Dynamic Strategies for $\mathrm{T_CT}$

Manuel Schneckenreither

Department of Computer Science,
Computational Logic Group,
University of Innsbruck, Austria
email: manuel.schneckenreither@uibk.ac.at

The Twenty First International Workshop on Logic and Computational Complexity Summer 2020

Motivation

The Tyrolean Complexity Tool, i.e. $\mathrm{T_CT}$

- Automatic tool to infer runtime (and derivational) complexity
- Allows (first or higher-order) functional programs as generalised form of term rewrite systems as input
- In case of success: $\mathrm{T_CT}$ outputs an asymptotic complexity and corresponding proof

[1] Landi, "Undecidability of static analysis", 1992

Motivation

The Tyrolean Complexity Tool, i.e. $\mathsf{T_CT}$

- Automatic tool to infer runtime (and derivational) complexity
- Allows (first or higher-order) functional programs as generalised form of term rewrite systems as input
- In case of success: $\mathsf{T_CT}$ outputs an asymptotic complexity and corresponding proof

Static Analyses

- Static analyses are undecidable [1]
- Thus: Some sort of creativity is required

[1] Landi, "Undecidability of static analysis", 1992

In T$_C$T

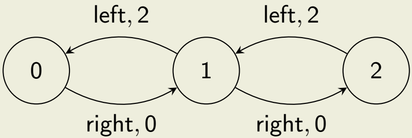
- T$_C$T uses Strategies to define:
  - what methods (e.g. poly./matrix interpretations) to use and
  - in which sequence these are used.
- Currently strategies are predetermined and fixed in T$_C$T
- Idea: Reinforcement learning for dynamic strategies
  - No training data needed (no predetermined strategies)
  - Provides a much larger solution space
  - Unexpected good behaviour might be discovered

Dynamic Strategies for T$_C$T

Table of Contents

Reinforcement Learning (RL)

Reinforcement Learning Processes [2]

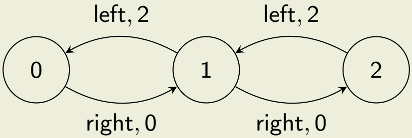


- Based on dynamic programming: states, actions, rewards

[2] Miller and Veinott, "Discrete Dynamic Programming with a Small Interest Rate", 1969

Reinforcement Learning (RL)

Reinforcement Learning Processes [2]



- Agent iteratively assesses state values $V_\gamma^\pi(s)$, where
- $0 < \gamma < 1$ is the discount factor, π the policy function, and
- for two consecutive observed states s_t, s_{t+1} with reward r_t

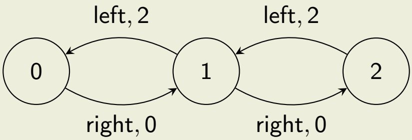
$$V_\gamma^\pi(s_t) \xleftarrow{\alpha} r_t + \gamma V_\gamma^\pi(s_{t+1})$$

($\xleftarrow{\alpha}$ is exp. smoothed updating with rate α)

[2] Miller and Veinott, "Discrete Dynamic Programming with a Small Interest Rate", 1969

Reinforcement Learning (RL)

Reinforcement Learning Processes [2]



- Thus, state values $V_\gamma^\pi(s)$ are

$$V_\gamma^\pi(s) = \lim_{N\to\infty} E[\sum_{t=0}^{N-1} \gamma^t R_t^\pi(s)]$$

where $R_t^\pi(s)$ the reward received at time t upon starting in state s by following policy π.

[2] Miller and Veinott, "Discrete Dynamic Programming with a Small Interest Rate", 1969

Aim in Discounted Reinforcement Learning

Aim of Discount Reinforcement Learning

- Finding an optimal policy $\pi^\star$
- $\pi^\star$ maximises the state value for all states s as compared to any other policy π:

$$V_\gamma^{\pi^\star}(s) - V_\gamma^{\pi}(s) \geqslant 0$$

Aim in Discounted Reinforcement Learning

Aim of Discount Reinforcement Learning

- Finding an optimal policy $\pi^\star$
- $\pi^\star$ maximises the state value for all states s as compared to any other policy π:

$$V_\gamma^{\pi^\star}(s) - V_\gamma^{\pi}(s) \geqslant 0$$

- This criteria: discounted-optimality as γ is fixed

Aim in Discounted Reinforcement Learning

Aim of Discount Reinforcement Learning

- Finding an optimal policy $\pi^\star$
- $\pi^\star$ maximises the state value for all states s as compared to any other policy π:

$$V_\gamma^{\pi^\star}(s) - V_\gamma^{\pi}(s) \geqslant 0$$

- This criteria: discounted-optimality as γ is fixed

The higher γ the more optimal are the solutions.

