NTNU

Norwegian University of
Science and Technology

Contributions to centralized dynamic channel allocation reinforcement learning agents

Torstein Sørnes

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer Science

Table of Contents

List of Tables

List of Figures

Abbreviations

BS	=	Base Station
CAC	=	Call Admission Control
RL	=	Reinforcement Learning
DP	=	Dynamic Programming
FCA	=	Fixed Channel Allocation
DCA	=	Dynamic Channel Allocation
DSA	=	Dynamic Spectrum Allocation
HCA	=	Hybrid Channel Allocation
BDCL	=	Borrow with Directional Channel Locking
(S)MDP	=	(Semi-)Markov Decision Process
A-(S)MDP	=	Average-Reward (Semi-)Markov Decision Process
ANN	=	Artifical Neural Network
TD	=	Temporal Difference
SINR	=	Signal to Interference plus Noise Ratio
HLA	=	Hand-off Look-Ahead
SGD	=	Stochastic Gradient Descent
TDC	=	TD(0) with Gradient Corrections

Symbols

I Number of cells in the grid.

$\mathcal{I}$ Set of all cells.

K Number of channels.

$\mathcal{K}$ Set of all channels.

$\mathbf{x}$ Allocation map.

$z(\mathbf{x}, e, a)$ Allocation map transition function, yields $\mathbf{x}'$.

$z_\phi(\mathbf{x}, \tilde{\mathbf{x}}, e, a)$ Feature representation transition function, yields $\tilde{\mathbf{x}}'$.

$\tilde{\mathbf{x}}$ Feature representation of allocation map.

ϕ Feature representation transformation function.

S State.

A Action.

R Reward.

$\mathcal{S}$ State space.

$\mathcal{A}(\mathrm{s})$ Action space in state s.

$\mathcal{P}^{\mathrm{a}}_{\mathrm{ss}\prime}$ Probability of transitioning to state s' given state s, action a.

$\mathcal{R}^{\mathrm{a}}_{\mathrm{s}}$ Expected reward given state s, action a.

G_t Return. Integration of possibly discounted rewards from time t on-wards.

γ Discount factor for MDP.

β Discount factor for SMDP.

$v_\pi(s)$ Expected return when starting in state s and following policy π.

$v_*(s)$ Expected return when starting in state s and following an optimal policy.

$V_\pi(s)$ Stochastic approximation of $v_\pi(s)$.

$V_{\boldsymbol{\theta}}(s)$ Function approximation with parameters $\boldsymbol{\theta}$ of $V_\pi(s)$.

$\bar{v}_\pi(s)$ Average-adjusted value-function (for average-reward MDPs).

α Learning rate for RL agent.

α^G Learning rate for gradient corrections.

α^A Learning rate for average reward.

δ Temporal Difference error.

$\bar{\delta}$ Differential Temporal Difference error (for average-reward MDPs).

ρ Average reward.

Chapter 1

Introduction

In 2017, the number of unique mobile subscribers was 5 billion and is expected to reach 5.9 billion by 2025. The number of mobile devices excluding cellular Internet-of-Things devices is expected to grow from 7.8 to 9.0 billion in the same period (Sivakumaran and Iacopino, 2018). In 2013, the United Kingdom government enjoyed a $£2.3$ billion gain by auctioning the 4G mobile spectrum to telecom operators (Ofcom, 2013). As the number of mobile cellular telecommunications devices is already staggering and expected to grow, the efficient use of the limited and expensive radio spectrum becomes even more important in order to satisfy the demands of the users.

In circuit switched networks carrying voice traffic, the fundamental limiting factor in system capacity on the radio spectrum is interference between different mobile callers (Katzela and Naghshineh, 1996). Nearby mobile callers using the same radio frequency causes co-channel interference which is the most prohibitive interference type. If a service request cannot be fulfilled without causing excessive interference, it is denied. Interference is managed through channel allocation techniques which aim to increase radio spectrum reuse efficiency by determining the pattern with which radio channels are used across the geographic radio coverage area. The area forms a grid divided into disjoint, fixed sized regions called cells, each served by its own base station (BS) which has a transceiver that wirelessly connects user equipment to the network providing voice transmission services. The system as a whole has access to a finitely sized communications resource, namely its allocation of radio spectrum bandwidth. The bandwidth is divided into bandwidth units, hereafter referred to as channels, using a multiple access technique such as frequency division multiple access (FDMA). In FDMA, the bandwidth is divided into narrow frequency bands with guard bands in between to avoid cross talk interference. Other schemes like time division multiple access (TDMA) and code division multiple access (CDMA) divide the spectrum using time slots and modulation codes respectively (Dutta et al., 2016). Each mobile caller must have exclusive access to a channel within its local area to communicate with the base station without interference. The minimum distance necessary between simultaneous reuse of a channel is termed the channel reuse constraint or reuse distance (Jordan, 1996). BSs spaced less than the reuse distance apart from a particular BS are its

interfering neighbors, while cells at or outside this distance are its *co-channel cells* and may use the same channels simultaneously as the cell in question. Since every cell has a single BS, we will use those terms interchangeably throughout this document.

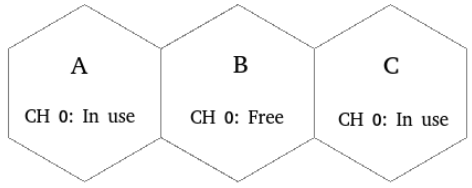


Figure 1.1: Three cells, with cell A and C both using the same channel without violating the reuse distance of 2. B is an interfering neighbor of A, while C is a co-channel cell of A.

The problem of channel allocation (channel assignment) is to select which channel to allocate to a caller upon a service request. In centralized systems, which this work considers, a BS receiving a service request queries a central switching centre for which channel to use. In the switching centre, the channel allocation policy selects a channel from the set of channels that are free for both the BS and its interfering neighbors, i.e. the *eligible channels*. If there are no eligible channels, the call must be blocked to avoid interference. Using a free channel *may* cause interference, while using an eligible channel will not.

Callers move around in the geographic area, creating call *hand-offs* between BSs. On hand-off from departure BS to arrival BS, the arrival BS may choose any eligible channel thus a hand-off is logistically equivalent to a call departure in one cell and a call arrival in a neighboring cell. Blocking a hand-off is considered less desirable than blocking a new call request, and there is a trend towards reducing the size of the area served by a base station which increases the hand-off frequency, thus handling hand-offs is increasingly important (El-Alfy et al., 2001).

The core objective of the channel allocation policy is to minimize the probability of blocking call service requests, or equivalently, to maximize service utilization. To aid in that objective, the channel allocation policy may reassign a call in progress from one channel to another when other calls depart in order to leave more channels eligible for future assignment. Preferably, hand-off blocking probability should be reduced below the probability of blocking new call requests. With a Fixed Channel Assignment (FCA) policy, each cell is pre-allocated a limited subset of the channels and may only assign channels from this subset. FCA is the most commonly used channel allocation policy in GSM networks (Lilith and Dogançay, 2004), but performs badly in low traffic conditions and is unable to prioritize hand-offs without resorting to techniques which cause a significant penalty to total system utilization (Wong, 2003). In Dynamic Channel Allocation (DCA), the policy allocates channels from a central pool where any channel can be used in any cell so long as the reuse constraint is satisfied. DCA policies have the potential to adapt to temporal and spatial variations in call traffic, to prioritize hand-offs, and handle non-fixed network topologies. Reinforcement Learning (RL) have been successfully applied to DCA and related problems in the radio network domain.

In this work, we investigate the formalization of the problem of DCA as a Markov De-

cision Process (MDP) and the design of the RL agent, and ask ourselves the questions: *Can hand-off blocking probability be improved without doing call admission control? Which objective function should be optimized in order to minimize cumulative call blocking probability? Does the problem of DCA exhibit any special characteristics that can be exploited in the design of the RL agent?*

We introduce a domain-specific policy improvement operator for reassigning channels during call hand-offs with the intent of reducing hand-off blocking probability. We construct an RL agent for maximizing average grid utilization, which uses a linear neural network as state value-function approximator and afterstates for action selection. A variant of TD(0) with gradient correction (TDC) (Sutton et al., 2009) is proposed for average-reward MDPs, which in conjunction with the policy improvement operator contributes decreased hand-off call blocking probability in a simulated centralized caller environment without any penalty to previously shown (Singh and Bertsekas, 1997) state of the art new call blocking probability. The policy improvement operator is also applied to the table-lookup based SARSA agent of Lilith and Dogançay (2004) where it shows state of the art performance in terms of hand-off blocking probability for an all-admission agent.

While this works considers centralized systems, the policy improvement operator is applicable to distributed agents so long as the channel usages of the interfering neighbors of the hand-off arrival BS are known to the hand-off departure BS.

Chapter 2

Background

In this work, we assume a cellular telephone network with a centralized agent operating as the sole user in a licensed band on the radio spectrum. In addition, we assume that the multiple access technique eliminates any significant adjacent channel interference. Adjacent channel interference is caused by two nearby mobiles using different channels that are nearby in the channel domain, e.g. nearby frequencies. Being the sole user of a network without adjacent channel interference implies that any interference is co-channel interference caused by using the same channel in close geographic distance, and that the network's own channel allocations is the only cause for any co-channel interference.

2.1 Channel Assignment Policies

Policies for assigning and reassigning channels can be categorized into three main types, depending upon whether all, some or none of the channels remain permanently pre-allocated to cells. Central to all of them is *channel packing*, which is the idea of using the same channel in cells that are as close as possible to each other without violating the reuse constraint.

2.1.1 Fixed Channel Assignment

In Fixed Channel Assignment (FCA), the grid is first labelled with the minimum amount of labels such that no two cells spaced less than the reuse distance apart from each other have the same label (Figure 2.1). Then, the channels are partitioned evenly into the labels. The partition of channels pre-allocated to a cell are called its *nominal channels*, and a cell may only use channels from this partition. Since all nominal channels in all cells can be used simultaneously without violating the reuse constraint, channel reassignment on call termination does not improve system utilization and is therefore not performed.

By convention, cellular grids are modelled with hexagonal cells. For certain hexagonal grid shapes, cells in some of the corners will have a set of interfering neighbors that, including the cell itself, does not span the entire range of labels. These corner cells should then expand their set of nominal channels to include the partitions that are not within the

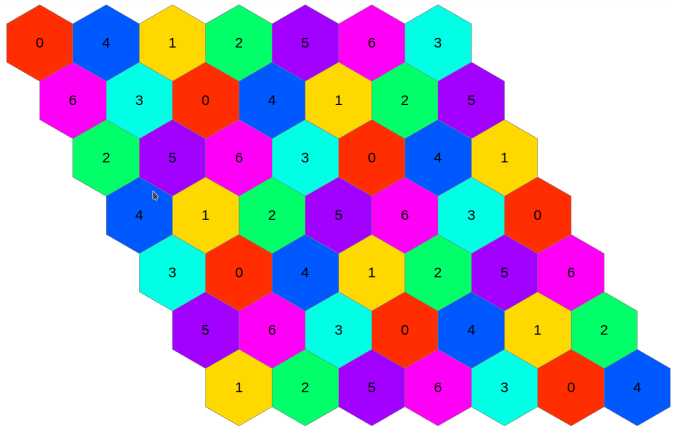


Figure 2.1: A 7×7 grid divided into 7 partitions. The number within a cell (and its color) designates which nominal channel partition it belongs to. No two cells with distance of 2 or less of each other have the same label. A reuse distance of 3 is therefore safe to use without risking interference.

reuse distance. In the rhombus shaped grid in Figure 2.1, this applies to the cell in the top left-hand corner and the cell in the bottom right-hand corner, assuming the reuse distance is 3.

Provided that corner cells have their set of nominal channels expanded where appropriate, FCA is optimal in the limit of infinite call traffic (McEliece and Sivarajan, 1994). Still, it is clear why FCA can perform sub-optimally under lighter traffic conditions. Suppose there is a hot spot in the grid with high traffic, and a surrounding area with low traffic, then a cell in the hot spot will be limited to serving as many mobile clients as its number of nominal channels. Cells in the neighboring low-traffic area might have unused channels, thus leading to sub-optimal grid usage. If the hot spot is not a temporary phenomenon, then the problem can be alleviated by partitioning channels unevenly into the cells. However, FCA is a static policy and unable to adapt to temporal variations in service demand. Furthermore, cellular systems are designed to operate at a fraction of their full capacity (Lilith, 2005), making low traffic performance more important.

2.1.2 Dynamic Channel Assignment

With Dynamic Channel Assignment (DCA) policies, cells are not restricted to use a subset of the channels. Every channel is available for assignment in every cell, unless using the channel violates the reuse constraint in the current grid conditions. The main idea behind all DCA policies is to rank the channels in each cell using a cost function, the choice of which is what differs between schemes (Katzela and Naghshineh, 1996).

