



LUND UNIVERSITY

Routing using Safe Reinforcement Learning

Nayak Seetanadi, Gautham; Årzén, Karl-Erik

Published in:
2nd Workshop on Fog Computing and the Internet of Things

2020

[Link to publication](#)

Citation for published version (APA):
Nayak Seetanadi, G., & Årzén, K.-E. (in press). Routing using Safe Reinforcement Learning. In *2nd Workshop on Fog Computing and the Internet of Things*

Total number of authors:
2

General rights

Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: <https://creativecommons.org/licenses/>

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

1 Routing using Safe Reinforcement Learning

2 Gautham Nayak Seetanadi 

3 Department of Automatic Control, Lund University, Sweden

4 gautham@control.lth.se

5 Karl-Erik Årzén 

6 Department of Automatic Control, Lund University, Sweden

7 karlerik@control.lth.se

8 — Abstract —

9 The ever increasing number of connected devices has led to a meteoric rise in the amount of data to be processed. This has caused computation to be moved to the edge of the cloud increasing the importance of efficiency in the whole of cloud. The use of this fog computing for time-critical control applications is on the rise and requires robust guarantees on transmission times of the packets in the network while reducing total transmission times of the various packets.

14 We consider networks in which the transmission times that may vary due to mobility of devices, congestion and similar artifacts. We assume knowledge of the worst case transmission times over each link and evaluate the typical transmission times through exploration. We present the use of reinforcement learning to find optimal paths through the network while never violating preset deadlines. We show that with appropriate domain knowledge, using popular reinforcement learning techniques is a promising prospect even in time-critical applications.

20 **2012 ACM Subject Classification** Computing methodologies → Reinforcement learning; Networks → Packet scheduling

22 **Keywords and phrases** Real time routing, safe exploration, safe reinforcement learning, time-critical systems, dynamic routing

24 **Digital Object Identifier** 10.4230/OASICS.Fog-IoT.2020.2

25 **Funding** The authors are members of the LCCC Linnaeus Center and the ELLIIT Strategic Research Area at Lund University. This work was supported by the Swedish Research Council through the project “Feedback Computing”, VR 621-2014-6256.

28 **1** Introduction

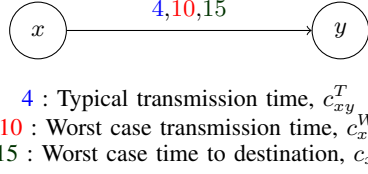
29 Consider a network of devices in a smart factory. Many of the devices are mobile and communicate with each other on a regular basis. As their proximity to the other devices change, the communication delays experienced by the device also change. Using static routing for such time-critical communications leads to pessimistic delay bounds and underutilization of network infrastructure.

34 Recent work [2] proposes an alternate model for representing delays in such time-critical networks. Each link in a network has delays that can be characterised by a conservative upper bound on the delay and the typical delay on the link. This dual representation of delay allows for capturing the communication behavior of different types of devices.

38 For example, a communication link between two stationary devices can be said to have equal typical and worst case delays. A device moving on a constant path near another stationary device can be represented using a truncated normal distribution. Adaptive routing techniques are capable of achieving smaller typical delays in such scenarios compared to static routing.

43 The adaptive routing technique described from [2] uses both delay information (typical and worst case) to construct routing tables. Routing is then accomplished using the tables





■ **Figure 1** Each link with attributes

45 which consider typical delays to be deterministic. This is however not the case as described
 46 above.

47 We propose using Reinforcement Learning (RL) [8, 12] for routing packets. RL is a
 48 model-free machine learning algorithm that has found prominence in the field of AI given its
 49 light computation and promising results [5, 9]. RL agents learn by exploring the environment
 50 around them and then obtaining a reward at the end of one iteration denoted one episode.

51 RL has been proven to be very powerful but it has some inherent drawbacks when considering
 52 its application to time-critical control applications. RL requires running a large number of
 53 episodes for an agent to learn. This leads to the possibility of deadline violations during
 54 exploration. Another drawback is the large state-space used for learning in classical RL
 55 methods that leads to complications in storage and search.