Aim in Discounted Reinforcement Learning

Aim of Discount Reinforcement Learning

- Finding an optimal policy $\pi^\star$
- $\pi^\star$ maximises the state value for all states s as compared to any other policy π:

$$V_\gamma^{\pi^\star}(s) - V_\gamma^{\pi}(s) \geqslant 0$$

- This criteria: discounted-optimality as γ is fixed

The higher γ the more optimal are the solutions.

Usual values: $\gamma \geqslant 0.99$

Laurent Series Expansion of Discounted State Values

Laurent Series Expansion of Discounted State Values[3]

The Laurent series expansion of $V_\gamma^\pi(s)$ of discounted state values:

$$V_\gamma^\pi(s) = \frac{\rho^\pi}{1-\gamma} + V^\pi(s) + e_\gamma^\pi(s)$$

Average Reward ρ^π (simplified for unichain Processes)

The average reward is defined as

$$\rho^\pi = \lim_{N\to\infty} \frac{\mathbb{E}[\sum_{t=0}^{N-1} R_t^\pi]}{N}$$

where R_t^π is the reward received at time t.

[3] Miller and Veinott, "Discrete Dynamic Programming with a Small Interest Rate", 1969

Laurent Series Expansion of Discounted State Values

Laurent Series Expansion of Discounted State Values[3]

The Laurent series expansion of $V_\gamma^\pi(s)$ of discounted state values:

$$V_\gamma^\pi(s) = \frac{\rho^\pi}{1-\gamma} + V^\pi(s) + e_\gamma^\pi(s)$$

Bias value $V^\pi(s)$

The bias value is defined as

$$V^\pi(s) = \lim_{N\to\infty} \mathbb{E}[\sum_{t=0}^{N-1}(R_t^\pi(s) - \rho^\pi)]$$

It is the additional reward that sums up when starting in state s.

[3] Miller and Veinott, "Discrete Dynamic Programming with a Small Interest Rate", 1969

Laurent Series Expansion of Discounted State Values

Laurent Series Expansion of Discounted State Values[3]

The Laurent series expansion of $V_\gamma^\pi(s)$ of discounted state values:

$$V_\gamma^\pi(s) = \frac{\rho^\pi}{1-\gamma} + V^\pi(s) + e_\gamma^\pi(s)$$

Error term of e_γ^π [4]

Error term $e_\gamma^\pi(s)$ consists of infinitely many terms, but

$$\lim_{\gamma \to 1} e_\gamma^\pi(s) = 0$$

If $\gamma < 1$ it takes number of steps and reward values into account.

[4] Puterman, "Markov Decision Processes: Discrete Stochastic Dynamic Programming", 1994

Laurent Series Expansion of Discounted State Values

Laurent Series Expansion of Discounted State Values[3]

The Laurent series expansion of $V_{\gamma}^{\pi}(s)$ of discounted state values:

$$V_{\gamma}^{\pi}(s) = \frac{\rho^{\pi}}{1-\gamma} + V^{\pi}(s) + e_{\gamma}^{\pi}(s)$$

[3] Miller and Veinott, "Discrete Dynamic Programming with a Small Interest Rate", 1969

Laurent Series Expansion of Discounted State Values

Laurent Series Expansion of Discounted State Values[3]

The Laurent series expansion of $V^{\pi}_{\gamma}(s)$ of discounted state values:

$$V^{\pi}_{\gamma}(s) = \frac{\rho^{\pi}}{1-\gamma} + V^{\pi}(s) + e^{\pi}_{\gamma}(s)$$

Note

- Recall: High $\gamma \approx 1$ values required for optimal policies
- First term converges to ∞ as $\gamma \to 1$
- Recall: RL is an iterative method

[3] Miller and Veinott, "Discrete Dynamic Programming with a Small Interest Rate", 1969

