Institute of Formal Methods in Computer Science

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit Nr. 76

The Simultaneous Maze Solving Problems

André Nusser

Course of Study:	Informatik
Examiner:	Stefan Funke
Supervisor:	Sabine Storandt, Stefan Funke
Commenced:	3. December 2015
Completed:	3. June 2016
CR-Classification:	G.2.1

Abstract

A grid maze is a binary matrix where fields containing a 0 are accessible while fields containing a 1 are blocked. In such a maze there are four possible movements: up, down, left, and right. We call a sequence of such movements a Solving Sequence if we *visit* the lower-right corner starting in the upper-left corner. Finding a Solving Sequence for a grid maze is a problem that has been thoroughly considered. However, finding a single sequence such that all grid mazes in a given set are solved has not drawn great attention. Especially the formulation as a minimization problem – i. e., finding a *shortest* Solving Sequence for a set of mazes – is a challenging problem. We call this minimization problem the Simultaneous Maze Solving Problem (SIMASOP). Beside this general formulation, we also focus on a special case of SIMASOP called the All Simultaneous Maze Solving Problem (ASIMASOP). Given n and m, this problem requires us to find a shortest Solving Sequence for the set of all solvable grid mazes of size $n \times m$. In this thesis we analyze both problems theoretically as well as practically. Among other theoretical results, we prove that SIMASOP is NP-complete, that ASIMASOP is in PSPACE, and give a cubic upper bound for the length of a shortest Solving Sequence for ASIMASOP. On the practical side, we present algorithms to compute shortest and approximately shortest Solving Sequences. Additionally, we provide a non-naive algorithm for finding an unsolved maze given a non-Solving Sequence and ways to compute instance-based lower bounds. Finally, we evaluate the algorithms and compare the results of the different approaches as well as provide lower bounds. Surprisingly, for ASIMASOP with size 4×4, for which there exist 3828 solvable mazes, it is already difficult to find a shortest Solving Sequence. We are able to compute a Solving Sequence of length 29 and a lower bound of 26 for this instance.

Contents

1 Introduction

Mazes regularly appear in popular culture and have also been investigated in math and computer science. It is a common game to search for a path from start to goal in a maze. One can also search for the shortest path instead of just any path. This has already been investigated thoroughly and finding a shortest path in a two-dimensional maze can be considered completely solved from the perspective of computer science. Consider the first time we solve a certain grid maze. In this case we have to apply some algorithm to get a solution. This solution can then be represented as a sequence of moves. If we now give this sequence of moves to a person who has not solved the maze before, he or she can simply follow these instructions and solve the maze. However, what if we do not only have a single maze but multiple mazes? And what if, instead of solving them separately, we want exactly one sequence of moves that solves all of the mazes? By simply choosing a very long sequence of moves, one can create a Solving Sequence with high probability. Therefore, the interesting problem is to find a short or even shortest Solving Sequence. Surprisingly, this problem has never been thoroughly considered in computer science to our knowledge. However, this topic has already been discussed on two different websites: the XKCD forums [XKCa] and Stack Overflow [Staa]. While the discussion on the former is aimed at proving the existence of Solving Sequences and Perfect Solving Sequences (which we define in Section 2.1), the discussion on the latter focuses on providing algorithms that compute short Solving Sequences. There, a Solving Sequence of length 31 for all solvable mazes of size 4×4 is presented [Stab]. We use some of the results presented there and reference them accordingly. While there also already exist some tools in theoretical computer science to approach this problem, a consideration of this exact problem can yield deeper insights than considering it simply as a special case of more general problems.

Apart from the theoretical interest in this problem, it can also lay the foundations for more practical problems. Consider for example exploration or repair robots. Those robots are deployed in dangerous scenarios involving buildings which are not save to enter for humans – e. g., think of the ruin of the Fukushima nuclear power plant. In this setting it is a real possibility that sensors and communication fail during a mission. The robots that were sent into Reactor 3 of the Fukushima power plant in the year 2016 were destroyed because of the high radiation [Sci]. Due to the high cost of such robots, they should always be able to exit the building as long as self-powered movement is still possible. To evacuate them, an emergency protocol is needed that enables them to escape. Here, the solution to our problem gives an abstract formulation of such an emergency protocol. If we consider the building as a maze of a certain size, then using a sequence that solves all the mazes of this size induces a high probability of escaping. As often, in practice it is probably much harder. However, a rigorous theoretical formulation helps to develop such a system in the end.

Structure

This thesis is structured in the following way:

Chapter 2 – Basics: In this chapter we introduce all the definitions that are later needed in the thesis and already define the main problems that we want to investigate.

Chapter 3 – Related Problems: Instead of directly diving into solving our problems, we first have a look at related problems and see in what sense they already give a solution. While not implying perfect solving strategies for our problems, the problems introduced in this chapter help us in the theoretical as well as the practical analysis.

Chapter 4 – Theoretical Analysis: Before having a look at the algorithmic side of the problem, we investigate the theoretical properties. Among other properties, complexity bounds and existence of Solving Sequences are shown.

Chapter 5 – Practical Algorithms: In this chapter we present algorithms for solving the problems formulated earlier.

Chapter 6 – Practical Analysis: Using the algorithms defined in the last chapter, we analyze our problem in practice.

2 Basics

2.1 Definitions

In the following, we define the most important mathematical objects we will be working with.

Definition (Maze)**.** *A maze is a matrix $M \in \{0,1\}^{n \times m}$ where 0s symbolize free fields while 1s symbolize blocked fields. The single elements of the maze are denoted by $\rho_{i,j}$.*

If not noted otherwise, we use n as the size of the first dimension and m as the size of the second dimension of the maze(s). Figure 2.1 shows a 4×5 maze (i. e., a matrix) with its visualization. If not mentioned otherwise, we assume the start and goal position to always be in the upper-left and lower-right, respectively.

Definition (Position)**.** *A position in a maze is given by a tuple (i, j) where $1 \leq i \leq n, 1 \leq j \leq m$. This is the position of the element $\rho_{i,j}$ of the maze. Thus, the origin of the indices is the upper-left corner, i. e. the default starting position.*

For example, in Figure 2.1 the position $(1, 1)$ is marked red, the position $(4, 5)$ is marked green and the positions of the blocked fields are $(1, 2), (1, 4), (2, 4), (2, 5), (3, 2)$ and $(4, 2)$.

Definition (Move)**.** *A move η is an element of the set of moves $\{u, d, l, r\}$.*

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

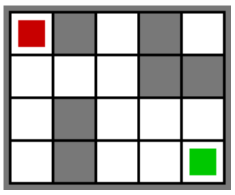

Figure 2.1: A maze with its corresponding visualization. The white fields are accessible while the gray ones are blocked. The red marked field is the start position and the green marked field the goal position.

The symbols in $\{u, d, l, r\}$ are abbreviations for "up", "down", "left" and "right". Those directions are interpreted as relative to the viewer of the maze.

Definition (Sequence)**.** *A sequence is a string $s \in \{u, d, l, r\}^*$, i. e. a concatenation of moves.*

To be able to navigate through a maze, we define a step function:

$$\text{step}_M((i,j),\eta) = \begin{cases} (i-1,j), & \text{if } \eta = u \wedge i > 1 \wedge \rho_{i-1,j} = 0 \\ (i+1,j), & \text{if } \eta = d \wedge i < n \wedge \rho_{i+1,j} = 0 \\ (i,j-1), & \text{if } \eta = l \wedge j > 1 \wedge \rho_{i,j-1} = 0 \\ (i,j+1), & \text{if } \eta = r \wedge j < m \wedge \rho_{i,j+1} = 0 \\ (i,j), & \text{otherwise} \end{cases}$$

The result of the step function is intuitive. If the adjacent field in movement direction is free, we enter it. If we are at the edge of the maze or adjacent to a blocked field and walk in this direction, then we stay on the current field. If it is obvious which maze we are referring to, we omit the index (e. g., M in the definition). Also, if we start at the upper-left, we omit the position (e. g., (i, j) in the definition) for sake of simplicity of the notation. We also define the step function on a sequence $s = s_1 \dots s_k$ as follows:

$$\text{step}((i,j), s) = \text{step}(\text{step}(\dots\text{step}(\text{step}((i,j), s_1), s_2), \dots), s_k)$$

As an example, let us apply the step function to the maze shown in Figure 2.1:

$$\text{step}((1,3), drll) = (2,1)$$

We are now ready to define Solving Sequences.

Definition (Solving Sequence)**.**

$$s = s_1 \dots s_k \textit{ is a Solving Sequence of } M \Leftrightarrow \exists i, 0 \le i \le k : \textit{step}_M((1,1), s_1 \dots s_i) = (n,m)$$

In other words, a sequence s is called a Solving Sequence of a maze M if we visit the position (n, m) during the traversal of M described by s starting from $(1, 1)$. We call s a Solving Sequence for a set of mazes $\mathcal{M}$, if s is a Solving Sequence for each $M \in \mathcal{M}$. If s is a Solving Sequence of M ($\mathcal{M}$), we also say that s solves M ($\mathcal{M}$).

Definition (Perfect Solving Sequence)**.**

$$s \textit{ is a Perfect Solving Sequence of } M \Leftrightarrow \textit{step}_M((1,1), s) = (n,m)$$

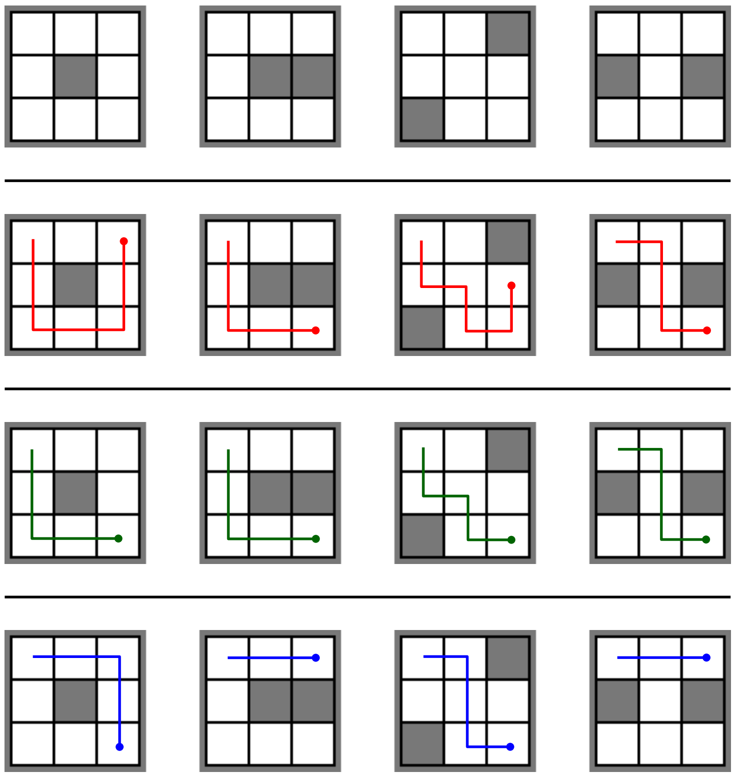

Figure 2.2: First row: The four example mazes. Solving Sequence but not Perfect Solving Sequence (second row, red): $ddrddruu$. Perfect Solving Sequence and thus also Solving Sequence (third row, green): $drddrr$. Neither Solving Sequence nor Perfect Solving Sequence (fourth row, blue): $rrddr$.

In other words, a Solving Sequence s of a maze M is called perfect if we end in position (n, m) after executing s on M starting in position $(1, 1)$. The definition also carries over to sets of mazes. Additionally, if s is a Perfect Solving Sequence of M ($\mathcal{M}$), we also say that s perfectly solves M ($\mathcal{M}$).

Consider Figure 2.2 for examples of Solving Sequences and Perfect Solving Sequences.

Definition (All Solving Sequence)**.** *A sequence s is an All Solving Sequence regarding the maze size $n \times m$ iff it solves every solvable maze of this size.*

For example, the sequence $drddruurrdd$ is an All Solving Sequence regarding the size 3×3. You can conveniently check this using Figure 2.3.

2.2 Notation and Conventions

To facilitate the reading of this thesis, the following consistent notation is used:

- $M, M_1, M_2, \ldots$ are used for single mazes
- $\mathcal{M}, \mathcal{M}_1, \mathcal{M}_2, \ldots$ are used for sets of mazes
- $\mathcal{M}_{n,m}$ is used for the set of all solvable $n \times m$ mazes
- $\eta \in \{u, d, l, r\}$ is a move
- $s \in \{u, d, l, r\}^*$ is a sequence of moves
- $\sigma(\mathcal{M})$ is the minimal length of a sequence solving all mazes in $\mathcal{M}$
- $\sigma(n, m) = \sigma(\mathcal{M}_{n,m})$

We assume that sets of mazes contain mazes of the same size if not stated otherwise. Many proofs also generalize to sets containing mazes of different sizes, but this would often compromise readability.

In the course of this thesis we also use graph algorithms on mazes and sometimes implicitly consider mazes as graphs. The graph corresponding to a certain maze is created in a straight forward manner: every free and reachable field corresponds to a node and all the different moves from a field (up, down, left and right) correspond to an accordingly labeled edge, which has as target the node corresponding to the field that is reached by the move.

2.3 The Simultaneous Maze Solving Problem

In the Simultaneous Maze Solving Problem (SIMASOP) we want to find the length of the shortest Solving Sequence for a given set of mazes. More formally, given a set of mazes $\mathcal{M}$, find the smallest number k such that there exists a Solving Sequence s for $\mathcal{M}$ with $|s| = k$. Note that s might not be unique but $|s|$ is. The corresponding decision problem to SIMASOP is defined as follows: Given a set of mazes $\mathcal{M}$ and an upper bound k, does a Solving Sequence s with regard to $\mathcal{M}$ exist such that $|s| \leq k$?

The result of the SIMASOP on the mazes in Figure 2.2 is 6, and a shortest Solving Sequence is $ddrddr$.

2.4 The All Simultaneous Maze Solving Problem

The All Simultaneous Maze Solving Problem (ASIMASOP) is a special case of the Simultaneous Maze Solving Problem where only the dimensions of the maze n, m are the inputs (given in unary encoding) and the set of mazes $\mathcal{M}$ of SIMASOP is set to $\mathcal{M}_{i,j}$ (i. e., *all solvable mazes* of size $n \times m$) – we then again search for the length of the shortest Solving Sequence. The corresponding decision problem is equivalent to the one of SIMASOP, i. e.: given a maze size $n \times m$ and an upper bound k, does a Solving Sequence s for the set of all solvable mazes of size $n \times m$ exist such that $|s| \leq k$?

For $n = m = 3$ the result of ASIMASOP is 11 and such a shortest Solving Sequence is $drddruurrdd$. Manually verifying that there indeed does not exist a shorter Solving Sequence is not trivial.

2.5 Counting Solvable Mazes

While not the central problem, it's also natural to ask for the number of solvable mazes of a certain size. Unfortunately, this doesn't seem to be straight forward calculable, but one can easily give some naive upper and lower bounds.

For the first naive *upper* bound note that neither the start nor the goal field can be blocked, but all others might be free or blocked. Thus, we have $nm - 2$ fields to choose (assuming $n > 1$ or $m > 1$), but not all choices imply a solvable maze. Therefore, a naive upper bound is 2^{nm-2}.

A little less naive *upper* bound is $\frac{9}{16}2^{nm-2}$ for $n, m > 2$. We again consider all fields that we can choose, but this time we check how many combinations of the neighboring fields of the start and goal field can lead to a solvable maze. Both, the start and the goal field have exactly two neighbors for $n, m > 2$. If both of the neighbors are blocked, the maze cannot be solved. Therefore, we have 3 out of 4 choices for each pair of neighbors such that the maze is not obviously unsolvable – i. e., 9 out of 16 choices in total, which explains the factor of the upper bound.

Let us now derive a naive *lower* bound. If we fix a solving path that goes along the left edge and then the bottom edge of the maze, we still have all fields in the upper right that can be arbitrarily chosen without making the maze unsolvable. The number of those fields is $(n-1)(m-1)$. It follows that $2^{(n-1)(m-1)}$ is a lower bound for the number of solvable mazes of size $n \times m$.

From the previous bounds it follows that:

$$2^{(n-1)(m-1)} \leq |\mathcal{M}_{n,m}| \leq \frac{9}{16} 2^{nm-2}$$

In Section 4.7 we present a way to calculate the exact number of mazes without explicitly enumerating them. Additionally, a more sophisticated lower bound calculation is shown. Also, I could not refrain from including a figure that shows the 51 solvable mazes of size 3×3: see Figure 2.3.

2.6 Markov Chains

If we consider random walks on a maze, we can use Markov chains as a mathematical model. As they have already been investigated thoroughly, we can use a lot of those results. Let us begin with defining Markov chains. We use less general definitions here for the sake of simplicity. See Section 6.2 in [MR10] as reference for more general versions of the following definitions.

Definition (Markov chain)**.** *A Markov chain is a directed graph $G = (V, E)$ of which the edges are labeled with transition probabilities $p : E \to (0, 1]$ with*

$$\forall v \in V : \sum_{e=(v,\cdot)\in E} p(e) = 1$$

It may contain loop edges but not multiple edges.

We also call V the *set of states* and $v \in V$ a *state*. A Markov chain is called *time-homogeneous* if the transition probabilities don't change over time. It is called *finite* if $|V| < \infty$. For the rest of this thesis we assume that a Markov chain is time-homogeneous and finite unless stated otherwise. There are several other terms and results relevant for this thesis, namely: irreducible, (a)periodic and stationary distribution.

Definition (irreducible)**.** *A Markov chain is called irreducible iff the graph G defining the Markov chain consists of exactly one strongly connected component, i. e., every node $v \in V$ can reach each other node $w \in V$.*

Definition (periodic)**.** *A state $v \in V$ of a Markov chain is called periodic iff for some initial distribution π_0 over the states, the probability of being in state v at time t is only greater than zero iff $t \in \{a + Ti \mid i \geq 0\}$, for some $a, T \in \mathbb{N}, a > 0, T > 1$. Else, v is called aperiodic.*

A Markov chain is called aperiodic iff every state of it is aperiodic.