Random assignment is the simplest example of a DCA policy, where a channel is chosen at random from the set of eligible channels, without any reassignment done on call termination. A random policy can adapt better to the hot spot scenario than FCA, because high-activity cells have the ability to use a greater portion of the total bandwidth than when the set of available channels remain fixed as in FCA.

2.1.3 Hybrid Channel Assignment

Hybrid Channel Assignment (HCA) policies pre-allocate some nominal channels to cells, while leaving some portion free to be used anywhere on the grid. This helps alleviate the issue highlighted in the previously mentioned hot spot example were neighboring cells have a different amount of traffic and therefore require a different number of channels. The best-performing example of such an approach is Borrowing with Direction Channel Locking (BDCL) (Ming and Yum, 1989). If a cell temporarily requires more channels than its the number of nominal channels, it may borrow a channel from a neighboring cell. Because the borrowing cell has a different set of interfering neighbors than the cell it borrows a channel from, a complex scheme of book-keeping makes sure that borrowing does not violate the reuse constraint. While BDCL is regarded as a powerful heuristic it is not practically implementable (Lilith, 2005) because a single call departure can cause an arbitrary number of reassignments in any cell (Singh and Bertsekas, 1997).

2.2 Reassignment policies

Because the domain cost of channel reassignment is not explicitly known to us, this work follows convention of only considering one reassignment per departure event. The reassignment is constrained to the same cell as the departure event.

A single channel reassignment per call termination is not sufficient to allow optimal grid utilization. Consider an example where the channels $[ch_1, ch_2, \ldots, ch_5]$ for a particular cell are valued in a strictly decreasing order. If both ch_1 and ch_2 are in use by an interfering neighbor when three calls arrive, ch_3 will be assigned first, then ch_4, and at last ch_5. If the neighboring calls on ch_1 and ch_2 both depart, and our call on ch_3 departs, the optimal action is to reassign the call on ch_5 — the minimally valued channel — to $ch1$. That leaves a call in progress on $ch4$, which is not the maximum valued free channel in the cell, and which could be reassigned to $ch2$ for a higher valued configuration, if we were to allow multiple reassignments.

2.3 Related tasks for cellular telephone networks

Another task related to channel allocation is Call Admission Control (CAC) (e.g. Chen and Jia (2009)), where a policy may decide to drop a call in progress or deny call requests even though eligible channels are available. The motivation for doing so can be to prioritize calls which generate more profit for the service provider, or to improve fairness between different categories of mobile clients. Also, if the bandwidth varies with time, denying call requests before a known or predicted bandwidth reduction will reduce the probability of dropping calls in progress. It is generally less desirable to drop a call in progress than to deny a service request. Many CAC agents are based on the guard channel approach (Hong and Rappaport, 1986). This approach reserves some number of channels which are only used for accepting hand-offs. The number of reserved channels can differ between cells, and need not remain fixed. During network operation, the CAC agent continuously adjusts the number of reserved channels in order to trade off hand-off versus new call blocking

probability according to an objective function given a priori. Using guard channels increases new call blocking probability because fewer channels are available to serve new calls (Wong, 2003).

Power control (e.g. Biggelaar et al. (2012)) is also done in caller environments. If a caller is close to the base station or uses high-grade equipment, the BS can reduce its transmit power. By regulating the transmit power for each cell and channel individually, the reuse constraint need not be set for the worst-case scenario but can vary for each cell-channel pair.

Instead of minimizing call blocking probability subject to limited bandwidth and a reuse constraint, alternate objectives are commonly studied. Graph theory (e.g. Lin and Shen (2018)) usually deals with minimizing the total number of channels used in a network given fixed service demand and reuse constraint, both of which must be fully satisfied. Other work (e.g. Kunz (1991)) aim to minimize interference given limited bandwidth while fully satisfying service demand. In this setting there is no reuse constraint, instead pairs of channel usages interfere with each other and the degree of interference is a function of geographic distance between cells, distance in the channel domain between the channels and transmit power.

We will not perform CAC or power control in this thesis, but look at some previous work since both problems and their solution methods share a lot of similarities with channel allocation.

2.4 Centralized and decentralized networks

In this work, it is assumed that the network is centralized where the state of every BS is known to a central agent. For large networks, the total amount of information that needs to be transferred to the central agent can be prohibitively large. Furthermore these networks rely on a single point of failure and are badly suited for cases where base stations are dynamically added and removed during network operation.

In decentralized systems, each BS has its own DCA agent. The degree of decentralization is a measure of the amount of information each agent has about its surroundings, and differs between proposed solutions. Pure distributed DCA systems has no information exchange between BSs (Morozs, 2015), while less decentralized system might exchange channel usages between nearest neighboring cells. The agents in a decentralized system have less information to act on and usually suffer a significant increase in blocking probability compared to their centralized counterparts because of it.

2.5 Call traffic modelling

Call traffic is by convention modelled by the memory-less exponential distribution (Ming and Yum, 1989). That is, call duration τ follows the distribution:

$$f(\tau;\mu) = \frac{1}{\mu}\exp^{-\tau/\mu} \tag{2.1}$$

where μ is the mean call duration. Similarly, mean time between two arriving calls, i.e. inter-arrival time Δt, is modelled independently for each cell by:

$$g(\Delta t; \lambda) = \lambda \exp^{-\Delta t \lambda} \tag{2.2}$$

where λ is the mean call rate. This is equivalent to modelling call arrivals following a Poisson distribution with parameter λ. The Poisson distribution is considered unsuited for modelling the data packet traffic (Paxson and Floyd, 1995), but this work considers voice traffic.

On scheduled call departure, calls hand off to a nearby cell with probability $p_{handoff}$ instead of terminating. The arrival cell is chosen uniformly at random from the set of neighbors with distance 1 from the cell of the departing call. Hand-offs have mean duration separate from regular calls, $\mu_{handoff}$, which is usually lower.

Call traffic is independent for each cell, and the collection of parameters for the whole grid defines the traffic pattern. The simplest traffic pattern is a uniform pattern where all cells share the same parameters for call duration and inter-arrival times. Non-uniform traffic patterns where the offered traffic varies between cells highlights how different policies handles spatial variations. Time-varying traffic and delayed equipment failure, i.e. disabling cells after some time period, are of special interest when comparing policies that learn.

2.6 Summary

As we continue forward, keep in mind our task of performing Dynamic Channel Allocation in a centralized network using a limited set of channels, given that the reuse constraint must be satisfied, and as a consequence, service demand may not. As such, the objective is to minimize the probability of blocking service requests, and preferably, to prioritize hand-off requests over regular new call service requests. Service requests will not be denied if they can be accepted without violating the reuse constraint, nor will any calls in progress be terminated before the caller decides to do so. The task of the DCA agent is then as simple as assigning and reassigning channels in a manner that minimizes the probability of not being able to accept service requests.

Chapter 3

Basic Theory

Reinforcement Learning (RL) is a set of methods concerned with how an agent ought to behave in an environment. The agent observes the state s_t of the environment, decides on an action a_t and executes it. In turn, the environment emits a numerical reward signal r_t and transitions to the next state, s_{t+1}. The goal of the agent is to maximize the reward output of the environment over a infinite time horizon or until the environment reaches a terminal state.

RL is suitable for online learning, that is, learning as data becomes available as opposed to learning techniques which learn on an entire data set at once. In general, knowledge of the environment dynamics is not required which contrasts to classical dynamic programming. Online learning is important in DCA due to the dynamic nature of the caller environment where temporal variations such as time-of-day fluctuations can arise.

In a DCA, the RL agent can be thought of as event-driven. The occurrence of call arrival, departure or hand-off events triggers the need for action, and, in conjunction with the current state of the system, determines which actions are possible to perform. RL has enticing facets for the problem of DCA in cellular networks, namely its ability to adapt to changing dynamic environments and ability to deal with state-spaces that are prohibitively large for traditional dynamic programming.

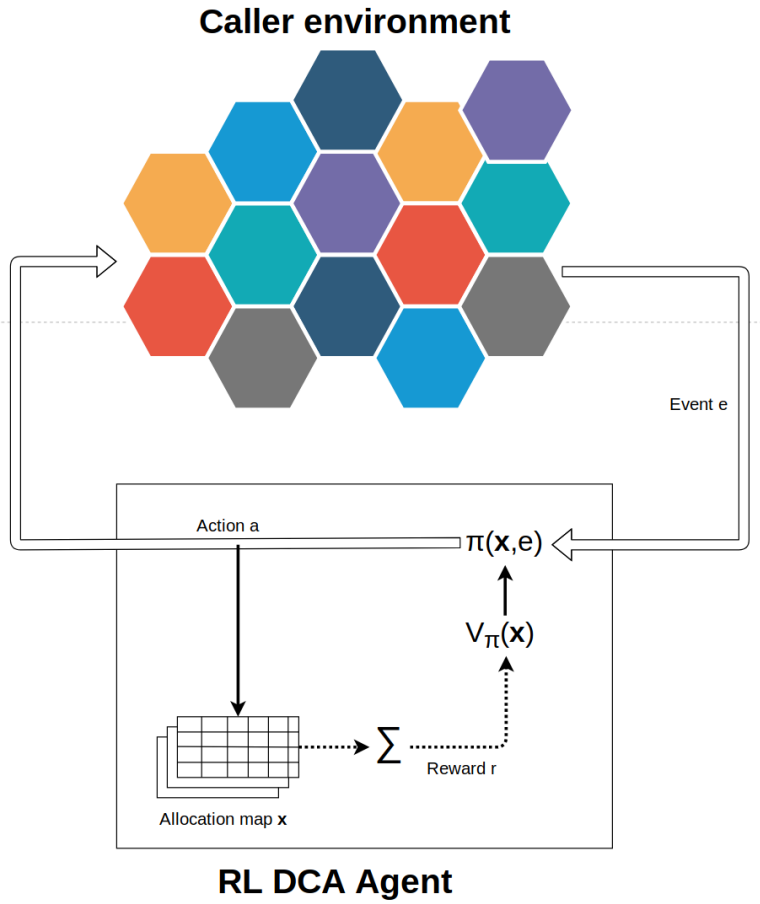
First, lets define the channel allocation problem formally in terms suitable for RL methods.

3.1 Environment specification

3.1.1 Markov Decision Processes

A Markov Decision Process (MDP) formally describes a fully observable, possibly stochastic environment. A MDP is defined by the tuple:

$$(\mathcal{S}, \mathcal{A}(\mathrm{s}), \mathcal{P}^{\mathrm{a}}_{\mathrm{ss}\prime}, \mathcal{R}^{\mathrm{a}}_{\mathrm{s}}, \gamma). \tag{3.1}$$



A state value-based reinforcement learning agent. Each time step, the agent receives an event e from the environment. A value-based policy π selects an action in response to the event using the allocation map which is an internal model of the state of the grid. The corresponding reward for that action is the sum of calls in progress after the action has been executed. The reward is used to update the value-function's (V_π) valuation of the pre-action allocation map.

Figure 3.1: RL DCA event-action-train cycle

Here, $\mathcal{S}$ is the finite set of environment states and $\mathcal{A}(\mathrm{s})$ is a function from a particular state $s \in \mathcal{S}$ to a finite set of actions available to the agent in that state. The environment dynamics are said to satisfy the Markov property, if and only if:

$$\Pr[S_{t+1} \mid S_t] = \Pr[S_{t+1}|S_1, \ldots, S_t]. \tag{3.2}$$

That is, the future (S_{t+1}) is conditionally independent of the past ($S_1, \ldots, S_{t-1}$) given the present (S_t), and knowledge of the past therefore does not help predict future states. The state transition probability function $\mathcal{P}^{\mathrm{a}}_{\mathrm{ss}\prime} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$

$$\mathcal{P}^{\mathrm{a}}_{\mathrm{ss}\prime} = \Pr[S_{t+1} = s' \mid S_t = s, A_t = a] \tag{3.3}$$

yields the probability of transitioning to state s' if the agent executes action a in state s. The reward function $\mathcal{R}^{\mathrm{a}}_{\mathrm{s}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$$\mathcal{R}^{\mathrm{a}}_{\mathrm{s}} = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] \tag{3.4}$$

is the *expected* reward for taking action a in state s. We use the notation $\mathcal{R}^{\mathrm{a}}_{\mathrm{s}}$ for the expectation of reward, r and r_t for an observed or fixed reward, and R_t for reward as a random variable, and similar notation for the other quantities. Lastly, the discount factor $\gamma \in [0, 1)$ represents the trade-off between short-term versus long-term reward, where $\gamma = 0$ fully prioritizes immediate, short-term rewards.

