Solving the 15-Puzzle Game Using Local Value-Iteration

Bastian Bischoff[1], Duy Nguyen-Tuong[1], Heiner Markert[1], and Alois Knoll[2]

[1] Robert Bosch GmbH, Corporate Research,
Robert-Bosch-Str. 2, 71701 Schwieberdingen, Germany
[2] TU Munich, Robotics and Embedded Systems,
Boltzmannstr. 3, 85748 Garching at Munich, Germany

Abstract. The 15-puzzle is a well-known game which has a long history stretching back in the 1870's. The goal of the game is to arrange a shuffled set of 15 numbered tiles in ascending order, by sliding tiles into the one vacant space on a 4×4 grid. In this paper, we study how Reinforcement Learning can be employed to solve the 15-puzzle problem. Mathematically, this problem can be described as a Markov Decision Process with the states being puzzle configurations. This leads to a large state space with approximately 10^{13} elements. In order to deal with this large state space, we present a local variation of the Value-Iteration approach appropriate to solve the 15-puzzle starting from arbitrary configurations. Furthermore, we provide a theoretical analysis of the proposed strategy for solving the 15-puzzle problem. The feasibility of the approach and the plausibility of the analysis are additionally shown by simulation results.

1 Introduction

The 15-puzzle is a sliding puzzle invented by Samuel Loyd in 1870's [4]. In this game, 15 tiles are arranged on a 4×4 grid with one vacant space. The tiles are numbered from 1 to 15. Figure 1 left shows a possible configuration of the puzzle. The state of the puzzle can be changed by sliding one of the numbered tiles – adjacent to the vacant space – into the vacant space. The action is denoted by the direction, in which the numbered tile is moved. For each state, the set of possible actions A_s is therefore a subset of $\{up, down, left, right\}$. The goal is to get the puzzle to the final state shown in Fig. 1 right by applying a sequence of actions. A puzzle configuration is considered as *solvable*, if there exists a sequence of actions which leads to the goal configuration. This holds true for exactly half of all possible puzzle configurations [10].

Solving the 15-puzzle problem has been thoroughly investigated in the optimization community [5]. Search algorithms, such as A^* and IDA^* [6], can be employed to find feasible solutions. The major difficulty of the game is the size of the state space. The 15-puzzle has $(16)! \approx 2 \cdot 10^{13}$ different states. Even optimal solutions may take up to 80 moves to solve the puzzle. Because of the huge size of the state space, a complete search is difficult and the 15-puzzle problem is one of the most popular benchmarks for heuristic search algorithms [5].

In this paper, we study how Reinforcement Learning (RL) [7, 13] can be employed to solve the 15-puzzle problem. Thus, we seek to learn a general strategy for finding solutions for all solvable puzzle configurations. It is well-known that RL suffers from the problem of large state spaces [7]. It is therefore difficult to straightforwardly employ state-of-the-art RL algorithms for solving the 15-puzzle problem. However, in [1] Pizlo and Li analysed how humans deal with this enormous state space of the 15-puzzle. Their study shows that humans try to locally solve the puzzle tile by tile. While this approach does not always provide optimal solutions, it significantly reduces the complexity of the problem. Inspired by the work in [1], we propose a local variation of the Value-Iteration approach appropriate for solving the 15-puzzle problem. Furthermore, we provide some theoretical analysis of the proposed strategy in this study.

The remainder of the paper is organized as follows: in the next section, we introduce a formal notation for the 15-puzzle problem. In Section 3, we briefly review the basic idea behind RL. In Section 4, we introduce the local Value-Iteration approach to solve the puzzle, followed by a section with an analysis of the puzzle and the proposed approach. In Section 6, simulation results are provided supporting the analysis and showing the feasibility of the proposed approach. A conclusion is given in Section 7.

2 Notation

Let Π_{16} be the set of all possible permutations on the set $\{1, 2, \ldots, 16\}$. If we map the vacant space of the 15-puzzle to 16 and all tiles to its corresponding number, we can interpret a given puzzle configuration as a permutation $\pi \in \Pi_{16}$ by reading the tile-numbers row-wise. For example, for the left puzzle in Fig. 1 we have $\pi = (15, 10, 16, 13, 11, 4, 1, 12, 3, 7, 9, 8, 2, 14, 6, 5)$. As shown in [10], exactly half of the permutations correspond to *solvable* puzzles, i.e. puzzles that can be brought to ascending order by a sequence of sliding actions ([10] also provide a simple criterion to check for solvability). We define the state space of the 15-puzzle as the set $S_{15} \subset \Pi_{16}$ of all solvable puzzles. The tile on position i, i.e. i-th entry of the permutation $s \in S_{15}$, is denoted by $s(i)$. The position of tile i is written as $s^{-1}(i)$ (note that $s^{-1}(16)$ gives the position of the vacant space). This implies $s(4) = 13$ and $s^{-1}(7) = 10$ for the example given in Fig. 1 on the left. The goal state of the 15-puzzle is defined as state $s_{\text{goal}} \in S_{15}$ with $s_{\text{goal}}(i) = i$ for all $i = 1, \ldots, 16$, as shown in Fig. 1 on the right. Depending on the configuration of the puzzle, the possible action set is a subset of $\{\mathit{up}, \mathit{down}, \mathit{left}, \mathit{right}\}$. In the permutation s, each action corresponds to a transposition, i.e. a switch of two elements of the permutation. Formally, a transposition is defined as permutation τ with $\tau(i) \neq i$ for exactly two elements i, j and therefore is denoted by $\tau = (i, j)$. Thus, the application of each action *left*, *right*, *up*, *down* (which describes the movement direction of the numbered tile into the vacant space) corresponds to the concatenation of the state permutation s with a corresponding transposition τ, $s \circ \tau$. Given a state s, we define transpositions corresponding to each actions and provide conditions when each action is applicable.