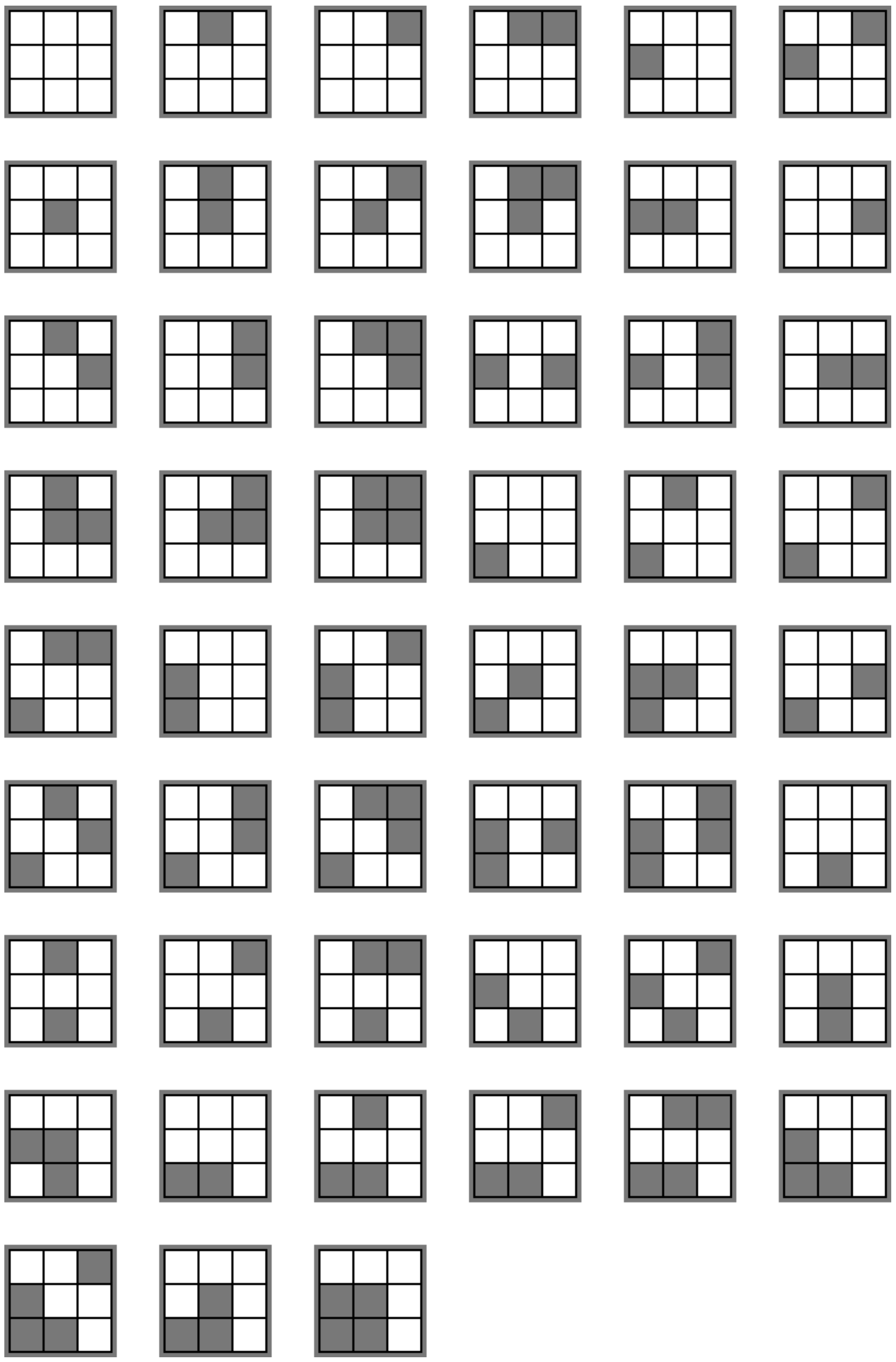

Figure 2.3: All solvable mazes of size 3×3. Beautiful, isn't it?

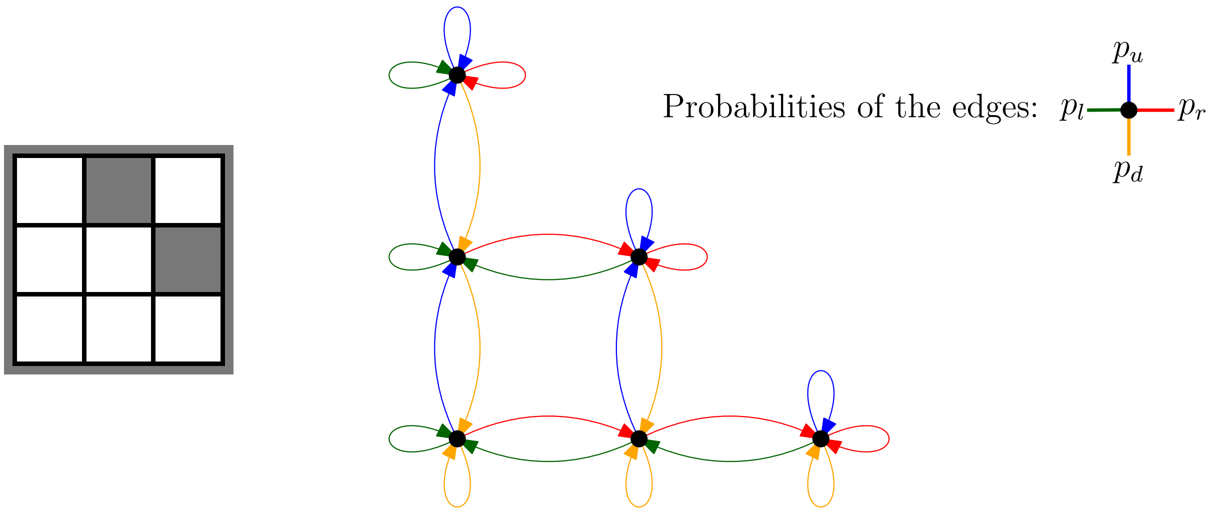

Figure 2.4: A maze with its corresponding Markov maze. Note that the graph of the Markov maze is restricted to the (strongly) connected component of the start and goal states. The edges are color-coded regarding their transition probability, which is induced by their direction.

Definition (stationary distribution)**.** *A stationary distribution of a Markov chain is a probability distribution* $\pi_v \in [0, 1]$ *over the states* $v \in V$ *such that*

$$\sum_{v \in V} \pi_v = 1 \text{ and } \forall v \in V : \pi_v = \sum_{v' \in V} \pi_{v'} p((v', v))$$

The fundamental theorem of Markov chains states that any irreducible, finite, and aperiodic Markov chain has the following properties [MR10]:

1. There is a unique stationary distribution π with $\pi_v > 0, \forall v \in V$.
2. Every initial distribution π_0 converges against the unique stationary distribution π over time.

Given a maze and transition probabilities (for all non-blocked fields), it is straightforward to derive the corresponding Markov chain. Simply convert the maze into a graph (walking against a wall/block creates a loop edge) and then label the edges according to the given probabilities. A formal description of the construction is left out, as it is obvious. As we are only interested in random walks on solvable mazes, we restrict the Markov chain in this case to the strongly connected component that contains the start and goal state. We call the Markov chain of such a maze a *Markov maze*. You can see an example of a maze and its corresponding Markov maze in Figure 2.4.

Claim. *A Markov maze with probabilities* $p_u, p_d, p_l, p_r > 0$*, which are the probabilities of moving up, down, left, and right respectively, has a unique stationary distribution every initial distribution converges to.*

Proof. To prove the claim we only have to show that the Markov maze is irreducible, finite and aperiodic. We can then apply the fundamental theorem of Markov chains.

As we reduce the Markov maze to the strongly connected component of the start and goal state – which are in the same component due to the maze being solvable – it is *irreducible*.

Furthermore, the Markov maze is also *finite* because a maze has a finite number of fields.

It remains to show that a Markov maze is always *aperiodic*. First note that in every arbitrary maze we can reach a field adjacent to a wall (possibly of a blocked field) in at most $\left\lfloor \frac{\min(n,m)}{2} \right\rfloor$ steps. Obviously, we can also return from that field in the same number of steps. It follows that in every maze from every arbitrary position p there always exists a path of length at most $\min(n,m)$, starting and ending in p, that touches a wall. By repeatedly walking in direction of the wall while standing next to it, we can lengthen the path by an arbitrary number of steps. Thus, for every number greater than or equal to $\min(n,m)$ there exists a circle, starting and ending in p, with that length. As all the probabilities p_u, p_d, p_l, p_r are greater than zero, also the probability of taking such a path is greater than zero. Thus, Markov mazes are aperiodic as $\{\min(n,m) + i \mid i \geq 0\} \not\subseteq \{a + Ti \mid i \geq 0\}$ for any $a > 0, T > 1$.

□

3 Related Problems

In this section we show the relation of SIMASOP and ASIMASOP to other problems of theoretical computer science. Even though all of them have been thoroughly investigated, none of them completely solves our problems. Still, they provide important components for algorithms and proofs later in this thesis.

3.1 The Shortest Path Problem

Both problems, SIMASOP and ASIMASOP, can be seen as a shortest path search on the Cartesian product of the graphs of all the mazes in $\mathcal{M}$. This means that the state space consists of tuples of positions in the single mazes:

$$(p_1, \ldots, p_{|\mathcal{M}|})$$

As we start in position $(1, 1)$ in every maze, the starting state of our search is:

$$\underbrace{((1,1), \ldots, (1,1))}_{|\mathcal{M}|}$$

However, contrary to a Perfect Solving Sequence, we do not have to end up in the goal state for every maze after executing a Solving Sequence. Thus, we have to somehow modify our graph to be able to define a correct goal state for our search. By adding a loop edge for every possible outgoing move from the goal state for every maze, we ensure that we stay in the goal field after once visiting it in a maze. After performing this change to the search graph, we can define the goal state as:

$$\underbrace{((n,m), \ldots, (n,m))}_{|\mathcal{M}|}$$

Every shortest path in this graph from the defined start state to the defined goal state induces a shortest Solving Sequence for $\mathcal{M}$.

While the relation between the problems means we could use the standard machinery of solving the shortest path problem, unfortunately the state space becomes too large very

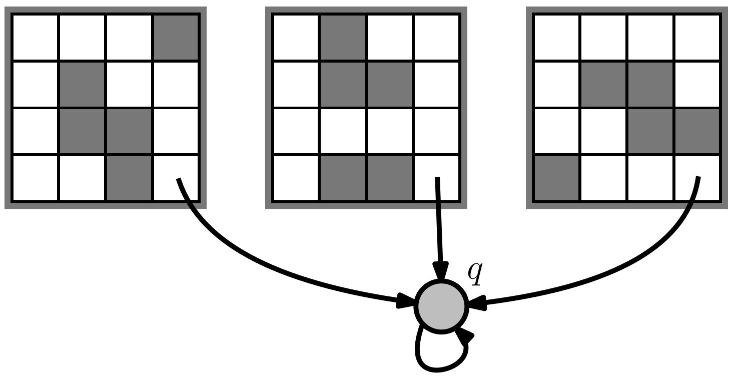

Figure 3.1: Schema of the graph for that we want to find a Synchronizing Sequence.

quickly such that this is only feasible for a very small set of mazes. In Section 5.1 we have a closer look at the relation between the topic of the thesis and the Shortest Path Problem.

3.2 Synchronizing Sequence

We can also formulate SIMASOP and ASIMASOP as problems of finding a Synchronizing Sequence [Hen64] for a large graph. A Synchronizing Sequence of a labeled graph implies a traversal that ends up in one and the same state for every starting state. That means, it synchronizes the state independent of the current state such that after executing the sequence, we know exactly which state we are in. The graph for which we want to find a Synchronizing Sequence is the union of all the mazes of $\mathcal{M}$. Also, we add one further state q with which all the goal states are connected with all labels. The state q then also has a loop edge to itself; we therefore stay in this state if we enter it once. See Figure 3.1 for a schematic representation. If we now find a Synchronizing Sequence for this graph, we know that with this sequence we always end up in q as this is the only state reachable from all the other states. One step before reaching q, we must have entered the goal state of a maze. Therefore, this Synchronizing Sequence *without the last symbol* is a Solving Sequence for the mazes in $\mathcal{M}$ as all states are considered as starting states, in particular the upper-left corner of every maze. Note, thought, that a shortest Synchronizing Sequence might be longer than a shortest Solving Sequence.

Property	UTS	Solving Sequence
start	arbitrary	upper-left
goal	visit all nodes	visit lower-right
graph structure	arbitrary connected d-regular graph	grid maze
d	arbitrary	4
labeling	arbitrary	consistent with the grid

Table 3.1: This table shows how the problem of finding a UTS is more general than finding a Solving Sequence.

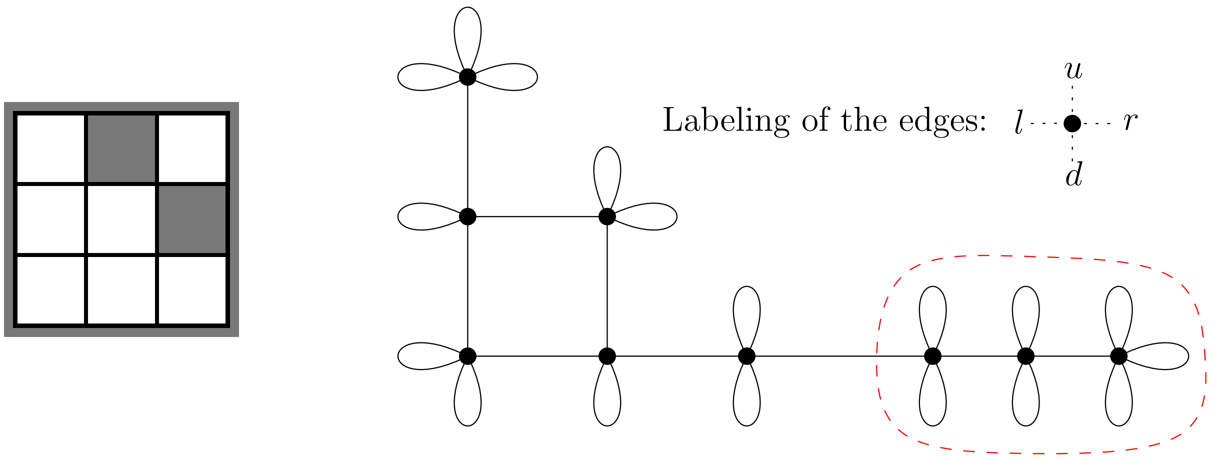

Figure 3.2: A maze with its corresponding UTS graph. The three red-circled nodes correspond to the two blocked fields and the one field that is not in the same component as start and goal (i. e., field $(1, 3)$).

3.3 Universal Traversal Sequence

Universal Traversal Sequences are a concept first researched in the context of the reachability problem in graphs [AKL+79]. A Universal Traversal Sequence (UTS) is a sequence of labels that implies a complete traversal of every connected and arbitrarily labeled graph with a certain number of nodes and a certain fixed degree. More formally, s is a UTS if we traverse every undirected, labeled, d-regular graph with n nodes from every arbitrary starting node completely. A d-regular graph is called *labeled* if all adjacent edges of a node v get assigned a unique label of the set $\{0, \ldots, d-1\}$. Note that this labeling doesn't have to be consistent, i. e., for the edge vw, the node v might label the edge differently than the node w. The problem of finding a UTS is more general than the problem of finding a Solving Sequence.

We can show that every UTS is also a Solving Sequence for certain parameters. In Table 3.1 you can see a comparison of the two concepts. More concrete, every UTS for graphs of size $n \cdot m$ and for fixed degree $d = 4$ is also a Solving Sequence for the set of all solvable $n \times m$ mazes. To show this we just have to convert an arbitrary solvable maze into a 4-regular graph with $n \cdot m$ nodes which is *connected*, such that a complete traversal of the resulting graph would imply solving the initial maze. While the size and the regularity of the converted graph are straight-forward (we just keep the nodes and at walls we insert a loop edge which is only labeled from one side), we still have to make sure that the converted graph is connected. Consider the node corresponding to a blocked field: If we simply insert loop edges at all nodes that correspond to adjacent fields, then it is not reachable anymore. Thus, we have to somehow connect it to the graph without creating new paths in the maze (as this might change the possible Solving Sequences). The same issue arises for fields that are not in the connected component of the start and goal field. We solve this by simply appending all nodes corresponding to blocked or non-reachable fields to a chain, which is then connected to the node that corresponds to the goal field. For the up and down movements we insert loop edges for all such nodes. The last node in the chain has an extra loop edge for the right move. See Figure 3.2 for an example. Thus, we can only enter those nodes if we already (implicitly) solved the maze and therefore a complete traversal of the graph also implies solving the initial maze.

An important result shown in [AKL+79] is that there always exists a UTS of polynomial length. The proof leverages the probabilistic method. As, to our knowledge, there unfortunately does not exist a derandomization, the proof cannot be used to explicitly construct such a sequence. However, they not only show that there exists a UTS of length $\mathcal{O}\left(|V|^3 \log(|V|)\right)$, but also that a random sequence of this length is a UTS with high probability.

4 Theoretical Analysis

In this section we analyze different theoretical aspects of SIMASOP as well as ASIMASOP. Results regarding solvability and complexity of the two problems are presented. Additionally, we show upper and lower bounds on the length of a shortest Solving Sequence. While all the already mentioned results are w. r. t. Solving Sequences, we also show different results on the number of solvable mazes.

4.1 SIMASOP is NP-complete

Recall that the decision problem of SIMASOP, as already stated in Section 2.3, is defined as:

Given a tuple $(\mathcal{M}, k)$ where $\mathcal{M}$ is a set of mazes and $k \in \mathbb{N}$. Then

$$(\mathcal{M}, k) \in \text{SIMASOP} \Leftrightarrow \exists s, |s| \leq k : s \text{ solves all } M \in \mathcal{M}$$

To show NP-completeness it has to be shown that the problem is contained in the complexity class NP and that it is also NP-hard. Let us first consider SIMASOP $\in$ NP. By using the "guess and check" method, we can check if there exists a Solving Sequence shorter than k nondeterministically in polynomial time: we simply guess a sequence $s, |s| \leq k$ and then verify if it really is a Solving Sequence by naively checking this property for every maze in $\mathcal{M}$. Note that the mazes are part of the input and thus contribute to the input size. This is not the case for ASIMASOP where the input is only n and m, and therefore we cannot prove that ASIMASOP $\in$ NP using the same argument.

Now it remains to show NP-hardness for SIMASOP. This is done by providing a polynomial time reduction from CNFSAT.

Definition (CNFSAT). *Given a Boolean formula F in conjunctive normal form (CNF), i. e., $F = \bigwedge_{i=1}^{c} \bigvee_{j=1}^{c_i} A_{i,j}$. The decision problem CNFSAT asks if there exists a satisfying assignment for F.*

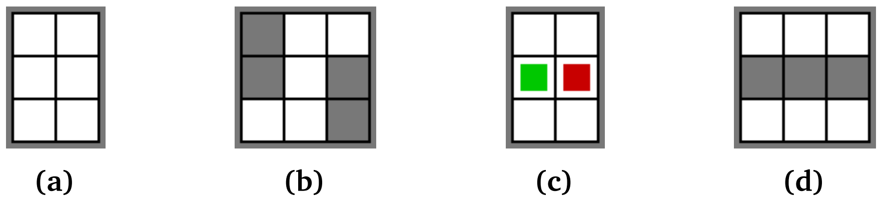

Figure 4.1: The different parts out of which we build the mazes created by the reduction function.

Given a formula in CNF

$$F = \bigwedge_{i=1}^{c} \bigvee_{j=1}^{c_i} A_{i,j}$$

we define the function f that realizes the reduction as

$$f(F) = (\{M\} \cup \{M_i | 1 \leq i \leq c\}, 9n - 6)$$

where n is the number of *distinct* variables $X_1, \ldots, X_n$ in F and M and M_i are mazes as described in the following.

First we describe the high-level idea of the reduction and then provide the details. The maze M is a special maze that somewhat fixes the path that must be taken through all the other mazes. The length of its shortest path from the upper-left corner to the lower-right corner is $k = 9n - 6$. From the fields we visit during the traversal of M, we deduce the assignment of the variables of F. Every clause C_i in F corresponds to one maze M_i whose structure depends on the variables contained in C_i. The length of the mazes only depends on n – the number of distinct variables in F. The idea of the M_i is that we stay in the first row of the maze as long as we haven't found a literal that satisfies the clause C_i and switch to the last row as soon as we find one.

Let us now have a look at the details of the structure of the mazes M and M_i starting with the former.

The maze M consist of an empty 3×2 block for every variable X_i in F; see Figure 4.1a. These blocks appear in order of the index of the variable and in between two of those blocks is one 3×3 block as shown in Figure 4.1b. The mazes M_i have a similar structure. They consist of a block like shown in Figure 4.1c for every variable X_j, where the green field is *not* blocked if and only if X_j is contained in the clause C_i while the red field is *not* blocked if and only if $\neg X_j$ is contained in the clause C_i. In between those blocks there is also a 3×3 block like the one shown in Figure 4.1d.

Consider for example the formula $F = (A \vee B) \wedge (\neg B \vee C) \wedge (A \vee \neg A \vee \neg C) \wedge (D)$. The mazes contained in $f(F)$ can be seen in Figure 4.2.

It remains to show that f is indeed a reduction.