56 In this paper, we augment classical reinforcement learning with safe exploration to
 57 perform *safe* reinforcement learning. We use a simple (Dijkstras[4]) algorithm to perform
 58 safe exploration and then use the obtained information for safe learning. Using methodology
 59 described in section 4 we show that safety can be guaranteed during the exploration phase.
 60 Using safe RL also restricts the state-space reducing its size. Our decentralised algorithm
 61 allows each agent/node in the network to make independent decisions further reducing the
 62 state space. *Safe* reinforcement learning explores the environment to dynamically sample
 63 typical transmission times and then reduce delays for future packet transmissions. Our
 64 Decentralised approach allows each node to make independent and safe routing decisions
 65 irrespective of future delays that might be experienced by the packet.

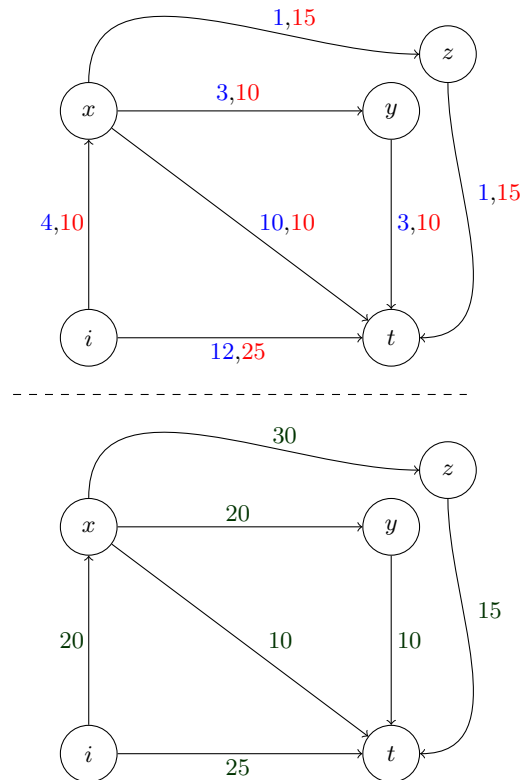
66 2 System Architecture

67 Consider a network of nodes, where each link $e : (x \rightarrow y)$ between node x and y is described
 68 by delays as shown in Figure 1.

- 69 ■ **Worst case delay (c_{xy}^W):** The delay that can be guaranteed by the network over each
 70 link. This is never violated even under maximum load.
- 71 ■ **Typical delay (c_{xy}^T):** The delay that is encountered when transmitting over the link
 72 and varies for each packet. We assume this information to be hidden from the algorithm
 73 and evaluated by sampling the environment.
- 74 ■ **Worst case delay to destination (c_{xt}):** The delay that can be guaranteed from node
 75 x to destination t . Obtained after the pre-processing described in section 4.1.

76 A network of devices and communication links can be simplified as a Directed Acyclic
 77 Graph as shown in Figure 2. The nodes denote the different devices in the network and the
 78 links denote the connections between the different devices. For simplicity we only assume
 79 one-way communication and consider a scenario of transmitting a packet from an edge device
 80 i to a server, t at a location far away from it.

81 As seen from the graph, many paths exist from the source i to t destination that can be
 82 utilised depending upon the deadline D_F of the packet.



■ **Figure 2** Example of graph and corresponding state space for the reinforcement learning problem formulation.

83 The values of c_{xy}^T and c_{xy}^W are shown in blue and red respectively for each link $e(x \rightarrow y)$.
 84 We also show the value of c_{xt} in green obtained after the pre-processing stage described in
 85 the following section.

86 3 Reinforcement Learning

87 Reinforcement Learning is the area of machine learning dealing with teaching agents to learn
 88 by performing actions to maximise a reward obtained [8] RL generally learns the environment
 89 by performing actions (safe actions) and evaluating the reward obtained at the end of the
 90 episode. We use Temporal-Difference (TD) methods for estimating state values and discover
 91 optimal paths for packet transmission.