A strategy $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for choosing actions when navigating an environment defined by a MDP, termed the *policy*, assigns action probabilities for each action in each possible state:

$$\pi(a \mid s) = \Pr[A_t = a \mid S_t = s] \qquad \text{with} \qquad \sum_{a \in \mathcal{A}(\mathrm{s})} \pi(a \mid s) = 1. \tag{3.5}$$

The policy defines the behavior of an agent in the environment. A policy is memory-less; it depends on the current state and not on the history of states. Policies can be deterministic, in which case a single action has probability one in each state and we shorten the notation to $a = \pi(s)$.

An agent executing actions according to policy π while interacting with the environment receives a reward at each time step, possibly zero. One measure for evaluating how well a policy performs is the sum of discounted rewards which an agent receives from time t on-wards, named the *return*:

$$G_t \doteq \sum_{i=t}^{\infty} \gamma^i R_{t+i} \tag{3.6}$$

The corresponding objective is to find a policy π as to maximize the return G_0.

3.1.2 Dynamic Channel Allocation as a MDP

We define a stochastic caller environment as a MDP by letting the state consist of the allocation map $\mathbf{x}_{\mathrm{t}}$ and the event e_t that causes the state transition to the next state s_{t+1}; i.e. $s_t = (\mathbf{x}_{\mathrm{t}}, e_t)$.

The allocation map $\mathbf{x}_\mathrm{t} \in \{0,1\}^{I \times K}$ is a matrix which describes the state of the network by specifying which channels are in use at each cell (using function notation for indecies):

$$\mathbf{x}_\mathrm{t}(i,k) \doteq \begin{cases} 1, & \text{if channel } k \text{ is in use in cell } i \text{ at time } t \\ 0, & \text{otherwise} \end{cases} \tag{3.7}$$

for all I cells, $\forall i \in \mathcal{I} = \{1,2,\ldots,I\}$, and K channels, $\forall k \in \mathcal{K} = \{1,2,\ldots,K\}$. Sometimes we also talk about the allocation map of a specific cell, e.g. $\mathbf{x}_\mathrm{t}(i,\cdot)$ which is a K-length vector specifying the channel usages at cell i.

State transitions occur when calls arrive, depart, or hand off; we model the latter as a call departure event in one cell and an arrival event in another. Arrival events consist of the cell number i for the service request and time of arrival t. Departure events, in addition, specify the channel k of the departing call.

$$e_t \in \{\mathrm{NEW}_{\mathrm{i,t}}, \mathrm{END}_{\mathrm{i,k,t}}\} \tag{3.8}$$

We use the term event-cell for the cell i where the current event takes place.

On call arrival, the set of available actions $\mathcal{A}(\mathrm{s})$ is limited to assigning a channel that is free in the given cell and its interfering neighbors, or to block the call if no channels are eligible. On call departure, we define the set of actions to reassigning a call in progress to the newly freed channel, or doing no reassignment at all. Reassignments are only considered in the event-cell.

Let $d(i,j)$ denote the (symmetric) distance between cell i and cell j, such that the distance from a cell to itself is zero, and from itself to an immediate neighbor is one. Given a reuse distance of three, the set of interfering cells for cell i is:

$$\mathcal{I}_{IF}(i) \doteq \{j \in \mathcal{I} : d(i,j) < 3\} \tag{3.9}$$

which for convenience also contains cell i itself.

The action space for channel assignment actions corresponds to the set of eligible channels, $\mathcal{K}_\mathrm{EL}$:

$$\mathcal{A}(\mathbf{x}_\mathrm{t}, \mathrm{NEW}_{\mathrm{i,t}}) = \mathcal{K}_\mathrm{EL}(\mathbf{x}_\mathrm{t}, i) \doteq \{k \in \mathcal{K} : \sum_{j \in \mathcal{I}_{IF}(i)} \mathbf{x}_\mathrm{t}(j,k) = 0\}, \tag{3.10}$$

where an empty set indicates that the call must be blocked. Similarly, on call departure the action specifies the channel of a call in progress to be reassigned to channel k:

$$\mathcal{A}(\mathbf{x}_\mathrm{t}, \mathrm{END}_{\mathrm{i,k,t}}) = \{l \in \mathcal{K} : \mathbf{x}_\mathrm{t}(i,l) = 1\}, \tag{3.11}$$

where action k is always present in the set and equivalent to doing no reassignment at all.

Note that executing any action a_t on $\mathbf{x}_\mathrm{t}$ deterministically results in $\mathbf{x}_{\mathrm{t}+1}$ given the current event e_t. Let z denote this deterministic transition function, such that $\mathbf{x}_{\mathrm{t}+1} = z(\mathbf{x}_\mathrm{t}, e_t, a_t) = z(s_t, a_t)$. Assignment actions simply sets the channel specified by the action to 1 (in use) in the event-cell:

$$z(\mathbf{x}_\mathrm{t}, \mathrm{NEW}_{\mathrm{i,t}}, a_t)(j,l) \doteq \begin{cases} 1 \text{ if } j = i \text{ and } l = a_t \\ \mathbf{x}_\mathrm{t}(j,l) \text{ otherwise,} \end{cases} \tag{3.12}$$

while reassignment actions on departure events specify which channel to set to 0 (free):

$$z(\mathbf{x}_\mathrm{t}, \mathrm{END}_\mathrm{i,k,t}, a_t)(j,l) \doteq \begin{cases} 0 \text{ if } j = i \text{ and } l = a_t \\ \mathbf{x}_\mathrm{t}(j,l) \text{ otherwise.} \end{cases} \tag{3.13}$$

Logistically, the reassignment action specifies which channel to mark as free. If a call on channel k departs, and the action is to reassign a call on channel a to channel k, this is equivalent to letting channel k stay marked as in use while marking channel a as free.

Let $c(\mathbf{x})$ be the total number of calls currently in progress system-wide:

$$c(\mathbf{x}_\mathrm{t}) \doteq \sum_{i=1}^{I} \sum_{k=1}^{K} \mathbf{x}_\mathrm{t}(i,k). \tag{3.14}$$

We define rewards $r_t = r(s_t, a_t)$ as the number of calls in progress immediately after executing action a_t on the allocation map $\mathbf{x}_\mathrm{t}$ of the state s_t:

$$r(s_t, a_t) \doteq c(z(s_t, a_t)) \qquad \text{(calls in progress } \textit{after} \text{ executing } a \text{ on } s) \tag{3.15}$$

$$= \mathcal{R}_{s_t}^{a_t} \qquad \text{(deterministic rewards)} \tag{3.16}$$

and are deterministic due to the functions z and c being deterministic. The corresponding discounted return for this reward definition is then:

$$G_t \doteq \sum_{i=t}^{\infty} \gamma^i c(\mathbf{x}_{t+i+1}). \tag{3.17}$$

Provided we know the model of the environment, $\mathcal{P}_{\mathrm{ss}\prime}^{\mathrm{a}}$ and $\mathcal{R}_{\mathrm{s}}^{\mathrm{a}}$, the above MDP is solvable by dynamic programming methods such as value iteration (see e.g. Sutton and Barto (2018)), which has complexity $O(mk^2)$ per iteration where $m = |A|$ is the size of the action space and $k = |\mathcal{S}|$ is the state space size.

Given I cells and K channels, the state space is 2^{I*K} in size, including illegal states with channel assignments that violate the reuse constraint. For a 7-by-7 grid with 70 channels, this yields a state space of size $2^{7*7*70} = 2^{3430}$, which is much too large to solve by dynamic programming. In fact, the problem of DCA as stated above is equivalent to the Euclidian graph coloring problem and is therefore NP-hard (Brunato, 1999).

3.1.3 Semi-Markov Decision Processes

In a MDP as defined above, state transitions emit a single, instantaneous reward at the beginning of each state transition. Semi-Markov Decision Processes (SMDPs) generalize MDPs, where reward is continuously accumulated during a state visit and the length of the time interval from one transition to the next is stochastic and possibly continuous (Bertsekas, 2005). In general, state transitions themselves may be continuous — e.g. increasing the speed of a car by 5 km/h — but in DCA and most related caller environments tasks, state transitions are discontinuous. Formulating a problem as a SMDP is necessary when the reward depends on the time spent in a state and when this duration is not fixed.

The return G_t^S of an agent operating in a SMDP caller environment from time t onwards is defined as:

$$G_t^S = \int_t^\infty e^{-\beta\tau} r(S(\tau), A(\tau)) d\tau \tag{3.18}$$

where the function $S(t_k) = S_k$ such that $S(t) = S_k$ for $t_k \leq t < t_{k+1}$ and similarly for $A(t_k)$. The discount factor $\beta > 0$ has a similar but not identical role to γ in MDPs, while $r(\cdot)$ is interpreted as a reward *rate*, e.g. dollars per day or accepted calls per hour.

We define DCA as a SMDP by using the number of calls in progress as reward rate, i.e. $r(s, a) \doteq c(z(s, a))$ as before, while the duration of state transitions are determined by the occurrence of call events. The state and action space remain the same as in the MDP, but state transition probabilities incorporate the time spent in a transition:

$$\mathcal{P}^a_{s\tau s'} = \Pr[S_{k+1} = s', t_{k+1} - t_k \leq \tau \mid S_k = s, A_k = a]. \tag{3.19}$$

Because the number of calls in progress is constant between call events, the expected reward for a single transition from state s_k at time t_k to the next state $s_{k+1} = (\cdot, e_{k+1})$ at time t_{k+1} with *sojourn time* $\Delta t = t_{k+1} - t_k$ can be simplified (see e.g. Bertsekas (2005)):

$$\mathcal{R}^{a_k}_{s_k} = \mathbb{E}_{e_{k+1}}[\int_{t_k}^{t_{k+1}} e^{-\beta\tau} r(S(\tau), A(\tau)) d\tau \mid S_k = s_k, A_k = a_k] \tag{3.20}$$

$$= r(s_k, a_k)\, \mathbb{E}_{\Delta t}[\int_0^{\Delta t} e^{-\beta\tau} d\tau \mid S_k = s_k, A_k = a_k] \tag{3.21}$$

$$= c(z(s_k, a_k))\, \mathbb{E}_{\Delta t}[\frac{1 - e^{-\beta\Delta t}}{\beta} \mid S_k = s_k, A_k = a_k]. \tag{3.22}$$

Unlike in the MDP formulation, rewards are stochastic because they depend on the time spent in a state, which given current state and action, is determined by the time of the next event.

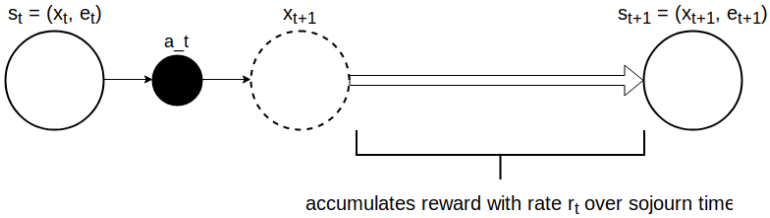


Figure 3.2: SMDP transition diagram. Selecting an action a_t and executing it on the grid $\mathbf{x}_t$ which transitions to $\mathbf{x}_{t+1}$ is assumed to be instantaneous. The reward rate, which is the sum of calls in progress on the grid $\mathbf{x}_{t+1}$, accumulates over the sojourn time determined by the occurrence of the next event e_{t+1}.

Maximizing the SMDP return (Equation 3.18) is a potentially biased way of minimizing call blocking probability, because the duration of calls influence their reward. If, for example, the agent is allowed to drop calls in progress in order to accept new ones, then ignoring call duration by using MDP returns (Equation 3.17) will not dissuade the agent

from dropping a call in order to accept a new one on the same channel, an action which decreases call blocking probability, but is not preferable in terms of user satisfaction.

We will visit both MDP and SMDP formulations in related work (chapter 4), but assume the simpler MDP formulation if not stated otherwise. SMDP formulations have rewards which depend on sojourn time while MDP formulations do not, making it easy to distinguish them from each other.

3.1.4 The average reward optimality criterion

In a caller environment, there is no clear reason why blocking a call tomorrow is preferential to blocking a call today, as implicit in the discount factor of both the MDP and SMDP formulation of returns.

Instead, a policy π can be evaluated by its average reward ρ_π, which for a MDP (under some conditions) is given by (Sutton and Barto, 2018):

$$\rho_\pi \doteq \lim_{h\to\infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}[R_t \mid A_{0:t} \sim \pi] = \lim_{t\to\infty} \mathbb{E}[R_t \mid A_{0:t} \sim \pi] \tag{3.23}$$

where the expectation is over trajectories and possibly stochastic rewards.