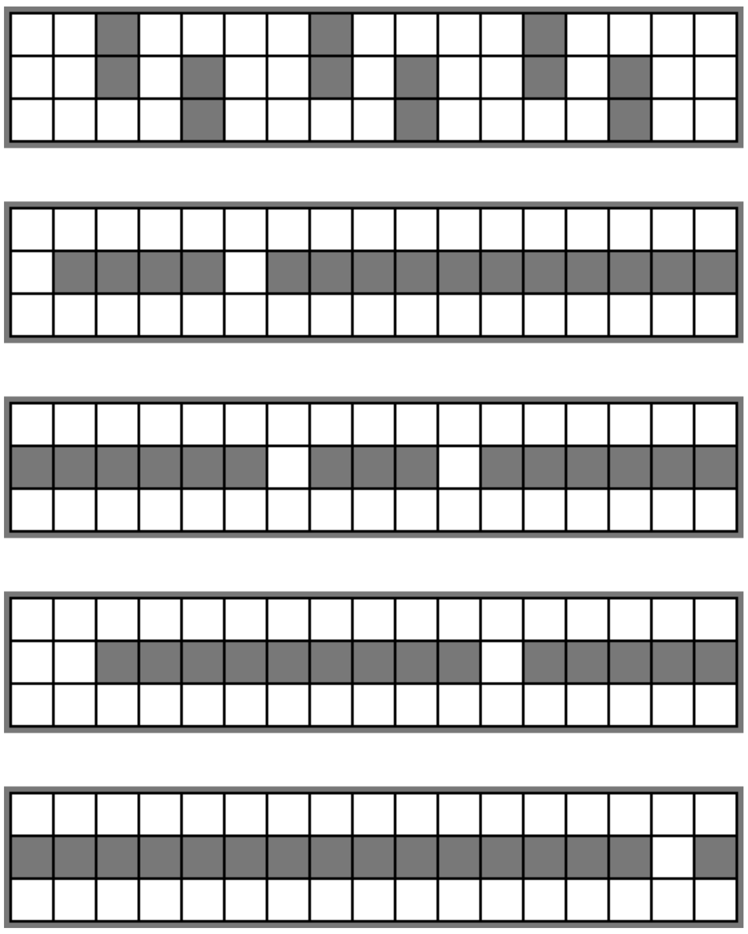

Figure 4.2: The mazes resulting from $f((A \vee B) \wedge (\neg B \vee C) \wedge (A \vee \neg A \vee \neg C) \wedge (D))$.

Claim. $F \in$ *CNFSAT* $\Leftrightarrow f(F) \in$ SIMASOP.

Proof. Let us first show that $F \in$ CNFSAT $\Rightarrow f(F) \in$ SIMASOP. Assume $F \in$ CNFSAT. It follows that there exists an assignment $\mathcal{A}$ under which F evaluates to *true*. From the assignment $\mathcal{A}$ we construct a Solving Sequence s as shown in Algorithm 4.1. Firstly, no matter how the if-statements evaluate, the returned sequence is a *shortest* Solving Sequence for the "path-forming" maze M of the reduction. That means, if we can show that the algorithm produces a Solving Sequence, we also know that it is a *shortest* Solving Sequence for all the mazes in $\mathcal{M}$. Now, note that the *up* movements in s never have an effect on the position in one of the M_i. Thus, we can only consider the *right* and *down* movements for the traversals of the M_i. As we never hit a wall during such a traversal (we are either in the first or last row where there are no obstacles when we move *right*), we are always in the same column in M and the M_i. Consider now the variable X_j – which might be negated – with the smallest index j that make the clause C_k evaluate to *true*. Let s' be the sequence that we construct until the loop variable in

Algorithm 4.1 is set to j. When executing s' in M_k we still reside in the first row of the maze. When now executing the moves that are appended when $i = j$ in the loop, we make a transition from the first row to the third row in the maze. That is because, if X_j occurs as a positive literal in C_k, there is an unblocked field in the second row right under the position we are currently residing at; when then executing the moves ddr we are not blocked. A similar argument holds if X_j occurs as a negative literal. As we are always in the same column in M and M_i and M is solved, we end up in the lower-right corner in all the M_i. Therefore, the s computed in Algorithm 4.1 is a Solving Sequence.

For the second half of the proof we have to show that $f(F) \in$ SIMASOP $\Rightarrow F \in$ CNFSAT. For this assume $f(F) \in$ SIMASOP and let $(\mathcal{M}, k) := f(F)$. Let $n \times m$ be size of the mazes in $\mathcal{M}$. Furthermore, let $s, |s| \leq k$ be the sequence that solves $\mathcal{M}$. By construction of the reduction $|s| = k$. Thus, s is a shortest Solving Sequence of M and therefore the number of *right* moves in s equals m, i. e., $|s|_r = m$. As s solves all the mazes in $\mathcal{M}$, this implies that we never walk against a wall or a blocked field in any of the traversals of M and M_i. Therefore, it again holds that the *up* moves in s only change the position in M but in none of the M_i as we always are in the same column in all the mazes. Now, as s solves all the mazes in $\mathcal{M}$, we have to cross the second row at some point of the traversal. The crossings of the middle row imply an assignment that satisfies F. Reconsider Figure 4.1c which shows one of the maze blocks that is inserted for every variable in every maze M_i. We have to cross the middle row in one of those blocks. Let this block be the one corresponding to variable X_j. If we enter both blocks, then every clause has to either contain X_j not at all or in positive and negative form; else the traversal would be blocked on one *right* movement. Thus, we can assume that the same field is entered for all M_i that cross the middle row at this columns. If it is the green field of Figure 4.1c, then we set $X_j = 1$, else we set $X_j = 0$. Thus, all the clauses corresponding to the mazes that cross the middle row in those moves evaluate to *true*. As all the traversals of the mazes M_i cross the middle row, all clauses are satisfied. Therefore, F is satisfied by the derived assignment.

$\square$

4.2 ASIMASOP $\in$ PSPACE

As we already showed in Section 3.3, a Universal Traversal Sequence (UTS) is a more general concept of a Solving Sequence for ASIMASOP. We showed that a UTS for graphs with $n \cdot m$ nodes which all have degree 4 is always a Solving Sequence for all mazes of size $n \times m$. In [AKL+79] the authors prove that there always exists a UTS with length polynomial in the number of nodes. It follows that the shortest Solving Sequence is also always of polynomial length in n and m. Simply calculating the shortest Solving

Algorithm 4.1 Construct a Solving Sequence from a satisfying assignment $\mathcal{A}$

Input: $\mathcal{A}$ = satisfying assignment for F
Output: s = Solving Sequence for $\mathcal{M}$

```
1:  if $\mathcal{A}(X_1) = 1$ then
2:      $s = ddr$
3:  else
4:      $s = rdd$
5:  end if
6:
7:  for $i \in \{2, \dots, |\text{Vars}(F)|\}$ do
8:      $s = s \cdot rruurr$
9:      if $\mathcal{A}(X_i) = 1$ then
10:         $s = s \cdot ddr$
11:     else
12:         $s = s \cdot rdd$
13:     end if
14: end for
15: return $s$
```

Sequence by brute-force is therefore possible in polynomial space. We only have to hold the current sequence and the maze we are currently checking in memory. This implies ASIMASOP $\in$ PSPACE.

In Section 4.8 we additionally show that the bound given in [AKL+79] can be improved by adapting the proof to our setting.

4.3 Existence of a Solving Sequence

While the existence of a Solving Sequence for every set of solvable mazes follows directly from the existence of a Universal Traversal Sequence, it unfortunately does not provide an algorithm to construct such a sequence. Also, to our knowledge, the proof has not been derandomized until now as already stated in Section 3.3. Because of this, we present a deterministic algorithm to construct a Solving Sequence for an arbitrary set of solvable mazes. The same algorithm already exists as the solution to similarly formulated riddles from [XKCa], [Puz], and other sources.

More formally described we now show the following: for any finite set of solvable mazes $\mathcal{M}$, there exists a sequence of moves s such that this sequence solves all mazes in $\mathcal{M}$. We give a proof in the form of an algorithm. The trick is to simply solve the mazes of

Algorithm 4.2 Find a Solving Sequence for a set of mazes $\mathcal{M}$

Input: $\mathcal{M}$ = set of mazes we want to solve
Output: s_{sol} = Solving Sequence for $\mathcal{M}$

```
 1: $s$ = empty sequence
 2: while $\mathcal{M} \neq \emptyset$ do
 3:     $M$ = arbitrary element of $\mathcal{M}$
 4:     $\mathcal{M} = \mathcal{M} \setminus \{M\}$
 5:
 6:     $pos$ = position in $M$ after executing $s$
 7:     $s'$ = shortest path from $pos$ to the goal field of $M$
 8:     $s = s \cdot s'$
 9: end while
10: return $s$
```

$\mathcal{M}$ in order. That means, we solve the first maze, then execute the Solving Sequence in the second maze and append moves to create a Solving Sequence for the second maze, then execute this sequence in the third maze, and so on. Note, we always append to the sequence. This preserves the Solving Sequence property. Therefore, by sweeping over the mazes in the set and solving them like that, we are guaranteed to have a Solving Sequence in the end. For pseudocode of the just described algorithm see Algorithm 4.2.

Let us apply the presented algorithm to a small example, which is shown in Figure 4.3. There, we solve four mazes of size 3×3 in order from left to right.

In Figure 4.4 three mazes are depicted. When constructing a Solving Sequence for those mazes using Algorithm 4.2, processing the mazes from left to right as depicted in the figure, we get the Solving Sequence $rrdrddurrrdullllddrdrr$ with length 21. The shortest Solving Sequence for those mazes has length 12 ($ddruurrdrddr$). This example shows that Solving Sequences constructed with Algorithm 4.2 might become much longer than the shortest Solving Sequence. The best theoretical upper bound on the length of the Solving Sequence produced by this algorithm is $\mathcal{O}(|\mathcal{M}| \cdot nm)$; for every maze in the set we might have to traverse almost the whole maze when it is solved.

4.4 Solving Order

The proof of the existence of a Solving Sequence is constructive and contains an algorithm to find such a sequence. As there are a lot of different solvable mazes for a certain size, one might assume that there even always exists and ordering of the mazes such

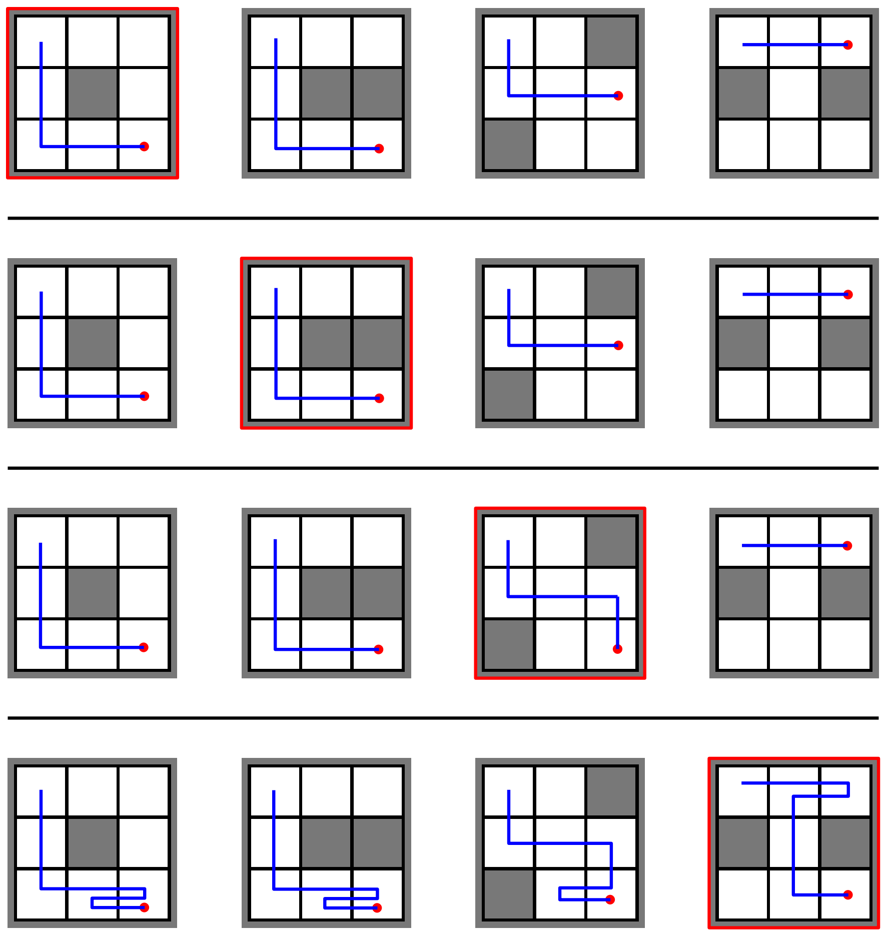

Figure 4.3: An example of the execution of Algorithm 4.2. Every row represents one pass through the while loop. The red-marked maze is the current M of the loop. The trajectories of the newly assigned s (at the end of the loop) are depicted in blue.

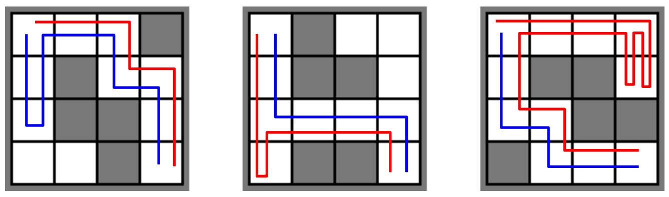

Figure 4.4: An example where the constructed Solving Sequence becomes very long when using Algorithm 4.2. In blue we show the trajectories of the shortest Solving Sequence and in red the trajectories of the sequence produced by Algorithm 4.2. Steps that are made after visiting the goal field are not shown due to clarity of the figure.

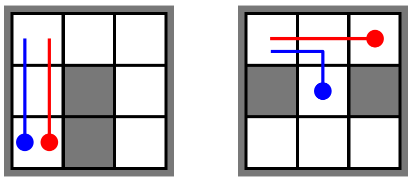

Figure 4.5: The two mazes used in the proof that there is not always an order of mazes that induces a shortest Solving Sequence. The red trajectories result from executing $ddrr$ or $drrd$. The blue ones from executing $drdr$.

that solving them in this order results in a shortest Solving Sequence. Unfortunately, this is not the case in general as we show in the remainder of this section.

Claim. *There does not always exist a permutation $M_{\pi(1)}, \dots, M_{\pi(k)}$ of the mazes $\mathcal{M} = M_1, \dots, M_k$ such that solving in this order (i. e., according to Algorithm 4.2) results in a shortest Solving Sequence.*

Proof. We will show the claim for $\mathcal{M} = \mathcal{M}_{3,3,}$. It follows that the claim holds for SIMASOP as well as ASIMASOP. The high-level idea of the proof is to show that the prefix of a shortest Solving Sequence is never a shortest Solving Sequence of a maze; and thus, no maze in $\mathcal{M}_{3,3}$ can be $M_{\pi(1)}$ in the ordering.

All solvable mazes of size 3×3 *only* have shortest Solving Sequences s of the form:

$$|s| = 4, |s|_d = 2, |s|_r = 2$$

That means, they consist of exactly two *down* and two *right* moves. Therefore, all the possible prefixes of a sequence calculated with Algorithm 4.2 are:

$$ddrr, drdr, drrd, rddr, rdrd, rrdd$$

For now, let us only consider the three sequences starting with a *down* move. When we execute those sequences on the mazes depicted in Figure 4.5, we end up in the positions marked red for $ddrr$ and $drrd$, and those marked blue for $drdr$. The shortest Solving Sequence starting at the red positions has length 9, while for the blue positions it has length 8. This can easily be verified by a shortest path query or even by hand. Because

$$4 + 9 > 11 \text{ and } 4 + 8 > 11$$

and 11 is the length of the shortest Solving Sequence for $\mathcal{M}_{3,3}$, which is shown in Chapter 6, none of those three sequences can be the start of a shortest Solving Sequence of $\mathcal{M}_{3,3}$. By transposing the mazes in Figure 4.5 (remember, a maze is simply a matrix), we get the same result for the sequences beginning with a *right* move. □

4.5 Existence of a Perfect Solving Sequence

Claim. *There exists a Perfect Solving Sequence for every set of solvable mazes $\mathcal{M}$.*

To my knowledge, the following beautiful proof has first appeared in the XKCD forums [XKCb], but has never been published in an academic context. We reworked the proof and present it here with significantly more details and also make the calculated results tighter. The proof leverages the probabilistic method. The probabilistic method randomizes the creation of an object to then show that the probability of an object with the desired property is larger than zero. This implies the existence of such an object as there exists a series of decision in the random process with that outcome – else, the probability wouldn't be larger than zero.

Proof. Let the set of mazes $\mathcal{M}$ be given; all of the contained mazes $M \in \mathcal{M}$ have size $n \times m, n, m \geq 2$. For $n < 2$ or $m < 2$ the claim is obviously true. The idea of the proof is to create a random sequence according to the probabilities p_u, p_d, p_l, p_r which are the probabilities of moving up, down, left, and right, respectively. This can be seen as a simultaneous traversal of all the Markov mazes corresponding to the mazes in $\mathcal{M}$, as defined in Section 2.6. By choosing the length of the random sequence large enough, we can show – because of convergence to the stationary distribution – that the probability of being in the goal state in all mazes $M \in \mathcal{M}$ is larger than $\frac{3}{4}$. Thus, there exists a sequence with the property of being a Perfect Solving Sequence. Let us now conduct the proof in all formal details.

Let the probabilities be given as $p_u = p_l = e$, $p_d = p_r = \frac{1}{2} - e$, $e = \left(\frac{1}{4}\right)^{nm} \leq \frac{1}{4}$. We claim that the stationary distribution of *every* Markov maze corresponding to a maze in $\mathcal{M}$ is given by

$$p_{i,j} \propto \left(\frac{\frac{1}{2} - e}{e}\right)^{i+j}$$

where $p_{i,j}$ denotes the probability of being in position (i, j) if the field is reachable from the starting field. Note that this distribution does not depend on any structure of the maze except the reachability of the fields from the starting state – which is incorporated in the proportionality.

Let us now show that this distribution is indeed the stationary distribution. We have to show that it fulfills the two properties in Definition 2.6. The first one – that the probabilities sum up to 1 – is trivially fulfilled due to the proportionality. For the second property – that the probabilities stay the same after one step – we have to show two things. Firstly, that it does not matter for the calculation that a field is next to a wall or a free field. And secondly, that the sum evaluates to the claimed value.

1. The first equation shows that the sum's value does not change if there is a wall or a free field above or to the left. The second equation shows the same for fields below and to the left.

$$p_{i,j-1}\cdot p_r = p_{i-1,j}\cdot p_d = \left(\frac{\frac{1}{2}-e}{e}\right)^{i+j-1}\cdot\left(\frac{1}{2}-e\right) = \left(\frac{\frac{1}{2}-e}{e}\right)^{i+j}\cdot e = p_{i,j}\cdot p_u = p_{i,j}\cdot p_l$$

$$p_{i,j+1}\cdot p_l = p_{i+1,j}\cdot p_u = \left(\frac{\frac{1}{2}-e}{e}\right)^{i+j+1}\cdot e = \left(\frac{\frac{1}{2}-e}{e}\right)^{i+j}\cdot\left(\frac{1}{2}-e\right) = p_{i,j}\cdot p_d = p_{i,j}\cdot p_r$$

2. Now show the stationarity. We use $v_{i,j}$ to denote the node of the Markov maze corresponding to position (i,j) of the maze.

$$\begin{aligned}\sum_{v_{k,l}\in V} p_{k,l}p((v_{k,l},v_{i,j})) &= 2\left(\frac{\frac{1}{2}-e}{e}\right)^{i+j}\cdot e + 2\left(\frac{\frac{1}{2}-e}{e}\right)^{i+j}\cdot\left(\frac{1}{2}-e\right)\\ &= \left(2e+2\left(\frac{1}{2}-e\right)\right)\cdot\left(\frac{\frac{1}{2}-e}{e}\right)^{i+j} = p_{i,j}\end{aligned}$$

From the fact that the probability of being in the goal state is not equal to 1, we can derive that: in the stationary distribution the (non-proportional) probability to be in a certain state that is not the goal state is less than $4e$:

$$\left(\frac{\frac{1}{2}-e}{e}\right)^{n+m} < 1 \Leftrightarrow \left(\frac{\frac{1}{2}-e}{e}\right)^{n+m-1} < \frac{e}{\frac{1}{2}-e}$$

and

$$\frac{e}{\frac{1}{2}-e} \le \frac{e}{\frac{1}{2}-\frac{1}{4}} \le 4e$$

As shown in Section 2.6, every initial distribution will converge to the stationary distribution. Thus, for a long enough sequence it holds for every maze that

$$P(\text{not in goal state}) < 4e(nm-1) = 4\left(\frac{1}{4}\right)^{nm}(nm-1)$$

where $(nm-1)$ is the maximal number of reachable fields other than the goal. The equality results from plugging in e. It then follows with Boole's inequality:

$$\begin{aligned}P(\text{not all in goal state}) &\le \sum_{M\in\mathcal{M}} 4\left(\frac{1}{4}\right)^{nm}(nm-1)\\ &\le 2^{nm-2}\cdot 4\left(\frac{1}{4}\right)^{nm}(nm-1)\\ &= \left(\frac{1}{2}\right)^{nm}(nm-1)\\ &\le \frac{1}{4}\end{aligned}$$

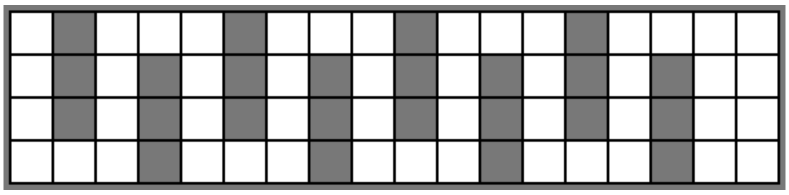
Figure 4.6: A zigzag maze with a long shortest Solving Sequence.