92 We model our problem of transmitting packets from source i to destination t as a Markov
 93 Decision Process (MDP) as is the standard in RL. An MDP is a 4-tuple $(\mathcal{S}, \mathcal{A}, P, R)$, where
 94 \mathcal{S} is a set of finite states, \mathcal{A} is a set of actions, $P : (s, a, s') \rightarrow \{p \in \mathbb{R} \mid 0 \leq p \leq 1\}$ is a
 95 function that encodes the probability of transitioning from state s to state s' as a result of
 96 an action a , and $R : (s, a, s') \rightarrow \mathbb{N}$ is a function that encodes the reward received when the
 97 choice of action a determines a transition from state s to state s' . We use actions to encode
 98 the selection of an outgoing edge from a vertex.

99 3.1 TD Learning

100 TD learning [8] is a popular reinforcement learning algorithm that gained popularity due to
 101 its expert level in playing backgammon [9]. This model-free learning uses both state s and
 102 action a information to perform actions from the state given by the Q -value, $Q(s, a)$. TD
 103 learning is only a method to evaluate the value of being in the particular state. It is generally
 104 coupled with an exploration policy to form the strategy for an agent. We use a special TD
 105 learning called one step TD learning that allows for decentralised learning and allows for
 106 each node to make independent routing decisions. The value update policy is given by

$$107 \quad Q(s, a) = Q(s, a) + \alpha \cdot (\mathcal{R} + \max(\gamma Q(s', a')) - Q(s, a)) \quad (1)$$

108 3.2 Exploration Policy

109 ϵ -greedy exploration algorithm ensures that the optimal edge is chosen for most of the packet
 110 transmissions while at the same time other edges are explored in search of a path with higher
 111 reward. The chosen action $a \in A$ is either one that has the max value V or is a random
 112 action that explores the state space. The policy explores the state space with a probability ϵ
 113 and the most optimal action is taken with the probability $(1 - \epsilon)$. Generally the value of
 114 ϵ is small such that the algorithm exploits the obtained knowledge for most of the packet
 115 transmissions. To ensure that the deadline D_F is never violated, we modify the exploration
 116 phase to ensure safety and perform *safe* reinforcement learning.

117 4 Algorithm

118 We split our algorithm into two distinct phases. A pre-processing phase that gives us the
 119 initial safe bounds required to perform safe exploration. A run-time phase then routes packets
 120 through the network.

121 At each node, the algorithm explores feasible paths. During the initial transmissions
 122 the typical transmission times are evaluated after packet transmission. During the following
 123 transmissions, the path with the least delay is chosen more frequently while also exploring
 124 new feasible paths for lower delays. All transmissions using our algorithm are guaranteed to
 125 never violate any deadlines as we use safe exploration.

126 4.1 Pre-processing Phase

127 The pre-processing phase determines the safe bound for the worst case delay to destination t
 128 from every edge $e : (x \rightarrow y)$ in the network. The algorithm used by our algorithm is very
 129 similar to the one in [2]. This is crucial to ensure that there are no deadline violations during
 130 exploration in the run-time phase and is necessary irrespective of the run-time algorithm
 131 used. Dijkstra's shortest path algorithm [7] [4] is used to obtain these values as shown in
 132 algorithm 1.

133 4.2 Run-time Phase

134 The run-time algorithm is run at each node on the arrival of a packet. It determines
 135 $e : (x \rightarrow y)$ the edge on which the packet is transmitted from the node x to node y . Then
 136 the node y executes the run-time algorithm till the packet reaches the destination.