15	10		13
11	4	1	12
3	7	9	8
2	14	6	5

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Fig. 1. A possible start state (left) and the goal state (right) of the 15-puzzle.

- *left*: $\tau_\ell = \left(s^{-1}(16),\, s^{-1}(16)+1\right)$ applicable iff $s^{-1}(16) \not\equiv 0 \bmod 4$
- *right*: $\tau_r = \left(s^{-1}(16),\, s^{-1}(16)-1\right)$ applicable iff $s^{-1}(16) \not\equiv 1 \bmod 4$
- *up*: $\tau_u = \left(s^{-1}(16),\, s^{-1}(16)+4\right)$ applicable iff $s^{-1}(16) \leq 12$
- *down*: $\tau_d = \left(s^{-1}(16),\, s^{-1}(16)-4\right)$ applicable iff $s^{-1}(16) \geq 5$

Elements of the set of possible actions $a \in A_s$ are the transpositions τ that perform the corresponding switch of elements. Instead of writing $s' = s \circ \tau$, the action corresponding to τ can also be given as $f(s,a) = s'$, where $f(\cdot)$ is the so-called transition function or dynamics model.

3 Reinforcement Learning: A Brief Review

In this section, we will briefly introduce RL, see [7, 13] for detailed introductions. Reinforcement Learning is a sub area of machine learning that deals with goal directed learning. The idea is to use an environment feedback about the desirability of states, and let the agent learn to find a task solution while optimizing the overall gathered reward.

The *state* of a learning agent is given by $s \in S$, in each state *actions* $a \in A_s \subseteq A$ can be applied. A *reward function* $r : s \mapsto \mathbb{R}$ defines the desirability of states and hence encodes the learning goal. The function $f : s, a \mapsto s'$ describes the *dynamics* of the considered system. The goal of the learning agent is to find a policy $\pi : s \mapsto a$ which maximizes the long term reward $\sum_{i=0}^{\infty} \gamma^i r(s_i)$. Here, $0 < \gamma < 1$ is a discount factor, s_0 is the start state and $s_i = f(s_{i-1}, \pi(s_{i-1}))$ holds. Many Reinforcement Learning algorithms are based on a value-function $V^\pi(s_0) = \sum_{i=0}^{\infty} \gamma^i r(s_i)$ with $s_i = f(s_{i-1}, \pi(s_{i-1}))$, which encodes the long term reward starting in state s_0 and acting according to policy π. The *optimal* value-function V is defined as $V(s) = \max_\pi \{V^\pi(s)\}$, the optimal policy can be derived by $\pi(s) = \arg\max_{a \in A_s} \{V(s') \mid s' = f(s,a)\}$. The optimal value-function $V(\cdot)$ is latent. In the literature, different ways to estimate $V(\cdot)$ can be found. A well-studied approach for the estimation of $V(\cdot)$ is *Value-Iteration* which is used in

various applications [14, 15]. It is given by an iterative application of

$$\forall s \in S : V_{t+1}(s) \leftarrow r(s) + \gamma \cdot \max_{a \in A_s} \{V_t(s') \mid s' = f(s,a)\}$$

until convergence of V_t. It is guaranteed that V_t converges to the optimal value-function V for $t \to \infty$ [7].

In the context of the 15-puzzle problem, the state space S corresponds to the puzzle-configurations S_{15}, an action $a \in A_s$ is the sliding of a numbered tile into the vacant space – as explained in the previous section. The reward function is defined as

$$r(s) = \begin{cases} 1, & \text{if } s = s_{goal} \\ 0, & \text{else.} \end{cases}$$

With these definitions, Value-Iteration can be performed. The policy $\pi(s) : s \mapsto a$ resulting from the learned value-function $V(\cdot)$ returns for every puzzle configuration s the sliding action a for reaching the goal-state.

While this approach theoretically gives us optimal solutions to all solvable puzzles, it is practically intractable. The reason is the size of the state space S_{15} with $|S_{15}| \approx 10^{13}$. The Value-Iteration approach will iterate over all possible states, which obviously cannot be performed. In the next section, we describe how the computational effort can be reduced significantly by decomposing the problem into tractable subproblems.