The second inequality holds as there are less than 2^{nm-2} solvable mazes of size $n \times m$. It follows that

$$P(\text{all in goal state}) = 1 - P(\text{not all in goal state}) \geq 1 - \frac{1}{4} = \frac{3}{4} > 0$$

□

4.6 Lower Bounds

It seems surprisingly hard to find non-naive lower bounds for the length of the shortest Solving Sequence for a given set of mazes $\mathcal{M}$ – also for $\mathcal{M}_{n,m}$.

For ASIMASOP we can easily see that the length of Solving Sequences for smaller sizes are lower bounds for the length of Solving Sequences for larger sizes, i. e. $\sigma(n,m) < \sigma(n+1,m)$ and $\sigma(n,m) < \sigma(n,m+1)$. This holds as the smaller mazes are contained as sub-mazes in the larger mazes and so already a subset of the larger mazes implies this lower bound. The relation is strict as we have to take at least one additional step to reach the new goal field.

Another rather naive lower bound is the length of the longest shortest path of a maze in $\mathcal{M}$. If $\mathcal{M}$ is individually given as in SIMASOP, this number depends on the specific instance. However, if we consider ASIMASOP, then we can easily give a lower bound on that number. By blocking the right fields we can always create a zigzag path as shortest Solving Sequence which goes all to the bottom, two steps to the right, all to the top, two steps to the right, and so on. The exact length of such a zigzag path is

$$\left(2\left\lfloor\frac{m-1}{4}\right\rfloor+1\right)(n+1)+((m-1 \mod 4)-2)$$

which can be verified easily; so, approximately $\frac{nm}{2}$. See Figure 4.6 for an example of such a zigzag maze.

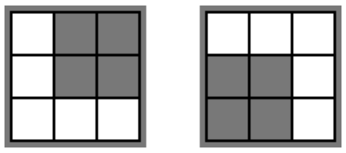

Figure 4.7: Two example mazes that imply a lower bound of length 6 for the shortest Solving Sequence of all solvable mazes of size 3×3.

n	1	2	3	4	5	6	7
#solvable mazes	1	3	51	3828	1225194	1636193228	9009490924794

Table 4.1: Number of solvable mazes of size $n \times n$ according to [The].

A set of mazes $\mathcal{M}_1$ for which the shortest Solving Sequence s is known can also be used to give a lower bound on the length of the shortest Solving Sequence for a set of mazes $\mathcal{M}_2$ with $\mathcal{M}_1 \subseteq \mathcal{M}_2$. This is because additional mazes will only make the Solving Sequence longer as more constraints are added to the minimization problem.

Consider, for example, the two mazes in Figure 4.7. Their shortest Solving Sequence has length 6 (one of the shortest Solving Sequences is $rrddrr$). This implies that any All Solving Sequence for size 3×3 mazes has at least length 6.

4.7 Number of Solvable Mazes

We calculate the number of solvable mazes by enumerating them. Because the number of solvable mazes of size $n \times n$ grows exponentially in n, as shown in Section 2.5, we have to enumerate mazes implicitly for our method to work on larger n. This, however, does not seem to be an easy task. We will describe two algorithms in the remainder of this section: an algorithm that calculates the exact number of solvable mazes, and an algorithm that calculates a lower bound for this number. An algorithm calculating a lower bound is necessary because of the exponential runtime of the algorithm that calculates the exact number of mazes. The exact number of solvable mazes has already been calculated before and is part of the On-Line Encyclopedia of Integer Sequences [The]. See Figure 4.1 for the sequence of the number of solvable $n \times n$ mazes. While the numbers are listed there, no explicit formula or algorithm to calculate them is given and they are also only calculated until $n = 7$. This suggests that the author did not find an efficient algorithm. We found an algorithm of similar efficiency and could verify the number of solvable mazes until 6×6 with a python implementation. A more efficient C++ implementation would have most probably allowed for the calculation of the number of solvable mazes up to size 7×7.

Exact Number of Solvable Mazes

To calculate the exact number of solvable mazes we enumerate the mazes via their *solving right-hand path*. The solving right-hand path of a maze is the path we get when solving the maze while always keeping a wall at our right-hand side, starting in the upper-left corner. This technique is guaranteed to produce a Solving Sequence for a maze as long as the maze is solvable. In Figure 4.8 the blue trajectories are all solving right-hand paths.

Note that every solvable maze has exactly one solving right-hand path. However, some solvable mazes have the same solving right-hand path. So, instead of enumerating the mazes themselves, we enumerate all the possible solving right-hand paths. Then we calculate the number of solvable mazes for every solving right-hand path and add them up. This obviously calculates the correct number of solvable mazes as we consider every solvable maze exactly once.

We still need to specify the details of the algorithm. We calculate all possible solving right-hand paths using a simple tree search. Every node of the tree contains a maze and a position in this maze. The root of the tree contains an empty maze with the current position being the start field. For every possible move in the maze of the parent node a child node node is created. A child is pruned if the move contradicts a solving right-hand path. Note that by performing the moves, we might have to set fields to be blocked or free for the right-hand path property to be preserved. A solving right-hand path is found when the goal field is reached. How do we now calculate the number of mazes for a given solving right-hand path? This is done by counting the fields in the maze which can still be arbitrarily set. There are two types of fields which cannot be arbitrarily set: the fields on the solving right-hand path (they have to be free) and the fields that define the solving right-hand path (they have to be blocked). A blocked field is said to define a solving right-hand path if the solving right-hand path changes if the field is not blocked. Let the number of fields that can still be arbitrarily set be a. Then the number of different mazes with a certain solving right-hand path is 2^a, i. e. all possible assignments of those fields. See Figure 4.8 for an example.

Lower Bound on the Number of Solvable Mazes

To compute a lower bound on the number of solvable mazes, we restrict ourselves to specifically shaped solving right-hand paths: those only containing *right* and *down* movements. Additionally, we assume all the fields below the path to be blocked, not only those that define the solving right-hand path. This means, we partition the maze into three components: a path, blocked fields which are below the path, and fields above

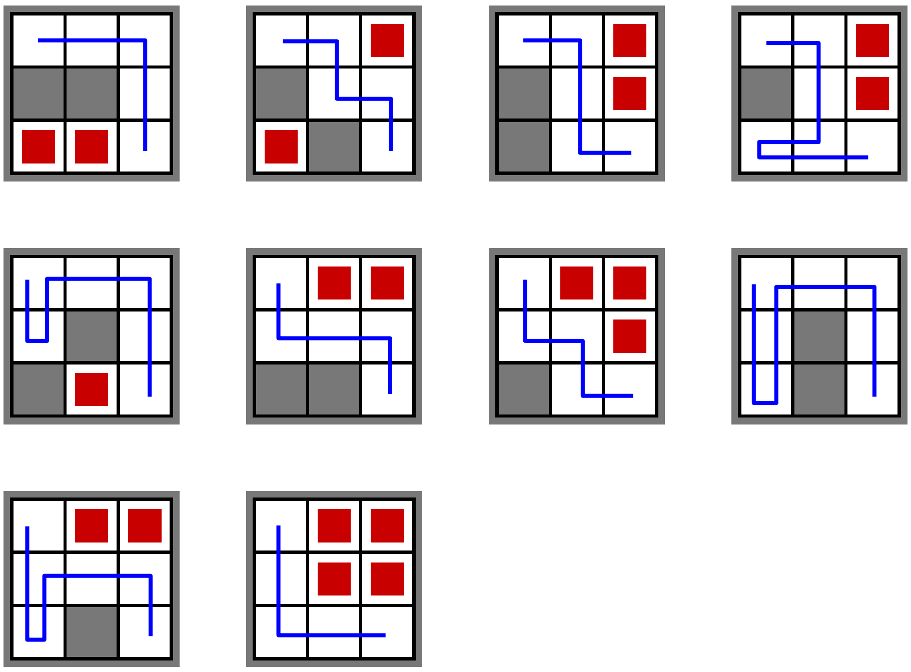

Figure 4.8: All the possible right hand paths in a 3×3 maze. The red-marked fields can be set arbitrarily as they do not change the right hand path and are not on it either. Thus, the number of solvable 3×3 mazes is: $2^2 + 2^2 + 2^2 + 2^2 + 2^1 + 2^2 + 2^3 + 2^0 + 2^2 + 2^4 = 51$. Every of the above mazes contributes one summand.

the path that can be arbitrarily set. We then calculate the lower bound in the same manner as we calculate the exact number of solvable mazes, i. e., by summing up all the possibilities.

Let z_s be the number of mazes that have s fields that can be arbitrarily set. If we can compute z_s efficiently, then we can calculate the lower bound via:

$$\sum_{s=0}^{(n-1)(m-1)} z_s \cdot 2^s$$

But how do we calculate z_s efficiently? This is done using the following recursive function (which is similar to the recursive function of the *partition numbers*):

$$S(t,r,k) = \begin{cases} 0, & \text{if } (t > 0 \text{ and } r = 0) \text{ or } t < 0 \text{ or } r < 0 \text{ or } k < 0 \\ 1, & \text{if } t = 0 \text{ and } r = 0 \\ \sum_{i=0}^{r} S(t-r,i,k-1), & \text{otherwise} \end{cases}$$

with $t, r, k \in \mathbb{Z}$. We claim that

$$z_s = S(s+n-1, n-1, m)$$

In the remainder of this section we argue why this is the case. The interpretation of S is:

$$S(t,r,k) = \left|\left\{\{s_1,\dots,s_r\}\middle|\sum_{i=1}^{r} s_i = t \text{ and } \forall i \in \{1,\dots,r\} : 0 < s_i \leq k\right\}\right|$$

That means, it is the number of (unordered) sums equaling t with r summands that are greater than zero and less than or equal to k each. The interpretation of the summands is the number of fields in the respective rows that can be arbitrarily assigned. The number of fields that can be arbitrarily assigned for row i is always less than or equal to the same number for the rows $1,\dots,i-1$. This is due to the step-like structure of the path that uses only *down* and *right* moves. From this, it follows that we only need to consider unordered sums as the summands can be interpreted as being "implicitly sorted". Now, consider again the definition of z_s. We have $n-1$ summands as we have $n-1$ rows that can contain fields that can be arbitrarily assigned (all but the last row). One would assume k to be m instead of $m-1$ as the maximal number of fields that can be arbitrarily assigned is $m-1$ per row. However, a row might contain no field that can be arbitrarily assigned, but we only considered summands greater than zero. Therefore, we want to decrease every summand by one such that we can also have zeros in our sum. Consequently, we have to increase the maximal summand (i. e., k) by one. Using this interpretation, we are now also searching for a larger sum as we (implicitly) decreased every summand by one; thus, we have to add $n-1$ (i. e., the number of summands) to the sum (i. e., t). It follows the definition of z_s.

4.8 Upper Bounds

An upper bound on the length of the Solving Sequence of SIMASOP is already given in Section 4.3:

$$\mathcal{O}(|\mathcal{M}| \cdot nm)$$

This upper bound can be improved for ASIMASOP. In the remainder of this section we prove a polynomial upper bound on the length of the shortest Solving Sequence of ASIMASOP.

Claim. *For every maze size $n \times m$ there exists a sequence of length $\mathcal{O}((nm)^3)$ that is a Solving Sequence of* ASIMASOP.

To prove this we modify a proof that gives an upper bound on the length of a Universal Traversal Sequence. The relationship between ASIMASOP and UTS has already been explained in Section 3.3. The proof that we will use as a basis is Theorem 8 from [AKL+79].

Before we conduct the main proof, we state an important result about random walks that is needed for the proof. The proof of this claim is included in [AKL+79] as Theorem 4 and in [MR10] as Theorem 6.8. We therefore omit it here.

Claim. *Let $G = (V, E), |V| = n, |E| = m$ be an undirected, connected graph and let $T(v, \cdot)$ be the cover time of G starting from $v \in V$, i. e., the expected number of edge traversals by a random walk starting in v until all vertices in G have been visited. Then $T(v, \cdot) \leq 2e(n-1)$.*

Now we can begin with the proof of the upper bound.

Proof. Let s be a random sequence with $|s| = 2 \cdot 4nm(nm-1)(nm+1) \in \mathcal{O}\left((nm)^3\right)$. The moves of s – which can be interpreted as random variables – are chosen independently and identically distributed with all moves being equally likely. Furthermore, let $X_M, M \in \mathcal{M}_{n,m}$ be a random variable such that:

$$X_M = \begin{cases} 0, & \text{if } s \text{ is a Solving Sequence of } M \\ 1, & \text{otherwise} \end{cases}$$

Additionally, let Y be defined as:

$$Y = \sum_{M \in \mathcal{M}_{n,m}} X_M$$

The sequence s is a Solving Sequence for $\mathcal{M}_{n,m}$ iff $Y = 0$. Therefore, if we can show that $E[Y] < 1$, it follows that there exists an assignment of the random variables X_M such that $Y = 0$. Which in turn means that there exists a Solving Sequence of length $|s|$. In the remainder of this proof we focus on showing that indeed $E[Y] < 1$.

Due to the linearity of the expected value we have:

$$E[Y] = E\left[\sum_{M \in \mathcal{M}_{n,m}} X_M\right] = \sum_{M \in \mathcal{M}_{n,m}} E[X_M]$$

We claim that $E[X_M] \leq 2^{-(nm+1)}$. With $|\mathcal{M}_{n,m}| \leq 2^{nm}$ it follows that

$$E[Y] = \sum_{M \in \mathcal{M}_{n,m}} E[X_M] \leq 2^{nm} \cdot 2^{-(nm+1)} = 2^{nm-nm-1} = 2^{-1} < 1$$

Therefore, it only remains to show that indeed $E[X_M] \leq 2^{-(nm+1)}$. For this, let us consider s as a concatenation of $nm+1$ random sequences $s_1, \ldots, s_{nm+1}$, each being of length $2 \cdot 4nm(nm-1)$. Let Z be a random variable that equals the number of transitions we need to visit all reachable fields in the maze, starting from an *arbitrary field*, when executing a random sequence. Using the second claim in this section, we know that:

$$E[Z] \leq 2 \cdot 2nm(nm-1)$$

With Markov's inequality we then get:

$$P(Z \geq 2 \cdot 4nm(nm-1)) \leq \frac{E[Z]}{2 \cdot 4nm(nm-1)} \leq \frac{2 \cdot 2nm(nm-1)}{2 \cdot 4nm(nm-1)} \leq \frac{1}{2}$$

In other words, the probability that during the execution of $s_i, 1 \leq i \leq nm+1$ we do not visit the goal field – starting on an arbitrary field – is at most $\frac{1}{2}$. Therefore, the probability that we do not visit the goal field when executing $s = s_1 \ldots s_{nm+1}$ is:

$$P(X_M = 1) \leq \left(\frac{1}{2}\right)^{nm+1} = 2^{-(nm+1)}$$

By plugging in the definition of the expected value we get:

$$E[X_M] = 0 \cdot P(X_M = 0) + 1 \cdot P(X_M = 1) = P(X_M = 1) \leq 2^{-(nm+1)}$$

$\square$

This result also applies to SIMASOP. Even if $\mathcal{M}$ contains mazes of different sizes, we can apply the proof using the size $n \times m$ of the maze in $\mathcal{M}$ that maximizes $n \cdot m$. The number of mazes in such a set is $\leq (nm)^2 \cdot 2^{nm}$. Using those values in the proof results in a Solving Sequence that has a length which is still in $\mathcal{O}((nm)^3)$.

5 Practical Algorithms

In this chapter we present different algorithms for solving the problems we defined earlier. The algorithms presented in the first two section are mainly used as building blocks for the algorithms presented in later sections. In those later sections we focus on algorithms for SIMASOP, ASIMASOP, and lower bounds.

5.1 Shortest Path Algorithms on Mazes

Finding a shortest Solving Sequence of a grid maze is a problem that has been thoroughly researched. It is a shortest path problem on a graph where state of the art techniques can handle large numbers of nodes [Sto13] [HKRS97]. However, finding a shortest Solving Sequence for a large *set of grid mazes* is a problem that seems to be significantly harder. The state space for this problem is exponential in the number of mazes. Thus, even for a small number of mazes the state space is normally too large to fit into RAM. It follows that we cannot store the distance to every state explicitly and especially exploring the whole state space is probably infeasible.

In the remainder of this section we have a look at different algorithms that find a shortest Solving Sequence on a set of mazes. As this problem seems difficult for a large number of mazes, we also present an approximation algorithm that is faster but might not return an optimal result.

Brute Force[1]

The naive way to calculate a shortest Solving Sequence is to try for every possible sequence, in ascending order of their length, if it is a Solving Sequence for the given set of mazes. The first Solving Sequence we find is also a shortest Solving Sequence

[1]Not to be confused with Brute Force by The Algorithm: https://youtu.be/CDS9gmdHtB8.

because of the order in which we checked them. You might have already guessed it; the runtime is huge: it is in

$$\mathcal{O}\left(4^{\sigma(\mathcal{M})} \cdot |\mathcal{M}|\right)$$

where $\sigma(\mathcal{M})$ is the length of the shortest Solving Sequence of $\mathcal{M}$. We have 4 different moves to choose from for all sequences up to the length of $\sigma(\mathcal{M})$, and for every sequence we have to check if it solves all the mazes in $\mathcal{M}$. Note that if $\mathcal{M}$ is not explicitly given as in ASIMASOP and the implicitly used $\mathcal{M}$ is too large to compute explicitly, we have to use some algorithm to find an unsolved maze. Obviously, this approach is computationally infeasible already for short shortest Solving Sequences and a small set of mazes.