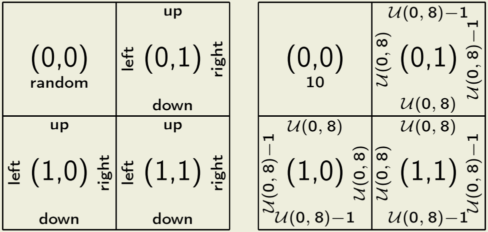
So What?

$$V_\gamma^\pi(s) = \frac{\rho^\pi}{1-\gamma} + V^\pi(s) + e_\gamma^\pi(s)$$

So What?

$$V^\pi_\gamma(s) = \frac{\rho^\pi}{1-\gamma} + V^\pi(s) + e^\pi_\gamma(s)$$

Consider this simple gridworld problem [4]



State/Action Space

Reward Function

[4] Manuel Schneckenreither, "Average Reward Adjusted Discounted Reinforcement Learning: Near-Blackwell-Optimal Policies for Real-World Applications", 2020

So What?

$$V_\gamma^\pi(s) = \frac{\rho^\pi}{1-\gamma} + V^\pi(s) + e_\gamma^\pi(s)$$

Same problem as 5x5 grid

(0,0) 101.260	23.113 67.160 (0,1) 61.783 37.331	162.243 49.812 (0,2) 349.515 153.653	354.839 306.761 (0,3) 399.560 351.907	398.391 399.376 (0,4) 398.444 399.539
...	...	...	...	...

- All states values completely assessed independently
- For $\gamma \approx 1$: $\rho^\pi/(1-\gamma) \gg V^\pi(s) + e_\gamma^\pi(s)$
- Thus, all values need to increase to $\approx \rho^\pi/(1-\gamma)$
- BUT: Greater $V_\gamma^\pi(s)$, then more likely to be picked

Average Reward Adjusted Reinforcement Learning [4]

Basic Idea

Separately assessing

- average reward ρ^{π}
- bias value $V^{\pi}(s)$

[4] Manuel Schneckenreither, "Average Reward Adjusted Discounted Reinforcement Learning: Near-Blackwell-Optimal Policies for Real-World Applications", 2020

Average Reward Adjusted Reinforcement Learning [4]

Basic Idea

Separately assessing

- average reward ρ^π
- bias value $V^\pi(s)$

Algorithm

$$\rho^\pi \xleftarrow{\alpha} r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$$

$$V^\pi(s_t) \xleftarrow{\beta} r_t + \gamma V^\pi(s_{t+1}) - \rho^\pi$$

Note: We can set $\gamma = 1$ here, as the subtraction of the average reward bounds the state values.

[4] Manuel Schneckenreither, "Average Reward Adjusted Discounted Reinforcement Learning: Near-Blackwell-Optimal Policies for Real-World Applications", 2020

Discounted vs. Average Reward Adjusted RL

Reconsider the 5x5 Gridworld [4]

Algorithm	Sum Reward	Avg. Steps
Avg. Rew. Adjusted $\gamma = 0.99$	**51894.094**	**5.039**
Avg. Rew. Adjusted $\gamma = 0.999$	51878.069	5.063
Avg. Rew. Adjusted $\gamma = 1.00$	51856.529	5.055
Standard Discounted $\gamma = 0.99$	34409.464	7661.833
Standard Discounted $\gamma = 0.999$	33931.917	7379.155
Standard Discounted $\gamma = 0.50$	30171.837	9999.000

(500k learning steps, 40 $\times$ 10k greedy evaluation steps)

[4] Manuel Schneckenreither, "Average Reward Adjusted Discounted Reinforcement Learning: Near-Blackwell-Optimal Policies for Real-World Applications", 2020

Average Reward Adjusted Reinforcement Learning

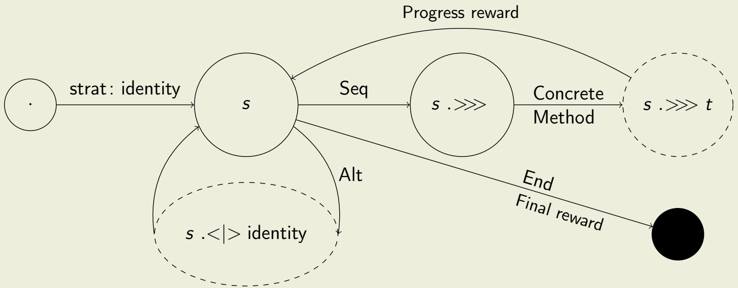
More Insights

Average Reward Adjusted Reinforcement Learning ...