4 Local Value-Iteration for Solving the 15-Puzzle

The work by Pizlo and Li [1] analysed how humans deal with the enormous state space of the 15-puzzle. The study shows that humans try to locally solve the puzzle tile by tile. While this approach does not provide optimal solutions, it significantly reduces the complexity of the problem [1]. The basic principle behind the solution strategy is to hierarchically divide the global problem into small local problems (e.g. move each tile to its correct position sequentially). As we will see later, it is not always sufficient to only consider one single tile. Instead of that, multiple tiles have to be considered at once in some scenarios (see Sect. 5). Inspired by the work in [1], we present a RL approach using local state space information, similar to the ideas given in [2, 3]. Thus, when learning a policy to move a given tile to its correct position, we consider only a limited number of possible puzzle configurations. Furthermore, the size of the local region can be set adaptively. After solving the local subproblems, the complete solution is obtained by sequential application of the local solutions.

A local subproblem can be defined by a set $G = \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, 15\}$ of k tiles which need to be moved to distinct positions $i_1, \ldots, i_k$ without moving tiles $R = \{j_1, \ldots, j_\ell\}$. For example, for $G = \{3\}$ and $R = \{1, 2\}$ the G, R-subproblem is to bring tile 3 to its correct position without moving tiles 1 and 2. Here, $R \cap G = \emptyset$ must hold, the free space must not be part of R and we restrict the action sets A_s according to R (to prevent that a tile from the set R is moved).

Algorithm 1 Value-Iteration to Solve the Local G, R-problem

function VALUE_ITERATION($S_{15}^{G,R}$)
  $V_0^{G,R} \leftarrow 0, \quad t \leftarrow 0$ ▷ Initial value-function
  repeat
    for all $s \in S_{15}^{G,R}$ **do** ▷ Apply Value-Iteration Update step on all states
      $V_{t+1}^{G,R}(s) \leftarrow r(s) + \gamma \max_{a \in A_s} \left\{ V_t^{G,R}(s') \,|\, s' = \overline{f}(s,a),\ s' \in S_n^{G,R} \right\}$
    end for
    $t \leftarrow t + 1$
  until $V_t^{G,R}$ has converged
  $\pi^{G,R}(\cdot) \leftarrow \arg\max_{a \in A_s} \left\{ V^{G,R}(s') \,|\, s' = \overline{f}(\cdot, a) \right\}$
    ▷ Derive optimal policy $\pi^{G,R} : S_{15}^{G,R} \rightarrow A$
  return $V_t^{G,R}, \pi^{G,R}$
end function

When moving the tiles in the local set G to their correct positions, we do not need to keep track of the positions of all other tiles of the puzzle. Thus, the state of the G, R-subproblem has the form $(i_1, \ldots, i_k, j)$, i.e. the positions of the k tiles in G including the position of the free space. The number of elements of the state space $S_{15}^{G,R}$ can be given as $|S_{15}^{G,R}| = \frac{1}{2}(16-\ell)(16-\ell-1)\ldots(16-\ell-k)$ with $\ell = |R|$. Note, that the factor $\frac{1}{2}$ bases on the fact that only half of all permutations are solvable [10]. In general, the $S_{15}^{G,R}$ state space is significantly smaller than S_{15}. For example, the state space to move tiles 6 and 7 to correct positions without moving tiles $\{1, \ldots, 5\}$ has only 495 elements (compared to 10^{13} elements in S_{15}).

As the G, R-subproblems have significantly smaller state spaces, we can employ state-of-the-art RL algorithms, such as Value-Iteration shown in Algorithm 1, for learning a policy to move a given tile to its desired position. Here, $\overline{f}$ is the dynamics function applied on elements $s \in S_{15}^{G,R}$, the reward function is defined as given in Sect. 3. The algorithm returns the value-function $V^{G,R}$ and the optimal local policy $\pi^{G,R}$. To solve the G, R-subproblem for a given puzzle configuration s, we subsequently apply actions $a = \pi^{G,R}(s)$ until all tiles in G are at their correct positions.

Until now, we described how Value-Iteration can be employed on local G, R-subproblems for learning an optimal local policy $\pi^{G,R}$. In the next step, we discuss an approach to determine the sets G and R for a given puzzle configuration. A naive approach would be to successively set $G = \{i\}$ for $i = 1, \ldots, 15$ and $R = \{1, \ldots, i-1\}$. That would mean solving the puzzle tile by tile, while fixing all lower numbered tiles. However, this simple approach does not work for many puzzles, as we will show in the next section. Thus, we need to adaptively vary the local region of G and possibly consider many tiles at once. A systematic approach is to successively move tiles from set R to G until a solution is found. Thus, if no solution is found for $G = \{i\}$ and $R = \{1, \ldots, i-1\}$, we first set $G = G \cup \{i-1\}$ and $R = R \setminus \{i-1\}$ and continue to increase G and decrease R by setting $G = G \cup \max(R)$ and $R = R \setminus \max(R)$. This procedure is done until