Dijkstra's Algorithm

Using Dijkstra's algorithm [Dij59] to find a shortest Solving Sequence is a little less naive. Instead of just trying out every possible sequence, Dijkstra's algorithm prunes all sequences with a prefix that is not a shortest sequence to the resulting target state. For example, let us consider the sequence $s = u$ (i. e., one single up move). The node in the search tree of Dijkstra's algorithm corresponding to s will be pruned as we are still in the starting state (of all mazes) after executing s. Because there is a trivial sequence of length 0 that leads to the starting state, s is not a shortest sequence. This, on the other hand, means that Dijkstra's algorithm will consider no sequence $s' = u\dots$, i. e., one starting with an up move, as it cannot be a shortest Solving Sequence. The runtime of Dijkstra's algorithm on a set of mazes $\mathcal{M}$ of size $n \times m$ is in

$$\mathcal{O}\left((nm)^{|\mathcal{M}|} \cdot |\mathcal{M}| \log(nm)\right)$$

A^*

The A^* algorithm [HNR68] adds a goal-directed component to Dijkstra's algorithm. While this might[2] help in practice, it does not improve the theoretical worst-case runtime. The goal direction results from the usage of a heuristic that estimates the distance to the goal. We use the longest path *of all mazes* to the goal state as heuristic. This heuristic is also suggested on [Stac] but without arguing why it is correct. For A^* to be correct (in the version of A^* where every node is at most inserted once into the priority queue),

[2]Depending on the graph, metric, and heuristic the A^* algorithm might even be slower than Dijkstra's algorithm in practice (e. g., on road networks with travel time as metric and euclidean distance as heuristic [GH05]).

the heuristic has to be consistent. A heuristic h is consistent iff $h(v) \leq c(v, w) + h(w)$ and $h(\text{goal state}) = 0$, where c is the cost function for the transitions. These conditions imply that we never overestimate the distance to a certain state, which in turn means that we never settle the distance of a state twice. Why is the heuristic we use consistent? From the intuition of never overestimating the distance to a state, it is directly clear that our heuristic is consistent. Let us check the formal conditions anyway. If we are in the goal state, then the longest shortest path is 0, i. e., $h(\text{goal state}) = 0$. If $c(v, w) = 0$, we stay in the same state ($v = w$) and the inequality holds. If $c(v, w) = 1$, we know that we took one step and the heuristic can decrease at most by one. Thus, the inequality holds again. As $c(v, w) \in \{0, 1\}$, it follows that our heuristic is consistent.

Greedy Lookahead Algorithm

As all algorithms we previously mentioned in this section have an infeasible runtime already on a small set of mazes, we present another algorithm that only approximates the shortest Solving Sequence. This algorithm greedily makes the best local decision and therefore has a very small memory footprint. Still, to determine the best local decision, we have to iterate over all the (unsolved) mazes. The states are evaluated by a cost function c, which has as input the current position p in all mazes. A natural cost function uses the shortest path distance to the goal of every single maze and penalizes large distances to the goal. Additional to the cost function, we use a list of lookahead sequences $\mathcal{L}$ from which we greedily choose the best one in every step, i. e., the one that minimizes the cost function. We then append this sequence to the already created sequence. The complete algorithm is shown in Algorithm 5.1. Note that the algorithm highly depends on its parametrization. This is especially important because the algorithm is in general not even guaranteed to terminate as we might get stuck in a local minimum. However, we can easily detect if we are stuck in a local minimum. For this, we check if the value of min_cost decreased in comparison to the last pass of the loop in line 3. When this is the case we do not use the best sequence from $\mathcal{L}$. Instead, we solve a random maze from the set of mazes and append this sequence to s (instead of min_cost) in line 14. This algorithm is also described on [Stad], but without a general lookahead set and avoidance of infinite loops.

5.2 Algorithms to Find Unsolved Mazes

There are several reasons why we need algorithms that find unsolved mazes given a sequence s. First, we can use those algorithms to check if s is a Solving Sequence. That is obviously the case if we do not find any unsolved maze, and the algorithms we

Algorithm 5.1 Heuristic algorithm for finding a short Solving Sequence

Input: c = cost function, $\mathcal{L}$ = lookahead sequences

Output: s = Solving Sequence

```
1: $s = \epsilon$
2: $p$ = starting position
3: while $p$ is not the goal position do
4:     $min\_seq = \epsilon$
5:     $min\_cost = \infty$
6:     for $s' \in \mathcal{L}$ do
7:         $p' = \text{step}(p, s')$
8:         if $c(p') < min\_cost$ then
9:             $min\_seq = s'$
10:            $min\_cost = c(p')$
11:        end if
12:    end for
13:    $p = \text{step}(p, min\_seq)$
14:    $s = s \cdot min\_seq$
15: end while
16: return $s$
```

present indeed always finds an unsolved maze if one exists. Secondly, we can construct a Solving Sequence using a routine that finds a yet unsolved maze – see 5.4 and 5.3 for the details.

We only consider the problem of finding an unsolved maze for ASIMASOP, as for SIMASOP the mazes are given in a set. For SIMASOP, we can then simply iterate over the mazes and test if they are solved by s. This is naively done in $\mathcal{O}(|s|\,|\mathcal{M}|)$.

If it is feasible to compute all mazes for ASIMASOP, then we can also use the naive way of finding an unsolved maze there. Otherwise, we have to think of a more sophisticated way. We present such an algorithm in the remainder of this section.

Recall that the input to the algorithm is a sequence s. The general idea is to perform a depth first search, where the depth of the tree corresponds to the position in s (i. e., the root node's level corresponds to the empty sequence, the next level to s_1, the next to $s_1 s_2$, e. t. c.). The children of a node are according to the following two cases: are we blocked in this step or not. Thus, in every node we have a position and a maze that has fields which are blocked, free or unknown. The maze of the root node has two free fields – start and goal – and all the others are unknown. The position is $(1, 1)$, i. e., the default starting position. Note that often a child node will not be created because it contradicts the maze of its parent or because it is not solvable. This is important as it prunes the tree and therefore greatly reduces the runtime of the search. If now, for example, the first move in s is a right move, we have two cases:

1. the field $(1, 2)$ is free
2. the field $(1, 2)$ is blocked

For the first case, we create a child node with position $(1, 2)$ and a maze like the one of the root node only with $(1, 2)$ being free instead of unknown. For the second case, we create a child node with position $(1, 1)$ and a maze like the one of the root node only with $(1, 2)$ being blocked instead of unknown. As soon as we find a node on level $|s|$ (where the level of the root is 0), and none of its parents has the goal position as position, we found a maze that is not solved by s. The unsolved maze is the maze of this node where the unknown fields can be arbitrarily set.

In Algorithm 5.2 you can see the pseudocode.

Lines 1 to 13 are a standard depth-first search. The only modification being that we cut branches when we reach the goal position (l. 6), and return an unsolved maze if we found one (l. 7-9). In the procedure EXPAND(e) (l. 15-39) we push the children of e to the stack. There, we handle the two cases if the move is executed (l. 21-25) or if it is blocked (l. 27-38). Only if the neighboring field is yet unknown, both children are created.

The correctness of the algorithm follows directly from its construction. We create all mazes that might be traversed differently by the input sequence s. Thus, we find an unsolved maze if and only if there exists one.

In Figure 5.1 you can see an example of the execution of Algorithm 5.2.

5.3 Exact Shortest Solving Sequence Algorithms

We show in Section 4.1 that SIMASOP is NP-complete. This unfortunately means that we cannot hope for an efficient algorithm to find a shortest Solving Sequence for an arbitrary set of mazes. While a search for such an algorithm for ASIMASOP has been conducted, we also could not find an *efficient* algorithm in that case.

As finding a shortest Solving Sequence is a shortest path problem, we can directly use the algorithms presented in Section 5.1. Those methods, however, quickly become infeasible when the size or the number of mazes increases. The main issue with finding another algorithm that computes the shortest Solving Sequence is to assure correctness of the solution, i. e., how can we make sure that there is no unsolved maze left and, also, that

Algorithm 5.2 Find an unsolved maze if one exists

Input: s = sequence for which we want to find an unsolved maze
Output: M = maze that is not solved by s. Or message that there is no unsolved maze.

```
1:  $Stack = \emptyset$
2:  $first\_element = (maze = initial\_maze, pos = (1,1))$
3:  $Stack.push(first\_element)$
4:  while $Stack \neq \emptyset$ do
5:      $e = Stack.pop()$
6:      if $e$ has not been expanded and $e.pos \neq (n,m)$ then
7:          if depth of $e = |s|$ then
8:              return $e.maze$
9:          end if
10:         EXPAND($e$)
11:     end if
12: end while
13: return there is no unsolved maze
14:
15: procedure EXPAND($e$)
16:     $M = e.maze$
17:     $pos_{old} = e.pos$
18:     $d$ = depth of $e$
19:     $pos_{new} = step_M(pos_{old}, s_{d+1})$
20:
21:     if $pos_{new} \neq pos_{old} \vee pos_{new}$ is unknown then
22:         $new\_maze = M$ with $pos_{new}$ being free
23:         $new\_element = (maze = new\_maze, pos = pos_{new})$
24:         $Stack.push(new\_element)$
25:     end if
26:
27:     if $pos_{new} = pos_{old} \vee pos_{new}$ is unknown then
28:         if $pos_{new}$ is unknown then
29:             $new\_maze = M$ with $pos_{new}$ being blocked
30:             if $new\_maze$ is not solvable then
31:                 return
32:             end if
33:         else
34:             $new\_maze = M$
35:         end if
36:         $new\_element = (maze = new\_maze, pos = pos_{old})$
37:         $Stack.push(new\_element)$
38:     end if
39: end procedure
```

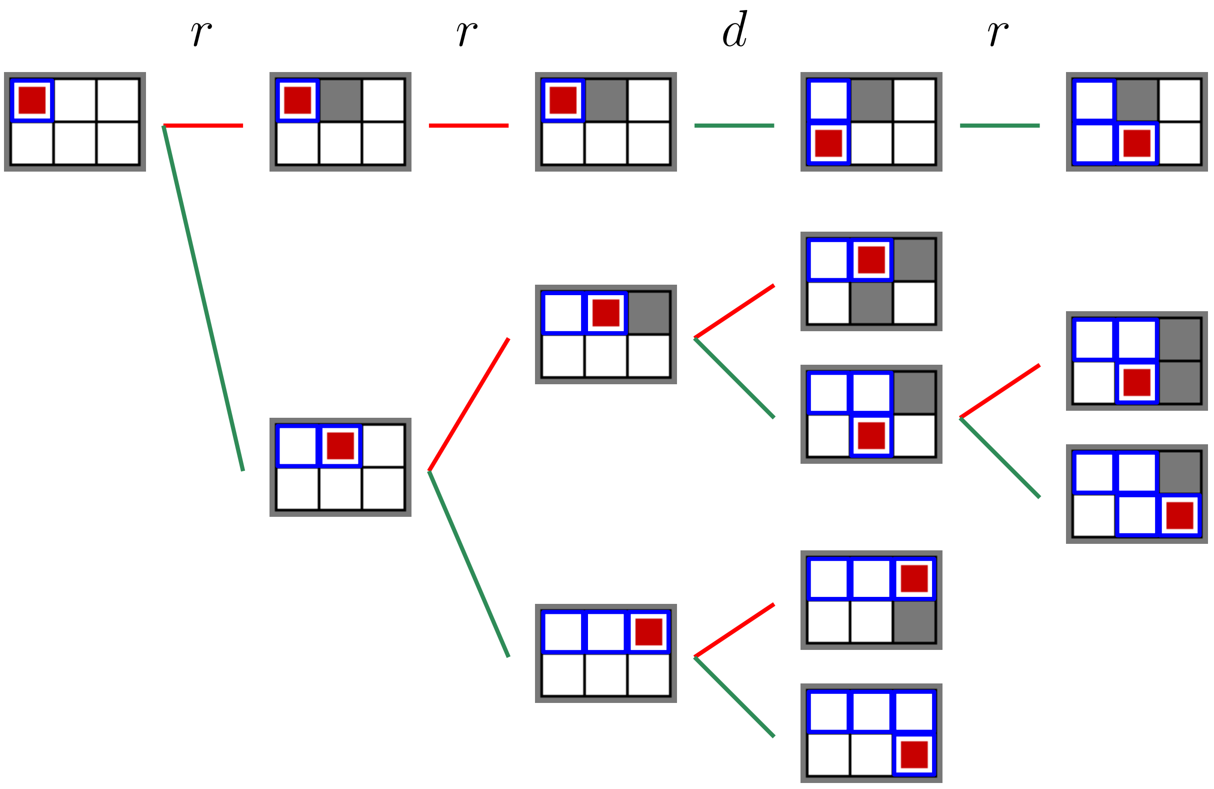

Figure 5.1: This figure shows the tree that results from the execution of Algorithm 5.2 for 2×3 mazes and $s = rrdr$. There are two different types of children in the tree. First, the ones connected with a red edge, which are those where the move is blocked. Secondly, the children connected with a green edge are the ones where the move is not blocked. The position is marked with a red square. The blocked fields are the ones that have to be blocked. The mandatory free fields are marked with blue color. The tree is constructed by a depth first search where we first explore the "green children". In two of the leaves we stop exploring because we found a solved maze; in three of the leaves we stop because the maze is not solvable anymore; and in the upper leaf we find an unsolved maze, which is then returned by the algorithm.

Algorithm 5.3 Calculates a shortest Solving Sequence

```
Input: $\mathcal{M}_{all}$ = set of mazes that should be solved
Output: $s$ = shortest Solving Sequence for $\mathcal{M}_{all}$
1: $\mathcal{M} = \emptyset$
2: $s = \epsilon$
3: while $U$ = FIND_UNSOLVED_MAZE($s$, $\mathcal{M}_{all}$) do
4:     $\mathcal{M} = \mathcal{M} \cup \{U\}$
5:     $s$ = FIND_SHORTEST_SOLVING_SEQUENCE($\mathcal{M}$)
6: end while
7: return $s$
```

there is no shorter Solving Sequence? Those questions lead directly to the design of Algorithm 5.3.

The main idea of Algorithm 5.3 is to iteratively calculate a shortest Solving Sequence of a subset of all the mazes as long as there is an unsolved maze left. Note that this formulation "answers" both of our questions: because we calculate a shortest Solving Sequence of a subset of all mazes, we know it is minimal w. r. t. all mazes; and as we continue as long as there is an unsolved maze left, we assure that the result is indeed a Solving Sequence for all mazes. We can use the algorithms presented in Section 5.1 as the function FIND_SHORTEST_SOLVING_SEQUENCE, and we can use the algorithms presented in Section 5.2 as the FIND_UNSOLVED_MAZE function.

Note that for the algorithm to be more efficient than a naive shortest path search, there has to exist a small subset of mazes that captures the difficulty of solving the complete set of mazes. In Chapter 6, where we conduct practical experiments, we show that this assumption is true for ASIMASOP but might not be for small sets of randomly generated mazes.

5.4 Approximate Shortest Solving Sequence Algorithms

In the last section we had a look on how to calculate a shortest Solving Sequence. Due to the NP-completeness of SIMASOP, we cannot hope that this is always possible efficiently. Naturally, we now present algorithms that approximate the shortest Solving Sequence. Often it is much easier to calculate a *minimal* solution (locally optimal) than a *minimum* solution (globally optimal). This is also the case for the two main problems of this thesis. By just iterating over a Solving Sequence and deleting moves that don't effect the solving property, we can create a *minimal* Solving Sequence out of some Solving Sequence. This method can be applied to any sequence that is a result of the following approximation algorithms. Also note that the length of the sequences are an upper bound on the length of the shortest Solving Sequence. We now present four different approximation algorithms.

Random Sequence

A completely naive way of finding a Solving Sequence is to create a long random sequence and then check if it solves all mazes. As the length of a random sequence increases, also the probability of it being a Solving Sequence increases; see Section 4.8. By minimizing the sequence, we can also reduce its length after checking that it indeed is a Solving Sequence. Even though one might think that this method does not work well in practice, it interestingly does produce good results as we show in Chapter 6.

Solve in Order

In Section 4.3 we constructively prove that there always exists a Solving Sequence. While we prove in Section 4.4 that there does not always exist an ordering of the mazes such that the algorithm returns a *shortest* Solving Sequence, it still does return a Solving Sequence. Therefore, we can simply use Algorithm 4.2 as an approximation algorithm for the shortest Solving Sequence. For a detailed explanation of the algorithm, see the corresponding section. By choosing different orderings, we can improve the approximation in pratice. This method requires that it is feasible w. r. t. runtime to iterate through all the mazes. That is, however, not necessary for the next algorithm we present.

Iteratively Append to Sequence

In the just presented algorithm we solve the mazes in order. However, if we are not interested in specifying the order in which the mazes are solved, we only need some unsolved maze in every iteration. This idea directly leads to the formulation of Algorithm 5.4. In this algorithm we iteratively compute an unsolved maze and then append moves to the already existing sequence to also make it a Solving Sequence for this maze. On the one hand, always appending to a sequence most probably makes it longer than necessary. On the other hand, by simply appending to the sequence, the only runtime intensive subroutine of this algorithm is finding an unsolved maze.

Algorithm 5.4 Approximate the shortest Solving Sequence

Input: $\mathcal{M}$ = set of mazes that should be solved
Output: s = Solving Sequence for $\mathcal{M}$

```
1: $s = \epsilon$
2: while $U$ = FIND_UNSOLVED_MAZE($s$, $\mathcal{M}$) do
3:     $pos$ = position in $U$ after executing $s$
4:     $s'$ = shortest path from $pos$ to the goal in $U$
5:     $s = s \cdot s'$
6: end while
7: return $s$
```

Use Greedy Lookahead in Algorithm 5.3

In the last section we presented algorithms that compute exact shortest Solving Sequences. The main algorithm, Algorithm 5.3, uses a function called FIND_SHORTEST_SOLVING_SEQUENCE to compute a shortest Solving Sequence for a subset of mazes. Instead of computing an exact shortest Solving Sequence with this function, we can also just use an approximation, e. g., the Greedy Lookahead algorithm presented in Section 5.1. This change should accelerate the algorithm significantly while compromising the length of the returned Solving Sequence.

5.5 Lower Bound Algorithms

In the previous section we considered approximation algorithms for shortest Solving Sequences. The length of the resulting sequences constitute an *upper* bound on the length of the shortest Solving Sequence. In this section we have a look at *lower* bounds on the length of the shortest Solving Sequence.

Using a Shortest Solving Sequence Algorithm

In most of the algorithms that compute a shortest Solving Sequence, we have intermediate results that can be used as a lower bound:

- **Brute Force:** As we iterate over all the sequences in increasing order, we can stop at any point in the algorithm and know that the length of the current sequence is a lower bound.

- **Dijkstra's Algorithm:** Here we can, as well, stop at any point in the algorithm and then the length of the sequence of the first element in the priority queue gives us a lower bound.
- **A*:** The argument regarding Dijkstra's algorithm carries over to A^*. This is due to the heuristic being consistent.
- **Algorithm 5.3:** The intermediate result of this algorithm is a set of mazes (a subset of all mazes) and a corresponding shortest Solving Sequence. The shortest Solving Sequence of all mazes has to be at least as long as those of arbitrary subsets of mazes. Therefore, the length of the sequence of the intermediate result is a lower bound.

Refined Algorithms

The algorithms that compute exact shortest Solving Sequences can be further refined to suit the problem of computing a lower bound better. For example, instead of choosing an arbitrary unsolved maze in Algorithm 5.3, we can always choose the maze that maximizes the shortest Solving Sequence computed in line 5. This means, we try to maximize the length of the shortest Solving Sequence of the mazes in $\mathcal{M}$ greedily.

Another way of choosing the unsolved maze in Algorithm 5.3 is by partitioning the mazes, using an already existing Solving Sequence. For this, we take the sequence and partition the mazes according to in which move of the sequence they are solved. Then, we iteratively choose the worst case maze of the next partition (i. e., the one that produces the longest Solving Sequence of $\mathcal{M}$) and put it into $\mathcal{M}$. The idea of this algorithm is that the mazes of a partition are somewhat similar in structure, and that including one of them in $\mathcal{M}$ represents the difficulty of solving the whole partition. Unfortunately, this assumption does not seem to hold in practice.