Corresponding to the discounted return (Equation 3.6), the *differential* MDP return $\bar{G}_t$ is given by:

$$\bar{G}_t \doteq R_t - \rho_\pi + R_{t+1} - \rho_\pi + R_{t+2} - \rho_\pi + \ldots = \sum_{i=t}^{\infty} (R_{t+i} - \rho_\pi). \tag{3.24}$$

Similar definitions exist for the SMDPs (section 4.9). Using the average reward as an optimality criterion avoids biasing the return towards short-term rewards, while keeping it bounded. Simply setting $\gamma = 1$ for discounted returns in an infinite-horizon environment results in an unbounded return.

Optimizing the average reward instead of the discounted sum of rewards therefore seems like the natural choice for a caller environment. However algorithms for optimizing the discounted case are by far the most studied in the context of RL and often come with convergence guarantees while their average-reward counterparts do not (Tadepalli, 2017).

Hereon, we will assume the simpler discounted case if not stated otherwise.

3.2 Reinforcement Learning

Continuing on from the definition of a MDP, the *value* of a state is defined as the expectation of the return:

$$v_\pi(s) \doteq \mathbb{E}_{A_t, S_{t+1}, A_{t+1}, \ldots}[G_t \mid S_t = s]. \tag{3.25}$$

The expectation is over actions from a possibly stochastic policy, $A_t \sim \pi(a \mid S_t = s)$, and state transitions according to the transition distribution $S_{t+1} \sim \mathcal{P}^{\mathrm{a}}_{\mathrm{ss}\prime}$.

Predictive reinforcement learning methods aim to estimate the value-function v_π under a fixed policy π, while control methods aim to learn a policy that maximizes the value-function for each state. The value of a state or state-action pair under a fixed policy is

known to satisfy the recursive Bellman expectation equation, which we obtain by expanding the return by one time step:

$$v_\pi(s) \doteq \mathbb{E}[G_t \mid S_t = s] = \mathbb{E}[R_t + \gamma v_\pi(S_{t+1}) \mid S_t = s] \tag{3.26}$$

$$= \sum_{a \in \mathcal{A}(\mathrm{s})} \pi(a \mid s)[\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss\prime}^a v_\pi(s')] \tag{3.27}$$

The state-action values, or *Q-values*, is the value of executing given action a in state s and then selecting actions according to a fixed policy π thereafter:

$$q_\pi(s,a) \doteq \mathbb{E}_{S_{t+1}, A_{t+1}, \ldots}[G_t \mid S_t = s, A_t = a] \tag{3.28}$$

$$= \mathbb{E}[R_t + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \tag{3.29}$$

$$= \mathcal{R}_{\mathrm{s}}^{\mathrm{a}} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{\mathrm{ss}\prime}^{\mathrm{a}} \sum_{a' \in \mathcal{A}(\mathrm{s})} \pi(a' \mid s') q_\pi(s', a') \tag{3.30}$$

Note that the value of a state and state-action pair can be expressed in terms of each other:

$$v_\pi(s) = \sum_{a \in \mathcal{A}(\mathrm{s})} \pi(a \mid s) q_\pi(s,a) \tag{3.31}$$

$$q_\pi(s,a) = \mathcal{R}_{\mathrm{s}}^{\mathrm{a}} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{\mathrm{ss}\prime}^{\mathrm{a}} v_\pi(s') \tag{3.32}$$

Value-based control methods aim to find the value-function of an optimal policy, which satisfies the Bellman optimality equation formulated as an optimal state value-function $v_*(s)$ or an optimal state-action value-function $q_*(s,a)$:

$$v_*(s) \doteq v_{\pi_*}(s) = \max_\pi v_\pi(s) \tag{3.33}$$

$$q_*(s,a) \doteq q_{\pi_*}(s,a) = \max_\pi q_\pi(s,a) \tag{3.34}$$

If the reward function is stationary (even if stochastic), there is guaranteed to exist a *deterministic* policy which is optimal for Equation 3.33 and Equation 3.34 (Puterman, 2005). The conditions under which the reward function is stationary in a caller environment is touched upon in section 4.13.

Assuming the existence of an optimal deterministic policy, we simplify by maximizing over actions instead of the policy:

$$v_*(s) = \max_{a \in \mathcal{A}(\mathrm{s})} [\mathcal{R}_{\mathrm{s}}^{\mathrm{a}} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{\mathrm{ss}\prime}^{\mathrm{a}} v_*(s')] = \max_{a \in \mathcal{A}(\mathrm{s})} q_*(s,a) \tag{3.35}$$

$$q_*(s,a) = \mathcal{R}_{\mathrm{s}}^{\mathrm{a}} + \gamma \sum_{s'} \mathcal{P}_{\mathrm{ss}\prime}^{\mathrm{a}} \max_{a' \in A(s')} q_*(s', a') = \mathcal{R}_{\mathrm{s}}^{\mathrm{a}} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{\mathrm{ss}\prime}^{\mathrm{a}} v_*(s'). \tag{3.36}$$

The optimal state-action value $q_*(s,a)$ yields the value of first executing an arbitrary given action a and then following the optimal policy π_* thereafter, which is why the first action remain fixed in Equation 3.36 while the second is maximized over.

Similarly, the average reward Bellman optimality equations follow from the definition of differential returns (Equation 3.24), the only difference being the subtraction of the average reward ρ_* from the reward:

$$\bar{v}_*(s) = \max_{a \in \mathcal{A}(\mathrm{s})} [\mathcal{R}^{\mathrm{a}}_{\mathrm{s}} - \rho_* + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{\mathrm{a}}_{\mathrm{ss}'} \bar{v}_*(s')] \tag{3.37}$$

$$\bar{q}_*(s,a) = \mathcal{R}^{\mathrm{a}}_{\mathrm{s}} - \rho_* + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{\mathrm{a}}_{\mathrm{ss}'} \max_{a' \in A(s')} \bar{q}_*(s', a') \tag{3.38}$$

These are termed average-adjusted value-functions.

In either case, the expectations over rewards and state transitions can be handled by integrating over all possible outcomes, termed a full width backup, in which case the model $\mathcal{R}^{\mathrm{a}}_{\mathrm{s}}$ and $\mathcal{P}^{\mathrm{a}}_{\mathrm{ss}'}$ is required and the equations define dynamic programming approaches. RL methods observe or execute an action and a sample reward and state transition is then received from a possibly stochastic environment.

Value-based RL methods iteratively learns an approximation to one of the value functions defined above. As an illustrative example, consider the predictive state value method TD(0) (Sutton, 1988):

$$V_{t+1}(s_t) = V_t(s_t) + \alpha[r(s_t, a_t) + \gamma V_t(s_{t+1}) - V_t(s_t)] = V_t(s_t) + \alpha\delta_t \tag{3.39}$$

where actions are sampled from the policy π that we wish to learn the value of. $\alpha \in (0, 1)$ is the learning rate and is usually decreased with time though we omit the time step subscript. The inner term δ is known as the temporal difference (TD) error, and contains the value target $r(s_t, a_t) + \gamma V_t(s_{t+1})$ which the value of state s_t is updated towards, where $V_t(s_{t+1})$ is known as the bootstrapping value.

The TD error is a sample of the Bellman error, which arises from subtracting the left hand side of a stochastic approximation of the Bellman expectation equation (Equation 3.27) from the right hand side:

$$\mathbb{E}[\delta_t] = -V_t(s_t) + \sum_{a \in \mathcal{A}(s_t)} \pi(a \mid s_t)\, \mathcal{R}^{\mathrm{a}}_{\mathrm{s}} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{\mathrm{a}}_{\mathrm{ss}'}\, V_t(s_{t+1}). \tag{3.40}$$

Model-based and model-free methods

Value-based methods have an implicit policy; actions are selected using the value-function. Policy-based methods on the other hand explicitly represent the action probabilities for each state, as in Equation 3.5. The reason to use state-action $q(s, a)$ values (or an explicit policy) instead of state values $v(s)$ is that a model of the environment dynamics is not necessary in order to select an action based on the estimated return.

Given a state value-function $v_\pi(s)$, the MDP environment model $\mathcal{R}^{\mathrm{a}}_{\mathrm{s}}$ and $\mathcal{P}^{\mathrm{a}}_{\mathrm{ss}'}$ is necessary to *greedily* choose an action:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}(\mathrm{s})} [\mathcal{R}^{\mathrm{a}}_{\mathrm{s}} + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}^{\mathrm{a}}_{\mathrm{ss}'}\, v_\pi(s')]. \tag{3.41}$$

The model is required in order to determine the expected reward and transition probabilities for the next states after taking each of the allowable actions in the current state.

The above equation is a non-strict policy improvement, that is, $V_{\pi'}(s) \geq V_\pi(s)$, which is satisfied as an equality if policy π is optimal, that is, if $V_\pi(s) \geq V_{\hat{\pi}}, \forall s, \forall \hat{\pi}$. Greedy action selection using $q_\pi(s, a)$, on the other hand, is model-free:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}(s)} q_\pi(s, a) \tag{3.42}$$

Model-free approaches are suitable when the MDP is unknown, or when the MDP is known but too large to store in memory. In the case of DCA, the reward function is known and deterministic but event probabilities are assumed unknown.

The model might be known a priori or learned online, and model-based RL covers a broad set of methods not limited to selecting actions using a model. If a model is known, it can e.g. be used to generate trajectories of state transitions and rewards for improving the policy or value-function. This is desirable if it is more expensive to generate experience from the learning environment, such as a simulator or a real-world environment, than it is to generate trajectories using a model.

On-policy and off-policy methods

In the simplest form of value-based RL methods, the value-function estimates the value of the same policy which generates the data it is trained on. Given a policy π, the trajectory data $S_0, A_0, R_0, S_1, A_1, R_1, \ldots$ is generated by policy π choosing actions while traversing the MDP, i.e. $A_t \sim \pi(a \mid S_t)$ where in turn π uses estimate of either q_π or v_π to select actions. This trajectory data is used to improve the estimates of q_π or v_π, and is termed *on-policy* training.

In off-policy training, the value-function estimates the value of a *target policy* which differs from the policy used to generate the trajectory data it is trained on. The latter is termed the *behavior policy*, and usually selects actions in a semi-random fashion in order to ensure exploration of the state space, while the target policy is strictly greedy (e.g. Equation 3.41).

If actions executed in the environment are always selected greedily, the agent might not explore parts of the state space which yield better returns than what the value-function currently deem the best states and actions. Greedy action selection does not imply selecting actions based on short trajectory returns as opposed to long trajectory returns, but on a short- versus long-term learning time frame.

Exploration is done by selecting actions according to a stochastic scheme. The simplest example is ϵ-greedy, which despite its simplicity performs well and is popular in RL applications (Silver, 2015). With an ϵ-greedy policy, a valid action is picked uniformly at random with probability ϵ while the highest-valued action is selected with probability $1-\epsilon$. The need for exploration decreases as the value of a larger part of the state-space is accurately estimated. For ϵ-greedy this is achieved by ϵ decaying over time, e.g. $\epsilon_t = \frac{\epsilon_0}{\log(t)}$.

3.2.1 Dealing with large state spaces

Storing a value-function $V_\pi(s), \forall s$, a state-action value-function $Q_\pi(s, a), \forall (s, a)$ or an explicit, deterministic policy $\pi(s), \forall s$ in memory might not be feasible. As determined at the end of subsection 3.1.2, storing a single value for each state in a table would require too

much memory for even a toy sized caller environment. RL by *table lookup*, as the approach is termed, results in too large tables and suffers from the same 'curse of dimensionality' as dynamic programming methods.

There are two main approaches to dealing with large state spaces, which are interrelated and may be combined. The first is to use a feature based representation of the state where a state s is mapped to an approximate representation $\tilde{s} = \phi(s)$ which captures only its essential features. For there to be any reduction in the size of the state space, the mapping must be non-injective, i.e. multiple states must map to the same feature representation, which implies that the environment becomes partially observable.

For DCA, a natural approach to feature representation is to only capture information about the event-cell and possibly its neighbors within some distance. In this case the feature representation is a local representation, which is a feasible approach under the assumption that the most important information for selecting an action resides in and around the cell where the action is to be executed. The condensed information of a cell might be the number of channels in use, the number of free channels, or the number of channels eligible for assignment.

A second approach to dealing with large state spaces is to use a parametrized function approximator with parameters θ for the value-function, i.e. $V_\theta(s) \doteq V_{\theta_\pi}(s) \approx V_\pi(s)$, the state value-function $Q_\theta(s, a) \approx Q_\pi(s, a)$ or the policy $\pi_\theta(s) \approx \pi(s)$. Artifical neural networks are popular function approximators in RL because their input-output computation scale linearly with the number of parameters in the network.