Algorithm 2 Local Value-Iteration for Solving the 15-puzzle

function SOLVE_PUZZLE(start_state)
  $s \leftarrow$ start_state
  for $i \leftarrow 1, \ldots, 15$ **do**
    $G_i \leftarrow \{i\}$, $R_i \leftarrow \{1, \ldots, i-1\}$ ▷ Successively move tiles to correct positions
    $V^{G_i,R_i}, \pi^{G_i,R_i} \leftarrow$ VALUE_ITERATION$(S_{15}^{G_i,R_i})$
    while $V^{G_i,R_i}(s) = 0$ **do** ▷ holds iff it is not possible to solve G_i, R_i for s
      $G_i = G_i \cup \max(R_i)$ ▷ move tile with highest number from set R_i to G_i
      $R_i = R_i \setminus \max(R_i)$
      $V^{G_i,R_i}, \pi^{G_i,R_i} \leftarrow$ VALUE_ITERATION$(S_{15}^{G_i,R_i})$
    end while
    ▷ Now solve the part of the puzzle corresponding to G_i, R_i
    while $\exists i \in G_i : s(i) \neq i$ **do** ▷ while at least one tile of G_i is not at its goal position
      $a_{\text{best}} \leftarrow \pi^{G_i,R_i}(s)$ ▷ Policy π returns the action to solve G_i, R_i
      $s \leftarrow f(s, a_{\text{best}})$ ▷ Apply sliding action
    end while
  end for
end function

a solution is found. If the puzzle is solvable, this approach guarantees to find a solution, because you will finally end up with $R = \emptyset$. This idea of increasing the local region gives us the following technique to solve an arbitrary puzzle:
Given a solvable starting puzzle configuration s, we successively try to solve $G = \{i\}$ for $i = 1, \ldots, 15$ and $R = \{1, \ldots, i-1\}$ while employing Value-Iteration on $S^{G,R}$, as given in Algorithm 1. We can move the tiles in G to their correct positions without moving tiles in the set R, if and only if $V^{G,R}(s) \neq 0$. In this case, we just apply the corresponding policy $\pi^{G,R}$ to bring the tiles to their correct positions. Otherwise, we increase the set G and decrease the set R, such as $G = G \cup \max(R)$ and $R = R \setminus \max(R)$. Subsequently, we go back to the Value-Iteration step with the new sets G, R. The resulting *local Value-Iteration* approach is summarized in Algorithm 2.

5 Analysis of Local Value-Iteration for the 15-Puzzle

As the proposed RL method bases on the well-known Value-Iteration [7], it guarantees to find a solution for any solvable 15-puzzle configuration. However, the proposed local approach involves the consideration of many tiles at times and, thus, can increase the computational complexity. In this section, we analyse the proposed algorithm and develop a bound for the maximum number of tiles that are considered at each step. Furthermore, we provide the maximal cardinality of involved state spaces and give bounds on the maximum number of actions to solve a given puzzle configuration.

Definition 1. *Consider a random puzzle $s \in S_{15}$. Let G_i be the set used in Local Value-Iteration to bring tile i to its correct position (see Algorithm 2). The*

probability, that $|G_i| = j$ holds is denoted by τ_i^j. The subset of puzzles $\mathfrak{S}^k \subseteq S_{15}$ is defined by $s \in \mathfrak{S}^k$, if and only if $\max(G_1, \ldots, G_{15}) = k$ holds. Finally, $\Phi(\mathfrak{S}^k)$ gives an upper bound on the number of actions, that are necessary to solve any puzzle $s \in \mathfrak{S}^k$ using Local Value-Iteration.

In Definition 1, the factor τ_i^j describes the probability that j tiles $\{i, \ldots, i-j+1\}$ need to be considered to bring tile i to its correct position given a random puzzle. For example, tile 2 can always be moved to position 2 without moving tile 1. Hence, it is $\tau_2^1 = 1$ and $\tau_2^\ell = 0$ for all $\ell > 1$. As we will see later, tile 4 can be moved to its correct position without moving tile 3 in $\tau_4^1 = \frac{1}{12}$ of the cases. In other cases, tile 3 needs to be moved in order to get tile 4 to the right position and, thus, $\tau_4^2 = \frac{11}{12}$. The proportions $\tau_i^1, \tau_i^2, \ldots$ always have to sum up to 1. The classes $\mathfrak{S}_k$ partition the set of all puzzle configurations. Informally, $s \in \mathfrak{S}_k$ states that the puzzle s can be solved with Local Value-Iteration by considering k tiles at once.

Theorem 1. *Given a puzzle $s \in S_{15}$, then the state spaces involved to solve the puzzle s with local Value-Iteration have at most* 10080 *elements. For $i \in \{1, 2, 3, 5, 6, 7, 9, 14, 15\}$ it holds that $\tau_i^1 = 1$. For $i \in \{4, 8, 10, 11, 12, 13\}$ τ_i^j is given as*

Tile i	τ_i^1	τ_i^2	τ_i^3	τ_i^4	τ_i^5	**Max. state space size**
4	$\frac{1}{12}$	$\frac{11}{12}$				$\left\lvert S_{15}^{\{3,4\},\{1,2\}} \right\rvert = 1092$
8	$\frac{1}{8}$	$\frac{7}{8}$				$\left\lvert S_{15}^{\{7,8\},\{1,\ldots,6\}} \right\rvert = 360$
10	$\frac{2100}{2520}$	$\frac{420}{2520}$				$\left\lvert S_{15}^{\{9,10\},\{1,\ldots,8\}} \right\rvert = 168$
11	$\frac{216}{360}$	$\frac{72}{360}$	$\frac{72}{360}$			$\left\lvert S_{15}^{\{9,10,11\},\{1,\ldots,8\}} \right\rvert = 840$
12	$\frac{15}{60}$		$\frac{30}{60}$	$\frac{15}{60}$		$\left\lvert S_{15}^{\{9,\ldots,12\},\{1,\ldots,8\}} \right\rvert = 3360$
13	$\frac{4}{12}$				$\frac{8}{12}$	$\left\lvert S_{15}^{\{9,\ldots,13\},\{1,\ldots,8\}} \right\rvert = 10080$

Proof. In the following, we provide the calculation for the probabilities τ_i^j, as well as the maximal cardinality of the state space involved for each tile i.

