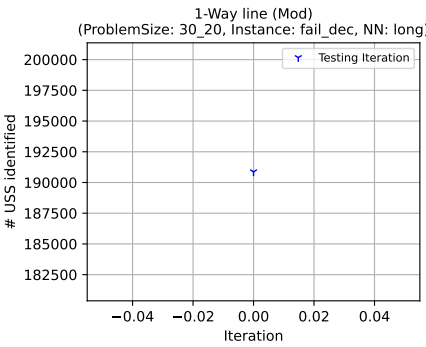

1-Way line (Mod)
(ProblemSize: 20_10, NN: long)
Testing Iteration
USS identified
Iteration

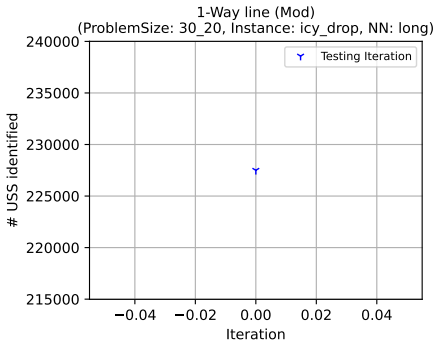

1-Way line (Mod)
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS identified
Iteration

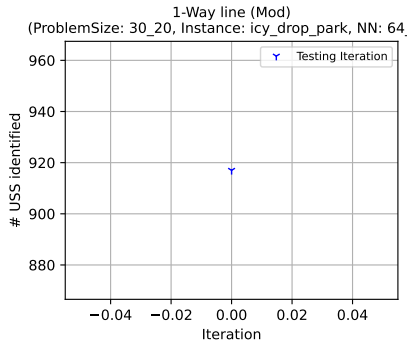

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
USS identified
Iteration

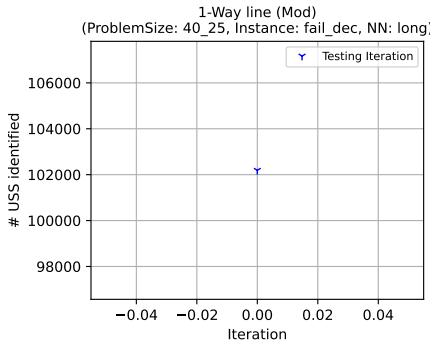

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 32_32)
Testing Iteration
USS identified
Iteration

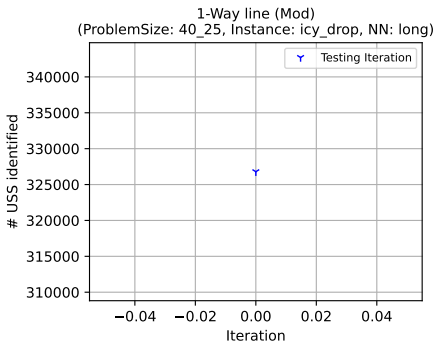

1-Way line (Mod)
(ProblemSize: 20_10, Instance: park, NN: 32_32)
Testing Iteration
USS identified
Iteration

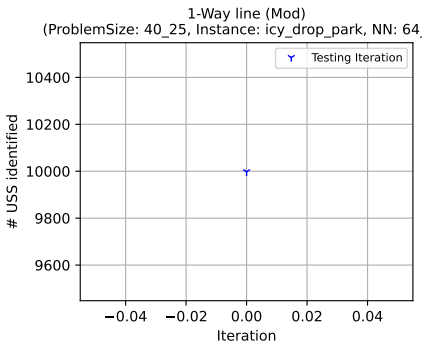

1-Way line (Mod)
(ProblemSize: 30_20, Instance: fail_dec, NN: long)
Testing Iteration
USS identified
Iteration

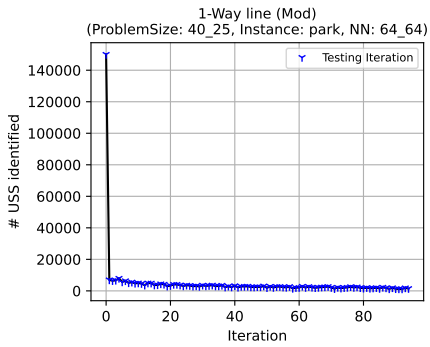

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop, NN: long)
Testing Iteration
USS identified
Iteration

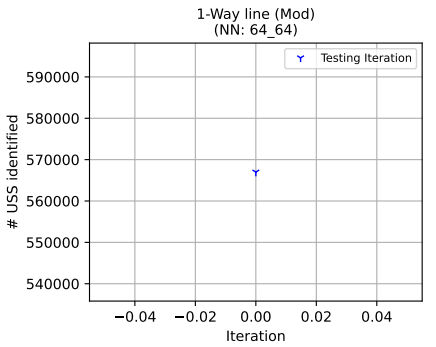

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS identified
Iteration

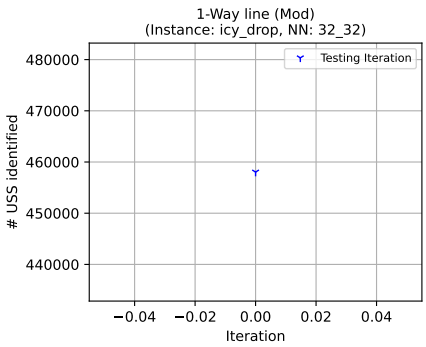

1-Way line (Mod)
(ProblemSize: 40_25, Instance: fail_dec, NN: long)
Testing Iteration
USS identified
Iteration

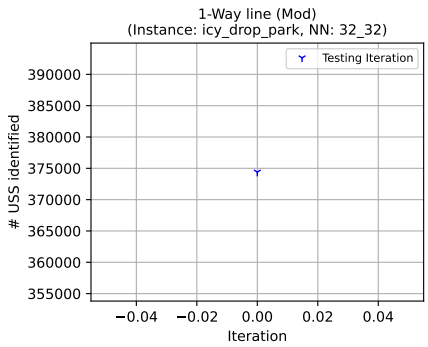

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop, NN: long)
Testing Iteration
USS identified
Iteration

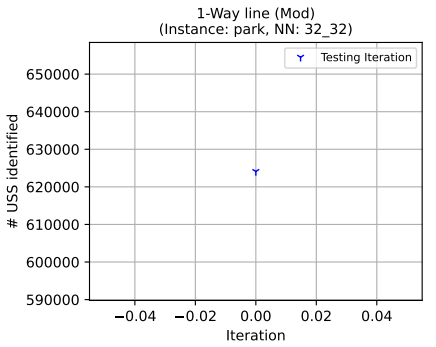

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS identified
Iteration

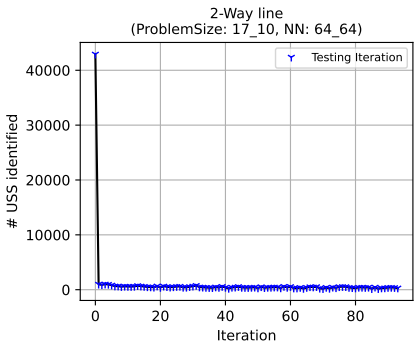

1-Way line (Mod)
(ProblemSize: 40_25, Instance: park, NN: 64_64)
Testing Iteration
USS identified
Iteration

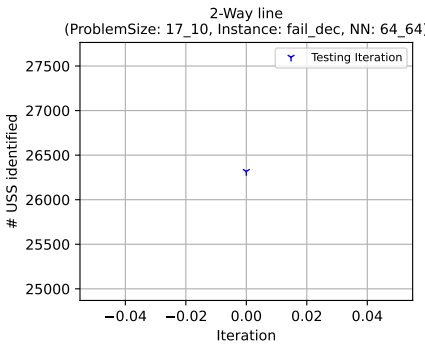

1-Way line (Mod)
(NN: 64_64)
Testing Iteration
USS identified
Iteration

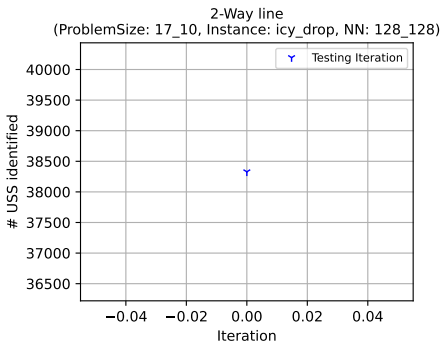

1-Way line (Mod)
(Instance: icy_drop, NN: 32_32)
Testing Iteration
USS identified
Iteration

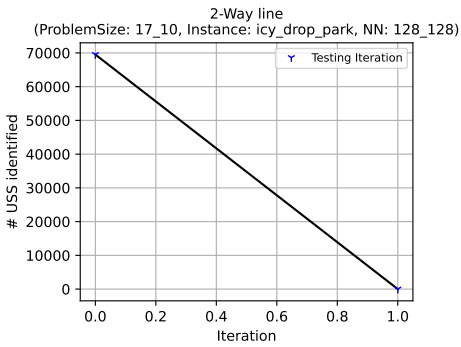

1-Way line (Mod)
(Instance: icy_drop_park, NN: 32_32)
Testing Iteration
USS identified
Iteration

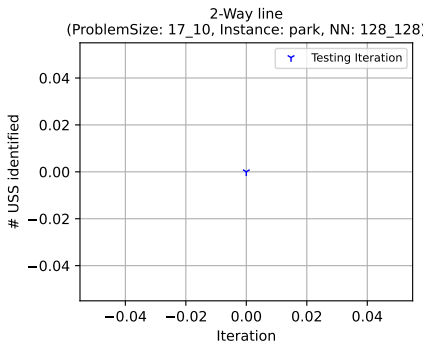

1-Way line (Mod)
(Instance: park, NN: 32_32)
Testing Iteration
USS identified
Iteration

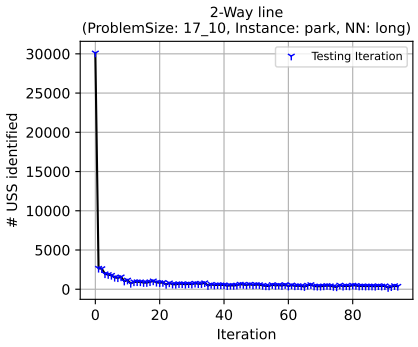

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Testing Iteration
USS identified
Iteration

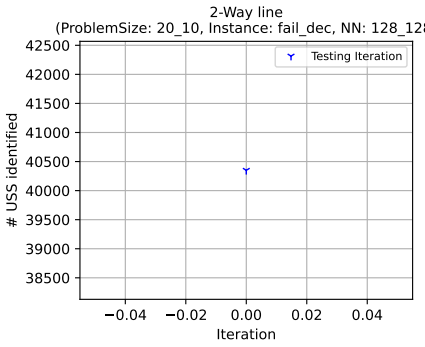

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS identified
Iteration

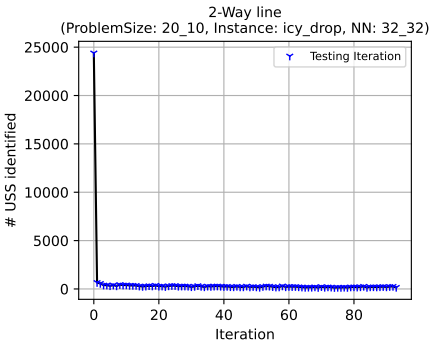

2-Way line
(ProblemSize: 17_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
USS identified
Iteration

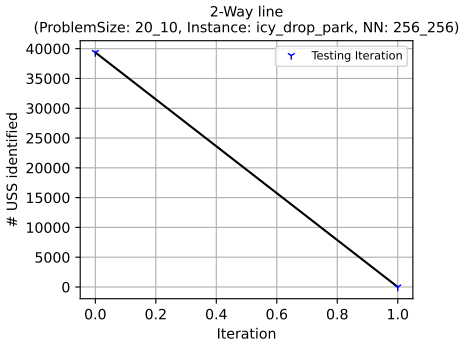

2-Way line
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS identified
Iteration

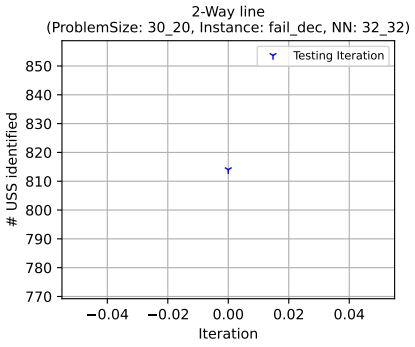

2-Way line
(ProblemSize: 17_10, Instance: park, NN: 128_128)
Testing Iteration
USS identified
Iteration

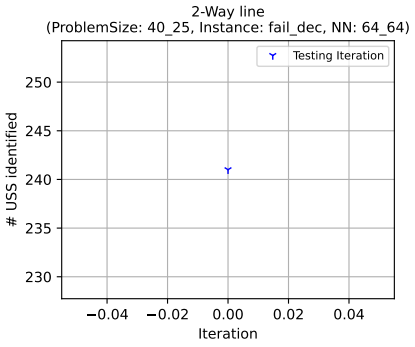

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
USS identified
Iteration

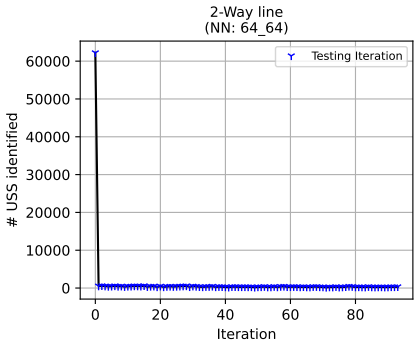

2-Way line
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS identified
Iteration

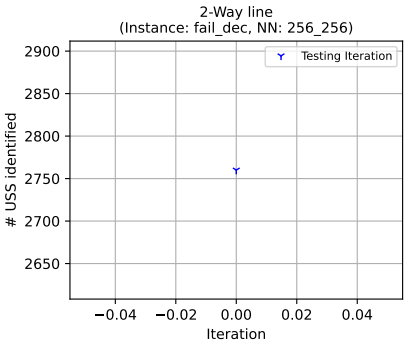

2-Way line
(ProblemSize: 20_10, Instance: icy_drop, NN: 32_32)
Testing Iteration
USS identified
Iteration

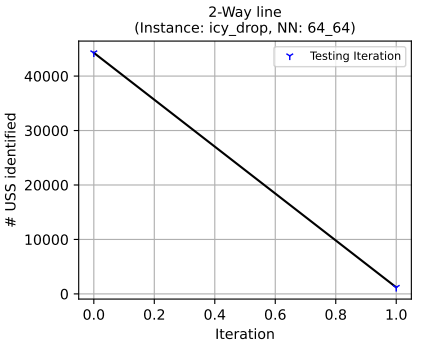

2-Way line
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 256_256)
Testing Iteration
USS identified
Iteration

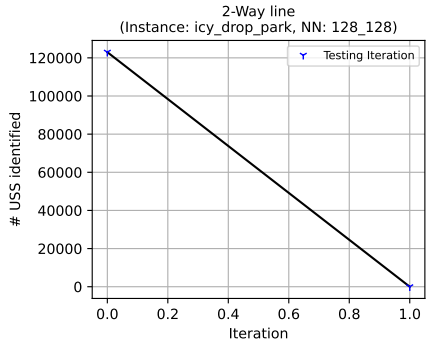

2-Way line
(ProblemSize: 30_20, Instance: fail_dec, NN: 32_32)
Testing Iteration
USS identified
Iteration

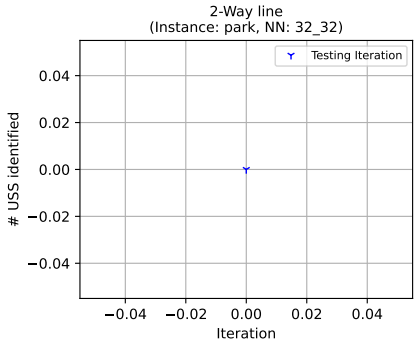

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS identified
Iteration

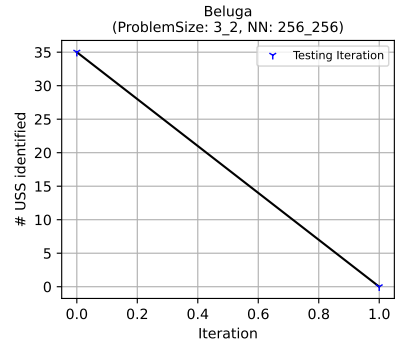

2-Way line
(NN: 64_64)
Testing Iteration
USS identified
Iteration

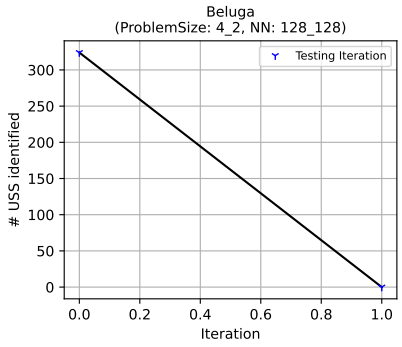

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
USS identified
Iteration

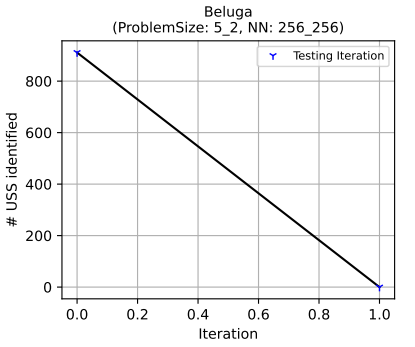

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
USS identified
Iteration

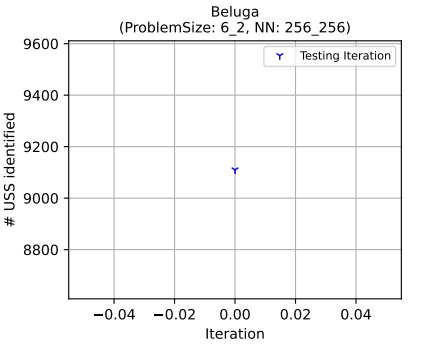

2-Way line
(Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS identified
Iteration

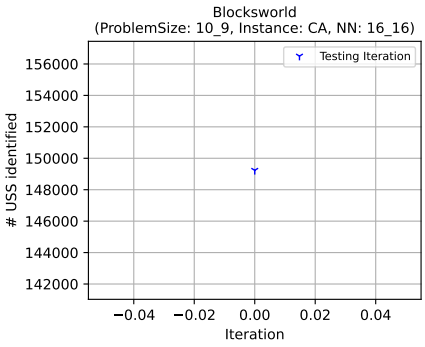

2-Way line
(Instance: park, NN: 32_32)
Testing Iteration
USS identified
Iteration