For a given set of features, table lookup will always give equal or better results than using function approximation, if enough time is given for the value-function to converge and the dynamics are stationary, because a function approximator can at best yield the same values as an optimal table. In practice, even if the table can be stored in memory, the feature space might be so large that convergence will take too long. It is too slow to learn the value of each state or state-action pair independent of each other.

3.3 Artificial Neural Networks in RL

Artifical Neural Networks (ANNs) are currently the most commonly used function approximator for RL (Silver, 2015). This combination has led to many impressive results, including super-human performance in Chess (Silver et al., 2017), Go (Silver et al., 2016), and a sizeable portion of the Atari 2600 games (Mnih et al., 2013). The neural network can be used as a function approximator for the state value-function, the state-action value-function, for a policy or combinations of these. It is particularly tempting to use them for state-action methods because the network architecture can be designed to output all q-values or action probabilities for a given state at once (Mnih et al., 2013), thus eliminating the need for multiple forward passes when selecting actions.

ANNs are universal function approximators loosely modelled after neurons in the brain (Goodfellow et al., 2017). Layers of nodes connect input nodes to output nodes, with connections in-between. Connections between nodes are weighted, and adjusting these weights allows the network to be trained. For feed-forward networks, which are the only kind considered in this work, each node is a function of the weighted sum of the nodes in the previous layer. Linear neural networks use the identity function while non-linear

neural networks use continuous or approximately continuous functions at each layer. The output $o_{l,i}$ of neuron i in layer l is given by:

$$o_{l,i} = f(\sum_{j=1}^{J} w_{l-1,j,i} o_{l-1,j} + b_{l,i}) \tag{3.43}$$

where f is the chosen activation function, J is the number of neurons in the upstream layer $l-1$, i.e. the layer closer to the input, $w_{l-1,j,i}$ is the weight from node j in the previous layer to node i, and $b_{l,i}$ is an optional bias neuron which itself has no inputs but outputs a constant value that can also be trained. Hereon we will use vector notation instead:

$$o_{l,i} = f(\mathbf{w}_{l-i,\cdot,i}^T \mathbf{o}_{l-1,\cdot} + b_{l,i}) \tag{3.44}$$

with $\mathbf{w}_{l-i,\cdot,i}, \mathbf{o}_{l-1,\cdot} \in \mathcal{R}^J$, which for a single-layer, single output linear neural network with inputs $\mathbf{x}$ can be further reduced to $o = \mathbf{w}^T\mathbf{x} + b$.

The number of layers between the input and the output layer, termed the *hidden layers*, determines the depth of the network. For linear neural networks, no more than a single hidden layer is necessary because the matrix multiplication of the weight matrices of successive linear layers yields a single weight matrix, and there is no activation function in between. (Goodfellow et al., 2017).

3.4 Summary

Moving forward, there is a few key concepts to keep in mind about the intersection between channel allocation, MDPs, and RL.

First, state transitions are partly deterministic. Given an action, an event and the current allocation map, the next allocation map is easily determined. If we have an allocation map and are tasked with modelling the acceptance of a call in a given cell on a given channel, it is as easy as marking the channel as in use. Events, on the other hand, are fully stochastic. The actions we perform on the grid have no impact on where and when the next call will arrive, nor on when any call in progress will depart, or if it will hand off.

Second, RL methods can be categorized into state based methods and state-action based methods. In the former category the only inhabitant is the state value-function $V_\pi(s)$ which yields a measure of the desirability of being in state s. In the latter category there is a choice between state-action value functions and explicit policies, both of which yield a measure of the desirability of being in a state and executing a particular action. We also draw a distinction between implicit and explicit policies. In implicit policies, actions are selected based on their value as determined by the value function. In explicit policies, an action probability is given for each action in each state without using a value-function as intermediary.

Third, there are multiple viable targets to optimize for; four in total. Discounted MDPs are the simplest and most widely studied case, where the time spent in a state does not matter but rewards received in earlier time steps matter more. In discounted SMDPs, a reward rate accumulates during the time period spent in a transition, e.g. servicing a call for 2 minutes is better than servicing a call for 1 minute. At last there is an average

reward formulation of either the two previous, which measures long-run performance of the system where rewards are not discounted. Discounting is natural to do in economic domains where it is better to earn $50 today than tomorrow, but the same cannot be said of servicing calls.

Chapter 4

Related work

4.1 Singh et al.

Singh and Bertsekas (1997) were probably the first to apply RL to the problem of Dynamic Channel Allocation. They used a SMDP formulation for maximizing the time-integrated, discounted sum of calls in progress (Equation 3.18).

The RL agent, hereafter referred to as SB-VNet, utilizes the state value-function TD(0) (Equation 3.39) modified to do control instead of prediction:

$$V_{t+1}(\tilde{\mathbf{x}}_t) = V_t(\tilde{\mathbf{x}}_t) + \alpha\Big[\frac{1-e^{-\beta\Delta t}}{\beta} r(\mathbf{x}_\mathrm{t}, e_t, a_t) + e^{-\beta\Delta t} \max_{\tilde{\mathbf{x}}'} V_t(\tilde{\mathbf{x}}') - V_t(\tilde{\mathbf{x}}_t)\Big] \quad (4.1)$$

where $\tilde{\mathbf{x}}_t$ is a feature-based representation of the allocation map $\mathbf{x}_\mathrm{t}$, and the reward rate $r(\mathbf{x}_\mathrm{t}, e_t, a_t)$ is the number of calls in progress immediately after executing action a_t on $\mathbf{x}_\mathrm{t}$. Δt is the sojourn time, that is, the time difference between the events e_{t+1} and e_t.

The feature-based representation is hand-crafted and for each cell consists of the number of free channels and for each cell-channel pair the number of times the channel is used in a 4-cell radius. These features are the inputs to a linear neural network with parameters $\boldsymbol{\theta}$ serving as a function approximator to the state value-function. The value network is simply the inner product $V(\tilde{\mathbf{x}}; \boldsymbol{\theta}) = \boldsymbol{\theta}^T \tilde{\mathbf{x}}$ where the feature representation is a column vector of length d, and the number of parameters must necessarily be the same since the output must be a scalar, i.e. $\tilde{\mathbf{x}}, \boldsymbol{\theta} \in \mathbb{R}^d$. To highlight that the feature representation is the input to the neural network and not the full state, we use the notation $V(\tilde{\mathbf{x}})$ instead of $V(s)$.

Reassignment actions are restricted to the cell where the call departure event takes place. Each ongoing call is considered for reassignment to the newly freed channel, and the reassignment with the highest valued resulting state is selected. The value of not reassigning a channel at all is also considered.

As shown in Equation 3.41, state value methods usually require knowing an explicit model of the environment dynamics in order to greedily select an action. Because call arrivals are always accepted provided there is an eligible channel, and because reassignments does not change the number of calls in process, the reward is independent of the choice of

action when conditioned on an allocation map and an event. Furthermore, because one-step allocation map transitions are deterministic, one-step look ahead is possible. Greedy action selection can then be done by creating a resulting allocation map $\mathbf{x}'$ for each of the possible actions $a \in A(\mathbf{x}, e)$ and selecting the action that results in the highest-valued next state. This approach is termed *afterstate value-functions* (Sutton and Barto, 1998), and is applicable when some or parts of one-step $(s, a) \rightarrow s'$ transitions are known but the full $\mathcal{P}^{\mathrm{a}}_{\mathrm{ss}\prime}$ dynamics are not. If we let ϕ be the feature representation transformation function such that $\tilde{\mathbf{x}} = \phi(\mathbf{x})$, then the resulting feature representation for an action and an event is $\tilde{\mathbf{x}}' = \phi(z(\mathbf{x}, e, a))$ where as before z is the deterministic allocation map transition function.

Each possible resulting feature representation $\tilde{\mathbf{x}}'$ is run through the network one by one in order to select an action using the implicit policy:

$$\pi(s) = \underset{a \in \mathcal{A}(\mathrm{s})}{\arg\max}\, V(\phi(z(s, a))). \tag{4.2}$$

The allocation map transition function is used to look ahead at the outcome of each potential action; the policy selects the action resulting in the most desirable outcome as measured by the value-function.

The system described by the authors chooses actions in a strictly greedy manner without exploration, which implies that Equation 4.1 is on-policy.

Online gradient descent, i.e. stochastic gradient descent with a sample size of 1, is used to train the network on the squared TD error δ:

$$L = [V_{t+1}(\tilde{\mathbf{x}}_t) - V_t(\tilde{\mathbf{x}}_t)]^2 \tag{4.3}$$

$$= \Big[\frac{1 - \gamma(\Delta t)}{\beta} r(\mathbf{x}_{\mathrm{t}}, e_t, a_t) + \gamma(\Delta t) V(\tilde{\mathbf{x}}_{t+1}; \boldsymbol{\theta}_t) - V(\tilde{\mathbf{x}}_t; \boldsymbol{\theta}_t)\Big]^2 = \delta_t^2 \tag{4.4}$$

where we have set $\gamma(\Delta t) \doteq e^{-\beta \Delta t}$ to point out the similar role β has in a SMDP as γ has in a MDP. Using $\nabla_{\boldsymbol{\theta}} V(\tilde{\mathbf{x}}) \doteq \nabla_{\boldsymbol{\theta}}\, \boldsymbol{\theta}^T \tilde{\mathbf{x}} = \tilde{\mathbf{x}}$, we take the gradient of the loss and use it to update the neural network parameters:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} L. \tag{4.5}$$

The authors does not specify whether the semi-gradient:

$$\nabla_{\boldsymbol{\theta}} L = 2\delta_t \big[- \nabla_{\boldsymbol{\theta}} V(\tilde{\mathbf{x}}_t; \boldsymbol{\theta}_t)\big] = -2\delta_t\, \tilde{\mathbf{x}}_t \tag{4.6}$$

or the naive residual gradient (Baird, 1995):

$$\nabla_{\boldsymbol{\theta}} L = 2\delta_t \big[\gamma(\Delta t) \nabla_{\boldsymbol{\theta}} V(\tilde{\mathbf{x}}_{t+1}; \boldsymbol{\theta}_t) - \nabla_{\boldsymbol{\theta}} V(\tilde{\mathbf{x}}_t; \boldsymbol{\theta}_t)\big] \tag{4.7}$$

$$= 2\delta_t (\gamma(\Delta t)\, \tilde{\mathbf{x}}_{t+1} - \tilde{\mathbf{x}}_t) \tag{4.8}$$

was used for updating the neural network parameters. The naive residual gradient is a *true* gradient method, in that it does not ignore the bootstrapping value's $V(\tilde{\mathbf{x}}_{t+1}; \boldsymbol{\theta}_t)$ dependence on $\boldsymbol{\theta}_t$, as is done with semi-gradients. Stochastic gradient descent on the residual gradient cannot diverge, but *in general* converges to a less desirable point than

when using semi-gradients, and does so more slowly (Sutton and Barto, 2018). The semi-gradient may not converge to a local minimum under off-policy updates but diverge with weights going to infinity and is therefore not a true gradient descent method.

The authors note that a non-linear neural network was tried as value-function approximator but yielded only a small performance improvement.

4.2 Nie et al.

Nie and Haykin (1999) also applied RL to the channel allocation problem. They used Q-learning (Watkins and Dayan, 1992) with a feature based state representation and compared table lookup to a linear neural network function approximator.

Q-learning is an off-policy, model-free temporal difference method for estimating the state-action value-function. It does so by recursively forming an approximation to the optimal value-function $q_*(s, a)$ (Equation 3.36):

$$Q_{t+1}(\tilde{x}_t, a_t) = Q_t(\tilde{x}_t, a_t) + \alpha_t \big[r(\mathbf{x}_\mathrm{t}, e_t, a_t) + \gamma \max_{a' \in \mathcal{A}(\mathrm{s})} Q_t(\tilde{x}_{t+1}, a') - Q_t(\tilde{x}_t, a_t) \big]. \tag{4.9}$$

Q-learning is proven to converge provided that table lookup is used, every state-action pair is visited infinitely often, and the learning rate is decreased in a suitable manner. The sequence of learning rates $\alpha_0, \alpha_1, \ldots$ should satisfy the Robbins-Monro conditions (Robbins and Monro, 1951):

$$\sum_{n=0}^{\infty} \alpha_n = \infty \quad \text{and} \quad \sum_{n=0}^{\infty} \alpha_n^2 < \infty. \tag{4.10}$$

The state is a feature based representation $\tilde{x}_t = (i_t, |\mathcal{K}_\mathrm{EL}(\mathbf{x}_\mathrm{t}, i_t)|)$ where i is the cell index of the event-cell and $|\mathcal{K}_\mathrm{EL}(\mathbf{x}_\mathrm{t}, i_t)|$ is the number of *eligible* channels in that cell. We will later describe how to efficiently find this set of channels (Algorithm 1); $|\mathcal{K}_\mathrm{EL}(\mathbf{x}_\mathrm{t}, i_t)|$ is the cardinality of this set for a given cell i at some time t. In their system, the action space consists of valid assignments; no reassignments were performed on call departure.