- Tiles $1, 2, 3, 5, 6, 7$: It is easy to check that these tiles can be moved to their goal position without moving lower numbered tiles. This implies $\tau_i^1 = 1$ for all $i \in \{1, 2, 3, 5, 6, 7\}$. The state spaces considered have the size $\left\lvert S_{15}^{\{i\},\{1,\ldots,i-1\}} \right\rvert = \frac{1}{2}(16 - i + 1)(16 - i)$, which is at most $\frac{1}{2} \cdot 16 \cdot 15 = 120$.
- Tiles $4, 8$: In $\frac{1}{13}$ (resp. $\frac{1}{9}$) of all states, tile 4 (resp. 8) is already on its correct position. Tile 4 (8) can also be brought to its correct position, if tile 4 (8) is in position 8 (12) and the vacant space is on position 4 (8). This applies to $\frac{1}{13} \cdot \frac{1}{12}$ (resp. $\frac{1}{9} \cdot \frac{1}{8}$) of all remaining puzzles with fixed tile 1 to 3. For the rest of the cases, one can check that it is not possible to bring tile 4 (resp. 8) to its position without moving lower numbered tiles. But it is sufficient to only move tile 3 (7): It is always possible to move tile 4 (8) below its correct position with the free space below tile 4 (8). The following sequence

will then bring tiles 3 and 4 (7 and 8) to its correct position: Down, Down, Right, Up, Left, Up, Right, Down, Down, Left, Up. This implies

$$\tau_4^1 = \frac{1}{13} + \frac{1}{13} \cdot \frac{1}{12} = \frac{1}{12},\ \tau_4^2 = \frac{11}{12},\ \tau_8^1 = \frac{1}{9} + \frac{1}{9} \cdot \frac{1}{8} = \frac{1}{8},\ \tau_8^2 = \frac{7}{8}.$$

The largest state space is $S_{15}^{\{3,4\},\{1,2\}}$ with $\left|S_{15}^{\{3,4\},\{1,2\}}\right| = \frac{1}{2}14 \cdot 13 \cdot 12 = 1092$.

- Tile 9: Tile 9 can be brought to its correct position by moving the vacant space circle wise, $\tau_9^1 = 1$ with $\left|S_{15}^{\{9\},\{1,\ldots,8\}}\right| = \frac{1}{2}8 \cdot 7 = 28$.
- Tiles $10, 11, 12, 13$: If the puzzle is solved up to tile 9, the remaining number of solvable puzzles is $\frac{1}{2}7! = 2520$ (as mentioned Sect. 2, only half of the states are solvable puzzles). We calculate the proportions τ by application of Algorithm 2 on all 2520 solvable puzzles.
- Tiles $14, 15$: As can be checked, it holds $\tau_{14}^1 = \tau_{15}^1 = 1$, the state spaces have 3 resp. 2 elements.

This concludes the proof for Theorem 1. The Theorem shows, that Local Value-Iteration considers in the worst-case a state space with 10080 elements – compared to 10^{13} elements of original Value-Iteration. In the next step, we investigate the puzzle classes $\mathfrak{S}^k$ introduced in Definition 1. Theorem 1 directly implies the following corollary.

Corollary 1. *It is $\tau_i^j = 0$ for all $1 \leq i \leq 15$, $j > 5$. Hence $\mathfrak{S}^k = \emptyset$ holds for all $k > 5$ and $\mathfrak{S}^1, \mathfrak{S}^2, \mathfrak{S}^3, \mathfrak{S}^4, \mathfrak{S}^5$ partition S_{15}.*

Thus, Local Value-Iteration considers at most 5 tiles at once to any given puzzle and puzzles of the class $\mathfrak{S}^k$ can be considered the most difficult to solve. With the next theorem, we investigate how the puzzles S_{15} split up into the difficulty classes $\mathfrak{S}^1$ to $\mathfrak{S}^5$.