6 Practical Analysis

In this chapter we test the algorithms presented in the previous chapter regarding their runtime and quality of their results. The focus of our implementation is to compare the different approaches and to get a rough idea which sizes of sets of mazes and sizes of mazes are feasible to handle. Therefore, we did the implementation using python. This allowed for a quick implementation of different algorithms. The obvious downside of this choice is the bad runtime in comparison to low-level languages like C++. Some parts were also implemented in C++. However, because of comparability, we only use python for the practical analysis. Additionally, due to the combinatorial explosion that occurs in (almost) all algorithms regarding this topic, an implementation with better runtimes would not enable us to solve (significantly) larger instances using the same algorithms. Also, note that practical algorithm engineering (i. e., minimizing the practical runtime of an algorithm given in theory) is not the main focus of this thesis. All tests are conducted on an Asus Zenbook UX303LN with an Intel Core i7-4510U CPU and 12GB of RAM (SODIMM DDR3, Synchronous, 1600 MHz). No parallelization was used, even though for several algorithms it would have been possible.

6.1 Shortest Path Algorithms on Mazes

In this section we compare the different algorithms that directly compute a shortest path for a set of mazes. Among those algorithms are three that give an exact result (brute force, Dijkstra's algorithm, A^*) and one that only approximates the shortest path (Algorithm 5.1, called the Greedy Lookahead algorithm). To put the results into perspective, we also include Algorithm 4.2 (called Solve in Order) in our tests as it can be seen as a very naive version of the Greedy Lookahead algorithm – where the cost function c is constant and we therefore solve a random maze in every iteration because of the avoidance of local minima. We compare the runtime of all algorithms as well as the quality of the approximated results by the Greedy Lookahead algorithm. As the set of lookahead sequences $\mathcal{L}$ used in the Greedy Lookahead algorithm, we use all possible sequences of length 3 in our experiments. This choice is a compromise between runtime and quality of the solution. If the lookahead is increased by one, the length of the resulting sequence is reduced in most cases and thus also the number of steps the algorithm has to take. However, the number of lookahead sequences we need to check in each step quadruples and thus the runtime increases significantly. With additional algorithm engineering by choosing a better set of lookahead sequences $\mathcal{L}$, the overall performance of the algorithm can certainly be further improved.

Comparison of Runtime

As the runtime of the brute force algorithm already explodes for small sizes of mazes and number of mazes, we do not include it in the tests. We do not provide any data because this behavior can already be derived from the theoretical runtime.

In the first test we want to see how the runtimes of the algorithms behave when we increase the number of mazes in the set. For this we create 100 random sets containing k mazes of size 4×4, and then measure the time a shortest path computation takes for each algorithm on average. This is done from $k = 1$ to $k = 10$. You can see the results in Figure 6.1; note that the y-axis uses a logarithmic scale. Clearly Dijkstra's algorithm has a much worse runtime than all the other algorithms. Interestingly, A^* is not much slower than the Greedy Lookahead algorithm. However, as already mentioned in Section 5.1, the runtime of the Greedy Lookahead algorithm heavily depends on the choice of parameters. Therefore, the performance of that algorithm might still be improved significantly. The implementation of Algorithm 4.2 (i. e., the one that solves the mazes one after another) easily outperforms all the others, but also produces the longest Solving Sequences as we see later in this section.

In the second test we increase the size of the mazes instead of the number of mazes. For this we create 100 random sets containing 3 mazes of size $n \times n$. This is done from $n = 1$ to $n = 10$. The results are shown in Figure 6.2; again we use a logarithmic scale and the runtime is the average per set. The situation is very similar to the previous test. Dijkstra's algorithm is the slowest by far; A^* and the Greedy Lookahead algorithm are very similar w. r. t. runtime; and Algorithm 4.2 outperforms all the other algorithms (again, with the drawback of producing a longer Solving Sequence).

Greedy Lookahead Sequence Lengths

We now test the quality (i. e., length) of the Solving Sequences that are produces by the Greedy Lookahead algorithm. To put the length of the Solving Sequence into perspective, we compare it with the length of the shortest Solving Sequence (computed using A^*) and the naive way of creating a Solving Sequence using Algorithm 4.2.

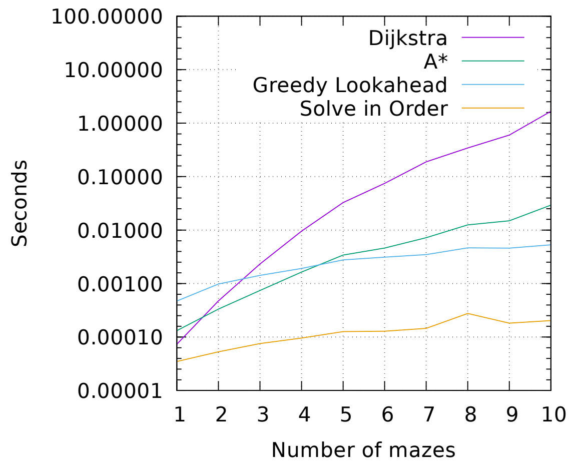

Figure 6.1: The runtime of the shortest path algorithms when the number of mazes increases.

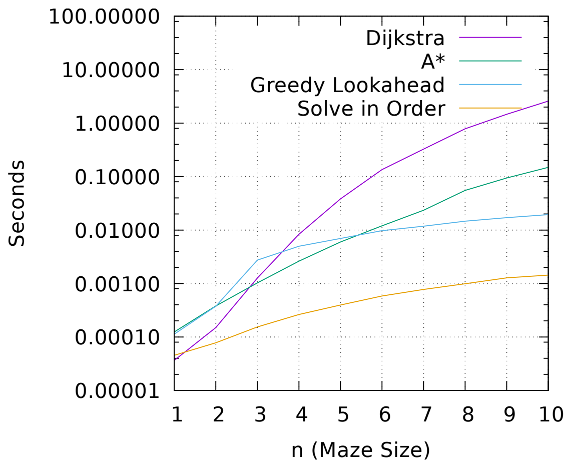

Figure 6.2: The runtime of the shortest path algorithms when the size of the mazes increases.

In the third experiment of this section we want to see how the length of the Solving Sequence behaves when increasing the number of mazes in the set. For this, we create 100 random sets containing k mazes of size 4×4, where k ranges from $k = 1$ to $k = 15$. You can see the results in Figure 6.3. As expected, the length of the Solving Sequence produced by the Greedy Lookahead algorithm is somewhere in between the length of the sequence produced by the other two algorithms. The difference between the naive way of creating a Solving Sequence and the Greedy Lookahead algorithm is significant and therefore justifies its worse performance w. r. t. runtime.

The next test paints a similar picture. There we increase the size of the mazes instead of the number of mazes. For this, we create 100 random sets containing 4 mazes of size $n \times n$ and let n go from $n = 1$ to $n = 10$. The results are shown in Figure 6.4. As always, we measure the runtime on average per set. Again, the performance w. r. t. quality of the Greedy Lookahead algorithm is closer to A^* than Algorithm 4.2.

6.2 Algorithms to Find Unsolved Mazes

In this section we analyze Algorithm 5.2 – the algorithm to find an unsolved maze of a certain size given a sequence s. We call this algorithm the Unsolved Maze Finder (UMF) in the following. A baseline is given by the naive way of finding an unsolved maze:

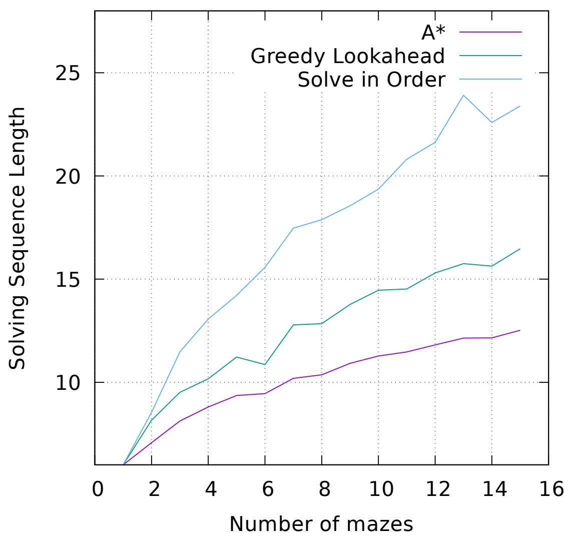

Figure 6.3: Development of the Solving Sequence length produced by the Greedy Lookahead algorithm when the number of mazes increases. The results of A^* and the naive method (Solve in Order) are shown for comparison.

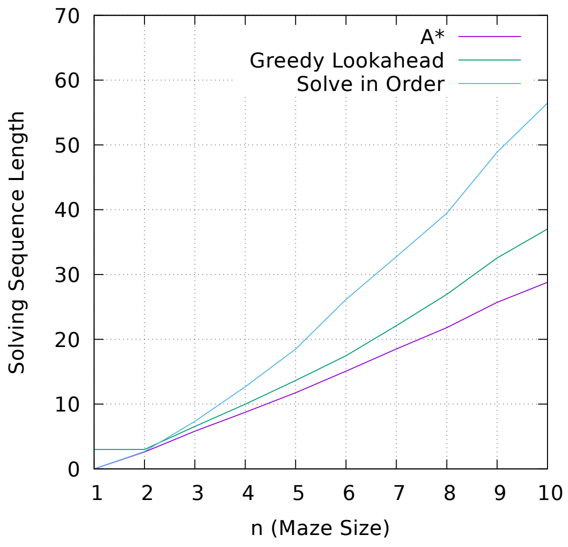

Figure 6.4: Development of the Solving Sequence length produced by the Greedy Lookahead algorithm when the size of the mazes increases. The results of A^* and the naive method (Solve in Order) are shown for comparison.

iterate through the set of all solvable mazes of this size and check one after another if they are solved by s. Note that all mazes of the specific size have to be created first. In the experiments this time is not measured, but only the time to find the first unsolved maze. However, as the mazes have to be created, the baseline algorithm becomes infeasible quickly when increasing the size.

Naive vs. UMF

Let us first compare our more sophisticated algorithm with the baseline algorithm. For this, we create random sequences of different lengths (from 1 to 49) and then measure the time it takes to find an unsolved maze of size 4×4. For every length we perform this test 1000 times and use the same sequences for both algorithms. The results are shown in Figure 6.5. Clearly, UMF outperforms the naive method.

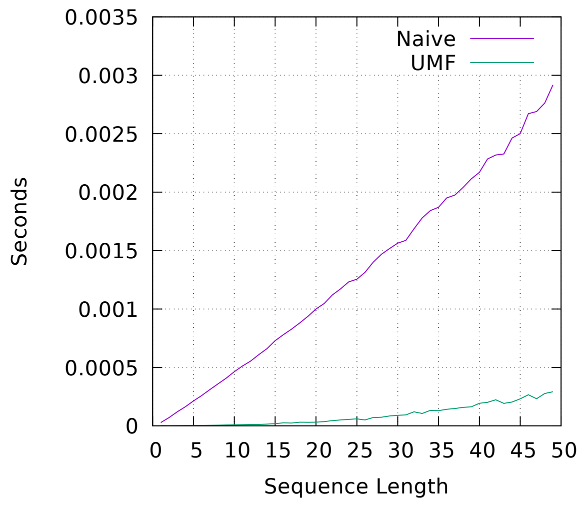

Figure 6.5: Comparison of the performance of the naive maze finding algorithm with the performance of UMF when the sequence length increases.

UMF

Let us now test which sizes and number of mazes UMF is capable of handling. First, we want to see how UMF behaves when the length of the sequence (for which we want to find an unsolved maze) is increased. We again create random sequences of different lengths (from 1 to 199) and then measure the time it takes to find an unsolved maze of size 10×10. For every length we perform this test 100 times. The results are shown in Figure 6.6. The runtime appears to be growing linearly in the length of the sequence. However, there are some spikes that can be explained by difficult instances, i. e., sequences that are almost Solving Sequences and where it takes a long time to find an unsolved maze.

In the second test of the sophisticated algorithm, we want to know how it behaves when the size of the mazes is increasing. For this, we create 100 random sequences of length 250 and then measure the time it takes to find an unsolved maze of size $n \times n$. We did this from $n = 1$ to $n = 19$. You can see the results in Figure 6.7. How can the local maximum at $n = 5$ be explained? It is probable that some of the random sequences solve most of the mazes of size 5×5. Thus, finding an unsolved maze requires us to explore a large part of the tree. After $n = 5$ the runtime declines again as the random sequences solve much less mazes of those sizes. However, as n increases further the runtime again increases – apparently in an almost linear fashion.

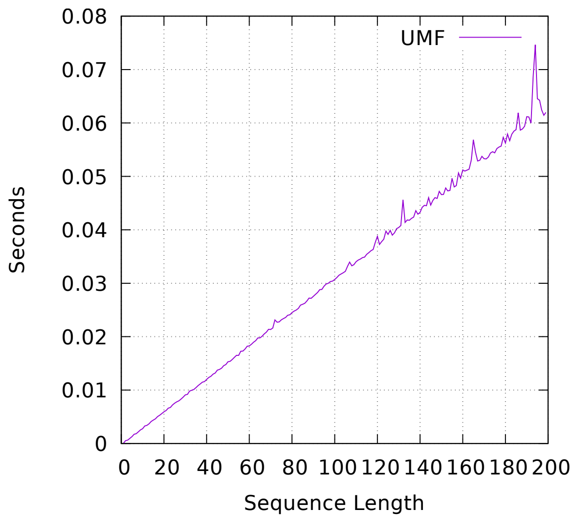

Figure 6.6: The runtime of UMF when the sequence length is increasing.

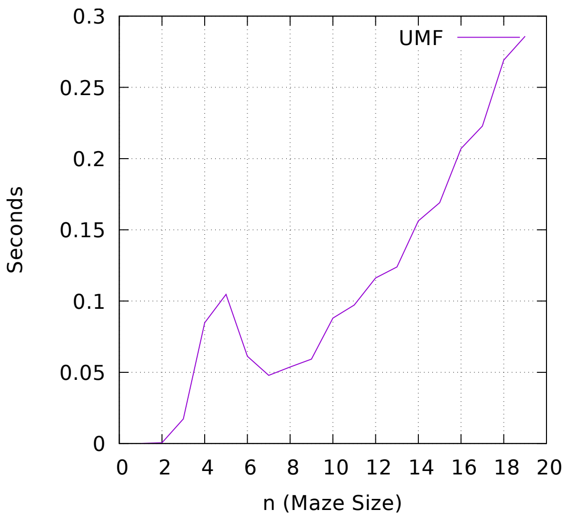

Figure 6.7: The runtime of UMF when the maze size is increasing.

n \ m	1	2	3	4
1	(0)	r (1)	rr (2)	rrr (3)
2	d (1)	rdr (3)	$rrdrr$ (5)	$rrrdrrr$ (7)
3	dd (2)	$ddrdd$ (5)	$rdrrdllddrr$ (11)	$rrdrrrdllllddrddrr$ (17)
4	ddd (3)	$dddrddd$ (7)	$ddrdddruuurrdrrdd$ (17)	–

Table 6.1: Shortest Solving Sequences calculated by A^* for ASIMASOP with size $n \times m$. The length of the sequences is denoted in brackets.

6.3 Exact Shortest Solving Sequence Algorithms

In this section we test the performance and show the results of the exact shortest Solving Sequence algorithms. First, we check what results we can obtain using A^*. As this has already been done for SIMASOP in Section 6.1, we only consider using A^* for ASIMASOP in this section. In the second part of this section we conduct experiments regarding the runtime of Algorithm 5.3 – called the Exact Solving Sequence algorithm (ESS) subsequently – for SIMASOP as well as ASIMASOP. To put the runtime of ESS into perspective we compare it with plain A^* on the set of mazes.

$n \backslash m$	1	2	3	4
1	0.000062	0.000125	0.000165	0.000213
2	0.000113	0.000462	0.001409	0.005914
3	0.000118	0.001167	0.091550	98.592423
4	0.000071	0.002280	99.169318	–

Table 6.2: Runtime (in seconds and real-time) of computing shortest Solving Sequences with A^* for ASIMASOP with size $n \times m$.

$n \backslash m$	1	2	3	4
1	0.000058	0.000296	0.000495	0.000652
2	0.000250	0.001684	0.003717	0.007201
3	0.000241	0.002443	0.032254	6.880254
4	0.000174	0.003242	3.221218	–

Table 6.3: Runtime (in seconds and real-time) of computing shortest Solving Sequences with ESS for ASIMASOP with size $n \times m$.

Shortest Path Algorithms

In Section 6.1 we test A^* on a set of mazes, which resembles the SIMASOP setting. We now want to test how A^* performs in the ASIMASOP setting. For this we let A^* run on the set of mazes $\mathcal{M}_{n,m}$ for $1 \leq n, m \leq 4$. Unfortunately, the runtime for $n = m = 4$ is infeasible for our implementation; maybe even in general on desktop computers. You can see the computed shortest Solving Sequences in Table 6.1 and the time it took to compute them using A^* in Table 6.2. Even though the algorithm is deterministic, we executed it 3 times and took the average runtime to compensate for small noise. Note the increase of the runtime by three orders of magnitude from the 3×3 to the 4×3 instance. The unusually low runtime for 4×1 (but not 1×4) is probably due to internal python or platform mechanisms as it cannot be explained from a theoretical standpoint or by our implementation.

ESS

Let us now analyze ESS. For the FIND_UNSOLVED_MAZE function we simply iterate over the set for SIMASOP but use Algorithm 5.2 for ASIMASOP. For the FIND_SHORTEST_SOLVING_SEQUENCE function we use A^* in both settings – SIMASOP as well as ASIMASOP. First, we test the performance of ESS for the SIMASOP setting. As a baseline we use the performance of A^* on the same sets of mazes. For this test, we choose 100 sets that contain k random mazes of size 4×4, with k going from $k = 1$ to $k = 20$. Then the average runtime for those 100 sets is computed. The results are shown in Figure 6.8. Interestingly, ESS has the same runtime as A^* in this setting. This suggests that there does not exist a small subset of mazes that captures the difficulty of solving the whole set. Thus, we still have to execute A^* on the complete set of mazes when using ESS.

Let us now test if there is an advantage of using ESS for ASIMASOP. For this, we let the algorithm run on $\mathcal{M}_{n,m}$ for $1 \leq n, m \leq 4$. Again, the runtime was infeasible[1] for 4×4 and we therefore do not report it. The experiment was repeated 5 times (even though the algorithm is deterministic) and then the runtime averaged. See Table 6.3 for the results. Compared with the entries of 3×4 and 4×3 in Table 6.2 there is a clear advantage of using ESS as it is more than one order of magnitude faster. Another interesting result is the asymmetry of the values for $a \times b$ and $b \times a$. If $a > b$, then consistently a better runtime is reported for $a \times b$ than $b \times a$. This behavior results from the asymmetry of our implementation of the A^* algorithm. The *down* edge is relaxed before the *right* edge. Therefore the computed shortest path in a maze might not be symmetric to the computed shortest path of its transposed maze. Apparently, the behavior of first exploring in the direction of the larger dimension is favorable for the general runtime of ESS.

6.4 Approximate Shortest Solving Sequence Algorithms

We can see in the last sections that solving SIMASOP as well as ASIMASOP becomes infeasible quickly as we increase the number of mazes or the size of the mazes. Because of this, we now analyze the approximation algorithms that are presented in Section 5.4. Those are (with abbreviations in brackets):

- Random Sequence with minimization (RS)
- Solve in Order (SO)
- Iteratively Append to Sequence (IAS)

[1]We let ESS run for several days on a server on the 4×4 instance and it did not terminate. However, we believe it is possible to calculate a shortest Solving Sequence using ESS with significant additional algorithm engineering on the implementation and the theoretical algorithm.

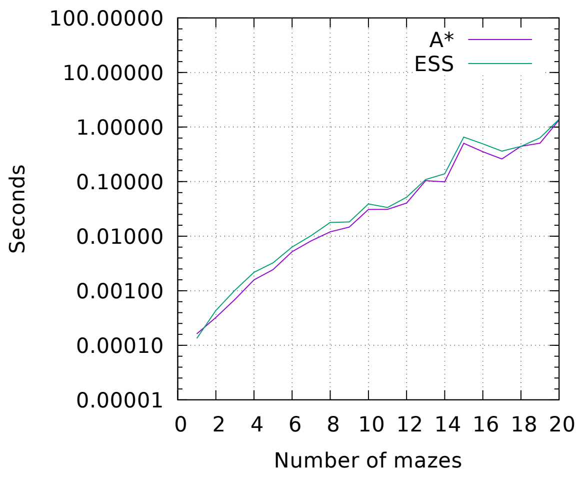

Figure 6.8: Comparison of the runtime of A^* and ESS with an increasing number of mazes.