Interestingly, they utilized a weighted, linear heuristic as reward function that rewarded compact channel usage patterns (channel packing), much in the same way as BDCL. In channel assignment, channel packing is not usually an end unto itself, but rather a heuristic to create or compare channel assignment policies.

The authors demonstrated similar performance between a linear neural network function approximator and table lookup, in both cases outperforming a non-learning algorithm named MAXAVAIL which is similar to BDCL but worse performing.

4.3 Lilith et al.

Lilith and Dogançay (2004) and Lilith (2005) used RL with table lookup. They used a state-action value-function approach termed SARSA (Rummery and Niranjan, 1994), which is the on-policy counterpart of Q-learning:

$$Q_{t+1}(\tilde{x}_t, a_t) = Q_t(\tilde{x}_t, a_t) + \alpha\big[r(\mathbf{x}_\mathrm{t}, e_t, a_t) + \gamma Q_t(\tilde{x}_{t+1}, a_{t+1}) - Q_t(\tilde{x}_t, a_t)\big] \quad (4.11)$$

The difference between Q-learning (Equation 4.9) and SARSA is that the latter uses target value of the action a_{t+1} chosen by the behavior policy, while Q-learning updates towards the value of the greedy action, which may differ from the executed action.

Rewards were defined as the number of calls currently in progress system-wide. They utilized the same reassignment strategy as Singh and Bertsekas (1997) by constricting reassignments to the cell of the departing call and to the newly freed channel. This approach allows for a clever trick when used in conjunction with a state-action value-function, namely that reassignments can be handled in much the same manner as ordinary assignments. Instead of choosing the highest valued eligible channel as is done on assignment, the lowest valued in-use channel is deterministically selected for reassignment to the newly freed channel:

$$\pi(\mathbf{x}_\mathrm{t}, \mathrm{END}_{\mathrm{i,k,t}}) = \underset{a \in A(\mathbf{x}_\mathrm{t}, \mathrm{END}_{\mathrm{i,k,t}})}{\arg\min} Q_t(\tilde{x}_t, a). \quad (4.12)$$

Therefore, any Q-value represents the value of assigning a given channel in a given cell — there is no need for separate Q-values for reassignment actions and assignment actions, or equivalently, for departure events and arrival events.

On call arrivals, actions are sampled from a stochastic *Boltzmann policy*, which has probability mass function:

$$\pi(\mathbf{x}_\mathrm{t}, \mathrm{NEW}_{\mathrm{i,t}}, a) = \frac{\exp\left(Q_t(\tilde{x}_t, a)/\tau\right)}{\sum_{a' \in A(\mathbf{x}_\mathrm{t}, \mathrm{NEW}_{\mathrm{i,t}})} \exp\left(Q_t(\tilde{x}_t, a')/\tau\right)} \quad (4.13)$$

where τ is the temperature. High temperature results in more exploration, less so as the temperature decays, and in the limit $\tau \to 0$ actions are chosen greedily. In contrast to ϵ-greedy exploration (section 3.2), the probability of picking a non-greedy action increases with its Q-value. The temperature was decayed as follows:

$$\tau_t = \frac{\tau_0}{\sqrt{t/s}} \quad (4.14)$$

where t is the elapsed time in seconds and τ_0 and s are hyperparameter constants.

The authors used a lookup table for value-function and a feature-based state representation is therefore necessary to make the size of the state space tractable. Three different feature representations were presented and benchmarked against each other. The first, as we shall denote Vanilla SARSA (V-SARSA), contains the cell index i_t of the event-cell, and the number of channels in use at that cell. Constructing this representation is therefore as easy as summing over the allocation map of the event-cell:

$$\tilde{x}_t = (i_t, \sum_{k=0}^{K} \mathbf{x}_\mathrm{t}(i_t, k)). \quad \text{(V-SARSA)}$$

As done by Nie and Haykin (1999), the feature representation only contains information about the event-cell. However, instead of counting the eligible channels in the cell, the feature representation counts the number of channels in use.

The second representation, Table-Trimmed SARSA (TT-SARSA), used the same approach as V-SARSA but aggregated the feature representation of states with 30 or more calls in progress. The reasoning behind this decision was an analysis of a high-traffic simulation showing that a cell using 30 or more channels at once was a rare occurrence.

$$\tilde{x}_t = (i_t, \max(30, \sum_{k=0}^{K} \mathbf{x}_\mathrm{t}(i_t, k))) \qquad \text{(TT-SARSA)}$$

In other words, the feature representation of the allocation map will be the same if event-cell has 30 calls in progress or more, regardless of which channels are in use.

A third representation, Reduced-State SARSA (RS-SARSA), reduced the state space even more aggressively by eliminating the state variable for the number of used channels entirely. The state then only consists of the cell index of the current event:

$$\tilde{x}_t = i_t. \qquad \text{(RS-SARSA)}$$

Despite RS-SARSA being the feature representation with the least amount of information, it was the best performing by a large margin. Lilith et al. argues that DCA systems that perform learning naturally tend to favour assigning certain channels to certain cells, and that the performance of the system is largely determined by the speed with which these associations can be formed. The smaller the size of the feature representation space, the quicker these associations can be formed.

State and state-action space size for the different feature representations are given in Table 4.1, where I is the number of cells and K is the number of channels. The state-action space size for a typical problem size of 7×7 cells with 70 channels is shown. In addition, reported new call blocking probability is shown for a simulation with hand-offs.

Version	$\lvert\tilde{\mathcal{X}}\rvert \times \lvert A\rvert$	Typical size	**New call block. prob.**	**Hand-off b.p.**
V-SARSA	$I \times K \times K$	240,100	20.6%	17.6%
TT-SARSA	$I \times 30 \times K$	106,330	20.4%	16.8%
RS-SARSA	$I \times K$	3,430	17.5%	14.2%

Table 4.1: Feature-action space size for the different feature representations and their size and reported performance on a 7×7 grid simulation with hand-offs

The state-action space size for RS-SARSA is only a small fraction of the other feature representations, and cell-channel associations are able to form and change more rapidly. We point out and emphasize that hand-offs have lower blocking probability than new calls, even though the agent is not defined with any explicit prioritization of hand-offs over new calls in the reward function or elsewhere. A contributing factor is that the channel of the departing call is guaranteed to be free at a large portion of the arrival cell's interfering neighbors, because it is eligible at the departure cell which shares a large portion of its interfering neighbors with the arrival cell. (Figure 4.1).

Noting that the state formulation of RS-SARSA only relies on local information (the event-cell index), the authors derived a distributed version with a separate agent in each

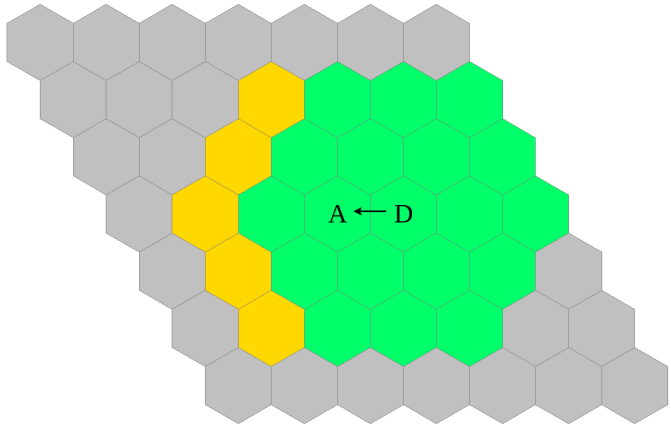


Figure 4.1: On hand-off from departure cell D to arrival cell A, the departure channel is guaranteed to be free in all interfering cells (green) of D. Since A must be an immediate neighbor of D, the departure channel is guaranteed to be free in the intersection of the interfering cells of A and the interfering cells of D. In the yellow cells, which are not interfering cells of D, no guarantee can be made of the status of the departure channel.

cell. The authors assumed that each BS has access to information about which channels are in use at its interfering neighbors. The reward, which for the centralized version is the number of calls in progress system-wide, was limited to count the number of calls in cells with a distance of 4 or less from the cell receiving the reward. In a high-traffic simulation, the distributed version incurred a call blocking probability increase of approximately 1.5%.

A non-linear neural network with 100 hidden nodes using the hyperbolic tangent function $f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$ with $f(x) \in (-1, 1)$ for the last two layers was compared to table lookup. While it performed better than the Random DCA algorithm (subsection 2.1.2), it was significantly worse than its table-based counterpart. The reason, the authors argue, is due to the irregularity of a well-performing value-function. Plotting state-action pairs versus their value from a table revealed that neither nearby states, as represented by the number of used channels, nor nearby actions had similar Q-values. If the neural network assumes a smooth relation between inputs and outputs, then it will struggle to learn a good value-function. This, according to the authors, renders function approximation with non-linear neural networks infeasible. We argue instead that the reason is the choice of activation function in the last layer. Assuming $\gamma = 0$, the true Q-value for a highly valued action will be roughly the same as the number of calls the system can serve simultaneously, which is $7 \times 7 \times 10 = 490$ for the environment in question. The upper bound for the range of the hyperbolic tangent is 1, thus the neural network cannot output larger Q-values. Any Q-value in any environment must be at least 1 since only Q-values for eligible actions are defined, and any eligible action results in an allocation map with at least 1 call in progress. Thus the network is totally unable to differentiate between desirable and undesirable actions, and as gamma increases, the problem compounds exponentially.

4.4 Kunz

In previously discussed work, we have discussed the objective of minimizing call blocking probability provided that no channel usage results in interference exceeding a threshold which is implicitly determined by the channel reuse constraint.

Kunz (1991) instead attempted to minimize the amount of interference given a limited set of channels while fully satisfying call service demand. The severity of interference for an allocation map $\mathbf{x}$ is given by the objective function:

$$F(\mathbf{x}) = \sum_{i=1}^{I}\sum_{k=1}^{K} \mathbf{x}(i,k) \sum_{j=1}^{I}\sum_{l=1}^{K} \mathbf{P}_{i,j,|k-l|}\, \mathbf{x}(j,l) \tag{4.15}$$

where $\mathbf{P}$ is a symmetric cost tensor containing the degree of interference between any two channel usages. $\mathbf{P}_{i,j,|k-l|}$ is the interference of using channel k at cell i and channel l at cell j, and $k-l$ is the distance between two channels in the channel domain, e.g. measured in MHz.

A Hopfield Neural Network (HNN) was used to minimize the objective function under the constraint of satisfying call demand. HNNs are a form of recurrent neural networks with binary threshold units and symmetric connections between every pair of neurons. The *energy function* for a network incorporates the objective function subjected to constraints, and is a function of the network's neurons, weights and biases:

$$E(\mathbf{x}) = F(\mathbf{x}) + C\sum_{i=1}^{I}(\sum_{k=1}^{K} \mathbf{x}(i,k) - D_i)^2. \tag{4.16}$$

The second term is a penalty term multiplied by a large constant $C > 0$, which then effectively serves as hard constraint on satisfying call service demand $D_i \geq 0$ for each cell i. Each neuron $\mathbf{x}(\cdot,\cdot)$ in the network corresponds to a cell-channel pair (hence the same notation as the allocation map) and is connected to every other neuron. The weight between two neurons is contained in the tensor $\mathbf{P}$ and yields the degree of interference if both cell-channel pairs are in simultaneous use. Call demand for cell i is represented by the weight D_i from a bias neuron to all neurons for cell i, i.e. $\mathbf{x}(i,k), \forall k \in \mathcal{K}$ where as before $\mathcal{K}$ is the set of channels.

A local energy minimum can be obtained by starting from any configuration and updating the neurons one at a time in arbitrary order. The update switches a neuron on or off whichever gives the lowest global energy. The contribution of a neuron to the global energy can be computed locally; it only depends on its own state and weights connected to it. HNNs are guaranteed to converge to a local optima, which may provide a poor solution. As a workaround, Smith and Palaniswami (1997) added random noise to the objective function allowing the HNN to perform hill climbing.

Though interference is a soft constraint in the energy function, the authors showed their method capable of finding interference-free configurations for the data sets they tested on.

4.5 Brunato et al.

Brunato (1999) and Battiti et al. (2001) used a weighted objective function which penal-

ized interference and rewarded channel packing. They proved that a global minimum of the heuristic function could be found in $O(K \log K)$ for K channels. The algorithm performed superior both in terms of call blocking probability and in terms of computational complexity over BDCL. The main contribution of this work is its low computational requirements while still achieving good performance.