Theorem 2. *Consider the classes $\mathfrak{S}^1, \mathfrak{S}^2, \ldots, \mathfrak{S}^5$ introduced in Definition 1. Then $\left|\mathfrak{S}^1\right| \approx 0,04\% \left|S_{15}\right|$, $\mathfrak{S}^2 \approx 6,62\% \left|S_{15}\right|$, $\mathfrak{S}^3 = 18,\overline{3}\% \left|S_{15}\right|$, $\mathfrak{S}^4 = 8,\overline{3}\% \left|S_{15}\right|$, $\mathfrak{S}^5 = 66,\overline{6}\% \left|S_{15}\right|$ holds.*

Proof. We compute the cardinalites using the proportions τ_i^j from Theorem 1:

$$\begin{aligned}
\left|\mathfrak{S}^1\right| &= \tau_4^1 \cdot \tau_8^1 \cdot \tau_{10}^1 \cdot \tau_{11}^1 \cdot \tau_{12}^1 \cdot \tau_{13}^1 \\
&= \frac{1}{2304} \left|S_{15}\right| \approx 0,04\% \left|S_{15}\right| \\
\left|\mathfrak{S}^2\right| &= \left[\tau_4^2 \cdot \left(\tau_8^1 + \tau_8^2\right) \cdot \left(\tau_{10}^1 + \tau_{10}^2\right)\right. \\
&\quad \cdot \left(\tau_{11}^1 + \tau_{11}^2\right) \\
&\quad + \tau_4^1 \cdot \tau_8^2 \cdot \left(\tau_{10}^1 + \tau_{10}^2\right) \cdot \left(\tau_{11}^1 + \tau_{11}^2\right) \\
&\quad + \tau_4^1 \cdot \tau_8^1 \cdot \tau_{10}^2 \cdot \left(\tau_{11}^1 + \tau_{11}^2\right) \\
&\quad \left. + \tau_4^1 \cdot \tau_8^1 \cdot \tau_{10}^1 \cdot \tau_{11}^2\right] \cdot \tau_{12}^1 \cdot \tau_{13}^1 \cdot \left|S_{15}\right| \\
&= \frac{763}{11520} \left|S_{15}\right| \approx 6,62 \left|S_{15}\right| \\
\left|\mathfrak{S}^3\right| &= \left[\tau_{11}^3 \left(\tau_{12}^1 + \tau_{12}^3\right)\right. \\
&\quad \left. + \left(\tau_{11}^1 + \tau_{11}^2\right) \tau_{12}^3\right] \tau_{13}^1 \left|S_{15}\right| \\
&= \frac{11}{60} \left|S_{15}\right| = 18,\overline{3}\% \left|S_{15}\right| \\
\left|\mathfrak{S}^4\right| &= \tau_{12}^4 \cdot \tau_{13}^1 \\
&= \frac{1}{12} \left|S_{15}\right| = 8,\overline{3}\% \left|S_{15}\right| \\
\left|\mathfrak{S}^5\right| &= \tau_{13}^5 \left|S_{15}\right| \\
&= \frac{2}{3} \left|S_{15}\right| = 66,\overline{6}\% \left|S_{15}\right|
\end{aligned}$$

According to Theorem 2, only 0.04% of all puzzles can be solved tile by tile without moving lower numbered tiles. On the other hand, $66,\overline{6}\%$ of all puzzle involve a situation, where 4 lower numbered tiles need to be considered to solve the puzzle. As already mentioned, the membership of a puzzle $s \in S_{15}$ in a class $\mathfrak{S}^k$ describes its difficulty in terms of how many tiles need to be considered at once to solve the puzzle with Local Value-Iteration. Another possible measure for the difficulty of a puzzle, is the number of sliding actions necessary to solve it. Now, our final step is to give upper bounds $\Phi(\mathfrak{S}^k)$ which give the maximal number of actions that need to be applied to solve any puzzle $s \in \mathfrak{S}^k$. As we will see, $\Phi(\mathfrak{S}^k)$ will go up with k meaning that puzzle from a higher class $\mathfrak{S}^k$ also potentially need more actions to be solved.

Theorem 3. *Given a puzzle $s \in \mathfrak{S}^k$. Then the maximal number of actions necessary to solve the puzzle s using Algorithm 2 is given by $\varphi(\mathfrak{S}^1) = 142$, $\varphi(\mathfrak{S}^2) = 220$, $\varphi(\mathfrak{S}^3) = 248$, $\varphi(\mathfrak{S}^4) = 255$, $\varphi(\mathfrak{S}^5) = 288$.*

Proof. We proof the statement by inspection of the value-function learned with Algorithm 1. An entry $V(s) = \sum_{t=k}^{\infty} \gamma^t$ implies, that k actions are necessary to reach the goal state given state s under an optimal policy. Let $M^{G,R}$ denote the maximal number of actions necessary to solve G, R. The relevant subproblems G, R are given in Theorem 1. By application of Algorithm 1, the following values for $M^{G,R}$ can be given:

$$
\begin{array}{lll}
M^{\{1\},\emptyset} = 21 & M^{\{8\},\{1,\ldots,7\}} = 1 & M^{\{12\},\{1,\ldots,11\}} = 1 \\
M^{\{2\},\{1\}} = 17 & M^{\{8,7\},\{1,\ldots,6\}} = 29 & M^{\{12,11,10\},\{1,\ldots,9\}} = 20 \\
M^{\{3\},\{1,2\}} = 20 & M^{\{9\},\{1,\ldots,8\}} = 15 & M^{\{12,\ldots,9\},\{1,\ldots,8\}} = 27 \\
M^{\{4\},\{1,2,3\}} = 1 & M^{\{10\},\{1,\ldots,9\}} = 9 & M^{\{13\},\{1,\ldots,12\}} = 1 \\
M^{\{4,3\},\{1,2\}} = 32 & M^{\{10,9\},\{1,\ldots,8\}} = 20 & M^{\{13,\ldots,9\},\{1,\ldots,8\}} = 34 \\
M^{\{5\},\{1,\ldots,4\}} = 17 & M^{\{11\},\{1,\ldots,10\}} = 6 & M^{\{14\},\{1,\ldots,13\}} = 1 \\
M^{\{6\},\{1,\ldots,5\}} = 13 & M^{\{11,10\},\{1,\ldots,9\}} = 14 & M^{\{15\},\{1,\ldots,14\}} = 1 \\
M^{\{7\},\{1,\ldots,6\}} = 18 & M^{\{11,10,9\},\{1,\ldots,8\}} = 23 &
\end{array}
$$