- Use Greedy Lookahead algorithm as FIND_SHORTEST_SOLVING_SEQUENCE function in the ESS algorithm (GL)

Note that SO and IAS produce the same results, as explained in Section 5.4. The difference is that SO is faster for SIMASOP but infeasible for ASIMASOP (for larger n, m), which in turn IAS is feasible for. Therefore, we use SO for the SIMASOP and smaller ASIMASOP tests and IAS for all ASIMASOP tests. Furthermore, we used a random sequence of length 250 for RS for all tests.

SIMASOP

In the first two experiments we want to test the behavior of RS, SO, and GL when the number of mazes increases. For this, we create 100 sets of k random mazes of size 4×4, where k ranges from $k = 1$ to $k = 50$. See Figure 6.9 for the average runtimes and Figure 6.10 for the average sequence length. Those values are averaged over the 100 created sets. The measurements show a much better runtime for SO than for RS and GL; the runtimes of the latter are in the same order of magnitude. However, while SO has a good runtime, it is significantly worse w. r. t. the length of the Solving Sequence. Again, RS's and GL's values are very similar.

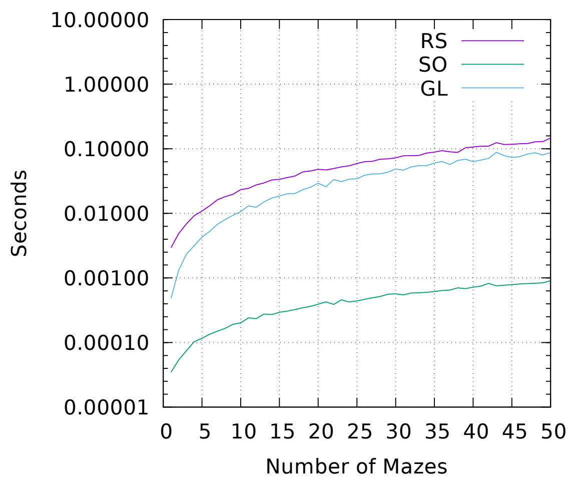

Figure 6.9: Runtime of the different algorithms to compute an approximate shortest Solving Sequence for SIMASOP with increasing number of mazes.

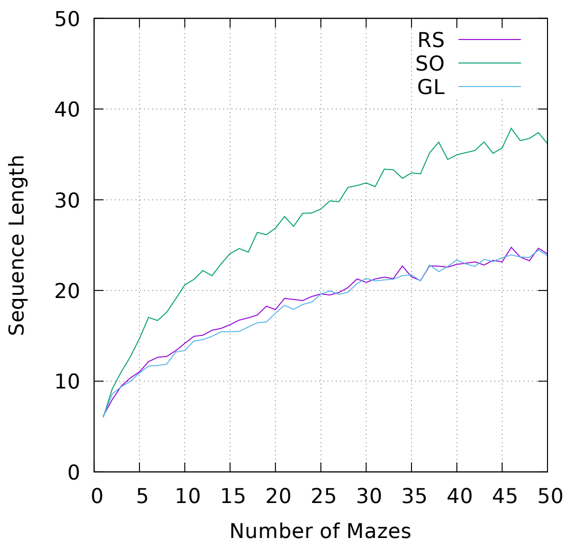

Figure 6.10: Length of the approximate shortest Solving Sequence of the different algorithms for SIMASOP with increasing number of mazes.

ASIMASOP

For the test of the algorithms on ASIMASOP, we compute the approximate shortest Solving Sequence on $n \times n, 1 \leq n \leq 4$. Due to the bad runtime of RS in this setting, we only repeat the test 10 times and average the results. They are shown in Table 6.4 for the runtime and in Table 6.5 for the sequence length. Note that RS always minimizes the sequences while the other algorithms do not by default. To allow for a better comparison, we give the times and sequence lengths including minimization in brackets. Additionally, as there are only 3828 solvable mazes of size 4×4, we can also use SO for this instance and the smaller instances. However, instead of just solving the mazes in a fixed order, we randomly permute them for every run. This obviously might change the result. Though, because of the similarity to IAS, no runtimes are reported. Also, we ran SO much more often than 10 times to get better results. As (almost) no randomization is used in IAS and GL, increasing the number of tests to get better results does not make sense for those two algorithms.

Without minimization, clearly IAS is much faster than GL, which in turn is much faster than RS. Considering the shortness of the computed Solving Sequences, the order is exactly the opposite. So, unfortunately there is a trade-off between runtime and

n \ Alg	RS	IAS	GL
1	0.004778	0.000015 (0.000104)	0.000012 (0.000097)
2	0.083224	0.000152 (0.001332)	0.000383 (0.002099)
3	2.732247	0.003703 (0.164220)	0.018921 (0.129637)
4	170.701078	0.269189 (97.665786)	8.697402 (61.317300)

Table 6.4: Runtime (in seconds and real-time) of the different algorithms to compute an approximate shortest Solving Sequence for ASIMASOP with size $n \times n$. In brackets we note the runtime *including* minimization of the sequence.

n \ Alg	RS	IAS	GL	SO
1	0.0	0 (0)	0.0 (0)	0 (0)
2	3.0	3 (3)	3.0 (3)	3 (3)
3	14.1	20 (17)	15.0 (14)	12 (11)
4	51.7	109 (53)	74.5 (53)	50 (29)
5	–	551	–	–
6	–	3728	–	–

Table 6.5: Approximate shortest Solving Sequence length of the different algorithms for ASIMASOP with size $n \times n$. In brackets we note the length of the minimized sequence.

quality of the Solving Sequence. Although, GL has the best runtime with subsequent minimization because of the initially computed short sequences which are minimized faster. The length of the computed Solving Sequence with minimization is roughly the same for all but SO. Because of the randomized approach and the good runtime, we can compute much shorter sequences with this algorithm. Note, though, that for those results we have to execute SO many times and then take the best result. The same holds for SO without minimization. The sequence lengths for 5×5 and 6×6 are reported without the length of a minimized sequence as the runtime of the minimization is too high for those instances. Due to the long runtime they were computed on a server. Even without minimization the runtime of GL is too high for the larger instances. See Appendix B for the best Solving Sequences we found for 4×4, 5×5, and 6×6.

6.5 Lower Bound Algorithms

In this section we test the quality of the lower bound algorithms described in Section 5.5 for the ASIMASOP setting. We do not provide any runtimes as the main goal of the lower bound algorithms is to find a large lower bound. We will compare the results of four algorithms (abbreviations in brackets):

- **A***
- Algorithm 5.3 (ESS)
- A greedy version of Algorithm 5.3 (GESS)
- Partition and iteratively choose the best maze from a set (PIC)

Let us first describe how exactly we use each of the algorithms to produce a lower bound. For A^* we simply let it run on the set of all mazes. We cannot compute those for larger n, m but it is still feasible for 4×4. Note that already for 4×4 the A^* algorithm does not finish in reasonable time. Therefore, instead of waiting for the final result, we use intermediate results to get a lower bound. Those intermediate results are the heuristic distances of the top elements of the priority queue as they never overestimate the distance to the goal state.

We always calculate intermediate results when running the standard ESS algorithm. Those are the current set of mazes and the corresponding shortest Solving Sequence. This sequence is a lower bound for the set of all mazes. Thus, we simply execute ESS and report the intermediate results during execution.

The GESS algorithm uses the structure of the ESS algorithm with one modification. Instead of calculating an unsolved maze in every iteration, we create k random mazes and choose the one that produces the longest shortest Solving Sequence with the mazes already in the set $\mathcal{M}$. The shortest Solving Sequence that is computed inside the loop is then used as the lower bound. This algorithm is an alternative to the first algorithm described in the Refined Algorithms part of Section 5.5. There, we choose the maze that increases the length of the shortest Solving Sequence the most in every round – which unfortunately has a huge runtime.

The PIC algorithm is already completely described at the end of Section 5.5. We use the minimized Solving Sequence computed by SO for 4×4 as input for PIC.

In the following experiments we do not calculate lower bounds for $n \times n$ with $1 \leq n \leq 3$ because we already know the shortest Solving Sequence for those sizes. See Table 6.6 for the results of the lower bound computations for $4 \leq n \leq 6$. Instead of using the machine described at the beginning of this chapter, we used a server for those computations. Especially the increased amount of RAM (~ 386 GB) enabled us to run A^* on larger sets

Alg / n	A^*	ESS	GESS	PIC	u.b.
4	21	26	23	20	29
5	–	29	24	–	551
6	–	31	27	–	3728

Table 6.6: Lower bounds on the length of the shortest Solving Sequence computed by the different algorithms for ASIMASOP with size $n \times n$. In the upper bound (u.b.) column we show the best upper bound (i. e., the length of the shortest Solving Sequence) we found in the previous sections.

of mazes. Those are the best results that we could compute using a reasonable amount of time. See Appendix B for the sets of mazes that induce the reported lower bounds of ESS for 4×4, 5×5, and 6×6.

7 Conclusion

In this thesis we analyzed the problems of finding a shortest Solving Sequence for a set of mazes (SIMASOP) and for the set of all solvable mazes of a certain size (ASIMASOP), which is a special case of the first problem. Additionally, we were interested in enumerating the solvable mazes of a certain size or at least know their number. While one might initially assume that those problems are easy to solve for small sizes, this does not seem to be the case.

On the theoretical side we showed that there always exists a Solving Sequence and even a Perfect Solving Sequence – where all mazes end up in the goal field at the end of executing the sequence (as opposed to only visiting it). Additionally, we could prove that SIMASOP is NP-complete by reduction from CNFSAT and that ASIMASOP is contained in PSPACE by using a result concerning Universal Traversal Sequences. Furthermore, we showed theoretical lower and upper bounds for shortest Solving Sequences. An algorithm for computing the exact number of solvable mazes much faster than naively is also given.

On the practical side, we first presented several algorithms and then evaluated their performance regarding solution quality and runtime. Some algorithms we presented were mainly introduced to use them as parts of more sophisticated algorithms in later sections. Those are the algorithms for the computation of a shortest path of a small set of mazes and the algorithm to find an unsolved maze of a certain size given a sequence. These algorithms are then used in the algorithms that compute exact or approximate shortest Solving Sequences for SIMASOP and ASIMASOP. Additionally, we presented algorithms that compute a lower bound on the length of the shortest Solving Sequence for both problems. Combined with the approximation algorithms, this can also be used to find exact shortest Solving Sequences. In the practical analysis we first showed that A^* is superior to Dijkstra's algorithm in this setting and surprisingly close to one of the approximation algorithms we presented regarding runtime. We additionally showed the clear advantage of using the sophisticated unsolved maze finding algorithm (UMF) in comparison to using the naive way. Furthermore, a comparison of the runtime showed that our main algorithm for finding a shortest Solving Sequence is much faster for ASIMASOP while for SIMASOP on a small number of random mazes the performance is similar to A^*. The best algorithm we found for approximating a shortest Solving

Sequence is the Solve in Order algorithm (SO) with subsequent minimization and a random order. As it is randomized, we get different results and can then choose the best one. Using this algorithm we found a Solving Sequence of length 29 for ASIMASOP with size 4×4. However, the SO algorithm relies on the enumeration of the mazes. For larger sizes (up to 6×6) the Iteratively Append to Sequence algorithm (IAS) can be used. Finally, the practical analysis identifies the Exact Solving Sequence algorithm (ESS) to be best suited to compute lower bounds for the length of Solving Sequences for ASIMASOP. With this algorithm we computed a lower bound of 26 for ASIMASOP with size 4×4.

There are still several problems to solve in future work. First, while we could calculate a Solving Sequence of length 29 for ASIMASOP with size 4×4, we were only able to show a lower bound of 26. Therefore, it still remains open what the exact length of the shortest Solving Sequence is. A natural area of research for future work is thus the search for better algorithms to compute shortest Solving Sequences as well as lower bounds. On the theoretical side, we are still lacking a lower bound for the runtime of ASIMASOP. As we know that ASIMASOP is in PSPACE, the natural questions are if ASIMASOP is PSPACE-hard or if one can show that it is not in NP. Maybe the problems is even much easier to solve and we just could not find such an algorithm. Another issue is that, while we could compute instance based lower bounds on the length of the Solving Sequence, our approximation algorithms lack a priori bounds. Our search for such an algorithm was not successful. Finally, a more efficient algorithm to calculate the number of solvable mazes of a certain size would be of interest or, alternatively, a proof of the hardness of computing those numbers.

A Deutsche Zusammenfassung

Ein Grid Maze ist eine binäre Matrix deren 1er Felder wir betreten können während die 0er Felder blockiert sind. Es gibt vier verschiedene Züge in solch einem Grid Maze: hoch, runter, links und rechts. Wir nennen eine solche Sequenz von Zügen eine *Lösungssequenz*, wenn wir in der linken oberen Ecke beginnend die Sequenz auszuführen und die rechte untere Ecke während der Ausführung *besuchen*. Eine solche Lösungssequenz für ein einzelnes Grid Maze zu ermitteln wurde schon ausgiebig erforscht. Im Gegensatz dazu wurde das Problem eine einzige Lösungssequenz zu finden, die alle Elemente einer Menge von Grid Mazes löst, noch kaum betrachtet. Besonders die Formulierung dieses Problems als Minimierungsproblem (finde eine *kürzeste* Lösungssequenz für eine Menge von Grid Mazes) ist ein schwieriges Problem. Wir nennen dieses Minimierungsproblem das Simultaneous Maze Solving Problem, kurz SIMASOP. Neben dieser allgemeinen Formulierung betrachten wir auch einen Spezialfall davon, den wir All Simultaneous Maze Solving Problem, kurz ASIMASOP, nennen. Dieses Problem ist definiert wie folgt: Gegeben n und m, finde eine kürzeste Lösungssequenz für die Menge aller lösbaren Grid Mazes der Größe $n \times m$. In dieser Abschlussarbeit analysieren wir die beiden Probleme theoretisch sowie praktisch. Neben anderen theoretischen Resultaten wird bewiesen, dass SIMASOP NP-vollständig ist, dass ASIMASOP in PSPACE ist und dass eine kubische obere Schranke für die Länge der kürzesten Lösungssequenz für ASIMASOP existiert. An praktischen Ergebnissen werden Algorithmen zur Berechnung der kürzesten oder approximiert kürzesten Lösungssequenz vorgestellt. Zusätzlich präsentieren wir einen Algorithmus der zu einer Sequenz, welche keine Lösungssequenz ist, ein ungelöstes Grid Maze findet, sowie verschiedene Algorithmen zur Berechnung von unteren Schranken. Schlussendlich evaluieren wir alle vorgestellten Algorithmen und vergleichen die Ergebnisse der verschiedenen Ansätze. Überraschenderweise ist es bereits schwer für ASIMASOP mit der Größe 4×4 eine kürzeste Lösungssequenz zu finden. Für diese Instanz können wir eine Lösungssequenz der Länge 29 mit einer entsprechenden unteren Schranke von 26 berechnen.

B Solving Sequences and Lower Bound Certificates

4×4

The shortest Solving Sequence we found for ASIMASOP with size 4×4 has length 29:

ddrddrdrurrluruurrdrrldrddrdd

In Figure B.1 you can see the set of mazes that induces a lower bound of 26 for ASIMASOP with size 4×4.

5×5

The shortest Solving Sequence we found for ASIMASOP with size 5×5 has length 551:

*ddddrrrrurrduurrdduuurrddduuuurrddddduuluurrdrddddduulluurrrdrddddulluururrdrddd
drddddlluuururrdrddddrddddlddrddrdddururrddddluururrdddurrddddullddrdrddurrdddll
uururrddrdddurrddddlddrlddrulllddddrrdrlddrrdrurrdrllddrdruullddddrdrullldddrrdrululld
ddrrdrurrrduurrddrdluurrdrdrllllddrrrrddurrdduullllddddrrdrrulullldddrrdrrurrdrrldllddr
drruulldlddrdrrluullddddrdrrullldddrrdrrullulldddrrdrrlululldddrrdrrururrddrddrullllddrd
rrrllulllddrdrrrlluullddddrdrrrurrdrrdllldddrrrlddrrrurrdllldddrrrdrruulllddlddrrrrdldlllddrrr
rllddrrrrluullddlddrrrrlulldldddrrrruururrddd*

In Figure B.2 you can see the set of mazes that induces a lower bound of 29 for ASIMASOP with size 5×5.

6×6

The shortest Solving Sequence we found for ASIMASOP with size 6×6 has length 3728:

*dddddrrrrurrduurrdduuurrddduuuurrdddduuuuurrddddduuuluurrdrddddduuulluurrrdr
dddddduuulluurrrrdrddddduulluururrrdrdddddrdddddulluuururrrdrdddddrdddddllluuu
ururrrdrdddddrddddurrrdrddddullluuururrrdrddddlddrdduuluuurrddrdddduuluuur
rddrdddddrddddrdddlddrrurrdrddddurrddddduulluurrurrdrddddurrdddduurrddrdddddlll
uururrurrdrddddurrddddduurrddrddddlddrddduluuuurrdrdddddldlluuurrrrdrddddlddll
uuuurrrrdrdddddldddlluuuurururrrdrddddrdddddldddrddullddrdrdddullddrrdrdddullluu
rruurrdrddddulllddrrrdrdddrdddlluururrrdrddrdddllluururrrdrddrddullllddrrrrdrdddu
rrddddduururrdrdddllddrrdrddlluluuurrdrrdrdduuuurrdrrdrdddlddrrdrddlddrrurrrdddllul
uuuurrddrrdrdduuuuurrddrrdrdddllddddrrurrrdddduurrddrddddurrdddddllluurrurrrdddduu
uuurrddrddddddlllddrrddrdluuuururrrddddldddrdllddrrddrdddluuurrddrdddduullddddrdrduu
lllddrdrdrdululllddrdrdrdduullllddrrdrdrdulullddrrdrdrddlulluurruurrdrdddduulllllddrrr
drdrdulullllddrrrdrdrddllluuuuurrdddrdrdduruurrdrddddllllddrdrdrdrdllluuurruurrdrd
dddduulldlddrrdrddluluuurrddrrdrddlddrrddrduulllddrdrrdrdullulllddrdrrdrdllullddrdrrd
rddlulllddrrdrrdrddurrdrldlddrdrlullllddddrrrrdrrurrdrulullllddddrrrdrdruullllldddrrrrd
rrurrdrululllllddddrrrdrrurrdrllllddrrrdrdrllllddrrrdrdrullulllddddrrrrdrrurrdrllullldddd
rrrrdrrurrdrullullullddddrrrrdrrurrdrllullddddrrrrdrrurrdruuulldddddrdrllldddddrrdrdrdruul
lllddddrrrdrdrurrddrdruuullllldddddrrrrdrrurrdrullldldddrrrdrulullldldddrrrdrlllddrrrrdru
ululldddddrrdruluullddddrrdrlulllddddrrrdrdrurrdrulululllddddrrrdrdrurrdrlulldldddrrrrdr
rurrdruullullldldddrrrrdrrurrdruullullldddddrrrrdrlulullldddddrrrrdruluullllddddrrrrdrurrdr
urrdrululllullddddrrrrdrururrddurrddrduuurrdddrduuuurrddddrduluurrdrddrlluururrdrd
drllddrrdrrdurrddluuluurrdrddrdrurrrddlddrdrluuluuurrddrddrdrldlddrdrdluluuururrrdd
lddrdrrdrdrldlddrrdrllllllddrrrrrddlluuuururrrddlddrdrddlddrdrdluuulldddddrdrrululldddddr
rdrrdrdrrllullllddddrrrdrrlullllddddrrrdrrluuulllllddddrrrdrrluluulldddddrrdrrulluullddddrr
drrlulllddddrrrdrrluulullllddddrrrdrrurrdrrdllulullldddddrrrdrrllulullldddddrrrdrrullulullldddd
rrrdrrllllldddrrrdrrurrdrrurrddrlddruuruurrddddrdurrdlddrrduuurrdrddruluurrdrddrruu
luluurrdrdrddddurrdddddrdddduulullluurrrdrdrddddurrddddddrdddulllddrddrrdlddrrdululu
uurrdrddrddddululluurrrurrddddduuulldldddrrdrlddrrdrluurrruuurrdddddduuulllllddrrddd
rrdrddrrdrurrduuuluurrdrdddddldldddrruuullllllddrrrdddrrdrluururrdddrrurrduuuluurrd
rddddduuuurrdddrdrrldlluurururrdrddddurrdrdddddlllllddddrrdrrrlulluururrdrrdddurrd
ddduurrddddddrdddlddrddduullddrddrdddlluuullddddrdrrrllllddddrrdrrrllulullddddrrdrrrurr
drrrlluuullllddddrrdrrrlluluullddddrrdrrrlulluullddddrrdrrrllluullddddrrdrrruuurrruurrdd
ddduurrddddduurrddddurrdddrddullddrdrrddrdrdrddurrdddrdlddrdlulluuurrdrdrrddrduu
urrdddddrdullllllddrrddrrrluuulldldddrddrrrdrrllddrrdrduuurrddrrdrdlulluuuurrrdrdddrdd
ddurrddddddlddrdduurrdddddlddrdddrddddlddrdrddllllddrdrdrdrdullddrdrdduuurrdddddu
ulllddrdrdrdulullddrdrdrddrdlddrdllllddrrrdrrrurrdlluluurrdrrdrddlulluurrrdrdrddddur
rddddddrdddrduluulllllddddrdrrrrluulullllddddrdrrrrllluuullddddrdrrrruruullllddddrdrrrrllul
uuurrddrrdrduluuuurrdddrrdrdurrddrduulllullddddrdrrrrulllulllddddrdrrrrlldddrrrdlddrr*