A distributed version of the algorithm showed no performance penalty. No reassignment was done on call departure in either version. Instead, on call arrival all calls in the cell were rearranged to achieve the best configuration. As a consequence the number of rearrangements might be large, but is bounded by the number of channels.

4.6 Results from previous work on DCA

The mentioned previous work on DCA uses a 7×7 grid with 70 channels and a channel reuse constraint of 3. Though they all used mean call duration μ of 3 minutes, the only common call rate λ between them was 200 calls per hour and so those are the results reported here (Table 4.2). These results are comparable to the simulations in chapter 6 where hand-offs are not performed. The average call rate is the same for all cells which results in a uniform traffic pattern. Results for hand-offs, non-uniform or time-varying traffic patterns are not included because different works used different parameters. Their reported performance with the FCA algorithm is shown to provide some confidence that other factors such as the random number generator and grid shape are not major differentiating factors. Results from Kunz (1991) and Smith and Palaniswami (1997) are not included because the tested their systems on a small data set instead of generating traffic in a simulator.

Authors	**Approach**	**Blocking probability, in %**
Singh & Bertsekas	FCA	21.4
Singh & Bertsekas	BDCL	14.3
Singh & Bertsekas	SB-VNet	12.3
Singh & Bertsekas (recreated)	SB-VNet	13.8
Nie et al.	FCA	21.5
Nie et al.	MAXAVAIL	20.0
Nie et al.	Q-learning	19.5
Brunato et al.	FCA	21.5
Brunato et al.	BDCL	14.2
Brunato et al.	BBB (obj. func)	13.4
Lilith et al. (recreated)	V-SARSA	18.6
Lilith et al. (recreated)	TT-SARSA	18.6
Lilith et al. (recreated)	RS-SARSA	15.5

Table 4.2: Results on a 7×7 grid with mean call rate of 200 calls/hour

Lilith et al.'s SARSA algorithms described in section 4.3 were recreated because the authors did not report results on simulations without handoffs. The reimplementation used

the hyperparameter values reported by the authors: $\tau_0 = 5$ and $s = 256$ for Boltzmann exploration (Equation 4.14), learning rate $\alpha = 0.05$ without decay and discount factor $\gamma = 0.975$.

The recreation of RS-SARSA achieved similar results on the reported benchmark with handoffs: 17.52% (standard deviation of 0.21%) for new calls versus the reported result of 17.52%. For hand-offs, 13.50% versus reported 14.38%. These results were averaged over 32 runs where each lasted for 24 hours of simulator time, the latter the same as in the paper.

SB-VNet of Singh and Bertsekas (1997) (section 4.1) is recreated because it will built further upon later, though there is a significant discrepancy between our recreation of SB-VNet and the reported results. The hyperparameter values for the learning rate α and discount factor β were not disclosed. Unlike the discount factor $\gamma \in [0, 1)$ for MDPs, $\beta > 0$ is unconstrained. The discount factor has a large impact on the magnitude of the TD errors, and thus must be optimized jointly with the learning rate. Reasonable results were achieved with β ranging from 10 to 2800. The result in the table used $\beta = 21$ and $\alpha = 5.1 \times 10^{-6}$, was averaged over 32 runs with a standard deviation of 0.20% and used semi-gradients (Equation 4.6). As in their paper, the learning rate was not decayed nor did the agent perform any exploration.

4.7 El-Alfy et al.

El-Alfy et al. (2001) developed a model-based Q-learning scheme for CAC with the intent of prioritizing hand-offs over new calls.

At each time step, the system updates the following estimates:

$\tau_t(s, a)$ running average of sojourn time, for each state-action pair

$\bar{r}_t(s, a)$ running average of time-integrated rewards for each state-action pair

ρ_t system-wide, time-integrated average reward

$\mathcal{P}_t(s', s, a)$ estimate of transition probabilities $\Pr[S_{t+1} = s' \mid S_t = s, A_t = a]$, for each state-action and resulting state triple

The transition probabilities are maximum likelihood estimates formed simply by counting transitions — that is, observing how often each s' is the resulting state after executing action a in state s. The model is updated online as transitions occur in the simulator.

The Q-value function is updated with an average-reward SMDP formulation of the environment:

$$\bar{Q}_{t+1}(s, a) = (1-\alpha)\bar{Q}_t(s, a) + \alpha\left[\bar{r}_t(s, a) - \rho_t\tau_t(s, a) + \sum_{s' \in S} \mathcal{P}_t(s', s, a) \max_{a' \in A(s')} \bar{Q}_t(s', a')\right] \tag{4.17}$$

For incoming new calls, the agent chooses between accepting or rejecting the call while hand-offs are always accepted. Contrary to regular Q-Learning, the target value is an expectation over state transitions which is attained by using the model. Note that none of the terms rely on the current state or executed action; (s, a) are arbitrary while the

remaining terms are current estimates from the model. Thus, the update equation can be used to update *any* state-action pair. The authors did not specify which scheme was used for selecting which state-action pairs to update. It is common to update the observed state and executed action, as in model-free approaches, in addition to state-action pairs thought to occur in the near future, which can be determined by using the transition probabilities (Silver, 2015).

The agent was tested on a system consisting of a single cell. In single-cell systems, no co-channel interference is possible, and the system state is therefore fully described by the number of channels in use. Also particular to single-cell systems is the property that without hand-off prioritization, new call and hand-off blocking probability must be the same. As previously discussed (section 4.3) this is not the case for multi-cell systems even if hand-offs are not explicitly prioritized by the agent.

4.8 Pietrabissa

Pietrabissa (2011) considered the case where the link capacity, i.e. the number of channels available for the system, varies with time, for instance due to weather conditions. If the link capacity decreases below the current utilization, calls in progress have to be dropped, which is less desirable than not granting service in the first place. To decrease the probability of dropping calls the authors implemented CAC using RL, with separate agents for call admission and call dropping control, both using table-lookup Q-learning. Call traffic was divided into multiple classes, each requiring a certain bitrate which is analogous to a single caller requiring more than one channel at once. The admission policy controls whether or not to accept call requests while the dropping policy decides which, if any, calls to drop when a cell uses full link capacity.

Even if service requests are always denied when the link capacity is fully used, call dropping cannot be avoided due to the link capacity varying with time. The objective of the system is to simultaneously minimize new call blocking probability, dropping probability and fairness in service time between the different call classes.

4.9 Usaha et al.

Usaha and Barria (2007) used RL for CAC and call *routing* in low Earth orbit satellite networks. These satellites accept calls from the ground, route them along satellites in space before they at last are connected back down to the ground. The satellites communicate with optical or radio intersatellite links which are broken and reestablished as satellites move in orbit and come in and out of range of each other. Current systems route calls along the shortest path (in number of hops) or the path with links most likely to stay up.

As in a regular caller environment, it is desirable to perform CAC because links may go down and there might not be sufficient unused bandwidth at active links to service all calls in progress, which necessitates dropping some of them. The decision of whether to accept or block a call is only made at the satellite where the call originates, i.e. is routed from the ground; calls routed from other nodes are always accepted. Hand-offs occur as

the satellites move in orbit, when a different satellite becomes physically nearer to a caller than the previous satellite that handled the call.

The RL agent performs CAC and routing simultaneously and on service requests either rejects the call or selects a route from the set of currently available routes. Rewards are integrated over sojourn time to encourage the selection of routes likely to stay up throughout the duration of the call. As in DCA, there is no inherent preference for short term rewards. The problem was therefore defined as a SMDP with differential returns. For average-reward SMDPs, the average reward ρ_π for a stationary policy π is averaged over cumulative continuous time instead of the number of time steps:

$$\rho_\pi = \lim_{N\to\infty} \frac{\sum_{k=0}^{N-1} \mathbb{E}[R_k \mid A_{0:k} \sim \pi]}{\mathbb{E}[t_N]} \tag{4.18}$$

where R_k is the (time-integrated) reward from transition k and t_N is the time of the N$^{\text{th}}$ time step. The Bellman optimality equation for average reward SMDPs, with expected sojourn time $\mathcal{T}_{\text{s}}^{\text{a}}$ for a particular state and action is given by:

$$\bar{v}_*(s) = \max_{a\in\mathcal{A}(\text{s})} [\mathcal{R}_{\text{s}}^{\text{a}} - \rho_* \mathcal{T}_{\text{s}}^{\text{a}} + \mathbb{E}[\bar{v}_*(s') \mid s, a]] \tag{4.19}$$

where ρ_* is the average reward under an optimal policy. The reward $\mathcal{R}_{\text{s}}^{\text{a}}$ was defined as the positive *constants* r_{NEW} and r_{HOFF} with $r_{HOFF} >> r_{NEW}$ for accepted new calls and hand-offs respectively, and zero otherwise. The average-adjusted value-function was used as a baseline for an actor-critic method using REINFORCE-style gradients (see section 4.12 and Equation 4.22).

4.10 Graph theory for FCA

In graph theory, a $L(j_1, j_2, \ldots j_m)$ graph labelling task concerns the issue of labelling (coloring) every vertex in a graph such that every pair of vertices with distance i between them, as measured by the number of edges of the shortest path, have labels that differ at least j_i. The usual graph coloring problem where adjacent vertices must have different colors is a $L(1)$ labelling task. Assigning partitions of nominal channels to cells, as is shown in subsection 2.1.1, is a $L(j_1, j_2, \ldots j_{d-1})$ labelling task with $j_1 = j_2 = \cdots = j_{d-1} = 1$ and reuse distance d, if we let any two partitions have distance 1 from another. Cells correspond to vertices where cells that are geographic neighbors are connected by an edge in the graph. Channel assignment has been studied in graph theory since the 1960s and is still actively studied (Calamoneri, 2011).

Recently, Lin and Shen (2018) considered the case where each pair of channel allocations in the same cell must be separated by some distance t in the channel domain (channel distance), and where every pair of channel allocations for neighboring cells within the reuse distance must differ by some arbitrary channel distance j. This is a multi-labelling problem because each vertex (cell) must be labelled (assigned) multiple labels (channels). These types of problems are defined as n-fold, t-separated $L(j_1, j_2, \ldots j_{d-1})$ graph labelling tasks with $j \doteq j_1 = j_2 = \cdots = j_{d-1}$ when each cell requires n channels, all of which have channel distance of t or greater between each other.

The authors provided an optimal solution requiring the least amount of channels. A further generalization of the problem allowing different channel distance j_i for each geographic distance i is NP-complete (Wong, 2003). Graph labelling in the context of channel allocation usually concerns assigning nominal channels for use with FCA. As we will see (subsection 5.2.7 and 6.4), a static assignment which is optimal under some fixed call traffic conditions can be useful even in the context of DCA.

4.11 Dynamic Spectrum Assignment and Cognitive Radio domains

The term Dynamic Spectrum Access (DSA) is often used to describe the problem of channel allocation in modern Orthogonal Frequency Division Multiple Access (OFDMA) data networks, though there are no strict definitions of what constitutes a DSA domain as opposed to a DCA domain. In packet switched data networks, voice traffic is transferred by data packets using Voice-over-IP. OFDMA divides the bandwidth into sub-carriers which is analogous to channels, though a single mobile client may use multiple sub-carriers at once depending on its data rate requirement. The objective is often, analogous to DCA, to minimize packet or file (a sequence of packets) retransmission probability.

In DCA, calls are blocked if there are no channels that can be used without interference. In DSA, packets are always transmitted but the transmission may fail due to excessive interference in which case the whole file is retransmitted. Furthermore, the data rate a client can achieve depends on the magnitude of the signal-to-interference-plus-noise (SINR) ratio, thus interference management is more important in typical DSA environments than in typical DCA environments. DCA (typically) has an implicit interference threshold determined by the reuse constraint, and further improving the SINR by reducing interference yields no improvement in user satisfaction. Some minimum SINR is required to support data transmission in DSA environments, and improving it further yields higher potential data rate. Some DSA policies therefore aim to maximize average SINR or data throughput. While results in DSA are not directly comparable to those in DCA, the solution methods are largely the same with the domain differences expressed in the reward formulation and state and action spaces.

In typical DCA caller environments, it is assumed that a single service provider has exclusive access to the bandwidth. The service provider is the primary user and the legal incumbent of a radio network. Other (secondary) users are traditionally only allowed to transmit if the primary user is inactive, in which case they can be ignored when doing channel allocation for the primary user. In proposed cognitive radio systems, this constraint is lifted and secondary users are allowed to transmit if they are located outside a protected geographic boundary or their interference on any primary transmitter, as a function of transmit powers and distances, is below a given threshold.

4.12 Morozs

Morozs (2015) applied distributed reinforcement learning to the problem of DSA for data file transfer in cognitive LTE networks.