137 The edge chosen can be one of two actions:

Algorithm 1 Pre-Processing:

```

1: for each node  $u$  do
2:   for each edge  $(u \rightarrow v)$  do
3:     // Delay bounds as described in Section 4.1
4:      $c_{uv} = c_{uv}^W + \min(c_{vt})$ 
5:     // Initialise the Q values to 0
6:      $Q(u, v) = 0$ 

```

Algorithm 2 Node Logic (u)

```

1: for Every packet do
2:   if  $u = \text{source node } i$  then
3:      $D_u = D_F$  // Initialise the deadline
4:      $\delta_{it} = 0$  // Initialise total delay for packet = 0
5:   for each edge  $(u \rightarrow v)$  do
6:     if  $c_{uv} > D_u$  then // Edge is infeasible
7:        $P(u|v) = 0$ 
8:     else if  $Q(u, v) = \max(Q(u, a \in A))$  then
9:        $P(u|v) = (1 - \epsilon)$ 
10:    else
11:       $P(u|v) = \epsilon / (\text{size}(\mathcal{F}) - 1)$ 
12:    Choose edge  $(u \rightarrow v)$  with  $P$ 
13:    Observe  $\delta_{uv}$ 
14:     $\delta_{it} += \delta_{uv}$ 
15:     $D_v = D_u - \delta_{uv}$ 
16:     $R = \text{Environment Reward Function}(v, \delta_{it})$ 
17:     $Q(u, v) = \text{Value iteration from Equation (1)}$ 
18:    if  $v = t$  then
19:      DONE

```

138 ■ **Exploitation action:** An action that chooses the path with the least transmission time
 139 out of all known feasible paths. If no information is known on all the edges, then an edge
 140 is chosen at random.

141 ■ **Exploration action:** An action where a sub-optimal node is chosen to transmit the
 142 packet. This action uses the knowledge about c_{xy} obtained during the pre-processing
 143 phase to ensure that the exploration is safe. This action ensures that the algorithm is
 144 dynamic by ensuring that if there exists a path with lower transmission delay, it will be
 145 explored and chosen more during future transmissions. Exploration also optimises for a
 146 previously congested edge that could be decongested at a later time.

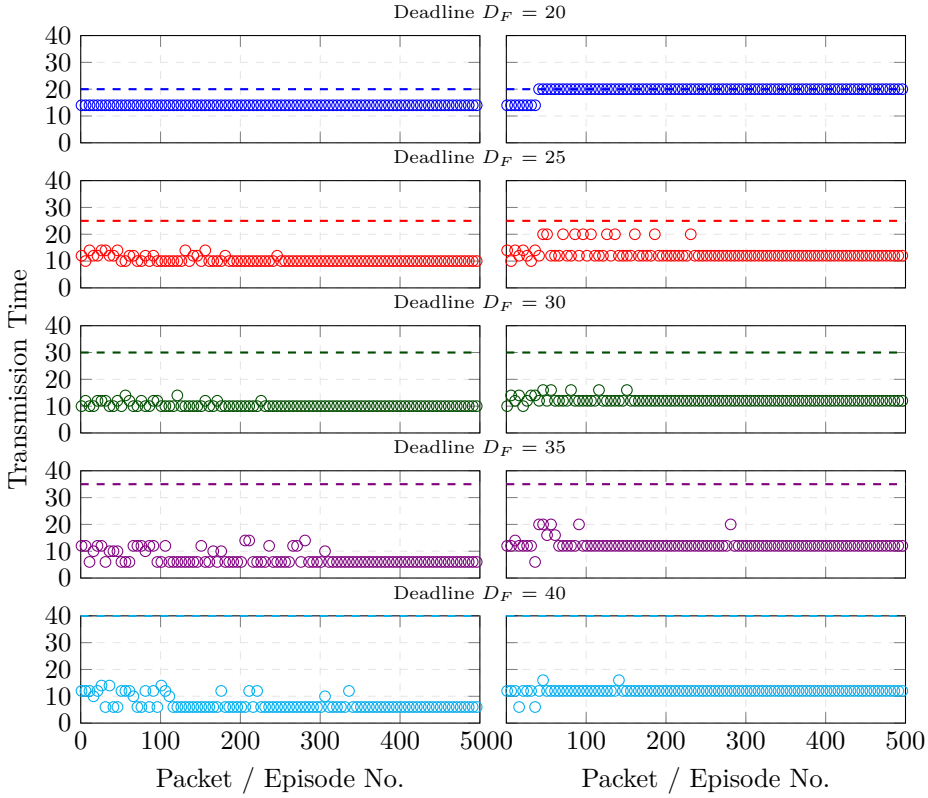
147 Algorithm 2 shows the pseudo code for the run-time phase. The computation is computationally
 148 light and can be run on mobile IoT devices.