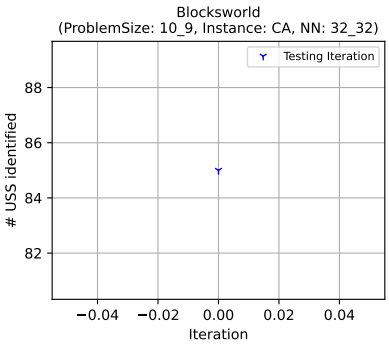

Beluga
(ProblemSize: 3_2, NN: 256_256)
Testing Iteration
USS identified
Iteration

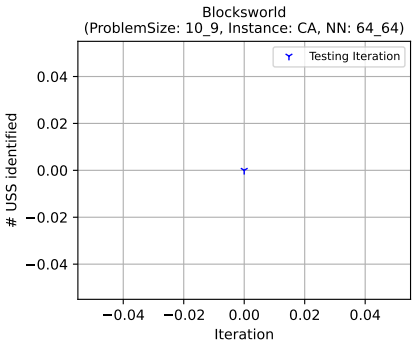

Beluga
(ProblemSize: 4_2, NN: 128_128)
Testing Iteration
USS identified
Iteration

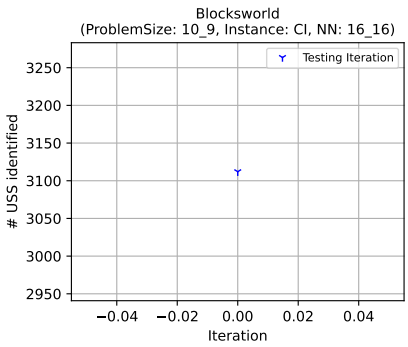

Beluga
(ProblemSize: 5_2, NN: 256_256)
Testing Iteration
USS identified
Iteration

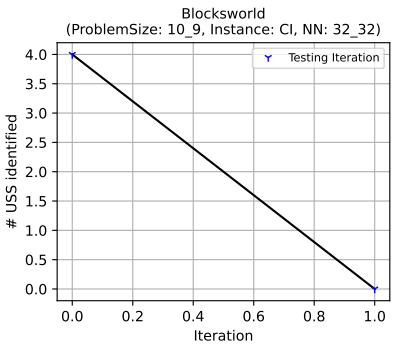

Beluga
(ProblemSize: 6_2, NN: 256_256)
Testing Iteration
USS identified
Iteration

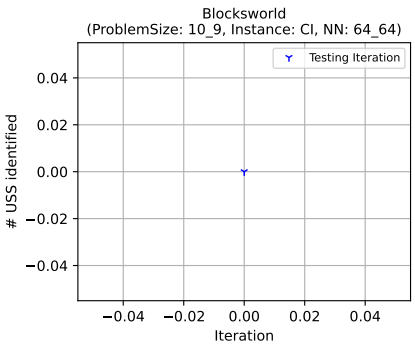

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

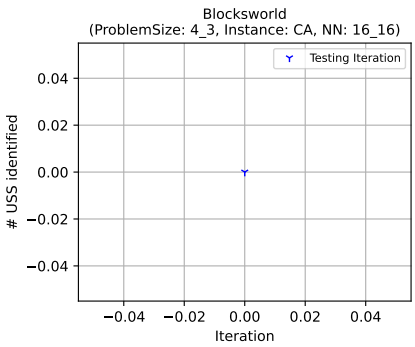

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

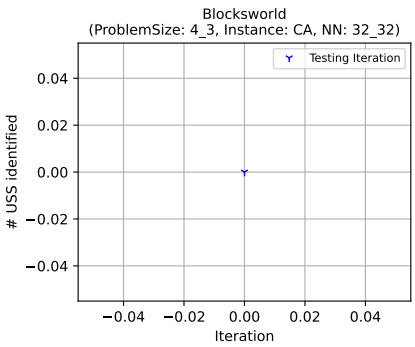

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

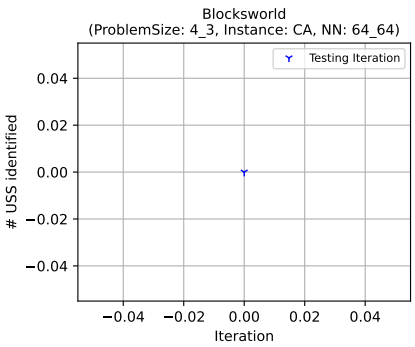

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

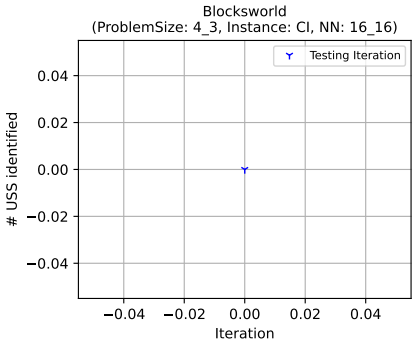

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

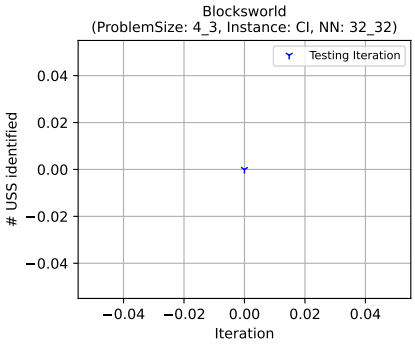

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

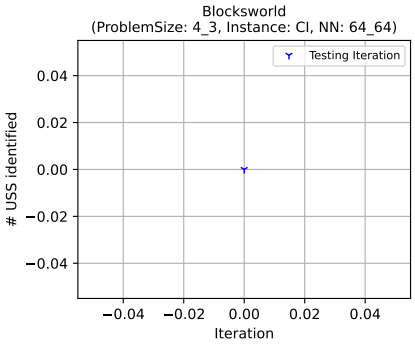

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

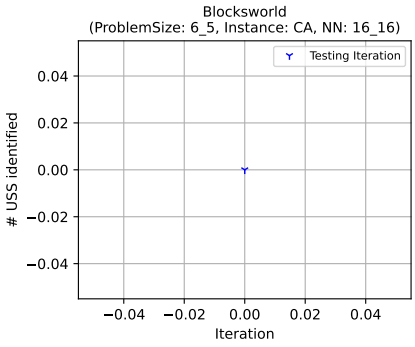

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

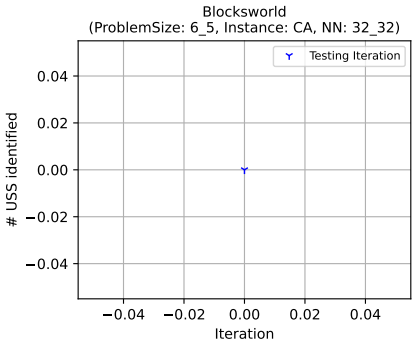

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

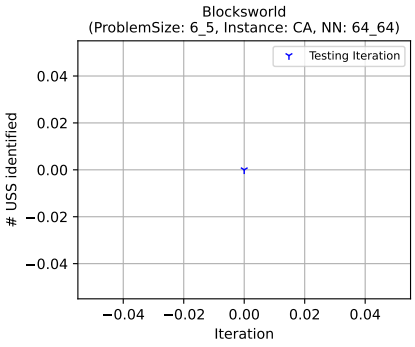

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

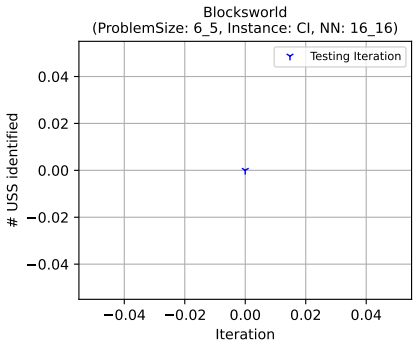

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

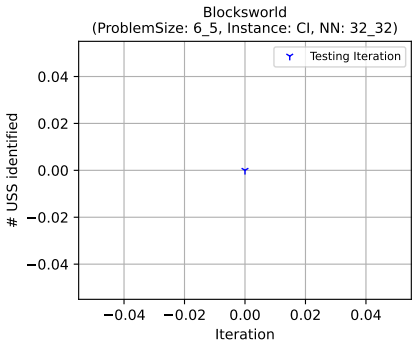

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

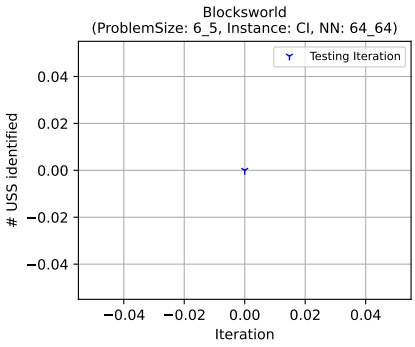

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

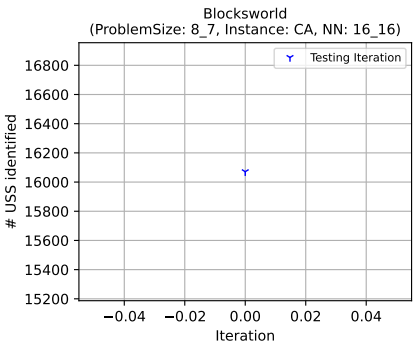

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

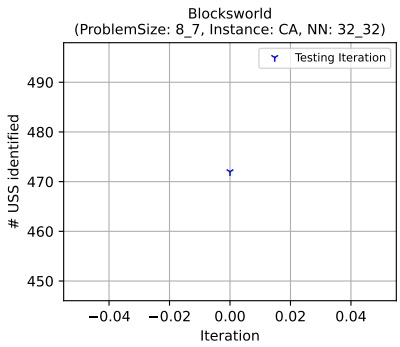

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

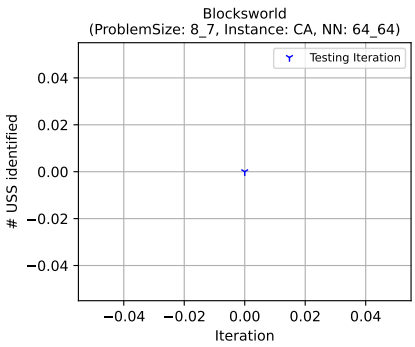

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

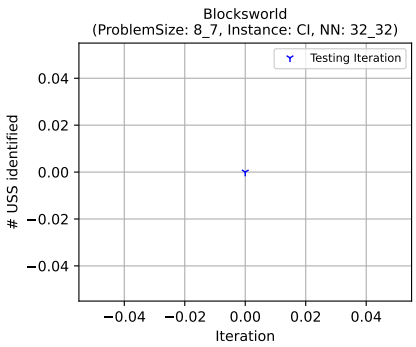

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

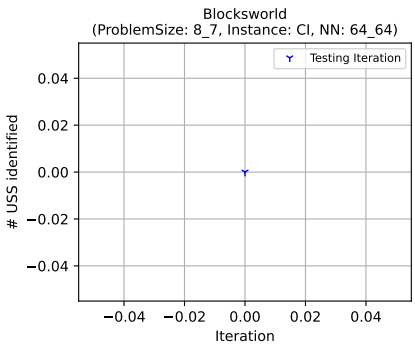

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

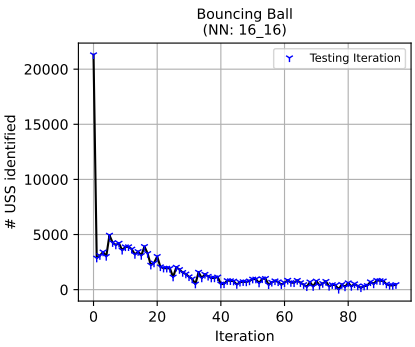

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

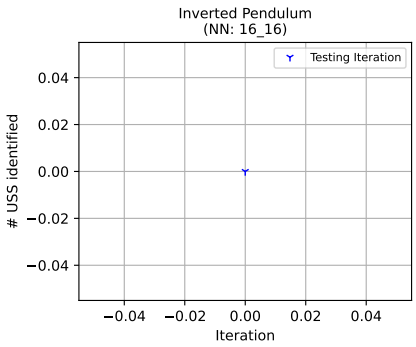

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

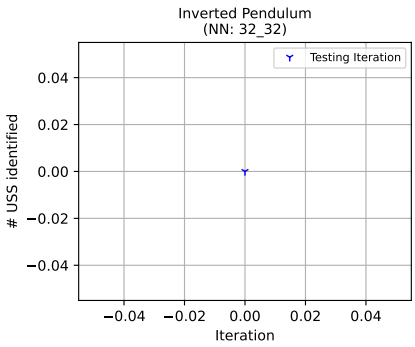

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

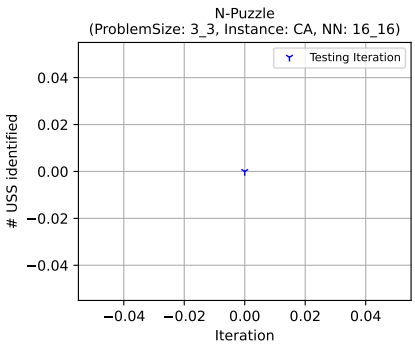

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
Iteration

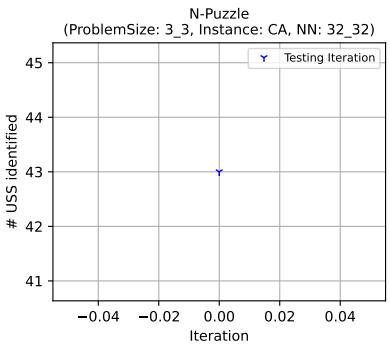

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
Iteration

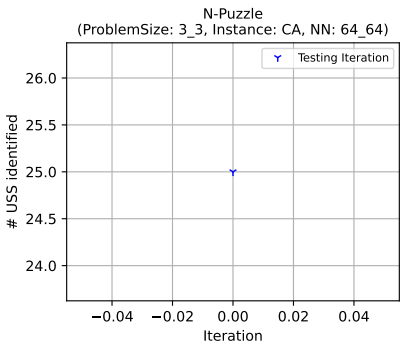

Bouncing Ball
(NN: 16_16)
Testing Iteration
USS identified
Iteration

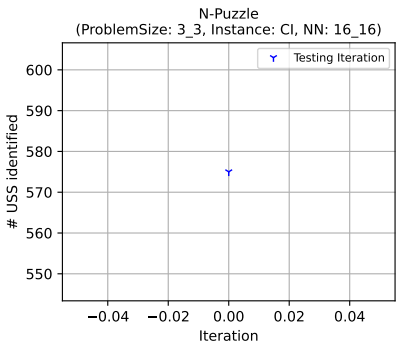

Inverted Pendulum
(NN: 16_16)
Testing Iteration
USS identified
Iteration

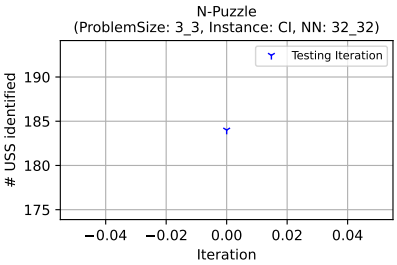

Inverted Pendulum
(NN: 32_32)
Testing Iteration
USS identified
Iteration

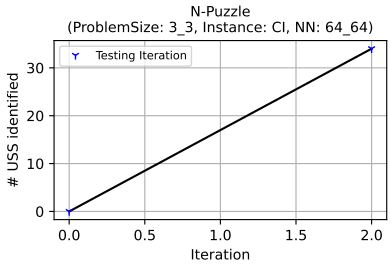

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Testing Iteration
USS identified
Iteration

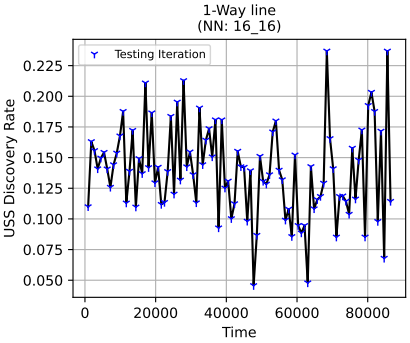

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Testing Iteration
USS identified
Iteration

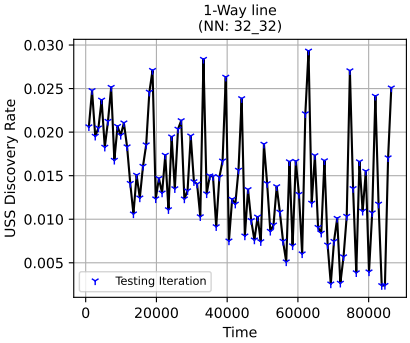

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Testing Iteration
USS identified
Iteration

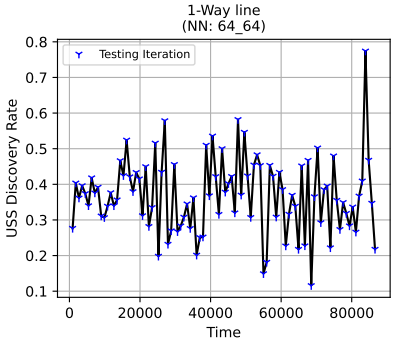

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Testing Iteration
USS identified
Iteration

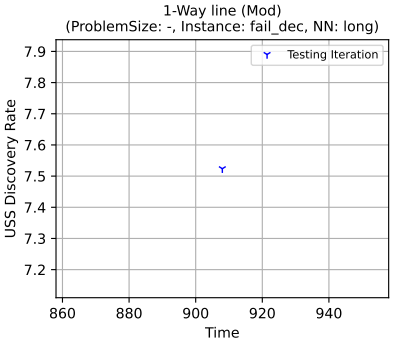

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Testing Iteration
USS identified
190
185
180
175
−0.04
−0.02
0.00
0.02
0.04
Iteration

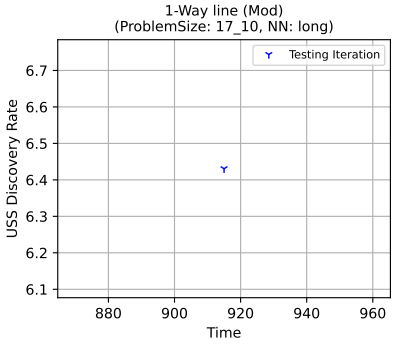

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Testing Iteration
USS identified
30
20
10
0
0.0
0.5
1.0
1.5
2.0
Iteration