1. requires fewer number of learning steps
2. allows specifying continuous feedback

Implementation in T$_C$T

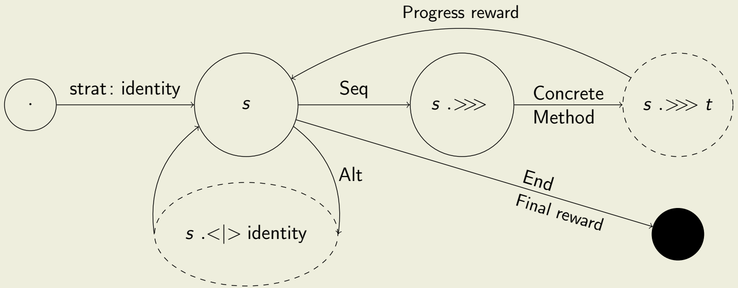
Process Schema



- Starting strategy: identity
- Strategy combinators: Alternative and Sequence

Implementation in TCT

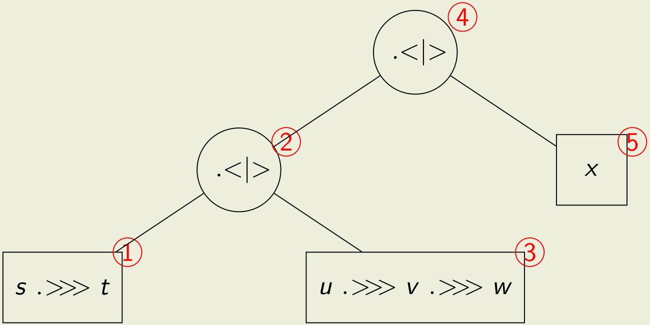
Process Schema



- Note: Continuous feedback (Progress reward)
- Required: Exponential growth of solution space (continuous and faster learning)

Implementation in T_CT

Binary Strategy Tree



- No ambiguity by enforcing right-associativity of $.{<}|{>}$
- Sorting of leaves on change of strategy:

$$(s \ .{>}{>}{>} \ t) \ .{<}|{>} \ (u \ .{>}{>}{>} \ v) \equiv (u \ .{>}{>}{>} \ v) \ .{<}|{>} \ (s \ .{>}{>}{>} \ t)$$

Implementation in $\mathsf{T_CT}$

State Space

- Characteristics of input problem, e.g.
  - number of rules
  - number of (root) symbols
  - left-linearity, right-linearity
  - SCC (recursion and size of argument in recursive call)
  - branching degree
  - ...
- Representation of current strategy
  - Counter specifying steps until last occurrence of concrete method

Implementation in $\mathrm{T_CT}$

Reward Function

Assume current strategy s_t at step t, a timeout of T seconds, a measured execution time of $t' \leqslant T$ seconds and resulting polynomial complexity C:

$$r(s_t) = \begin{cases} cT + t' & \text{if } C \text{ is } \mathrm{O}(n^c) \text{ and } c < P_{\max} \\ P_{\max}\, T & \text{otherwise} \end{cases}$$

where $P_{\max}$ is the maximum polynomial degree being considered.

Outlook

Reality everything is more complex

- Algorithm is more complex, e.g. incorporates the error term
- Artificial Neural Network (ANN) to approximate value function
- More strategy combinators than just .<|> and .>>>
- ...

Outlook

Reality everything is more complex

- Algorithm is more complex, e.g. incorporates the error term
- Artificial Neural Network (ANN) to approximate value function
- More strategy combinators than just $.\langle|\rangle$ and $.\ggg$
- ...

Current State (+ Expert info)

- Implemented to work with ANN (critic-only, experience replay memory, target+worker ANN, ...)
- We aim for an actor-critic approach (policy and value iteration)
- We plan to use a DB for caching (certain) I/O tuples
- Many open questions, as we are coding it right now

Outlook

Reality everything is more complex

- Algorithm is more complex, e.g. incorporates the error term
- Artificial Neural Network (ANN) to approximate value function
- More strategy combinators than just $.<|>$ and $.>>>$
- ...