149 4.3 Environment Reward

150 The reward \mathcal{R} is awarded as shown in algorithm 3. After each traversal of the edge, the
 151 actual time taken δ is recorded and added to the total time traversed for the packet, $\delta_{it} += \delta$.
 152 The reward is then awarded at the end of each episode and it is equal to the amount of time
 153 saved for the packet, $R = D_F - \delta_{it}$.

■ **Algorithm 3** Environment Reward Function(v, δ_{it})

-
- 1: Assigns the reward at the end of transmission
 - 2: **if** $v = t$ **then**
 - 3: $R = D_F - \delta_{it}$
 - 4: **else**
 - 5: $R = 0$
-



■ **Figure 3** Smoothed Total Delay for Experiment with (a) Constant delays and (b) Congestion at packet 40

154 **5 Evaluation**

155 In this section, we will evaluate the performance of our algorithm. We apply it to the
 156 network shown in Figure 2. The network is built using Python and the NetworkX package [6]
 157 package. The package allows us to build Directed Acyclic Graphs (DAGs) with custom
 158 delays. Each link $e : (x \rightarrow y)$ in the network has the constant worst case link delay c_{xy}^W visible
 159 to the algorithm but the value of c_{xy}^T although present is not visible to our algorithm. The
 160 pre-processing algorithm and calculates the value of c_{xt} . This is done only once initially and
 161 then Algorithms 2 and 3 are run for every packet that is transmitted and records the actual
 162 transmission time δ_{it} .

163 Figure 3 shows the total transmission times when the actual transmission times δ and
 164 typical transmission times c^T are equal. We route 500 packets through the network for
 165 deadline $D_F \in (20, 25, 30, 35, 40)$. For $D_F = 20$, the only safe path is $(i \rightarrow x \rightarrow t)$ and so has
 166 a constant δ_{it} for all packets. For the remaining deadlines, the transmission times vary as

167 new paths are taken during exploration. The deadlines are never violated for any packets
 168 irrespective of the deadline. Table 1 shows the optimal paths and the average transmission
 169 times compared to the algorithm from [2].

170 Figure 3 shows the capability of our algorithm in adapting to congestions in the network.
 171 Congestion is added on the link ($i \rightarrow x$) after the transmission of 40 packets. The transmission
 172 time over the edge increases from 4 to 10 time units and is kept congested for the rest of
 173 the packet transmissions. The algorithm adapts to the congestion by exploring other paths
 174 that might now have lower total transmission times δ_{it} . In all cases other than $D_F = 20$, the
 175 algorithm converges to the path ($i \rightarrow t$) with $\delta_{it} = 12$. When $D_F = 20$, ($i \rightarrow x \rightarrow t$) is the
 176 only feasible path.

177 6 Practical Considerations

178 In this section we will discuss some of the practical aspects when implementing the algorithms
 179 described in section 4.

180 6.1 Computational Overhead

181 The computational complexity of running our algorithm mainly arises in the pre-processing
 182 stage. This complexity is dependent on the number of nodes in the network. Dijkstras
 183 algorithm has been widely studied and have efficient implementations that reduce computation.
 184 The pre-processing has to be run only once for all networks given that there are no structural
 185 changes.

186 6.2 Multiple sources

187 The presence of multiple sources and thus multiple packets on the same link can be seen as
 188 an increase in the typical delays on the link. This holds true given that the worst case delay
 189 c_{xy}^W over each link is properly determined and guaranteed.