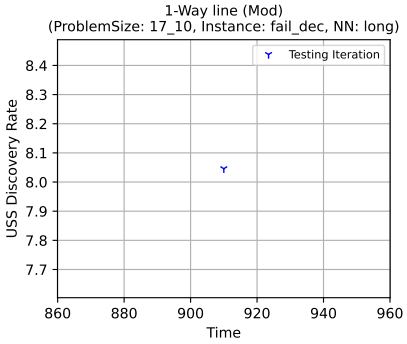

1-Way line
(NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

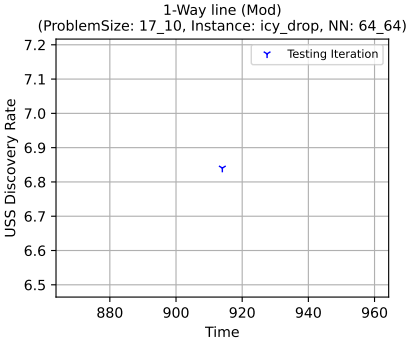

1-Way line
(NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

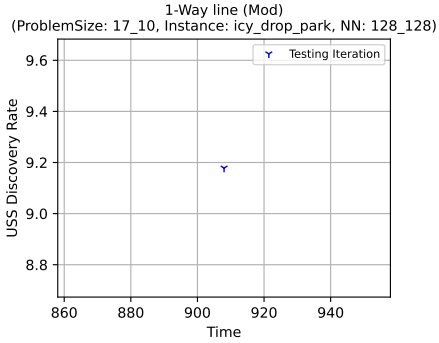

1-Way line
(NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

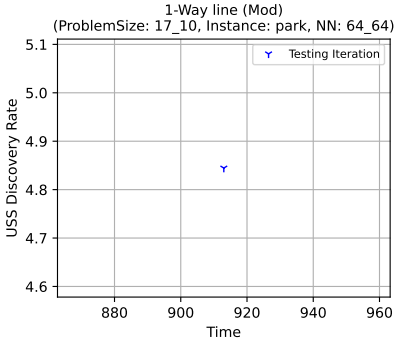

1-Way line (Mod)
(ProblemSize: -, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

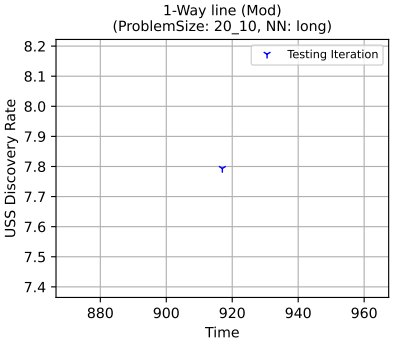

1-Way line (Mod)
(ProblemSize: 17_10, NN: long)
Testing Iteration
USS Discovery Rate
Time

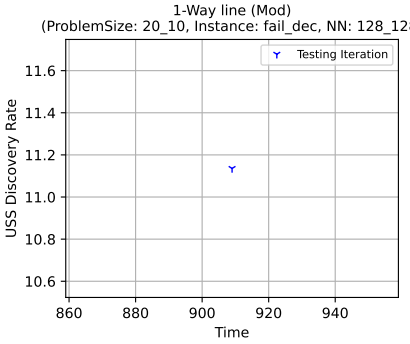

1-Way line (Mod)
(ProblemSize: 17_10, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

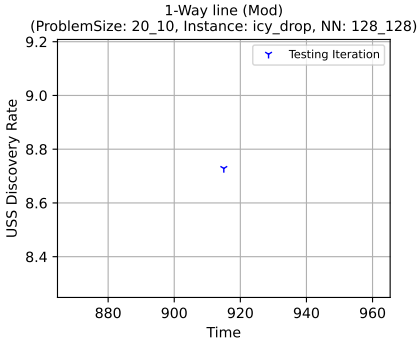

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

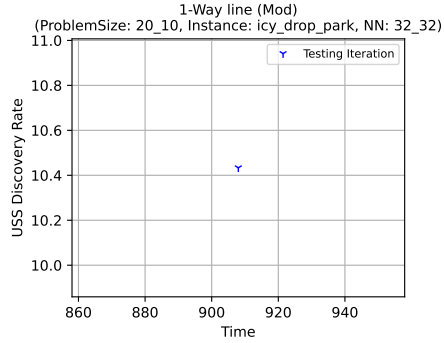

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

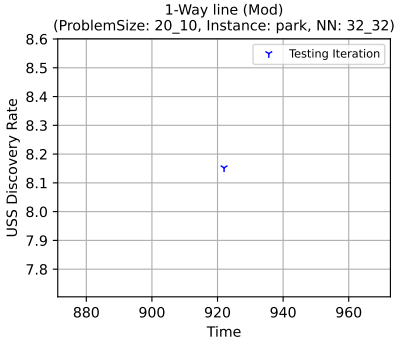

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

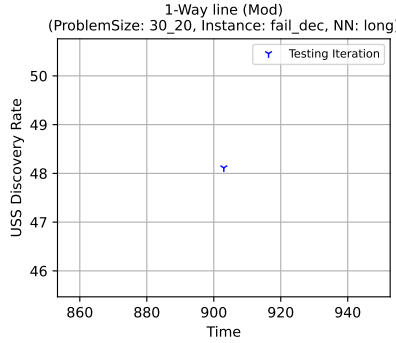

1-Way line (Mod)
(ProblemSize: 20_10, NN: long)
Testing Iteration
USS Discovery Rate
Time

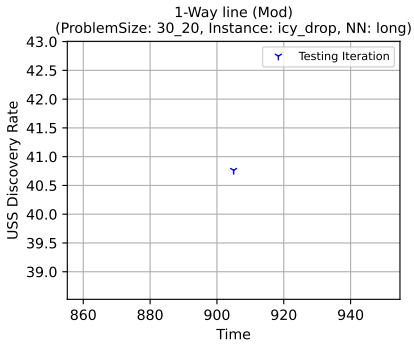

1-Way line (Mod)
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

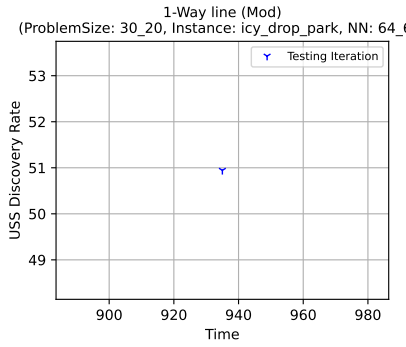

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

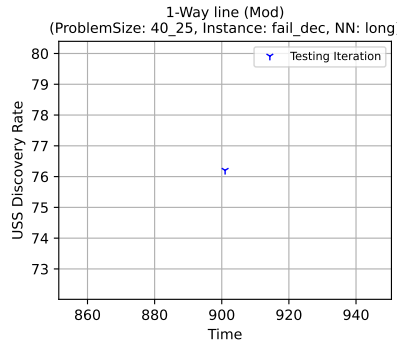

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

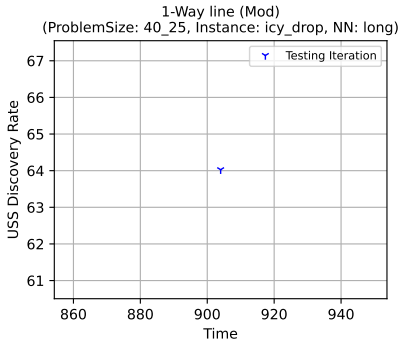

1-Way line (Mod)
(ProblemSize: 20_10, Instance: park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

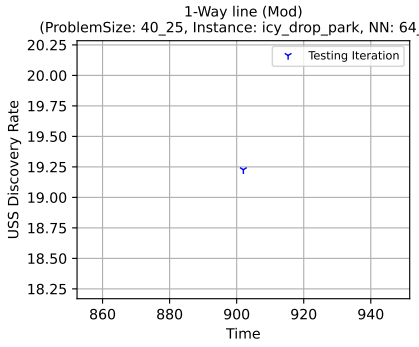

1-Way line (Mod)
(ProblemSize: 30_20, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

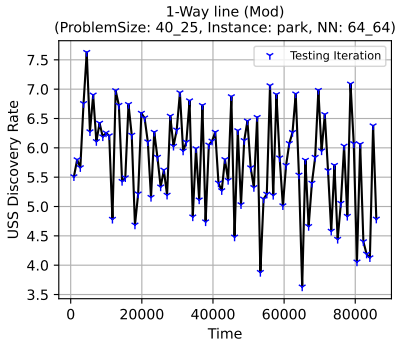

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop, NN: long)
Testing Iteration
USS Discovery Rate
Time

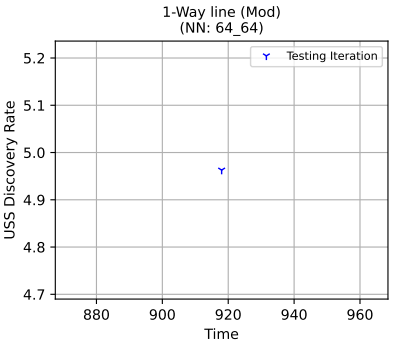

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

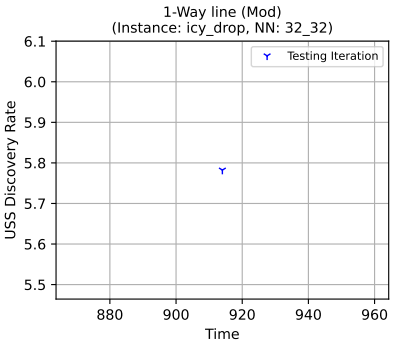

1-Way line (Mod)
(ProblemSize: 40_25, Instance: fail_dec, NN: long)
Testing Iteration
USS Discovery Rate
Time

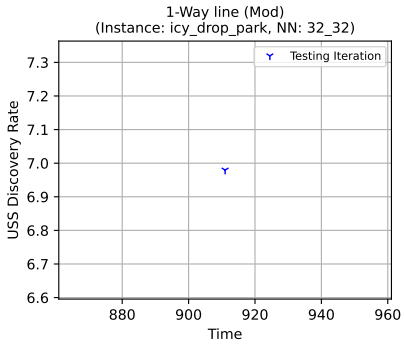

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop, NN: long)
Testing Iteration
USS Discovery Rate
Time

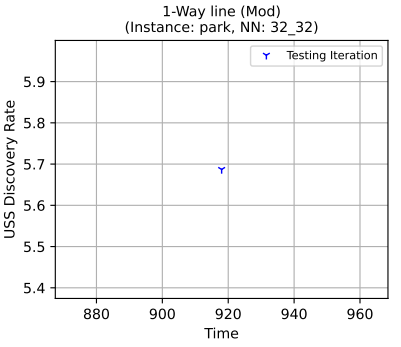

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

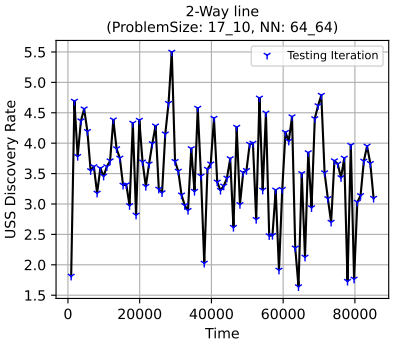

1-Way line (Mod)
(ProblemSize: 40_25, Instance: park, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

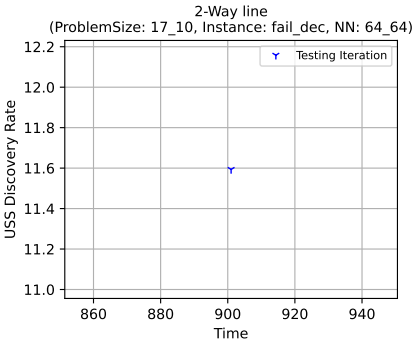

1-Way line (Mod)
(NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

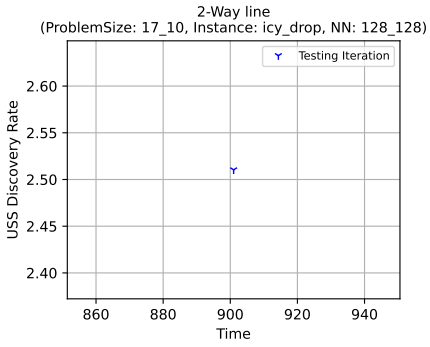

1-Way line (Mod)
(Instance: icy_drop, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

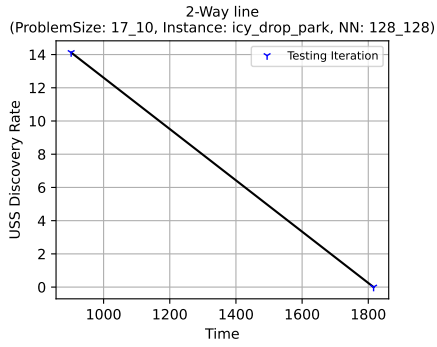

1-Way line (Mod)
(Instance: icy_drop_park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

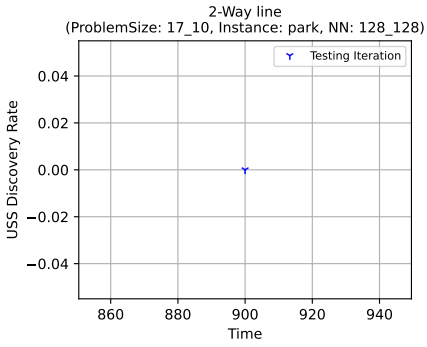

1-Way line (Mod)
(Instance: park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

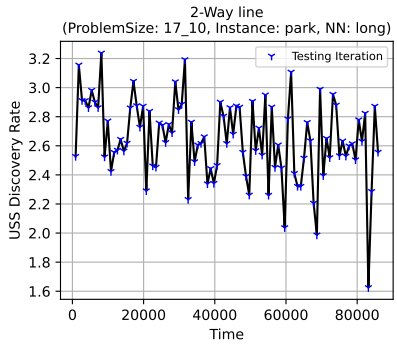

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

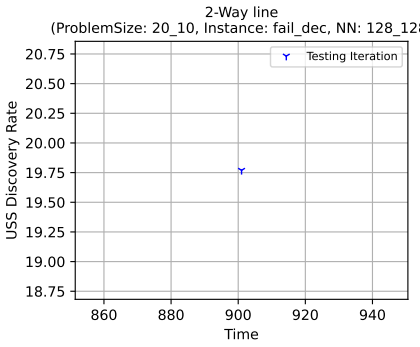

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

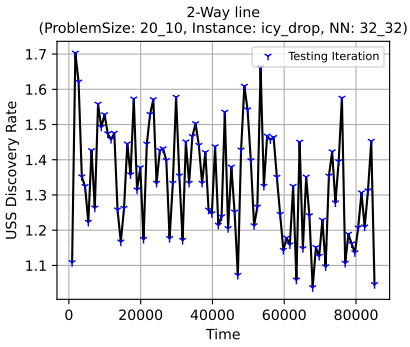

2-Way line
(ProblemSize: 17_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

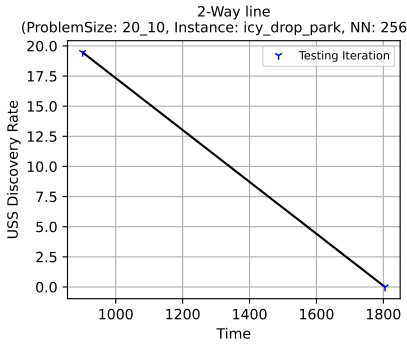

2-Way line
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

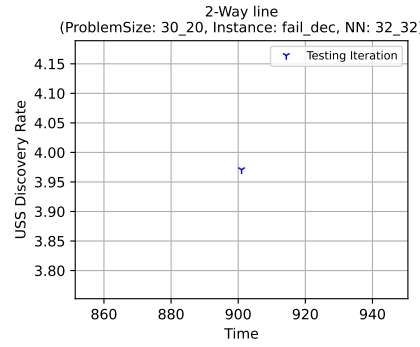

2-Way line
(ProblemSize: 17_10, Instance: park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

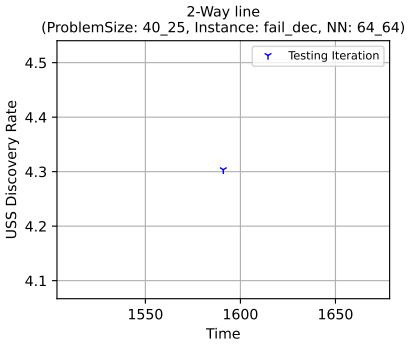

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
USS Discovery Rate
Time

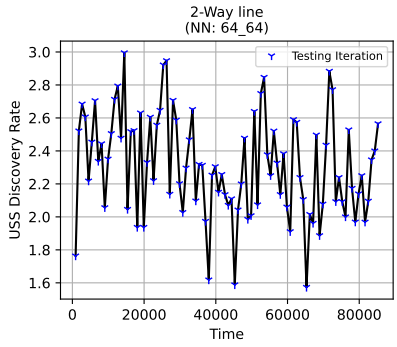

2-Way line
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

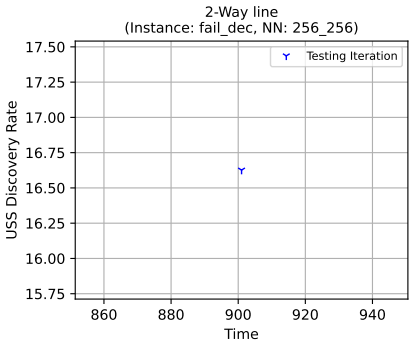

2-Way line
(ProblemSize: 20_10, Instance: icy_drop, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

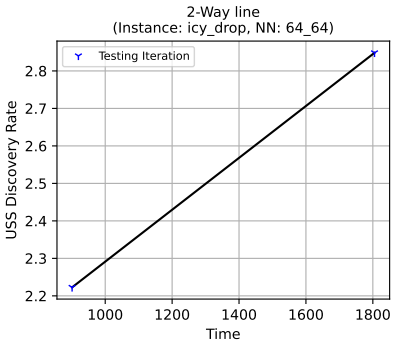

2-Way line
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

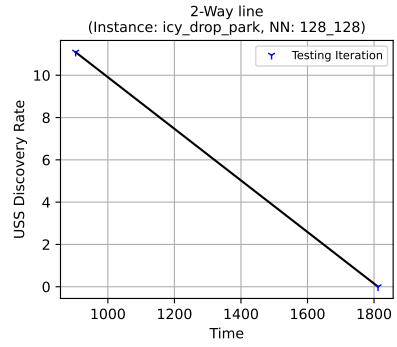

2-Way line
(ProblemSize: 30_20, Instance: fail_dec, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