ruurrdddrrrurrrdrlldllddrrrrlllllddrddrrrrurrdlldlddrrurrdululluuurrrdrddrddrddurrdddd duurrddddlddruuulldlllddrddrrrrluuuluurrdrddrrddurrddrdrddluluuluurrrdddrdrdduuul uurrrdddrdrddlddrdllddddrdrdurrddulldddrddrluuluuurrdrddrrddlululuuurrrdddrdrddllu uluuurrrdddrdrddurrddlldddrrddlllddrdrdrdrduuullddllddrrdrrllddrrdrrlddrrdrrdurrrrdlu lldlddrrrrdluulldlddrrdrrurrurrdduululldd ldddrrrrluuullddldddrrrrllulllddrdddrrrrurrrrd uluullldddd rdrrrlluuuluurrrdddrdrddlddrrrrrullddlddrrrrrurrdrluuulldldddd rrrrurrdrrulu ulldlddrrddrrdrurrddrrdrulldllllddrrrrrlddrrrrurrdrrlluulldldddrrrrrdrrrrlldllddrrrdrurrru rrdduurrdddrruulldllddrrdrrllullddldddrrrrr

In Figure B.3 you can see the set of mazes that induces a lower bound of 31 for ASimasop with size 6×6.

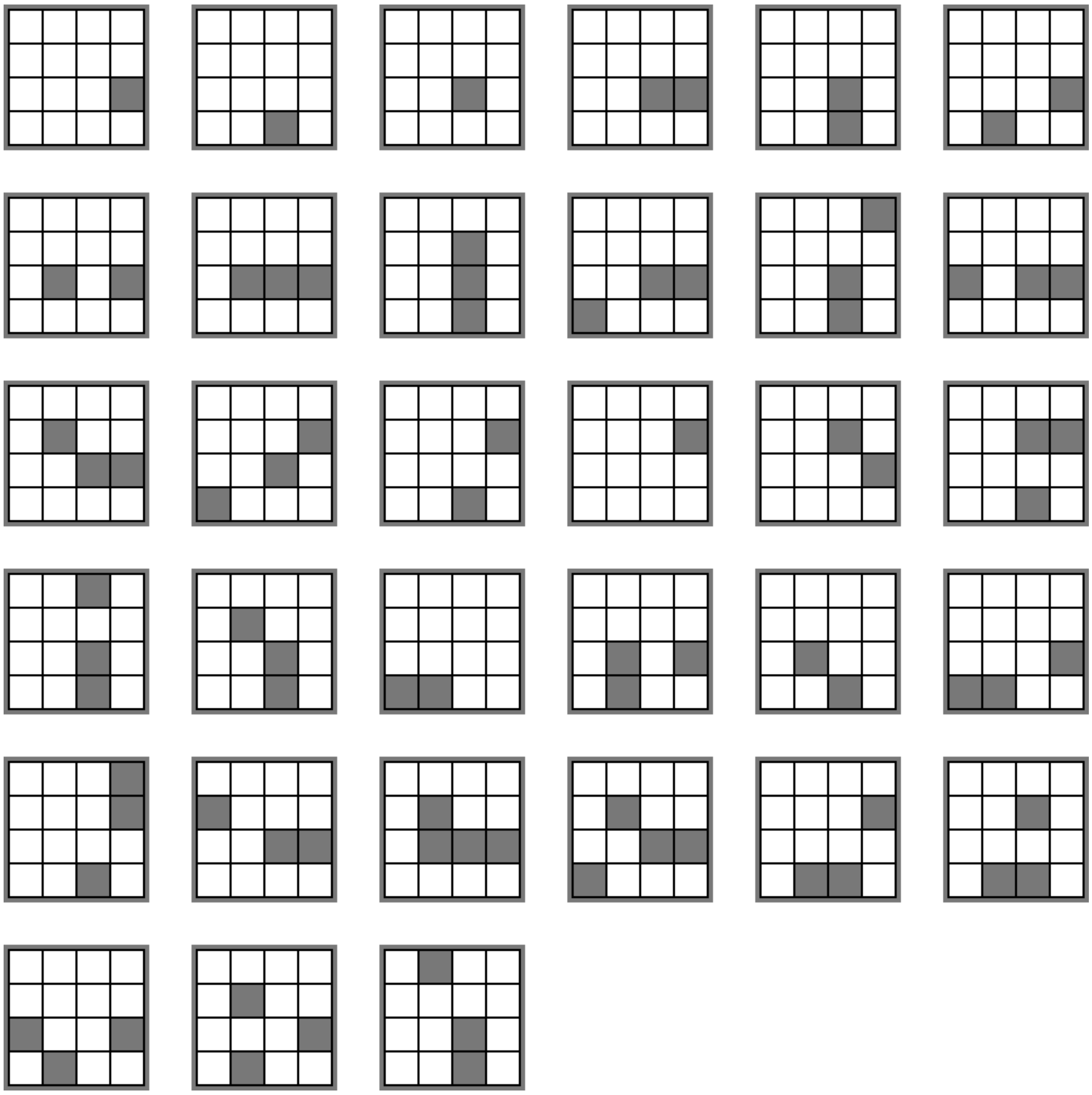

Figure B.1: Lower bound certificate for ASIMASOP with size 4×4. The shortest Solving Sequence of this set of mazes is 26.

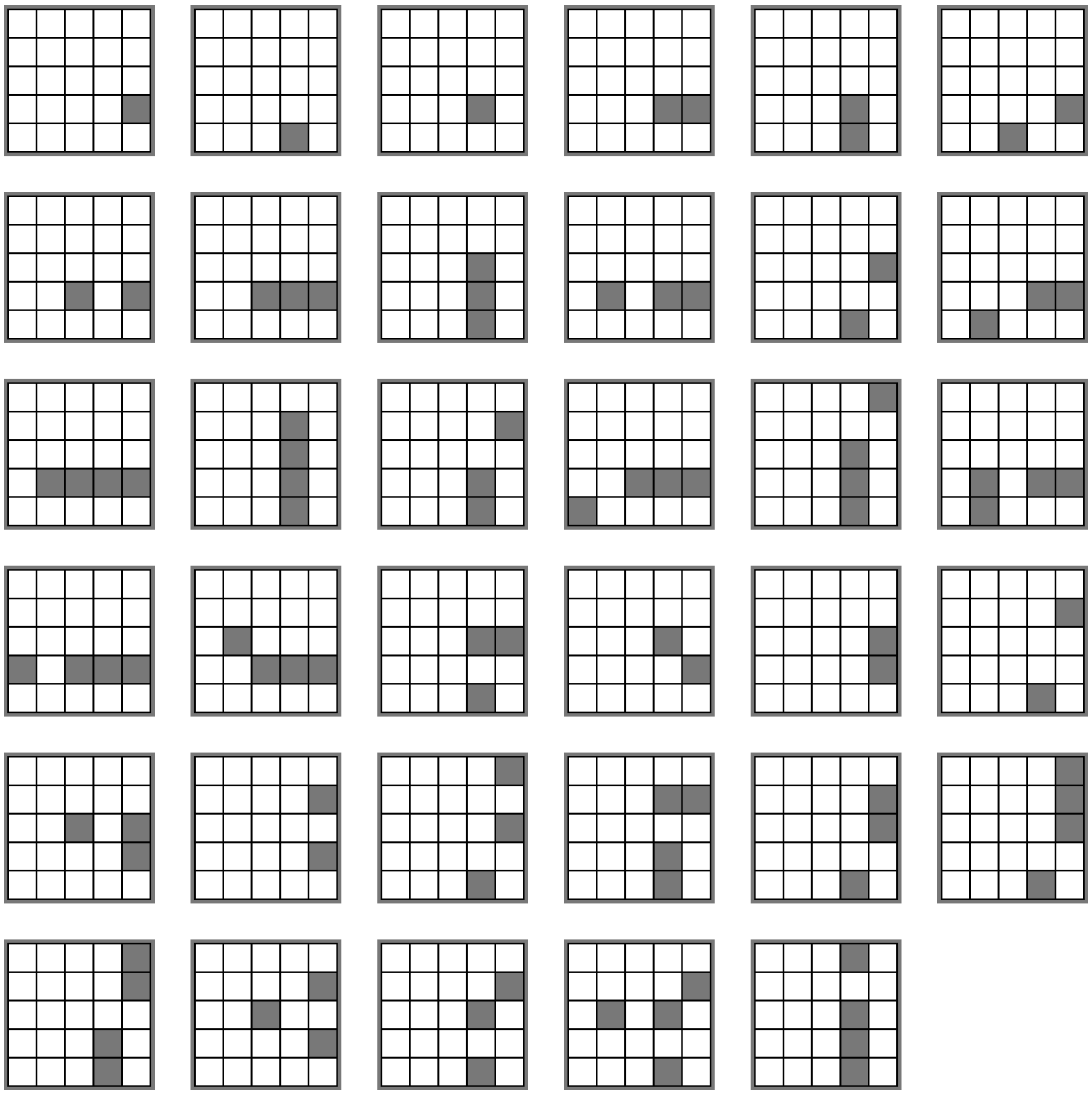
Figure B.2: Lower bound certificate for ASIMASOP with size 5×5. The shortest Solving Sequence of this set of mazes is 29.

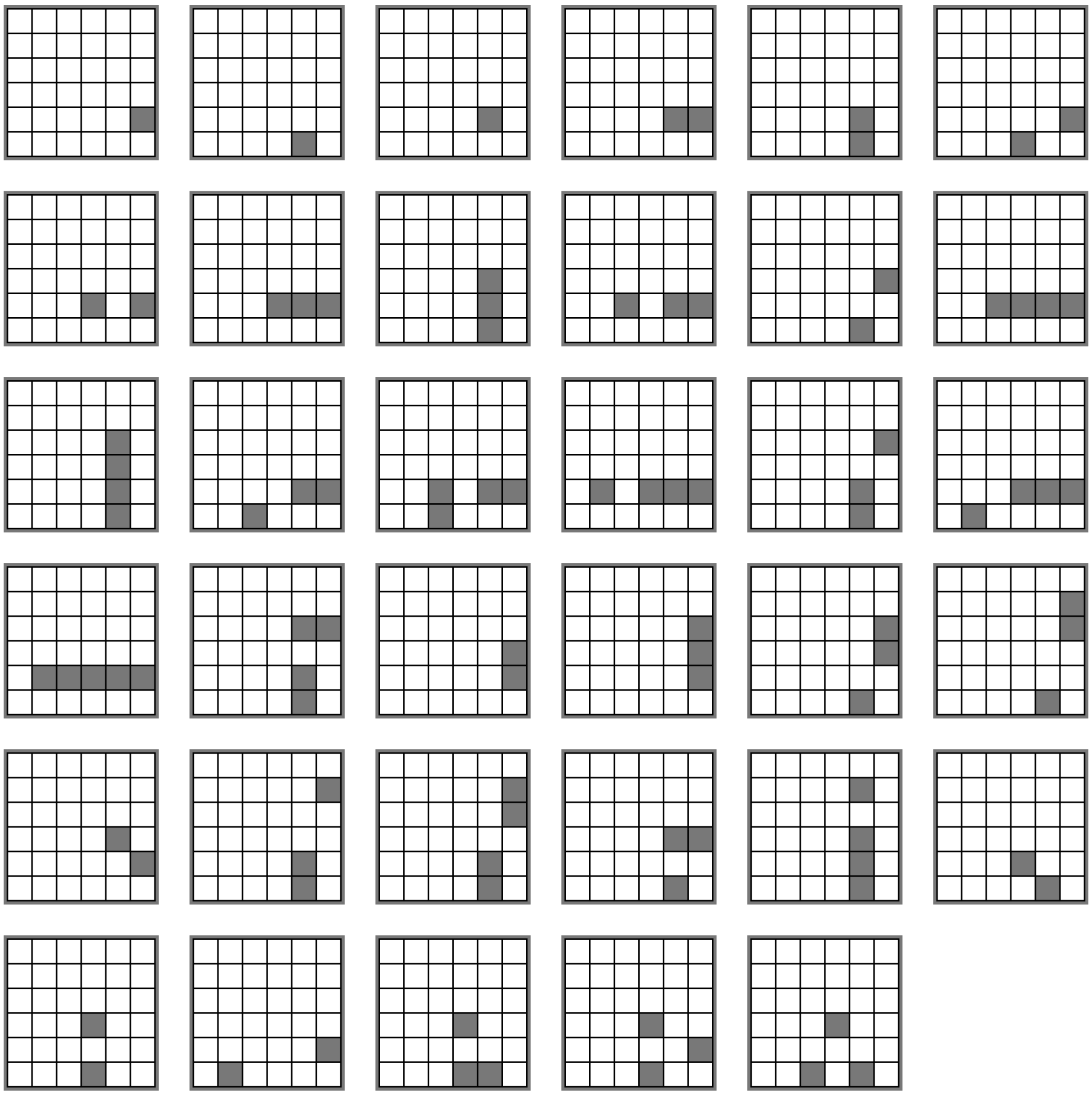

Figure B.3: Lower bound certificate for ASIMASOP with size 6×6. The shortest Solving Sequence of this set of mazes is 31.

Bibliography

[AKL+79] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, C. Rackoff. “Random walks, universal traversal sequences, and the complexity of maze problems.” In: *Foundations of Computer Science, 1979., 20th Annual Symposium on*. Oct. 1979, pp. 218–223 (cit. on pp. 21, 22, 26, 27, 38).

[Dij59] E. Dijkstra. “A Note on Two Problems in Connexion with Graphs.” ger. In: *Numerische Mathematik* 1 (1959), pp. 269–271. URL: http://eudml.org/doc/131436 (cit. on p. 42).

[GH05] A. V. Goldberg, C. Harrelson. “Computing the shortest path: A search meets graph theory.” In: *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2005, pp. 156–165 (cit. on p. 42).

[Hen64] F. C. Hennine. “Fault detecting experiments for sequential circuits.” In: *Switching Circuit Theory and Logical Design, 1964 Proceedings of the Fifth Annual Symposium on*. Nov. 1964, pp. 95–110 (cit. on p. 20).

[HKRS97] M. R. Henzinger, P. Klein, S. Rao, S. Subramanian. “Faster shortest-path algorithms for planar graphs.” In: *journal of computer and system sciences* 55.1 (1997), pp. 3–23 (cit. on p. 41).

[HNR68] P. E. Hart, N. J. Nilsson, B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (July 1968), pp. 100–107 (cit. on p. 42).

[MR10] R. Motwani, P. Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010 (cit. on pp. 14, 16, 38).

[Puz] Puzzling Stack Exchange (users: Mike Earnest, xnor). *Maze Solving Robot*. Original URL: http://puzzling.stackexchange.com/questions/18009/maze-solving-robot. Snapshot URL: http://web.archive.org/web/20160530205519/http://puzzling.stackexchange.com/questions/18009/maze-solving-robot. (cit. on p. 27).

[Sci] Science Alert (author: BEC CREW). *The robots sent into Fukushima have 'died'*. Original URL: http://www.sciencealert.com/the-robots-sent-into-fukushima-have-died. Snapshot URL: http://web.archive.org/web/20160530211428/http://www.sciencealert.com/the-robots-sent-into-fukushima-have-died. (cit. on p. 7).

[Staa] Stack Overflow (user: Olavi Mustanoja). *Solve all 4×4 mazes simultaneously with least moves*. Original URL: http://stackoverflow.com/questions/26910401/solve-all-4x4-mazes-simultaneously-with-least-moves. Snapshot URL: http://web.archive.org/web/20160530211403/http://stackoverflow.com/questions/26910401/solve-all-4x4-mazes-simultaneously-with-least-moves. (cit. on p. 7).

[Stab] Stack Overflow (user: schnaader). *Solve all 4×4 mazes simultaneously with least moves*. Original URL: http://stackoverflow.com/a/27231225. Snapshot URL: http://web.archive.org/web/20160530211635/http://stackoverflow.com/questions/26910401/solve-all-4x4-mazes-simultaneously-with-least-moves/27231225. (cit. on p. 7).

[Stac] Stack Overflow (user: templatetypedef). *Solve all 4×4 mazes simultaneously with least moves*. Original URL: http://stackoverflow.com/a/26914767. Snapshot URL: http://web.archive.org/web/20160530211649/http://stackoverflow.com/questions/26910401/solve-all-4x4-mazes-simultaneously-with-least-moves/26914767. (cit. on p. 42).

[Stad] Stack Overflow (user: Vincent van der Weele). *Solve all 4×4 mazes simultaneously with least moves*. Original URL: http://stackoverflow.com/a/26920984. Snapshot URL: http://web.archive.org/web/20160530211655/http://stackoverflow.com/questions/26910401/solve-all-4x4-mazes-simultaneously-with-least-moves/26920984. (cit. on p. 43).

[Sto13] S. Storandt. "Contraction hierarchies on grid graphs." In: *KI 2013: Advances in Artificial Intelligence*. Springer, 2013, pp. 236–247 (cit. on p. 41).

[The] The On-Line Encyclopedia of Integer Sequences (user: R. H. Hardin). *Number of $n \times n$ binary arrays with path of adjacent 1's from upper right corner to lower left corner*. Original URL: https://oeis.org/A069343. Snapshot URL: http://web.archive.org/web/20160530211705/https://oeis.org/A069343. (cit. on p. 34).

[XKCa] XKCD Forums (user: Lopsidation). *Solve ALL of the Mazes [solutions]*. Original URL: http://forums.xkcd.com/viewtopic.php?f=3&t=99534. Snapshot URL: http://web.archive.org/web/20160530211714/http://forums.xkcd.com/viewtopic.php?f=3&t=99534. (cit. on pp. 7, 27).

[XKCb] XKCD Forums (user: notzeb). *Solve ALL of the Mazes [solutions]*. Original URL: http://forums.xkcd.com/viewtopic.php?f=3&t=99534#p3249295. Snapshot URL: http://web.archive.org/web/20160530211714/http://forums.xkcd.com/viewtopic.php?f=3&t=99534. (cit. on p. 31).

All links were last followed on May 30, 2016.

Declaration