190 6.3 Network changes

- 191 ■ **Node Addition:** During the addition of a new node the pre-processing stage has to
 192 be run in a constrained space. The propagation of new information to the preceding
 193 nodes is only necessary if it affects the value of c_{xt} over the affected links. The size of the
 194 network affected has to be investigated further.
- 195 ■ **Node Deletion:** In the event of node deletion during the presence of a packet at the
 196 deleted node, the packet is lost and leads to deadline violation. However no further

■ **Table 1** Optimal Path for Different Deadlines

D_F	Optimal Path	Delays [2]	Average Delays (1000 episodes)
15	Infeasible	-	-
20	{i,x,t}	14	14
25	{i,x,y,t}	10	10.24
30	{i,x,y,t}	10	10.22
35	{i,x,z,t}	6	6.64
40	{i,x,z,t}	6	6.55

197 packages will be transmitted over the link as the reward \mathcal{R} is 0. Similar to the case of
 198 node addition, the pre-processing algorithm requires further investigations.

199 **7 Conclusion and Future Work**

200 In this paper we use *safe* reinforcement learning to routing networks with variable transmission
 201 times. A once used pre-processing algorithm is used to determine safe bounds. Then a safe
 202 reinforcement learning algorithm uses this domain knowledge to route packets in minimal
 203 time with deadline guarantees. We have considered only two scenarios in this paper but we
 204 believe that the algorithm will be able to adapt with highly variable transmission times and
 205 network failures. The use of low complexity RL algorithm makes it suitable for use on small,
 206 mobile platforms.

207 Although we show stochastic convergence in our results with no deadline violations, our
 208 current work lacks **formal guarantees**. Recent work has been published trying to address
 209 analytical safety guarantees of safe reinforcement learning algorithms [10, 11]. In [10], the
 210 authors perform safe Bayesian optimization with assumptions on Lipschitz continuity of
 211 function. While [10] estimates the safety of only one function, our algorithm is dependent on
 212 the continuity of multiple functions and requires more investigation.

213 The network implementation and evaluation using NetworkX in this paper have shown that
 214 using *safe* RL is a promising technique. An extension of this work would be implementation
 215 on a network emulator. Using network emulators (for example CORE [1], Mininet [3]) would
 216 allow us to evaluate the performance of our algorithm on a full internet protocol stack.
 217 Using an emulator allows for implementation of multiple flows between multiple sources and
 218 destinations.

219 **References**

-
- 220 **1** Jeff Ahrenholz. Comparison of core network emulation platforms. In *2010-Milcom 2010*
 221 *Military Communications Conference*, pages 166–171. IEEE, 2010.
 - 222 **2** Sanjoy Baruah. Rapid routing with guaranteed delay bounds. In *2018 IEEE Real-Time*
 223 *Systems Symposium (RTSS)*, pages 13–22, December 2018.
 - 224 **3** Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and
 225 Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks.
 226 In *2014 IEEE Colombian Conference on Communications and Computing (COLCOM)*, pages
 227 1–6. Ieee, 2014.
 - 228 **4** E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*,
 229 1(1):269–271, Dec 1959. doi:10.1007/BF01386390.
 - 230 **5** Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophile
 231 Weber, David Raposo, Adam Santoro, Laurent Orseau, Tom Eccles, et al. An investigation of
 232 model-free planning. *arXiv preprint arXiv:1901.03559*, 2019.
 - 233 **6** Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics,
 234 and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors,
 235 *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.
 - 236 **7** Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*.
 237 Springer Publishing Company, Incorporated, 1 edition, 2008.
 - 238 **8** Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An Introduction*. Adaptive
 239 computation and machine learning. MIT Press, 2018.
 - 240 **9** Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–
 241 68, March 1995. URL: <http://doi.acm.org/10.1145/203330.203343>, doi:10.1145/203330.
 242 203343.

- 243 10 Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration for interactive
244 machine learning. In *Proc. Neural Information Processing Systems (NeurIPS)*, December 2019.
- 245 11 Kim P Wabersich and Melanie N Zeilinger. Safe exploration of nonlinear dynamical systems:
246 A predictive safety filter for reinforcement learning. *arXiv preprint arXiv:1812.05506*, 2018.
- 247 12 Marco Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and*
248 *optimization*, 12:3, 2012.