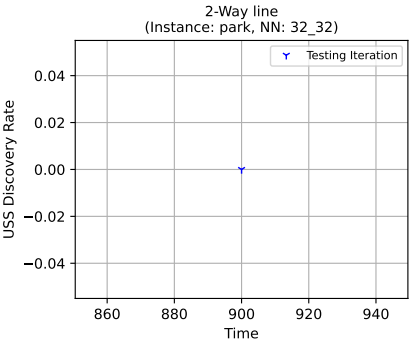

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

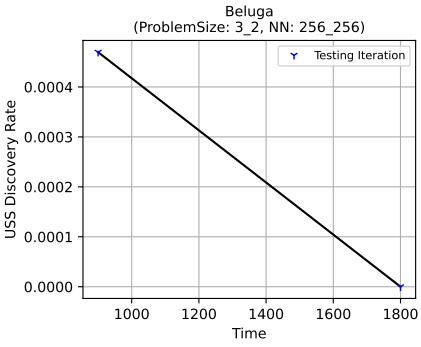

2-Way line
(NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

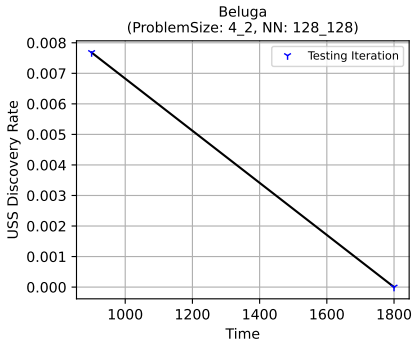

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

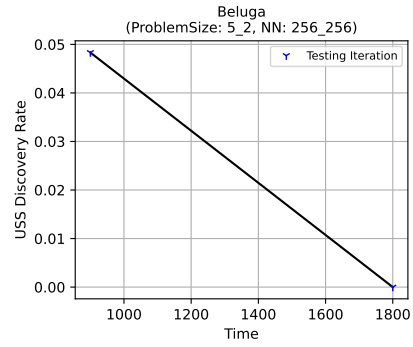

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

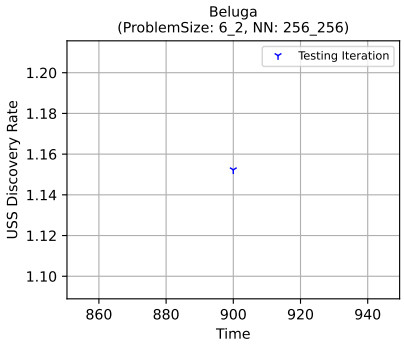

2-Way line
(Instance: icy_drop_park, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

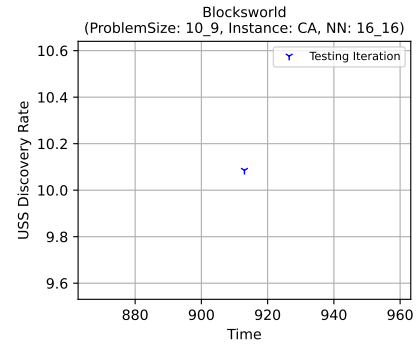

2-Way line
(Instance: park, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

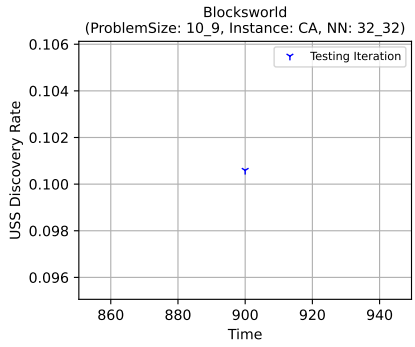

Beluga
(ProblemSize: 3_2, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

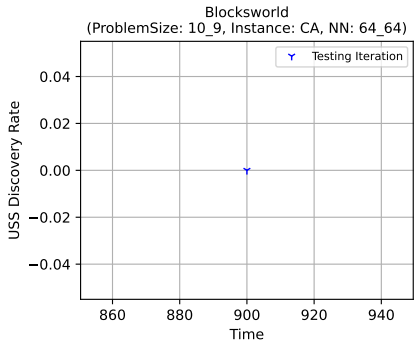

Beluga
(ProblemSize: 4_2, NN: 128_128)
Testing Iteration
USS Discovery Rate
Time

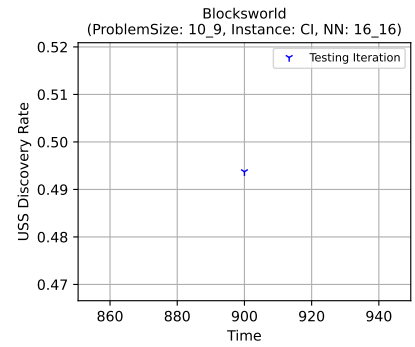

Beluga
(ProblemSize: 5_2, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

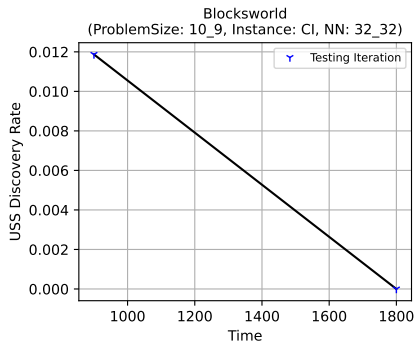

Beluga
(ProblemSize: 6_2, NN: 256_256)
Testing Iteration
USS Discovery Rate
Time

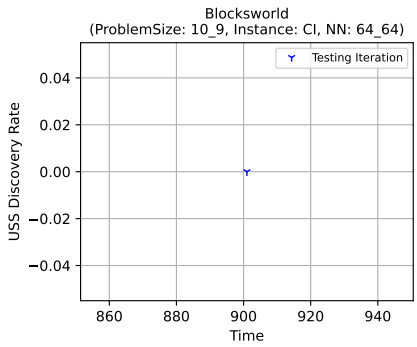

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

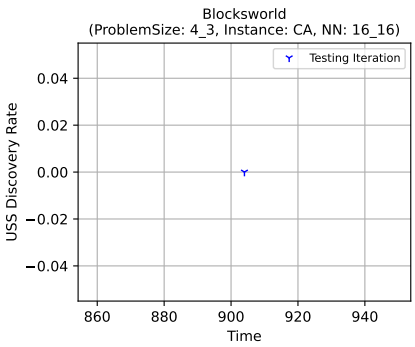

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

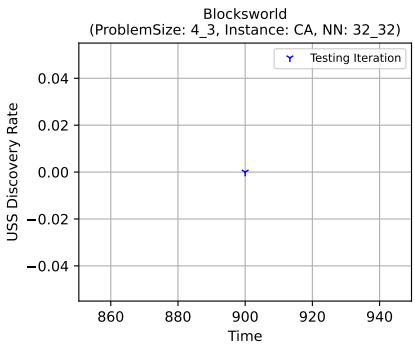

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

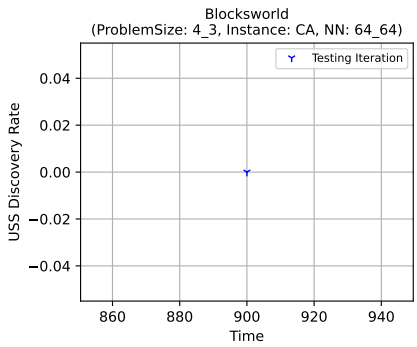

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

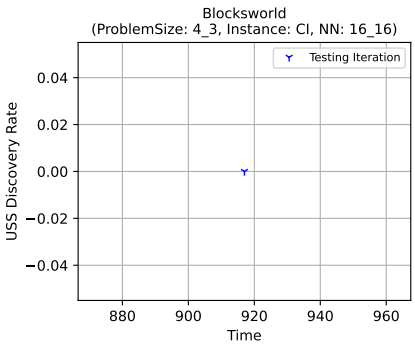

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

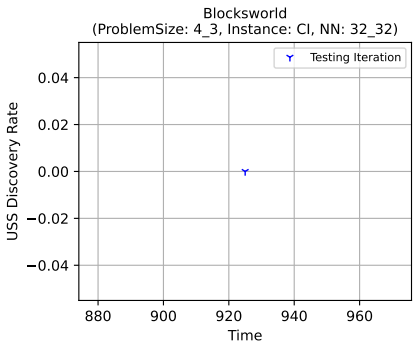

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

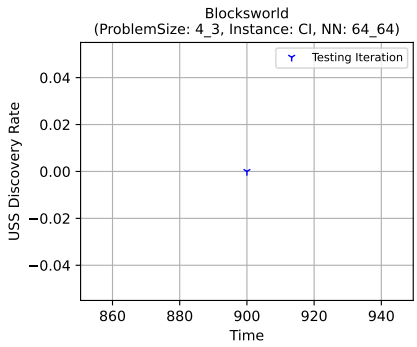

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

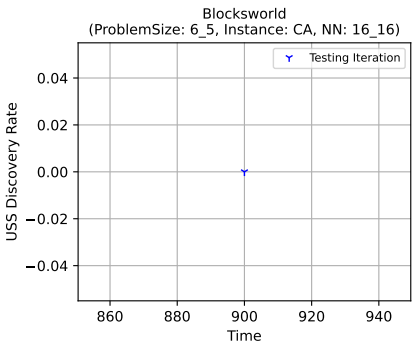

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

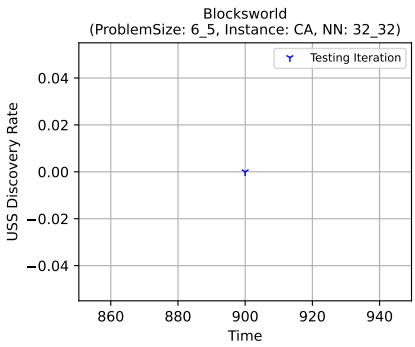

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

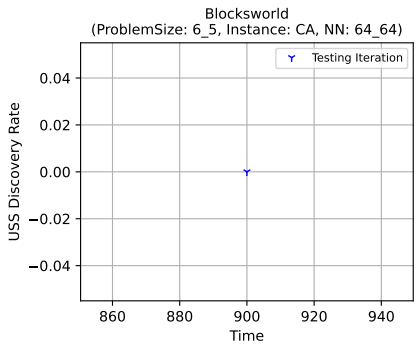

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

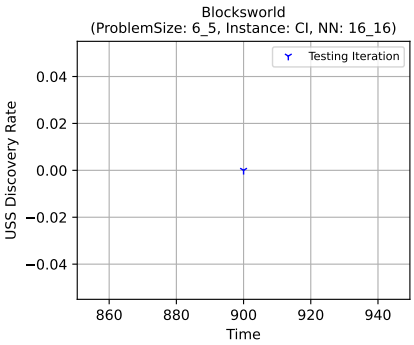

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

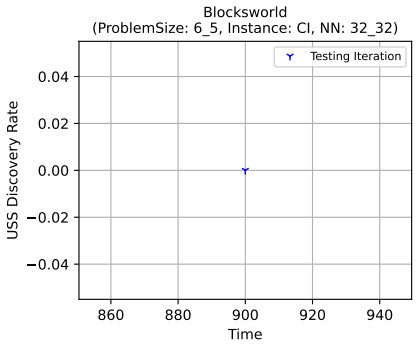

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

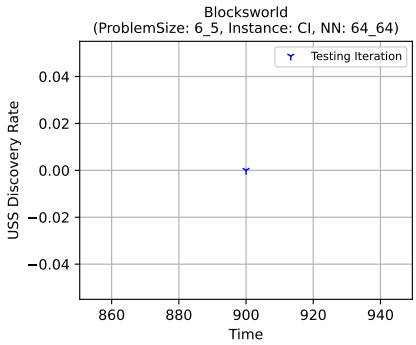

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

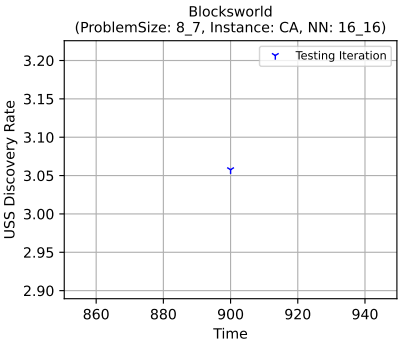

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

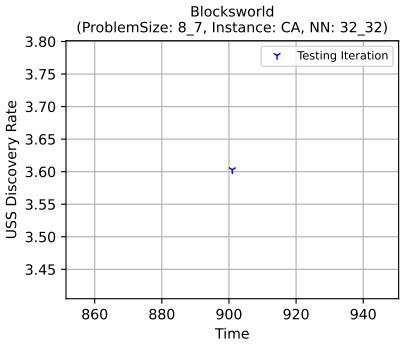

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

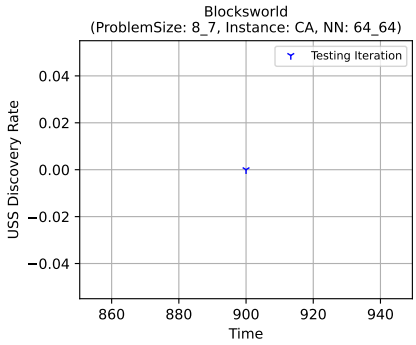

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

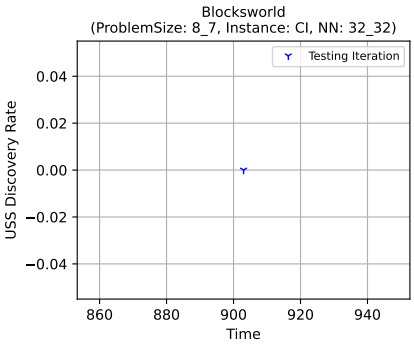

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

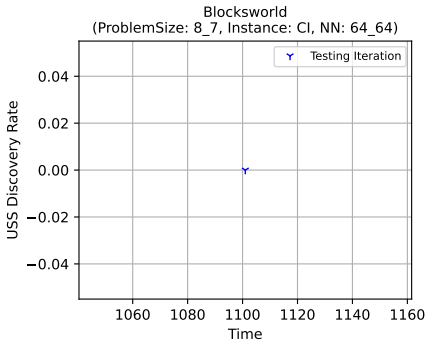

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

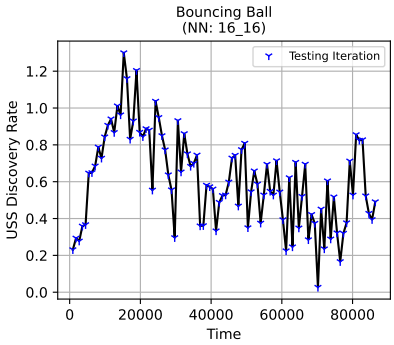

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

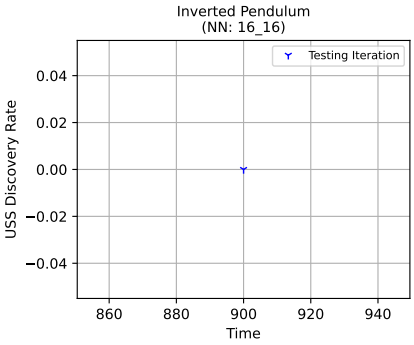

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

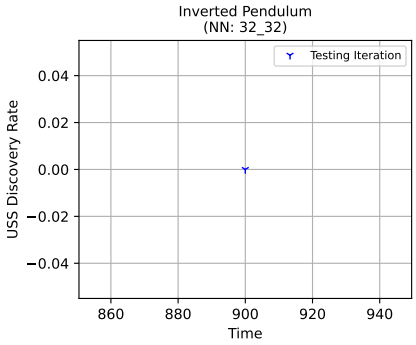

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

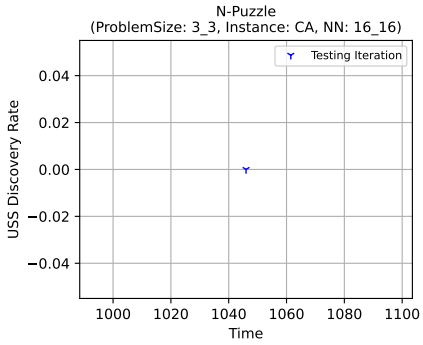

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

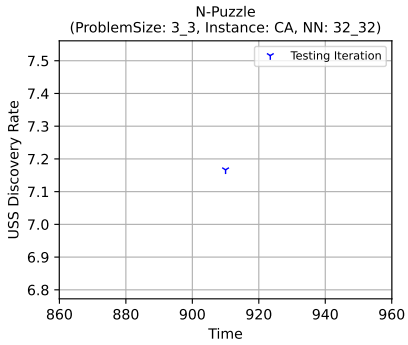

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

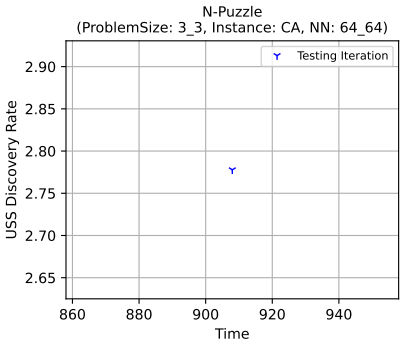

Bouncing Ball
(NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

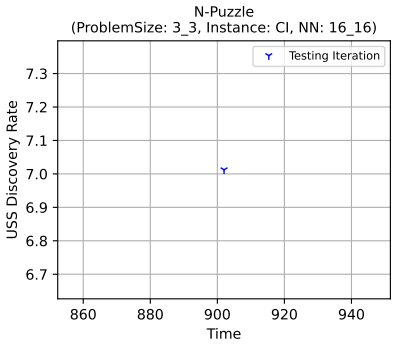

Inverted Pendulum
(NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

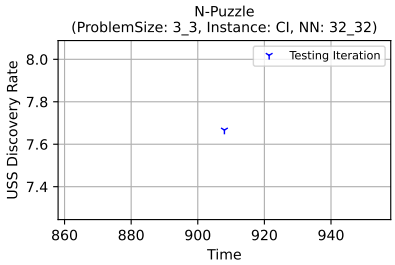

Inverted Pendulum
(NN: 32_32)
Testing Iteration
USS Discovery Rate
Time

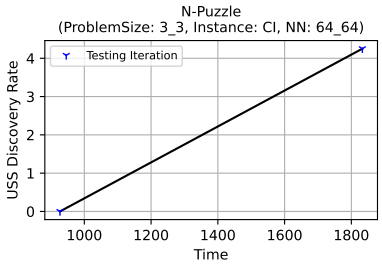

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Testing Iteration
USS Discovery Rate
Time