Given these maximal numbers of actions to solve a subproblem G, R, we can now estimate the worst-case, i.e. the maximal number of actions, to solve a puzzle $s \in \mathfrak{S}^k$. Therefore, we compute

$$\varphi(\mathfrak{S}^k) = \sum_{i=1}^{15} \max \left\{ M^{\{i\},\{1,\ldots,i-1\}}, \ldots, M^{\{i,\ldots,i-k+1\},\{1,\ldots,i-k\}} \right\}$$

where the max-operator only considers entries $M^{G,R}$ that are given above (other subproblems G, R are not relevant according to Theorem 1).

6 Simulation Results

This section contains experimental results when Local Value-Iteration is applied to solve the 15-Puzzle. The experiments will show the feasibility of the Local Value-Iteration approach and will also support the analysis provided in the previous section.

We will use the Local Value-Iteration Algorithm 2 to solve instances of the 15-Puzzle. Here, subproblems G, R will recur for different puzzles. Hence, we will learn the value-function and the corresponding policy only once for a given subproblem G, R. On a standard desktop PC, the training for all involved G, R-problems took 155 seconds. After the training, arbitrary puzzles can be solved. We build a set $\tilde{S}$ of 100000 random puzzles, all puzzles $s \in \tilde{S}$ could successfully be solved with Algorithm 2 with an average of approximately 0.25 seconds to solve one puzzle.

In the analysis section, Theorem 2 states the proportions of the classes $\mathfrak{S}^k$ relative to the set of all puzzles. These proportions can also be estimated by experiment: for each puzzle $s \in \tilde{S}$, we keep track of the maximum number of tiles k considered at once to solve the puzzle and subsequently sort them in the corresponding class $\tilde{\mathfrak{S}^k}$. The results after solving all 100000 random puzzles in $\tilde{S}$ are

$$\left|\tilde{\mathfrak{S}^1}\right| = 42 = 0.042\% \left|\tilde{S}\right| \qquad \left|\tilde{\mathfrak{S}^4}\right| = 8446 = 8.446\% \left|\tilde{S}\right|$$
$$\left|\tilde{\mathfrak{S}^2}\right| = 6539 = 6.539\% \left|\tilde{S}\right| \qquad \left|\tilde{\mathfrak{S}^5}\right| = 66611 = 66.611\% \left|\tilde{S}\right|$$
$$\left|\tilde{\mathfrak{S}^3}\right| = 18362 = 18.362\% \left|\tilde{S}\right|$$

These empirical proportions correspond to the values given in Theorem 2 up to errors of less than 0.15%. For each of the five classes $\tilde{\mathfrak{S}^k}$, we compute the average and maximum number of actions necessary to solve a puzzle. The results are given in Table 1 and correspond to the statements of Theorem 3. The results show, that when the number of tiles which need to be considered increases, the number of actions to solve the puzzle also increases.

Table 1. We solved 100000 puzzles with Local Value-Iteration and classified them into the classes $\mathfrak{S}^k$. The table gives the average and maximum number of actions necessary to solve puzzles of the subsequent classes.

	$\mathfrak{S}^1$	$\mathfrak{S}^2$	$\mathfrak{S}^3$	$\mathfrak{S}^4$	$\mathfrak{S}^5$	overall
average # actions	68.85	100.68	112.16	122.92	128.71	123.35
maximum # actions	107	156	165	174	202	202

So far, we start with one tile, i.e. $G_i = \{i\}$, and increased the local region, if the subproblem could not be solved. While Value-Iteration provides optimal solutions for the subproblems, the overall solution is in general not optimal

(with respect to the number of actions necessary to solve a puzzle). On the other hand, solving the complete puzzle at once, i.e. $G = \{1, \ldots, 15\}$, will give us the optimal solution, but is computationally intractable. In the next step, we vary between those two extremes and increase the initial size of the local region to find better solutions – in terms of fewer actions necessary to solve the puzzle –, while accepting higher computational efforts. For an initial local size ℓ, we define $G_i = \{\ell i - \ell + 1, \ldots, \ell i\}$. If we increase, for example, the size to $\ell = 2$, this will give us $G_1 = \{1, 2\}$, $G_2 = \{3, 4\}$, $G_3 = \{5, 6\}$ and so on. We solved the 100000 random puzzles in $\tilde{S}$ again, now with initial size from 1 to 5. Figure 2 shows on the left, that the average as well as the maximal number of actions to solve a puzzles declines as expected when we increase the region size. The cost of this improvement is shown in Fig. 2 on the right – the computation time grows exponentially. The detailed results can be found in Table 2.