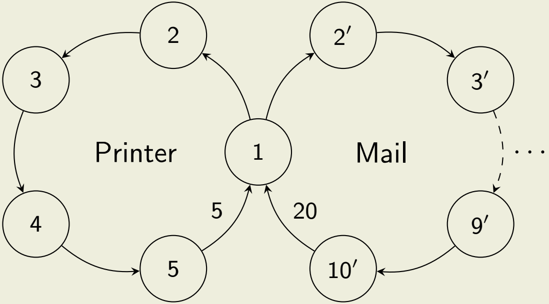
Current State (+ Expert info)

- Implemented to work with ANN (critic-only, experience replay memory, target+worker ANN, ...)
- We aim for an actor-critic approach (policy and value iteration)
- We plan to use a DB for caching (certain) I/O tuples
- Many open questions, as we are coding it right now

Thank you for your attention.

Let's discuss.
I am glad for any feedback, ideas or issues you have or see!

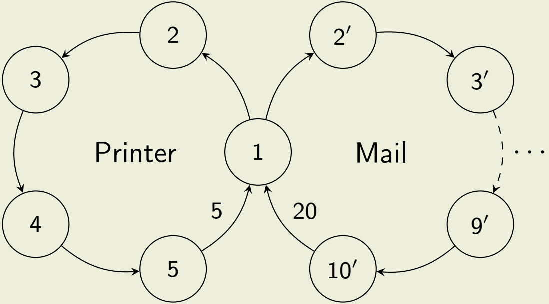
The issue of discounted-optimality illustrated [5]



- Average reward received is
  - 1 for the printer-loop
  - 2 for the mail-loop

[5] Adapted from Mahadevan, "Optimality criteria in reinforcement learning", 1996

The issue of discounted-optimality illustrated [5]

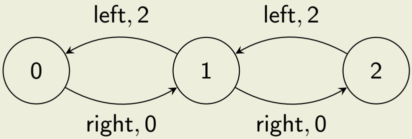


- Average reward received is
  - 1 for the printer-loop
  - 2 for the mail-loop
- BUT: if $\gamma < 3^{-\frac{1}{5}} \approx 0.8027$ the agent prefers the printer loop

[5] Adapted from Mahadevan, "Optimality criteria in reinforcement learning", 1996

Reinforcement Learning (RL)

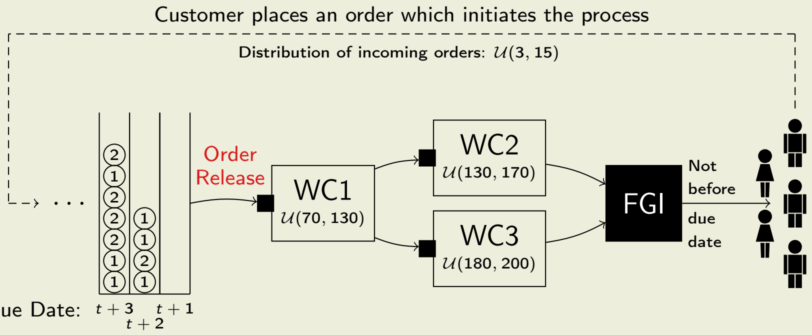
Markov Decision Processes [6]



- Based on dynamic programming: states, actions, rewards
- Markov Prop.: transitions with probability $p(s_{t+1}, r_t \mid s_t, a_t)$
- RL processes: *Markov decision processes* (MDPs)

[6] Miller and Veinott, "Discrete Dynamic Programming with a Small Interest Rate", 1969

Motivation: More Applications

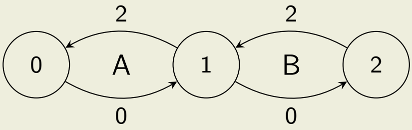


- WIP, Holding and Backorder costs ⇒ Aim: min. Costs
- A3C Reinforcement Learning variant (= highly sophisticated): Results were OK, but by far not optimal

M. Schneckenreither and Haeussler, "RL Methods for OR Applications", 2019

Add: Normed state values have a higher spread (Slide 13)

Reconsider the Example



- Both policies are have an average reward of 1 (gain-optimal).
- But only π_A is bias-optimal: $V^{\pi^\star}(s) - V^{\pi}(s) \geqslant 0$.

s	$V^{\pi_A}(s)$	$V^{\pi_B}(s)$	$V^{\pi_A}_{0.99}$	$V^{\pi_B}_{0.99}$
0	-0.5	-1.5	99.4975	98.5025
1	0.5	-0.5	100.5025	99.4975
2	1.5	0.5	101.4975	100.5025