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Testing Iteration
USS Discovery Rate
Time

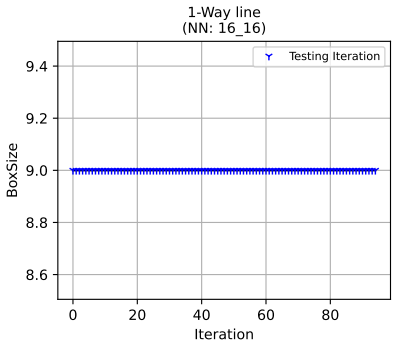

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Testing Iteration
USS Discovery Rate
Time

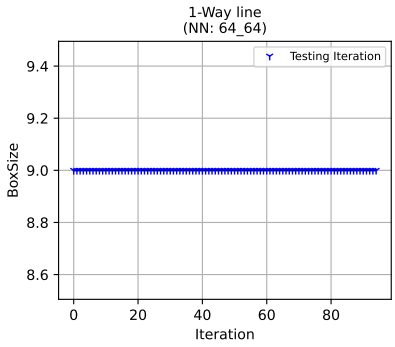

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Testing Iteration
USS Discovery Rate
8.0
7.8
7.6
7.4
860
880
900
920
940
Time

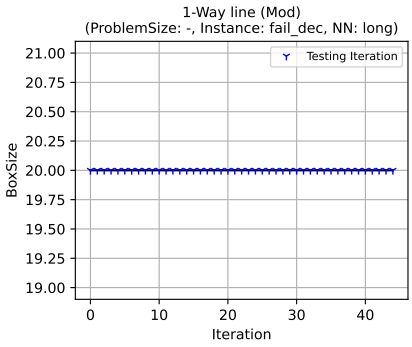

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Testing Iteration
USS Discovery Rate
4
3
2
1
0
1000
1200
1400
1600
1800
Time

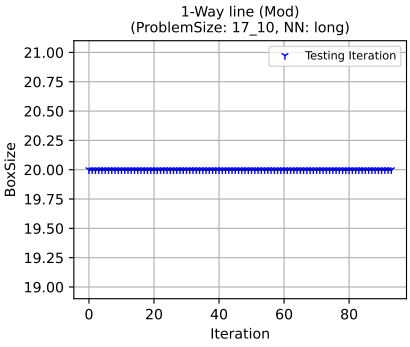
- Results for chapter 5 -

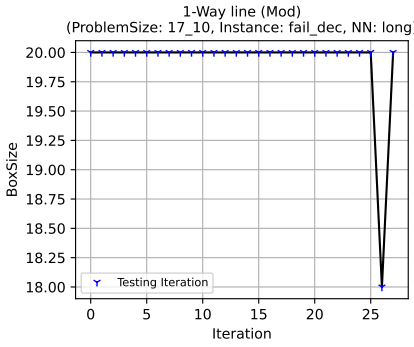
Invariant Strengthening

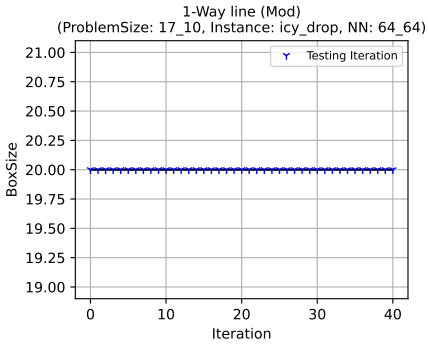

1-Way line
(NN: 16_16)
Testing Iteration
BoxSize
Iteration

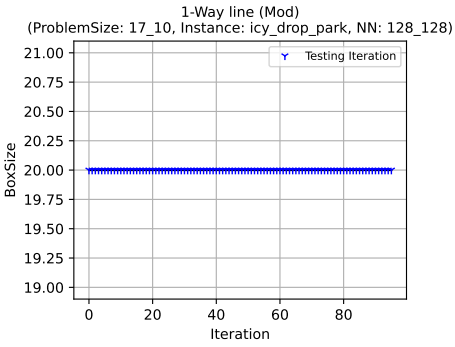

1-Way line
(NN: 64_64)
Testing Iteration
BoxSize
Iteration

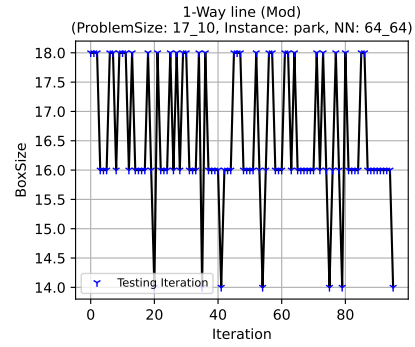

1-Way line (Mod)
(ProblemSize: -, Instance: fail_dec, NN: long)
Testing Iteration
BoxSize
Iteration

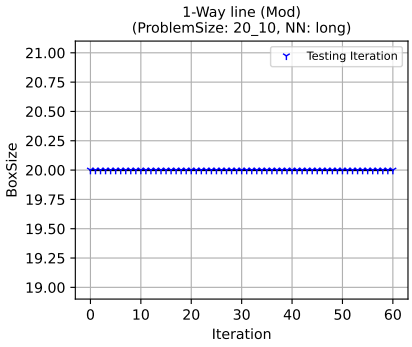

1-Way line (Mod)
(ProblemSize: 17_10, NN: long)
Testing Iteration
BoxSize
Iteration

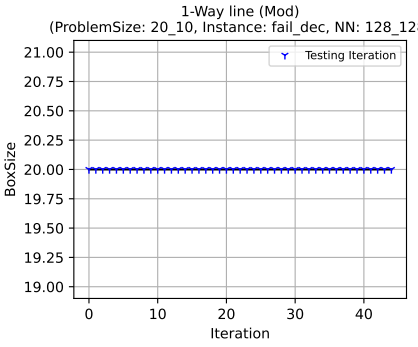

1-Way line (Mod)
(ProblemSize: 17_10, Instance: fail_dec, NN: long)
Testing Iteration
BoxSize
Iteration

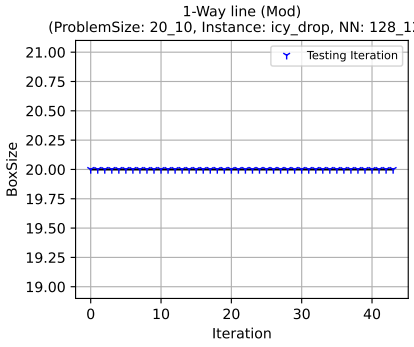

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Testing Iteration
BoxSize
Iteration

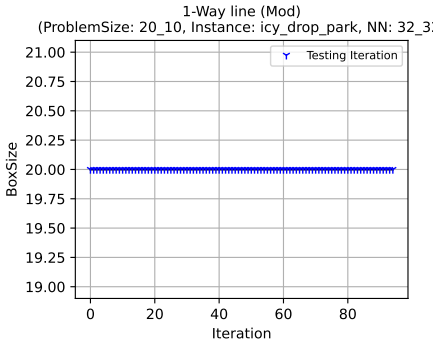

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
BoxSize
Iteration

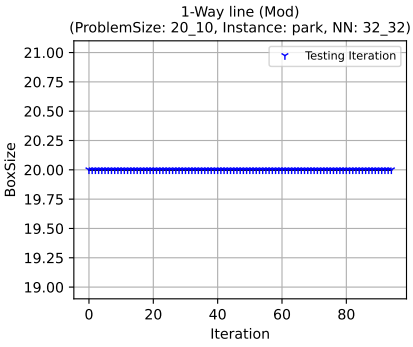

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Testing Iteration
BoxSize
Iteration

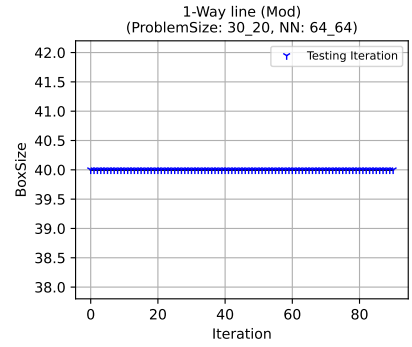

1-Way line (Mod)
(ProblemSize: 20_10, NN: long)
Testing Iteration
BoxSize
Iteration

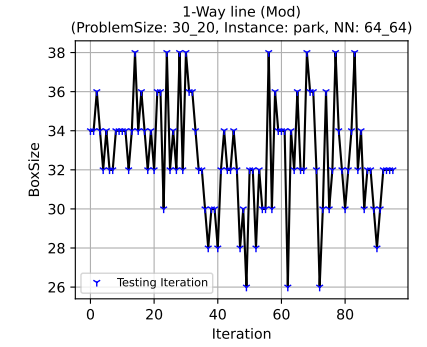

1-Way line (Mod)
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
BoxSize
Iteration

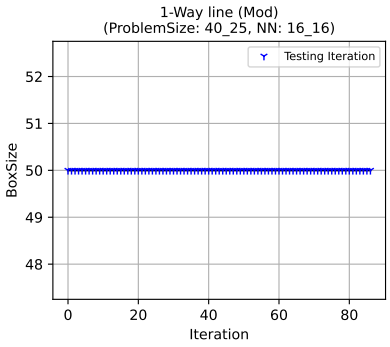

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
BoxSize
Iteration

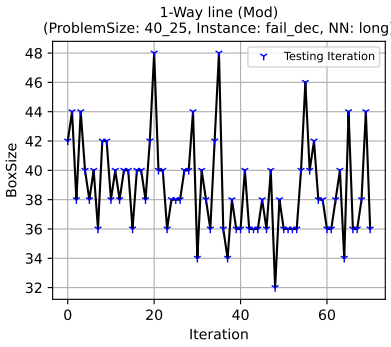

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 32_32)
Testing Iteration
BoxSize
Iteration

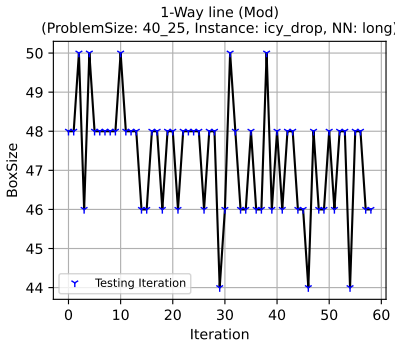

1-Way line (Mod)
(ProblemSize: 20_10, Instance: park, NN: 32_32)
Testing Iteration
BoxSize
Iteration

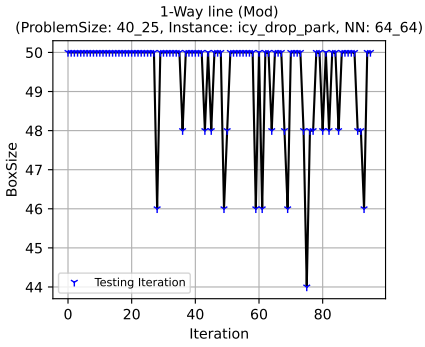

1-Way line (Mod)
(ProblemSize: 30_20, NN: 64_64)
Testing Iteration
BoxSize
Iteration

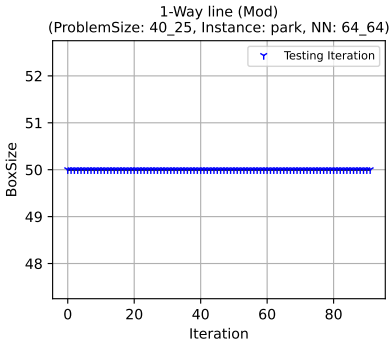

1-Way line (Mod)
(ProblemSize: 30_20, Instance: park, NN: 64_64)
Testing Iteration
BoxSize
Iteration

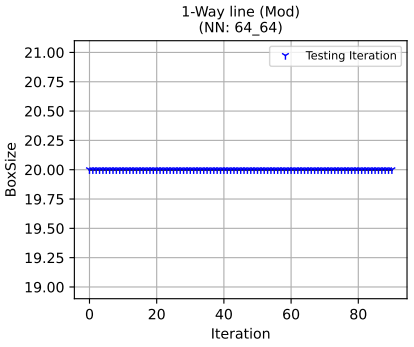

1-Way line (Mod)
(ProblemSize: 40_25, NN: 16_16)
Testing Iteration
BoxSize
Iteration

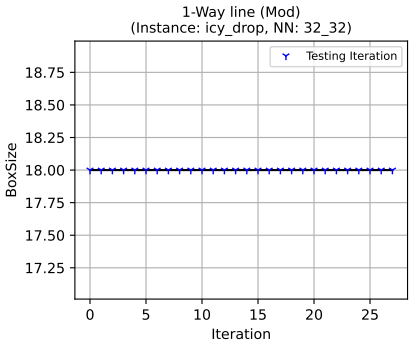

1-Way line (Mod)
(ProblemSize: 40_25, Instance: fail_dec, NN: long)
Testing Iteration
BoxSize
Iteration

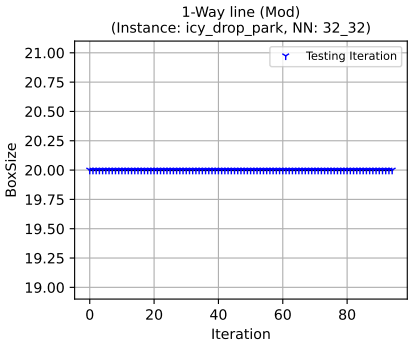

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop, NN: long)
Testing Iteration
BoxSize
Iteration

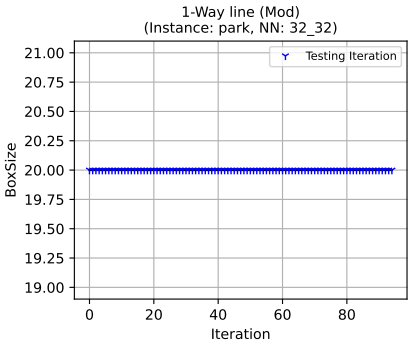

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
BoxSize
Iteration

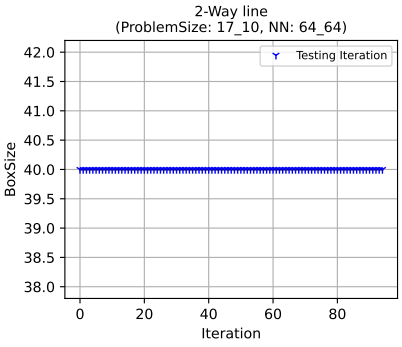

1-Way line (Mod)
(ProblemSize: 40_25, Instance: park, NN: 64_64)
Testing Iteration
BoxSize
Iteration

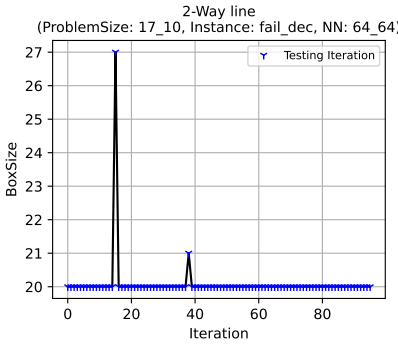

1-Way line (Mod)
(NN: 64_64)
Testing Iteration
BoxSize
Iteration

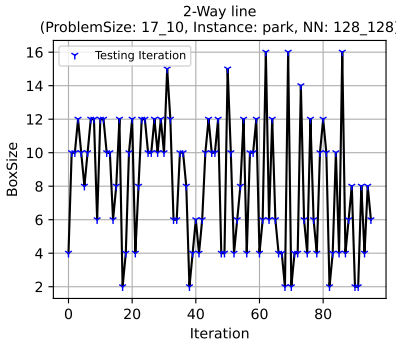

1-Way line (Mod)
(Instance: icy_drop, NN: 32_32)
Testing Iteration
BoxSize
Iteration

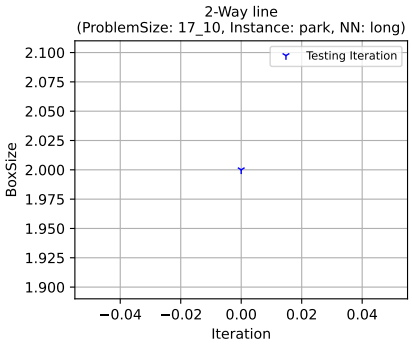

1-Way line (Mod)
(Instance: icy_drop_park, NN: 32_32)
Testing Iteration
BoxSize
Iteration

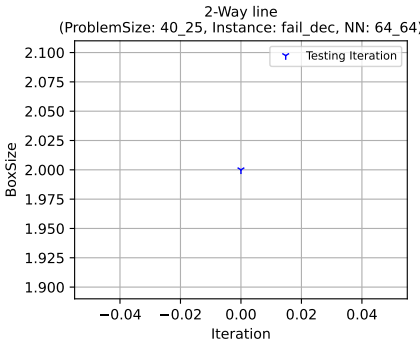

1-Way line (Mod)
(Instance: park, NN: 32_32)
Testing Iteration
BoxSize
Iteration

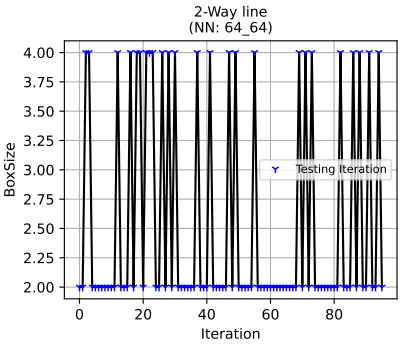

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Testing Iteration
BoxSize
Iteration

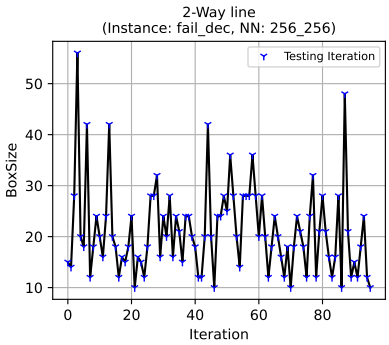

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Testing Iteration
BoxSize
Iteration