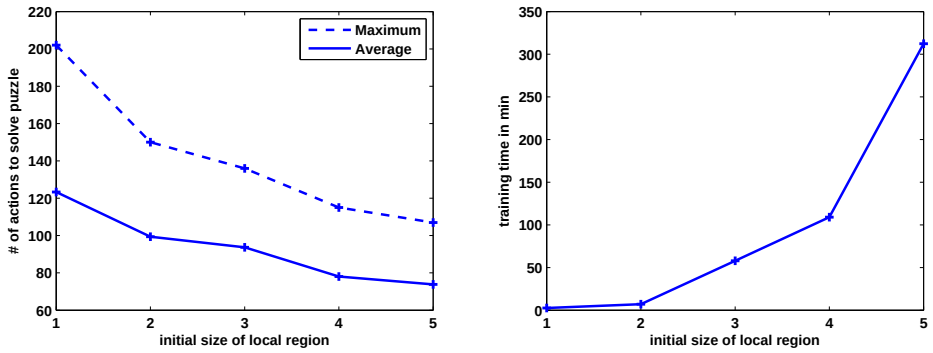

Fig. 2. The initial size of the local region G for Local Value-Iteration is varied from 1, i.e. initial $G_i = \{i\}$, up to 5, i.e. initial $G_i = \{5i - 4, \ldots, 5i\}$. The left figure shows the average as well as maximal number of actions to solve 100000 random puzzles. On the right, the necessary overall training time for each initial local size is shown.

Table 2. We adapted the initial local size from 1, i.e. $G_i = \{i\}$ for $i = 1, \ldots, 15$, up to 5, i.e. $G_i = \{5i - 4, \ldots 5i\}$ for $i = 1, \ldots 3$. We solved 100000 puzzles with each initial local size, the table gives the average and maximum number of actions necessary to solve solve the puzzles. The last row states the necessary training time on a standard desktop PC.

default size local region	1	2	3	4	5
average # actions	123.35	99.36	93.69	78.07	73.82
maximum # actions	202	150	136	115	107
training time in minutes	2.58	7.10	57.87	109.03	312.33

7 Conclusion

In this study, we investigate the possibility of using RL to solve the popular 15-puzzle game. Due to the high state space dimension of the problem, it is difficult to straightforwardly employ state-of-the-art RL algorithms. In order to deal with this large state space, we proposed a local variation of the well-known Value-Iteration appropriate to solve the 15-puzzle problem. Our algorithm is inspired by the insight that humans use to solve the 15-puzzle game locally, by sequentially moving tiles to their correct positions. Furthermore, we provide a theoretical analysis showing the feasibility of the proposed approach. The plausibility of the analysis is supported by several simulation results.

References

1. Pizlo, Z., Li, Z.: Solving Combinatorial Problems: The 15-Puzzle. Memory & Cognition 33 (2005) 1069–1084
2. Borga, M.: Reinforcement Learning Using Local Adaptive Models. ISY, Linköping University, ISBN 91-7871-590-3, LiU-Tek-Lic-1995:39 (1995)
3. Zhang, W., Zhang, N. L.: Restricted Value Iteration: Theory and Algorithms. Journal of Artificial Intelligence Research 23 (2005) 123–165
4. Archer, A. F.: A Modern Treatment of the 15 Puzzle. American Mathematical Monthly 106, AAAI Press (1999) 793–799
5. Burns, E. A., Hatem, M. J., Ruml, W.: Implementing Fast Heuristic Search Code. Symposium on Combinatorial Search (2012)
6. Korf, R. E.: Depth-first iterative-deepening: an optimal admissible tree search. Artificial Intelligence 27 (1985) 97–109
7. Sutton, R. S., Barto, A. G.: Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning). The MIT Press (1998)
8. Surynek, P., Michalik, P.: An Improved Sub-optimal Algorithm for Solving $N^2 - 1$-Puzzle. Institute for Theoretical Computer Science, Charles University, Prague (2011)
9. Ratner, D., Warmuth, M. K.: NxN Puzzle and Related Relocation Problem. Journal of Symbolic Computation 10 (1990) 111-138
10. Hordern, E.: Sliding Piece Puzzles. Oxford University Press (1986)
11. Parberry, I.: A Real-Time Algorithm for the (n^2-1)-Puzzle. Inf. Process. Lett. 56 (1995) 23-28
12. Korf, R. E., Schultze, P.: Large-scale parallel breadth-first search. Proceedings of the 20th national conference on Artificial intelligence 3 (2005) 1380–1385
13. Wiering, M. and van Otterlo, M.: Reinforcement Learning: State-of-the-Art. Springer (2012)
14. Porta, M. J., Spaan, M. T. J. and Vlassis, N.: Robot Planning in Partially Observable Continuous Domains. Robotics: Science and Systems, MIT Press (2005) 217–224
15. Asadian A., Kermani M.R., Patel R.V.: Accelerated Needle Steering Using Partitioned Value Iteration. American Control Conference, ACC 10, Baltimore, MD, USA, 2010