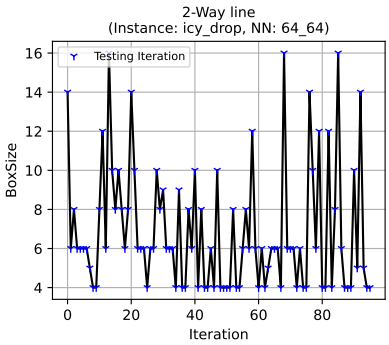

2-Way line
(ProblemSize: 17_10, Instance: park, NN: 128_128)
Testing Iteration
BoxSize
Iteration

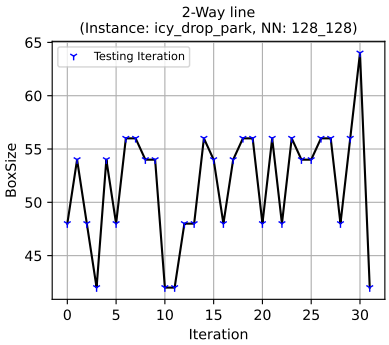

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
BoxSize
Iteration

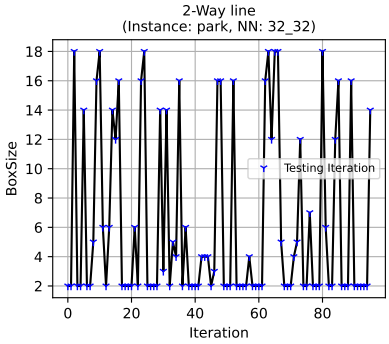

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Testing Iteration
BoxSize
Iteration

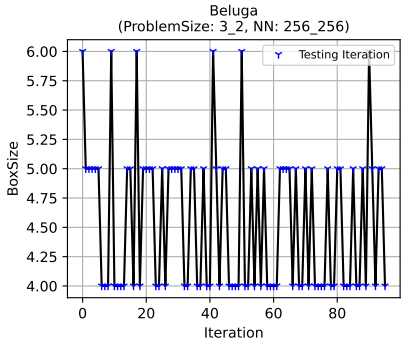

2-Way line
(NN: 64_64)
Testing Iteration
BoxSize
Iteration

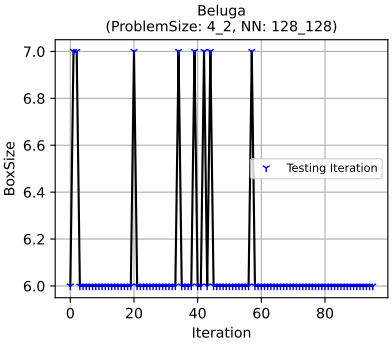

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
BoxSize
Iteration

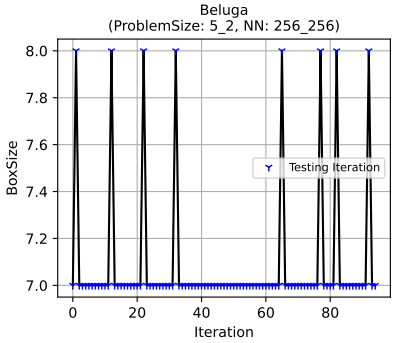

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
BoxSize
Iteration

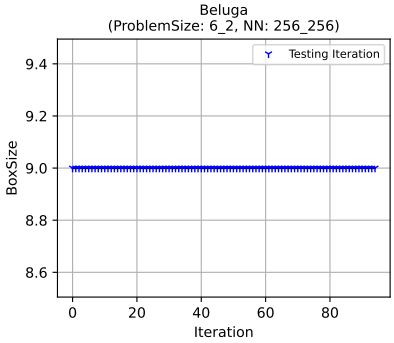

2-Way line
(Instance: icy_drop_park, NN: 128_128)
Testing Iteration
BoxSize
Iteration

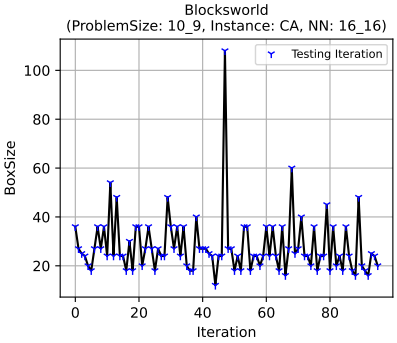

2-Way line
(Instance: park, NN: 32_32)
Testing Iteration
BoxSize
Iteration

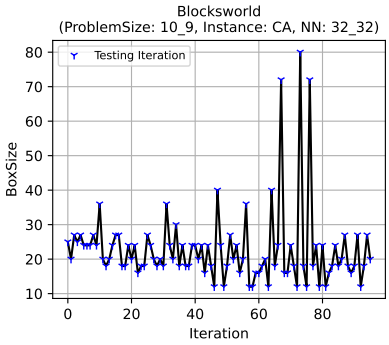

Beluga
(ProblemSize: 3_2, NN: 256_256)
Testing Iteration
BoxSize
Iteration

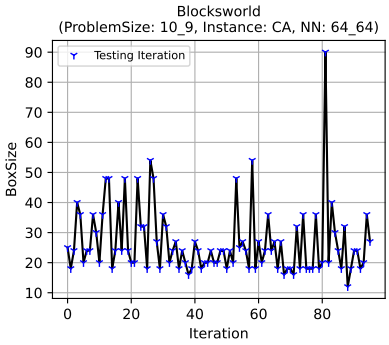

Beluga
(ProblemSize: 4_2, NN: 128_128)
Testing Iteration
BoxSize
Iteration

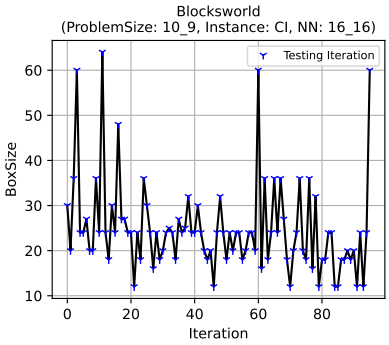

Beluga
(ProblemSize: 5_2, NN: 256_256)
Testing Iteration
BoxSize
Iteration

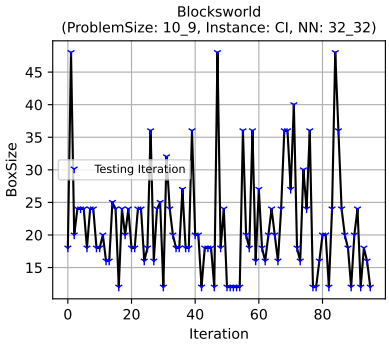

Beluga
(ProblemSize: 6_2, NN: 256_256)
Testing Iteration
BoxSize
Iteration

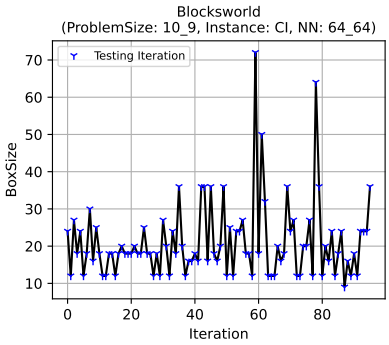

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Testing Iteration
BoxSize
Iteration

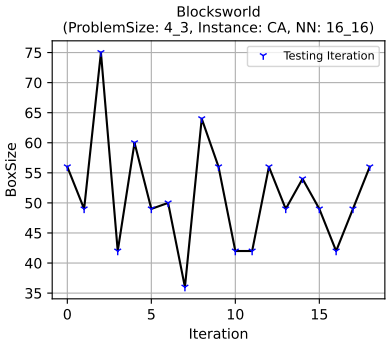

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Testing Iteration
BoxSize
Iteration

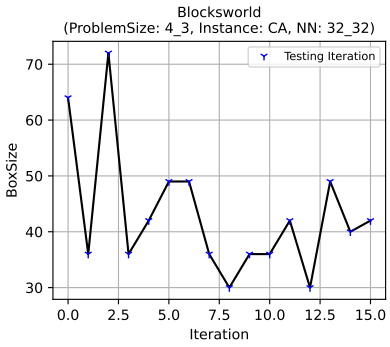

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 64_64)
Testing Iteration
BoxSize
Iteration

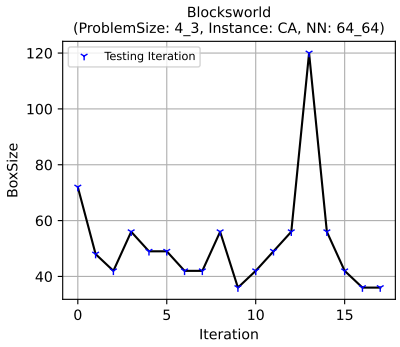

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Testing Iteration
BoxSize
Iteration

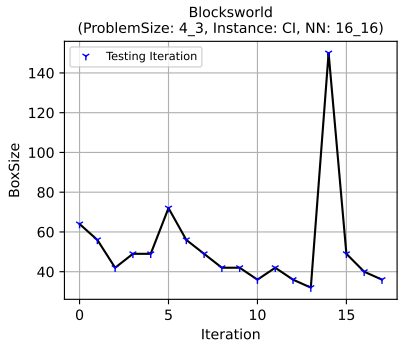

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Testing Iteration
BoxSize
Iteration

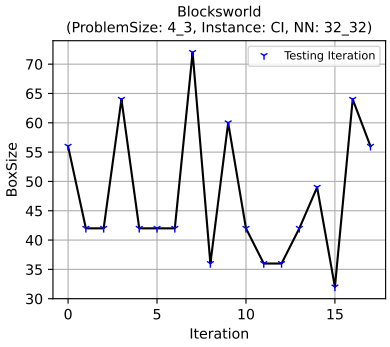

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 64_64)
Testing Iteration
BoxSize
Iteration

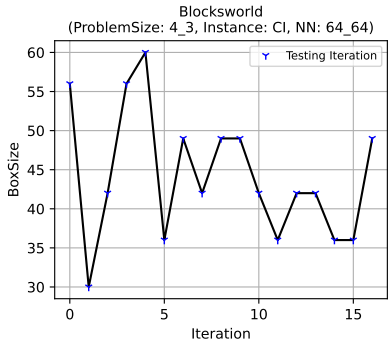

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 16_16)
Testing Iteration
BoxSize
Iteration

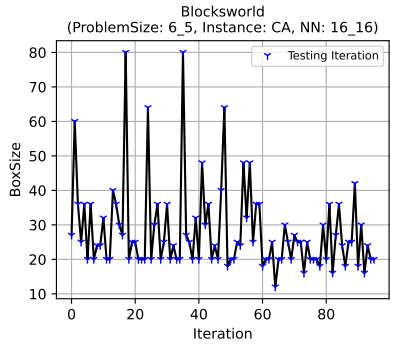

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 32_32)
Testing Iteration
BoxSize
Iteration

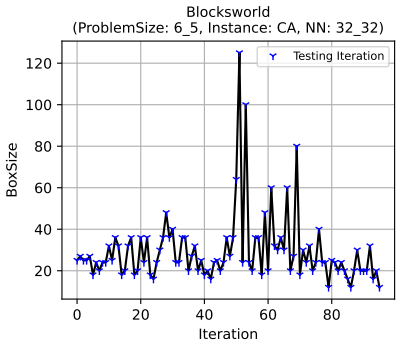

Blocksworld
(ProblemSize: 4_3, Instance: CA, NN: 64_64)
Testing Iteration
BoxSize
Iteration

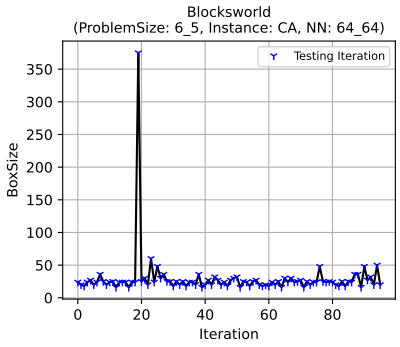

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 16_16)
Testing Iteration
BoxSize
Iteration

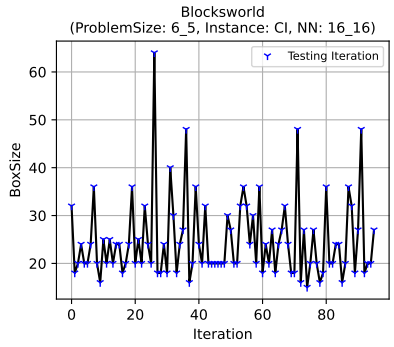

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 32_32)
Testing Iteration
BoxSize
Iteration

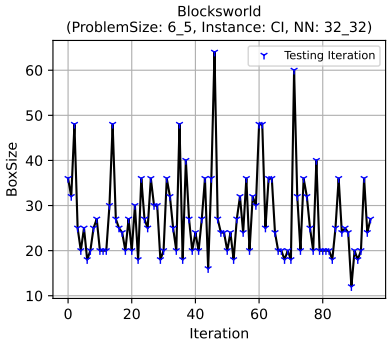

Blocksworld
(ProblemSize: 4_3, Instance: CI, NN: 64_64)
Testing Iteration
BoxSize
Iteration

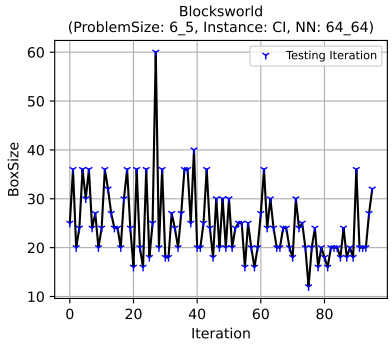

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 16_16)
Testing Iteration
BoxSize
Iteration

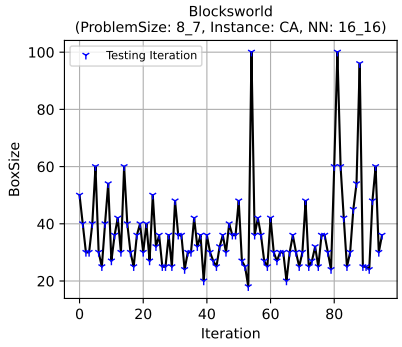

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 32_32)
Testing Iteration
BoxSize
Iteration

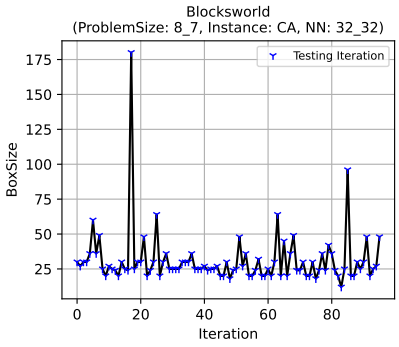

Blocksworld
(ProblemSize: 6_5, Instance: CA, NN: 64_64)
Testing Iteration
BoxSize
Iteration

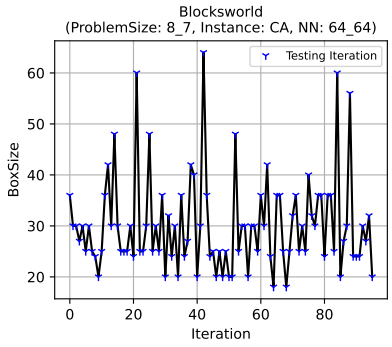

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 16_16)
Testing Iteration
BoxSize
Iteration

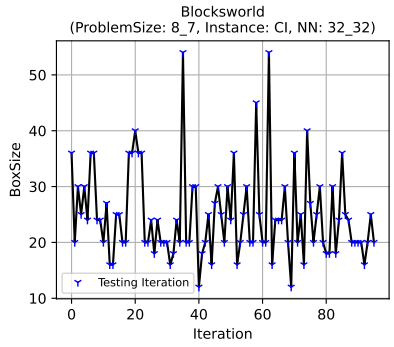

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 32_32)
Testing Iteration
BoxSize
Iteration

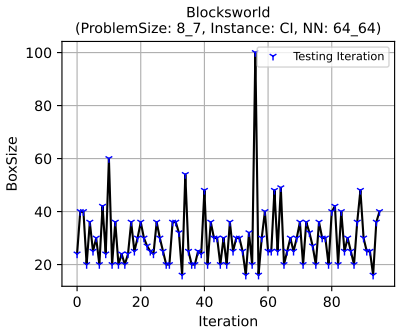

Blocksworld
(ProblemSize: 6_5, Instance: CI, NN: 64_64)
Testing Iteration
BoxSize
Iteration

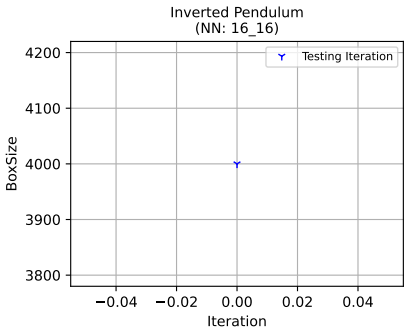

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
BoxSize
Iteration

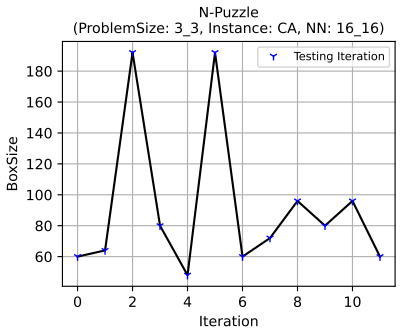

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 32_32)
Testing Iteration
BoxSize
Iteration

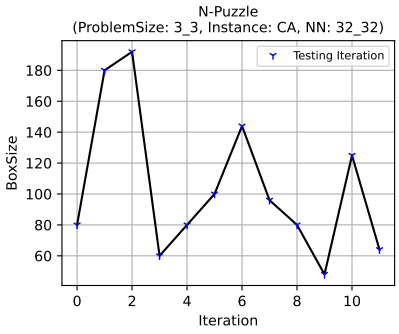

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 64_64)
Testing Iteration
BoxSize
Iteration

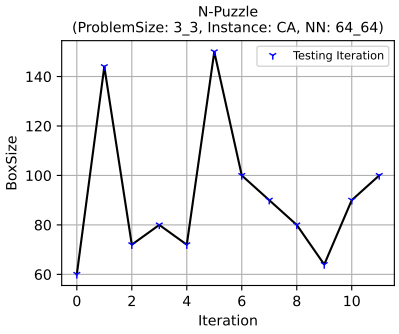

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 32_32)
Testing Iteration
BoxSize
Iteration

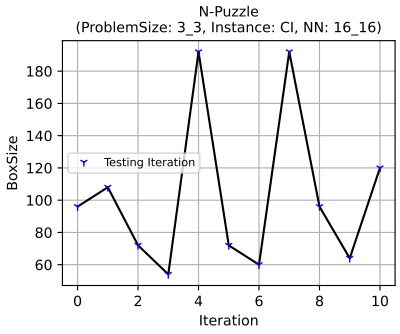

Blocksworld
(ProblemSize: 8_7, Instance: CI, NN: 64_64)
Testing Iteration
BoxSize
Iteration

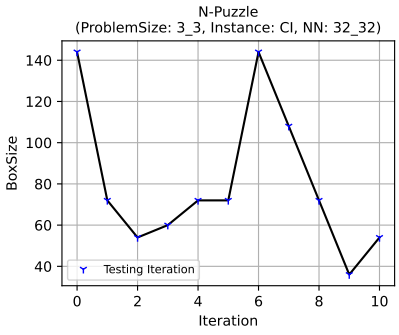

Inverted Pendulum
(NN: 16_16)
Testing Iteration
BoxSize
Iteration

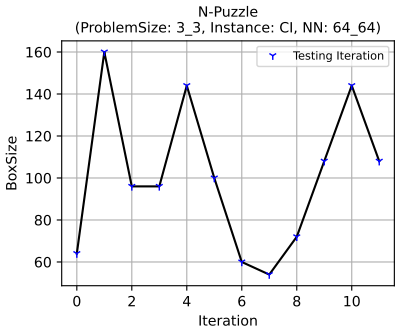

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Testing Iteration
BoxSize
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Testing Iteration
BoxSize
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Testing Iteration
BoxSize
Iteration

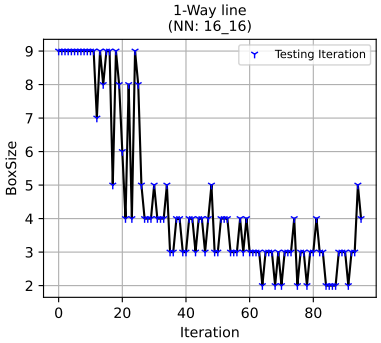

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Testing Iteration
BoxSize
Iteration

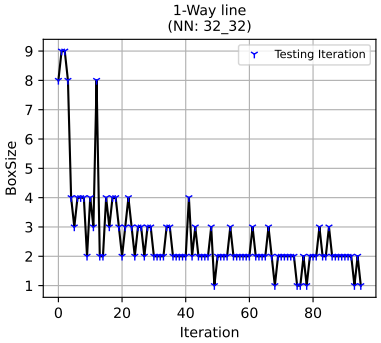

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Testing Iteration
BoxSize
Iteration

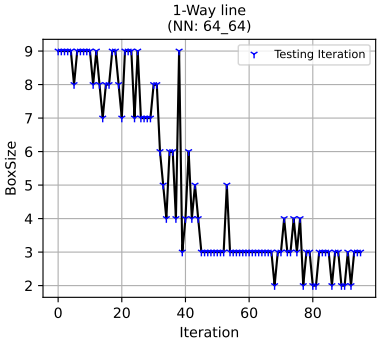

N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Testing Iteration
BoxSize
Iteration

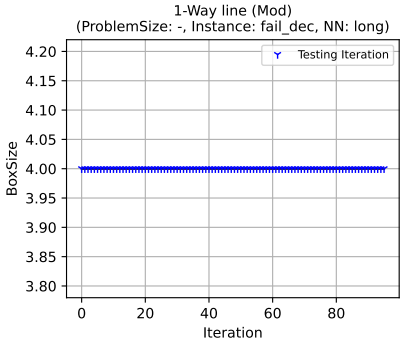
- Results for chapter 5 -

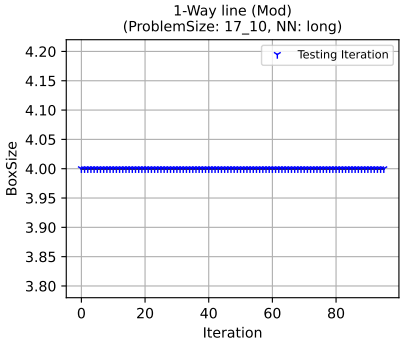
Start Condition Strengthening

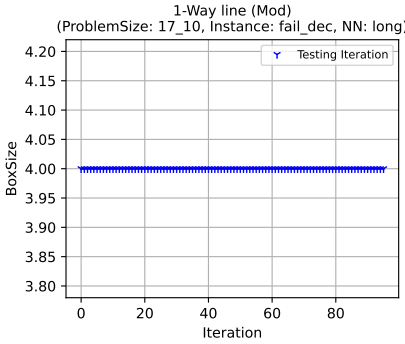

1-Way line
(NN: 16_16)
Testing Iteration
BoxSize
Iteration

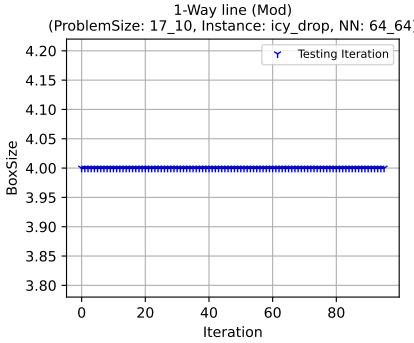

1-Way line
(NN: 32_32)
Testing Iteration
BoxSize
Iteration

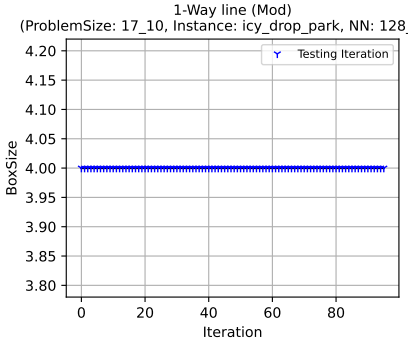

1-Way line
(NN: 64_64)
Testing Iteration
BoxSize
Iteration

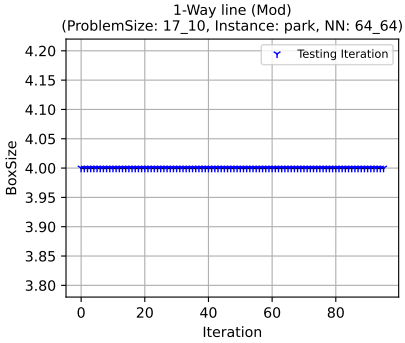

1-Way line (Mod)
(ProblemSize: -, Instance: fail_dec, NN: long)
Testing Iteration
BoxSize
Iteration

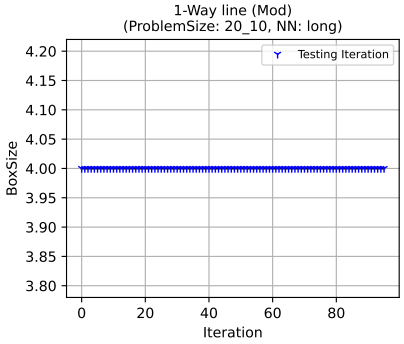

1-Way line (Mod)
(ProblemSize: 17_10, NN: long)
Testing Iteration
BoxSize
Iteration

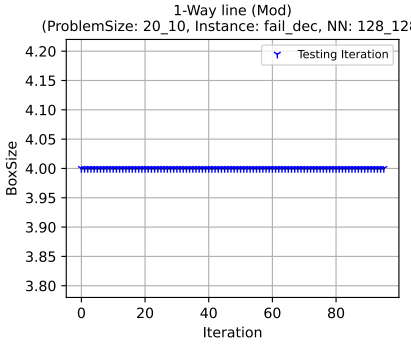

1-Way line (Mod)
(ProblemSize: 17_10, Instance: fail_dec, NN: long)
Testing Iteration
BoxSize
Iteration

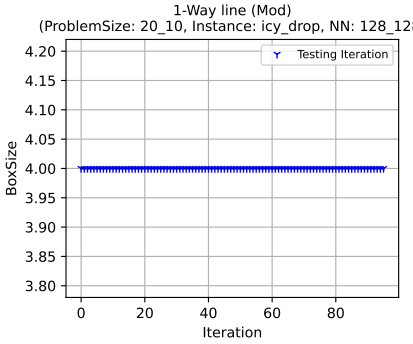

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop, NN: 64_64)
Testing Iteration
BoxSize
Iteration

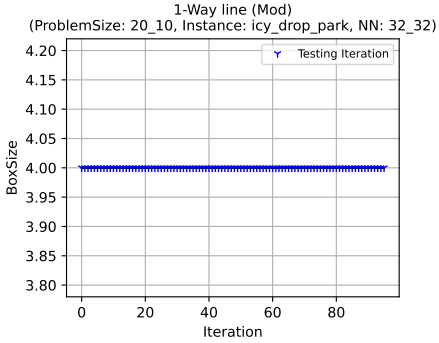

1-Way line (Mod)
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
BoxSize
Iteration

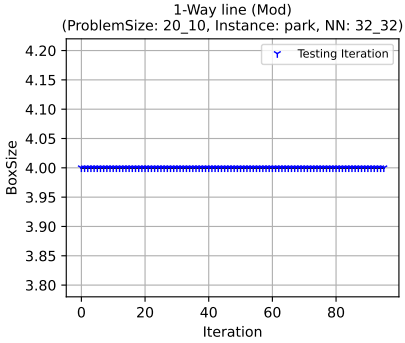

1-Way line (Mod)
(ProblemSize: 17_10, Instance: park, NN: 64_64)
Testing Iteration
BoxSize
Iteration

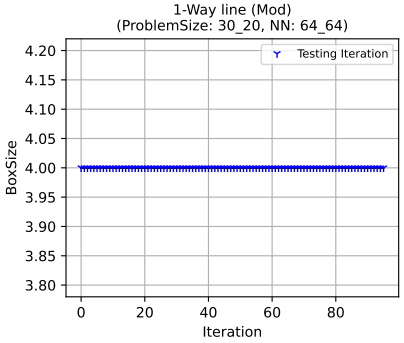

1-Way line (Mod)
(ProblemSize: 20_10, NN: long)
Testing Iteration
BoxSize
Iteration

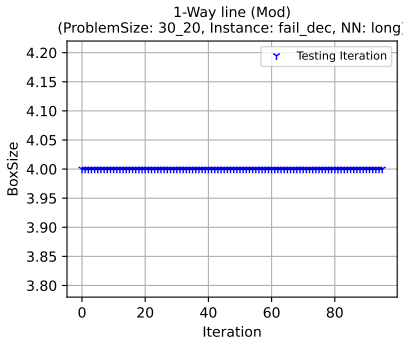

1-Way line (Mod)
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
BoxSize
Iteration

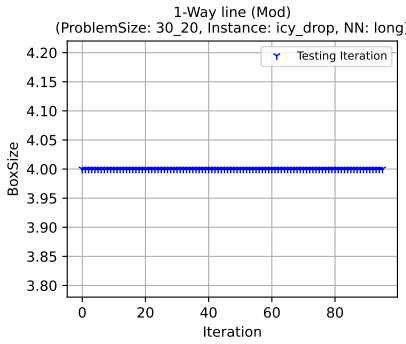

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
BoxSize
Iteration

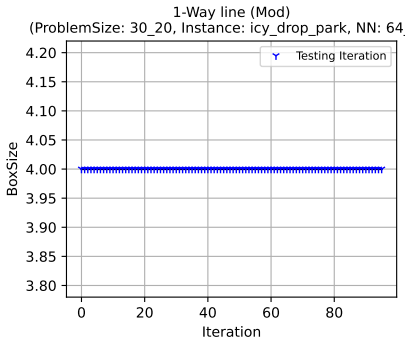

1-Way line (Mod)
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 32_32)
Testing Iteration
BoxSize
Iteration

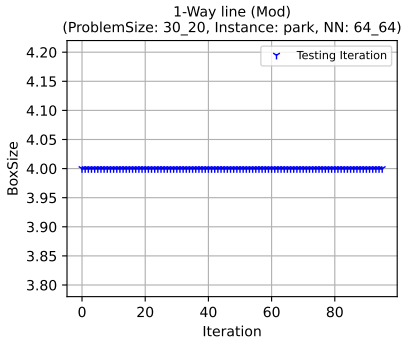

1-Way line (Mod)
(ProblemSize: 20_10, Instance: park, NN: 32_32)
Testing Iteration
BoxSize
Iteration

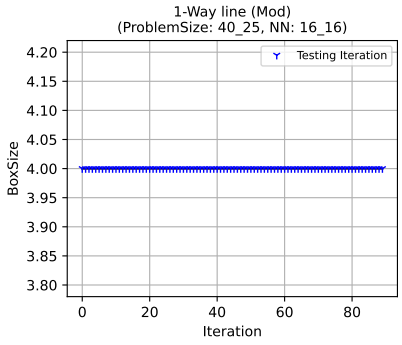

1-Way line (Mod)
(ProblemSize: 30_20, NN: 64_64)
Testing Iteration
BoxSize
Iteration

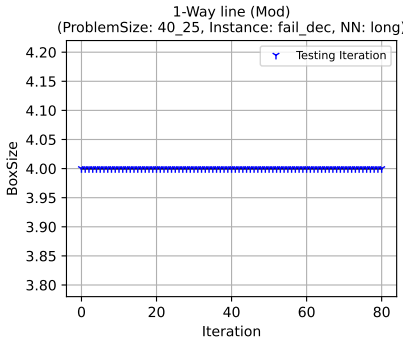

1-Way line (Mod)
(ProblemSize: 30_20, Instance: fail_dec, NN: long)
Testing Iteration
BoxSize
Iteration

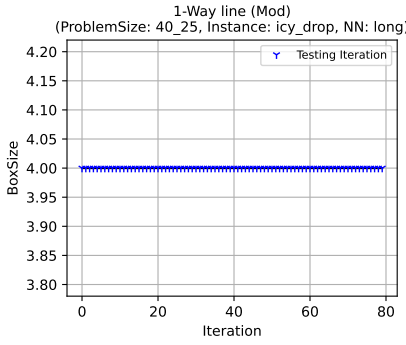

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop, NN: long)
Testing Iteration
BoxSize
Iteration

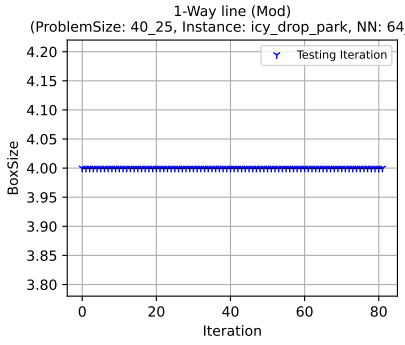

1-Way line (Mod)
(ProblemSize: 30_20, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
BoxSize
Iteration

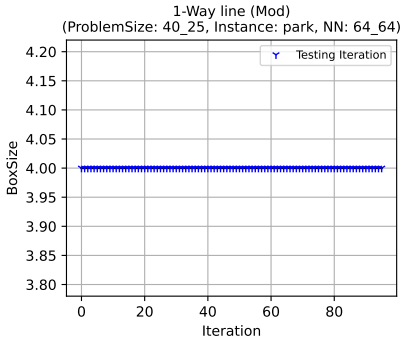

1-Way line (Mod)
(ProblemSize: 30_20, Instance: park, NN: 64_64)
Testing Iteration
BoxSize
Iteration

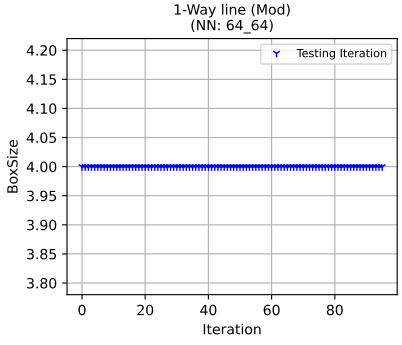

1-Way line (Mod)
(ProblemSize: 40_25, NN: 16_16)
Testing Iteration
BoxSize
Iteration

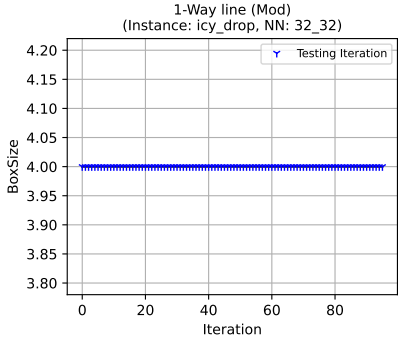

1-Way line (Mod)
(ProblemSize: 40_25, Instance: fail_dec, NN: long)
Testing Iteration
BoxSize
Iteration

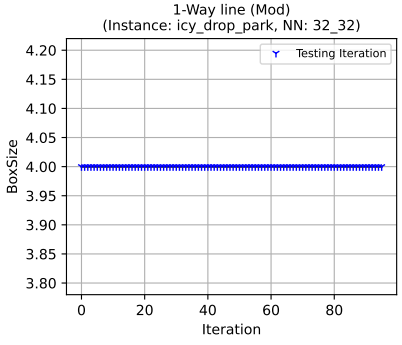

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop, NN: long)
Testing Iteration
BoxSize
Iteration

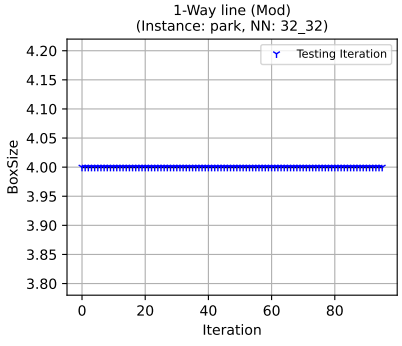

1-Way line (Mod)
(ProblemSize: 40_25, Instance: icy_drop_park, NN: 64_64)
Testing Iteration
BoxSize
Iteration

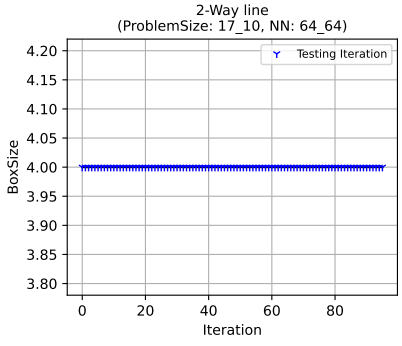

1-Way line (Mod)
(ProblemSize: 40_25, Instance: park, NN: 64_64)
Testing Iteration
BoxSize
Iteration

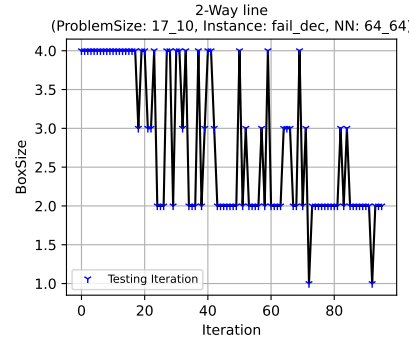

1-Way line (Mod)
(NN: 64_64)
Testing Iteration
BoxSize
Iteration

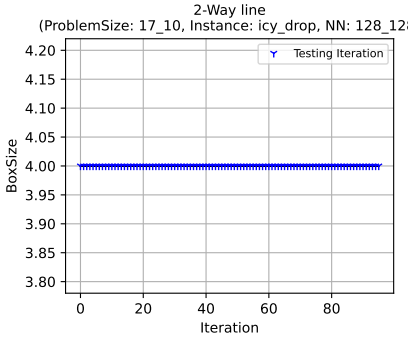

1-Way line (Mod)
(Instance: icy_drop, NN: 32_32)
Testing Iteration
BoxSize
Iteration

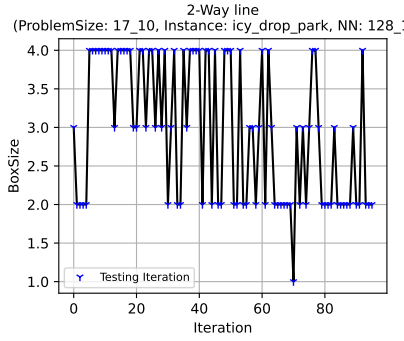

1-Way line (Mod)
(Instance: icy_drop_park, NN: 32_32)
Testing Iteration
BoxSize
Iteration

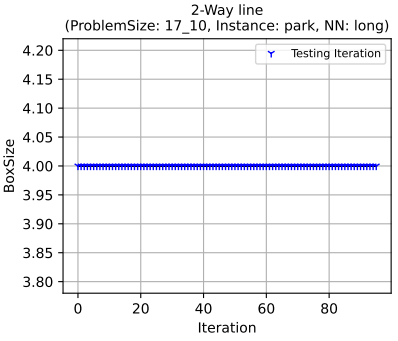

1-Way line (Mod)
(Instance: park, NN: 32_32)
Testing Iteration
BoxSize
Iteration

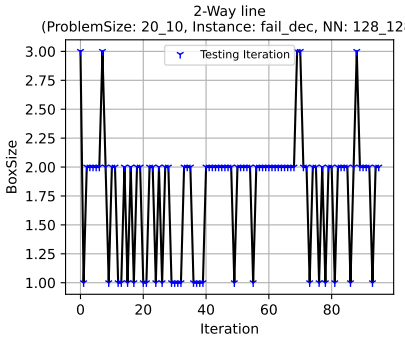

2-Way line
(ProblemSize: 17_10, NN: 64_64)
Testing Iteration
BoxSize
Iteration

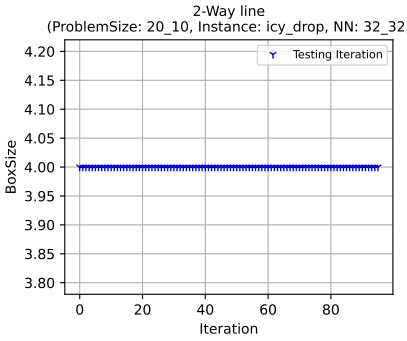

2-Way line
(ProblemSize: 17_10, Instance: fail_dec, NN: 64_64)
Testing Iteration
BoxSize
Iteration

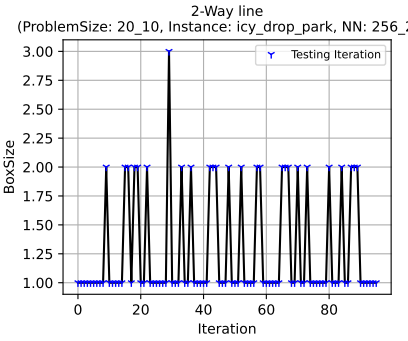

2-Way line
(ProblemSize: 17_10, Instance: icy_drop, NN: 128_128)
Testing Iteration
BoxSize
Iteration

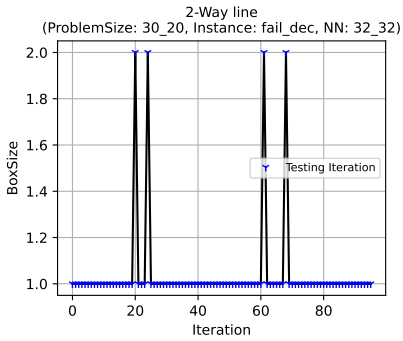

2-Way line
(ProblemSize: 17_10, Instance: icy_drop_park, NN: 128_128)
Testing Iteration
BoxSize
Iteration

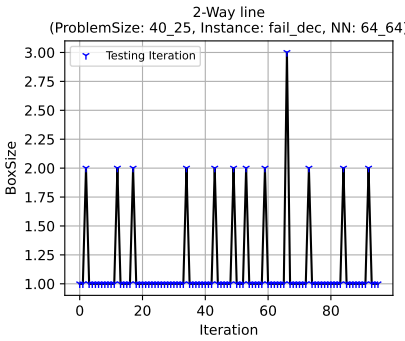

2-Way line
(ProblemSize: 17_10, Instance: park, NN: long)
Testing Iteration
BoxSize
Iteration

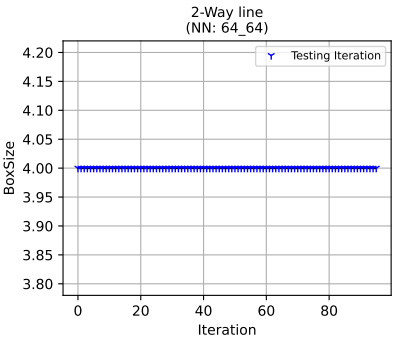

2-Way line
(ProblemSize: 20_10, Instance: fail_dec, NN: 128_128)
Testing Iteration
BoxSize
Iteration

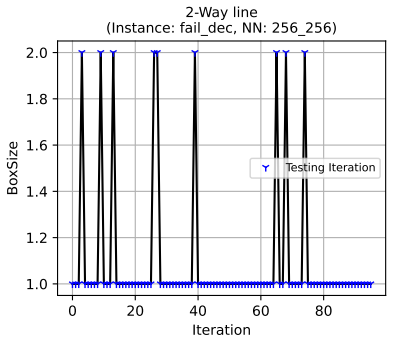

2-Way line
(ProblemSize: 20_10, Instance: icy_drop, NN: 32_32)
Testing Iteration
BoxSize
Iteration

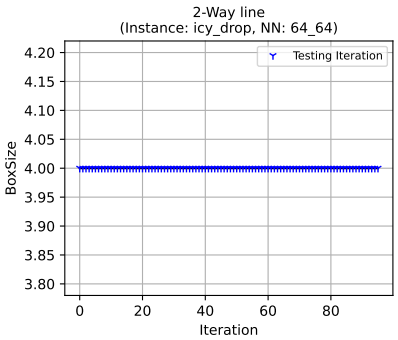

2-Way line
(ProblemSize: 20_10, Instance: icy_drop_park, NN: 256_256)
Testing Iteration
BoxSize
Iteration

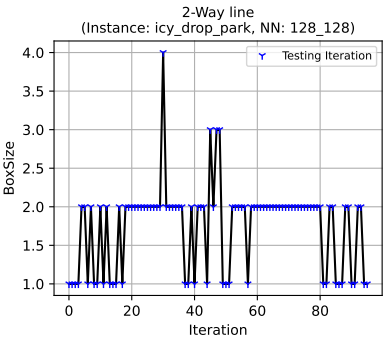

2-Way line
(ProblemSize: 30_20, Instance: fail_dec, NN: 32_32)
Testing Iteration
BoxSize
Iteration

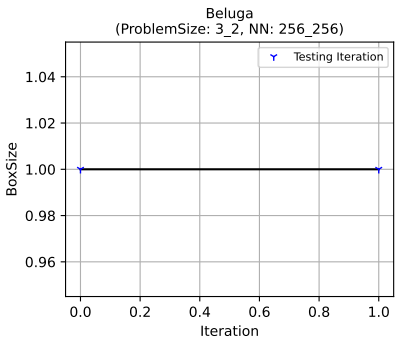

2-Way line
(ProblemSize: 40_25, Instance: fail_dec, NN: 64_64)
Testing Iteration
BoxSize
Iteration

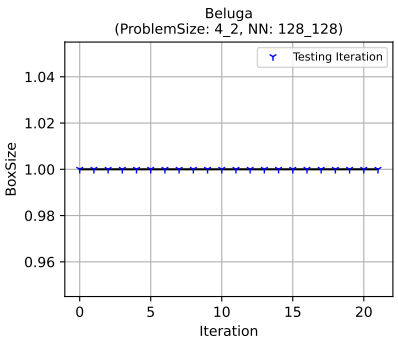

2-Way line
(NN: 64_64)
Testing Iteration
BoxSize
Iteration

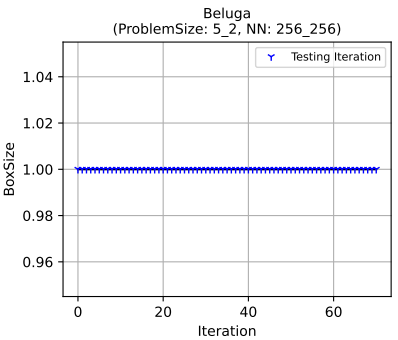

2-Way line
(Instance: fail_dec, NN: 256_256)
Testing Iteration
BoxSize
Iteration

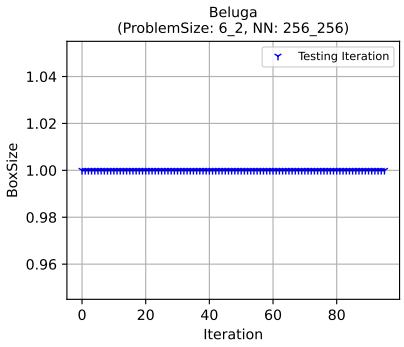

2-Way line
(Instance: icy_drop, NN: 64_64)
Testing Iteration
BoxSize
Iteration

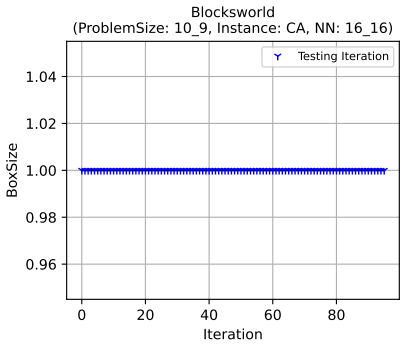

2-Way line
(Instance: icy_drop_park, NN: 128_128)
Testing Iteration
BoxSize
Iteration

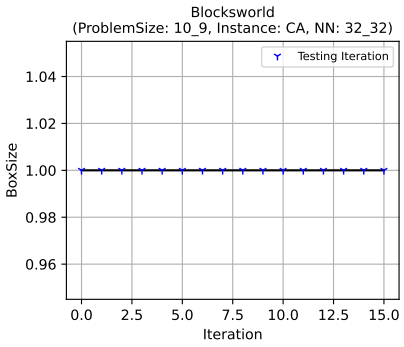

Beluga
(ProblemSize: 3_2, NN: 256_256)
Testing Iteration
BoxSize
Iteration

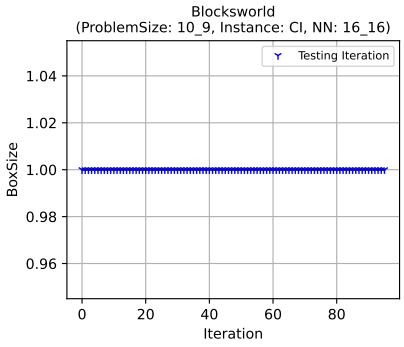

Beluga
(ProblemSize: 4_2, NN: 128_128)
Testing Iteration
BoxSize
Iteration

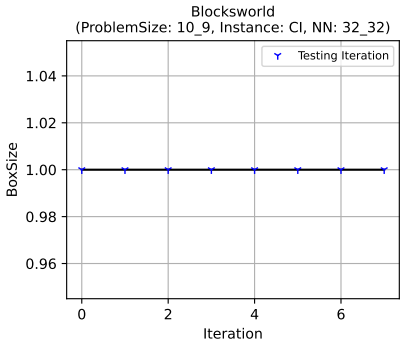

Beluga
(ProblemSize: 5_2, NN: 256_256)
Testing Iteration
BoxSize
Iteration

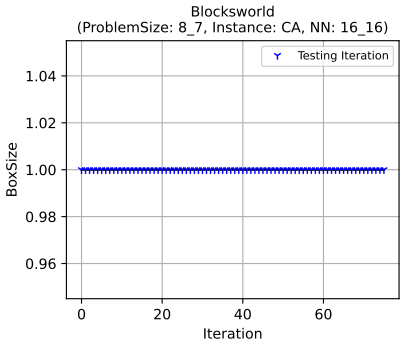

Beluga
(ProblemSize: 6_2, NN: 256_256)
Testing Iteration
BoxSize
Iteration

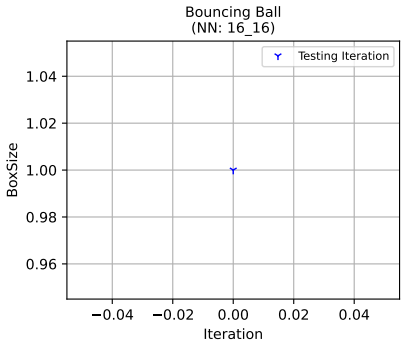

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 16_16)
Testing Iteration
BoxSize
Iteration

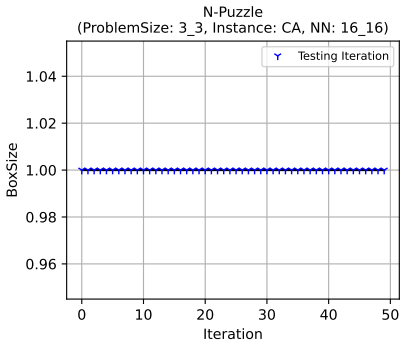

Blocksworld
(ProblemSize: 10_9, Instance: CA, NN: 32_32)
Testing Iteration
BoxSize
Iteration

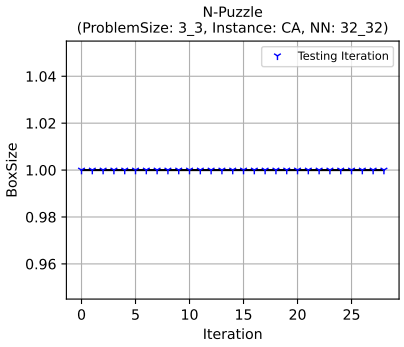

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 16_16)
Testing Iteration
BoxSize
Iteration

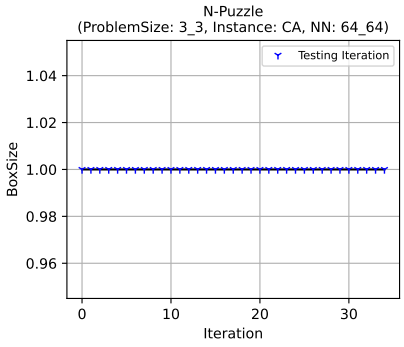

Blocksworld
(ProblemSize: 10_9, Instance: CI, NN: 32_32)
Testing Iteration
BoxSize
Iteration

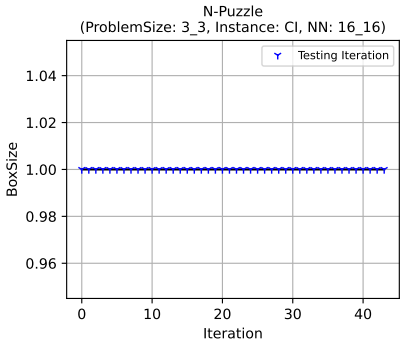

Blocksworld
(ProblemSize: 8_7, Instance: CA, NN: 16_16)
Testing Iteration
BoxSize
Iteration

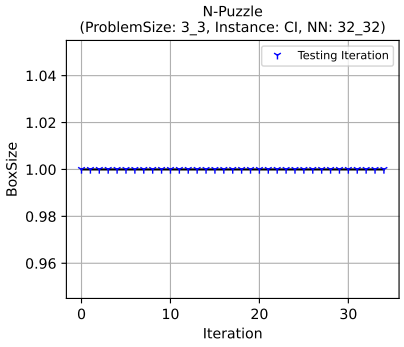

Bouncing Ball
(NN: 16_16)
Testing Iteration
BoxSize
Iteration

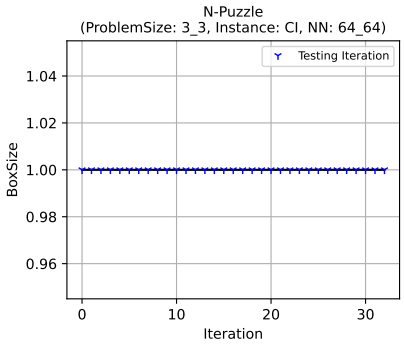

N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 16_16)
Testing Iteration
BoxSize
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 32_32)
Testing Iteration
BoxSize
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CA, NN: 64_64)
Testing Iteration
BoxSize
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 16_16)
Testing Iteration
BoxSize
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 32_32)
Testing Iteration
BoxSize
Iteration


N-Puzzle
(ProblemSize: 3_3, Instance: CI, NN: 64_64)
Testing Iteration
BoxSize
Iteration

Bibliography

[1] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[2] Marcel Vinzent, Siddhant Sharma, and Jöerg Hoffmann. Neural policy safety verification via predicate abstraction: Cegar. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 15188–15196, 2023.

[3] Chaahat Jain, Lorenzo Cascioli, Laurens Devos, Marcel Vinzent, Marcel Steinmetz, Jesse Davis, and Jörg Hoffmann. Safety verification of tree-ensemble policies via predicate abstraction. In *27th European Conference on Artificial Intelligence, 19–24 October 2024, Santiago de Compostela, Spain–Including 13th Conference on Prestigious Applications of Intelligent Systems (PAIS 2024)*, volume 392, pages 1189–1197. IOS Press, 2024.

[4] Michael Akintunde, Alessio Lomuscio, Lalit Maganti, and Edoardo Pirovano. Reachability analysis for neural agent-environment systems. In *Sixteenth international conference on principles of knowledge representation and reasoning*, 2018.

[5] Marcel Vinzent, Marcel Steinmetz, and Jörg Hoffmann. Neural network action policy verification via predicate abstraction. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pages 371–379, 2022.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[7] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[8] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*, pages 443–452. Springer, 2019.