They used distributed stateless Q-learning, which for each cell is of the form:

$$Q_{t+1}(a_t) = Q_t(a_t) + \alpha[r(a_t) - Q_t(a_t)]. \tag{4.20}$$

Stateless RL algorithms are often applied to multi-armed bandit problems. Bandits have no state and thus does not transition in state space. Stateless Q-learning is possible to use for pure distributed channel allocation because the action space varies with time and implicitly encodes some state; at the extreme if no channels are eligible then the action space is empty. Omitting the state variable reduces the number of Q-values that need to be learned, thus speeding up learning. The aim of their work was to improve the poor initial performance of table-based Q-learning agents such as the one in Lilith (2005), and further reducing the size of the state-action space is one approach for achieving that goal.

The reward was defined as ± 1 for successful and failed file transmissions respectively, with corresponding objective to minimize the probability of file retransmission. A file transmission consists of several packet transmissions and may fail because of excessive interference, in which case it is retransmitted. In this distributed, cognitive channel allocation domain, excessive interference may occur because cells only intermittently communicate their list of used channels, and the information of whether a channel is safe to use may therefore be outdated.

The system employed the variable learning rate technique Win-or-Learn-Fast, where the constant α_{win} is used for successful transmissions and α_{lose} with $\alpha_{lose} > \alpha_{win}$ is used for failed transmissions. This technique is of particular interest to non-stationary environments, because large state(-action) values are decreased in a shorter period of time if the network topology or call traffic distribution changes and decreases the desirability of certain actions.

Morozs et al. (2016) extends the above approach by incorporating Case Based Reasoning (CBR) for the purpose of improving performance for cellular environments with time-varying network topology. In this scenario, the environment may change with the addition of new base stations or the removal of existing ones. In real-world networks, base stations may be turned off to save power, in which case nearby BSs increase their transmit power to cover a larger area. Other scenarios include base stations which are only temporarily deployed in order to provide service for an event.

CBR is often described as "solving new problems using the solutions to similar problems solved in the past". In short, a new problem is compared to previously solved ones using a comparison function. The most similar one is retrieved with the corresponding solution from the case base. The retrieved solution is applied to the new problem, then modified based on its performance before it is stored back in the case base as a solution to the new problem.

Here, a network topology constitutes a case and a Q-table a solution. The paper considers a distributed approach where only the local network topology informs the decision of which Q-table to use. A binary string is constructed which embeds the status of BSs in cells within a radius of 2 or less; if a bit is switched on it signifies a currently active BS and vice versa. Cases, i.e. network topologies, are retrieved from the case base using a similarity measure defined as the sum reduction of the 'exclusive or' product of the current topology and each stored topology. The solution yielding the lowest sum reduction is retrieved with its corresponding Q-table. This similarity measure is position dependent; two

cases with the same number of active BSs have low similarity if the specific BSs which are active differ between the two topologies.

For every file transfer, the case base is queried and the most similar Q-table is retrieved and used to select an action. After a single regular Q-learning update (Equation 4.20), the Q-table is stored back into the case base as a solution to the *current* topology, which may differ from the case it was retrieved as in the first place.

The CBR approach was shown to outperform stateless Q-learning (Equation 4.20), particularly for environments of low complexity dynamics.

4.13 Bernardo et al.

Bernardo et al. (2009) applied distributed RL for chunk pre-allocation in OFDMA networks. A chunk is a predetermined, contiguous, fixed set of sub-carriers. In each cell, each chunk is given its own RL agent, resulting in multiple agents per cell. An agent outputs a single binary decision, signifying whether the channels in the chunk are made available for allocation in that cell. The reward is given on a per-cell basis and increases linearly with user throughput and decreases linearly with the number of used chunks. These two terms are weighted by constants which are hyperparameters of the system. Furthermore, the reward is zero if the average user throughput is below some predefined threshold; this is done to encourage a minimum quality of service. Decreasing the reward according to the number of used chunks discourages the agent from pre-allocating more chunks than its clients can use.

Each agent consists of a single-layer neural network with a single sigmoid output which serves as parameter for a Bernoulli unit. The sigmoid (logistic function) output is in general given by $p(\tilde{\mathbf{x}}, \boldsymbol{\theta}) = \frac{1}{1+e^{-\boldsymbol{\theta}^T \tilde{\mathbf{x}}}}$ with $\boldsymbol{\theta}, \tilde{\mathbf{x}} \in \mathcal{R}^d$ for input vector $\tilde{\mathbf{x}}$ and network weights $\boldsymbol{\theta}$.

The probability $p \in (0, 1)$ is used as parameter for a Bernoulli unit $g(a, p)$, which is the probability mass function:

$$g(a,p) \begin{cases} 1-p & \text{if } a = 0 \\ p & \text{if } a = 1. \end{cases} \tag{4.21}$$

The Bernoulli unit is used as policy, $\pi_\theta(a \mid \tilde{x}) \doteq g(a, p(\tilde{x}, \theta))$, thus the action is 1 (pre-allocate) with probability p and 0 (do not pre-allocate) otherwise. Each agent in a particular cell receives the same input scalar $\tilde{x}$ defined as the percentage of users in that cell with respect to the total number of users in the system. As a consequence each agent has a single scalar network parameter (weight) θ.

The network is updated using the general framework of REINFORCE policy gradients (Williams, 1992):

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{S_0,A_0,S_1,\ldots}[\sum_{t=0}^{\infty} r_t] = \mathbb{E}_{S_0,A_0,S_1,\ldots}[\sum_{t=0}^{\infty} (r_t - \psi_t) \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(a_t \mid s_t)] \tag{4.22}$$

where ψ_t is an optional baseline, such as the average reward ρ under the policy or $V(s_t)$, which reduces the variance of the gradient (see e.g. Schulman et al. (2015) for a thorough

overview). Regardless of the baseline used, so long as it is unbiased, the expectation of the policy gradient points in the direction of increase in average reward.

Every event, the Bernoulli unit is used to sample a binary action a_t using the sigmoid output. The policy network weights are then updated with REINFORCE using the average reward ρ as baseline:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha[(r_t - \psi_t)\nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(a_t \mid x_t)] \quad \text{general REINFORCE method} \quad (4.23)$$

$$\theta_{t+1} = \theta_t + \alpha[(r_t - \rho_{t-1})\nabla_{\theta} \ln(g(a_t, p_t))] \quad \text{single layer Bernoulli net} \quad (4.24)$$

$$\theta_{t+1} = \theta_t + \alpha[(r_t - \rho_{t-1})(a_t - p_t)x_t] \quad \text{see Williams (1992).} \quad (4.25)$$

Each agent has their own weights and are updated individually, with r being the cell-specific reward (cell and agent indecies are omitted). The cell-specific average reward ρ_π is not known beforehand, and must be estimated. There are several ways of doing so, in this work they used an exponentially weighted moving average of the (cell-specific) reward:

$$\rho_t = \alpha^A r_t + (1 - \alpha^A)\rho_{t-1} \quad (4.26)$$

for some secondary learning rate $\alpha^A \in (0, 1)$.

As mentioned in section 3.2, an optimal *deterministic* policy is guaranteed to exist provided the reward function is stationary. For a centralized DCA system, this holds true even if the call distribution is non-stationary. In multi-agent environments, this is not the case if the reward depends on the behavior of the other agents, which will change over time as they learn. As an illustrative example, consider two RL agents playing the game of rock-paper-scissors against each other. If agent A plays 'rock' more than a third of the time, agent B (should) play 'paper' with the same frequency. In turn agent A should decrease the frequency with which it plays 'rock', which would cause the reward for playing 'paper' for agent B to trend downwards. The optimal policy for either agent is to randomly play each of the alternatives with equal probability, which is the Nash equilibrium of the game.

For the system described above, the reward from pre-allocating a chunk in a cell given system load will trend downwards if interfering neighbors increase their probability of pre-allocating the same chunk. If more interfering neighbors pre-allocate the same chunk, then interference will increase causing decreased throughput and thus a smaller reward. A stochastic policy is more easily expressed by a policy function, and REINFORCE policy gradients are able to learn optimal stochastic policies (van Hasselt, 2016).

4.14 Biggelaar et al.

Biggelaar et al. (2012) proposed a decentralized Q-learning system for power control of secondary users in cognitive radio networks.

Their system consists of a RL agent for each secondary transmitter which controls the transmit power used for all channels in the cell. In order to use Q-learning for the continuous action space of transmission power, the full range was quantized into a set of 15 discrete values.

The objective was to *minimize* the SINR of each secondary agent under the constraints that the SINR for the primary and secondary transmitters both exceed their separate minimum thresholds. To achieve this goal, the reward formulation for a particular cell is of the

form:

$$r_t = \begin{cases} -(SINR_t^s - SINR_{TH}^s)^2, & \text{if } SINR_t^p \geq SINR_{TH}^p \\ -\infty, & \text{otherwise.} \end{cases} \quad (4.27)$$

$SINR_{TH}^s$ and $SINR_{TH}^p$ are the minimum thresholds for the secondary and primary transmitters respectively, and $SINR_t^s$, $SINR_t^p$ are the measured SINRs for the transmitters at time t. Higher than required secondary SINR is penalized, thus decreasing the chance of exceeding interference thresholds on the primary transmitters. The SINR requirement for the secondary transmitter is a soft constraint while the interference constraint on the primary transmitter is effectively a hard constraint due to the large negative reward. SINR measurements are local to each cell though we have omitted indecies; $SINR_{t,i}^s$ measures the SINR of cell i while $SINR_{t,i}^p$ yields the SINR of the nearest primary transmitter.

In a multi-agent setting it is not necessarily desirable that each agent attempts to maximize its own reward measure, unless the reward measure incorporates the performance of other agents. For example, maximizing the SINR at each secondary agent in isolation could lead to worse performance system wide (i.e. average performance) if the interference effect on the primary transmitters is multiplicative instead of additive. A work-around where each agent maximizes the system-wide performance instead would undermine the purpose of using multiple agents since reward information would need to be communicated over potentially long distances. In decentralized, multi-agent settings, cooperative reward functions of which the above (Equation 4.27) is an example are designed to increase the performance of other agents without requiring access to their reward signals.

The Q-learning algorithm was implemented with table-lookup and state information for each cell consists of the cell's transmit power, which is local information, and knowledge of the nearest primary transmitter's interference level, which needs to be communicated. The authors demonstrated significant improvement over the naive approach of maximizing secondary SINR for each agent.

4.15 Summary

For Dynamic Channel Allocation and related tasks in the cellular network domain, the most popular RL approach seems to be Q-learning and related state-action value-function methods, most often using a small feature representation of the state, and table-lookup as value-function. The feature representation is not restricted in size due to limited computational resources. Instead, the state-action space is reduced in size to allow the RL agent to learn rapidly and exhibit good performance while learning; characteristics which also indicate the agent's ability to deal with dynamic environments.

Recent work is largely focused on distributed and/or cognitive networks. There, knowledge of the grid state may be limited to the local neighborhood, outdated, or outright missing. These domains have different and more complex sets of assumptions than what is common in DCA domains, yet solution methods remain largely the same. The same solution methods are also used to achieve tasks other than channel allocation, or with different success criteria. Different tasks define their objective in the reward function, which is often shaped to amplify different aspects of the agent's behavior.

Chapter 5

Methodology

In this chapter, we first describe the details of the caller environment simulator. Then, using the SB-VNet agent of Singh and Bertsekas (1997) as starting point, we construct an improved agent by introducing the following three changes:

- Relaxation of the problem to an MDP and optimizing for average reward
- TD(0) gradient corrections for the average-reward MDP
- A policy improvement operator for look-ahead on hand-off departure reassignments

In addition, we present an algorithm for calculating the feature representation incrementally, which is done strictly for computational efficiency. At last, we introduce a policy with preference for using nominal channels, and which is later used as an aid in analyzing agent performance.

5.1 Simulator

The caller environment simulator is implemented as a discrete event simulator (Figure 5.1) (Hillier and Lieberman, 2010), where events are processed sequentially and the simulator clock is advanced according to the timestamp of the current event.

A priority queue stores event timestamps and unique event identifiers, sorted on the timestamps in ascending order. A hash table is used for mapping identifiers to actual event objects. In addition, a second hash table maps cell-index channel pairs to event identifiers. Using two hash tables and event identifiers is one approach for handling reassignments efficiently. If a call in progress is reassigned to another channel, its corresponding departure event in the priority queue — which is already generated and waiting for its turn to be processed — must be changed to reflect the channel reassignment. The only information available from the DCA agent is the cell index and target channel; that is, the departure event triggering a reassignment and the chosen action. To avoid searching sequentially through the entire priority queue for a departure event matching the cell and channel for the call to be reassigned, hash tables allow efficient lookup.

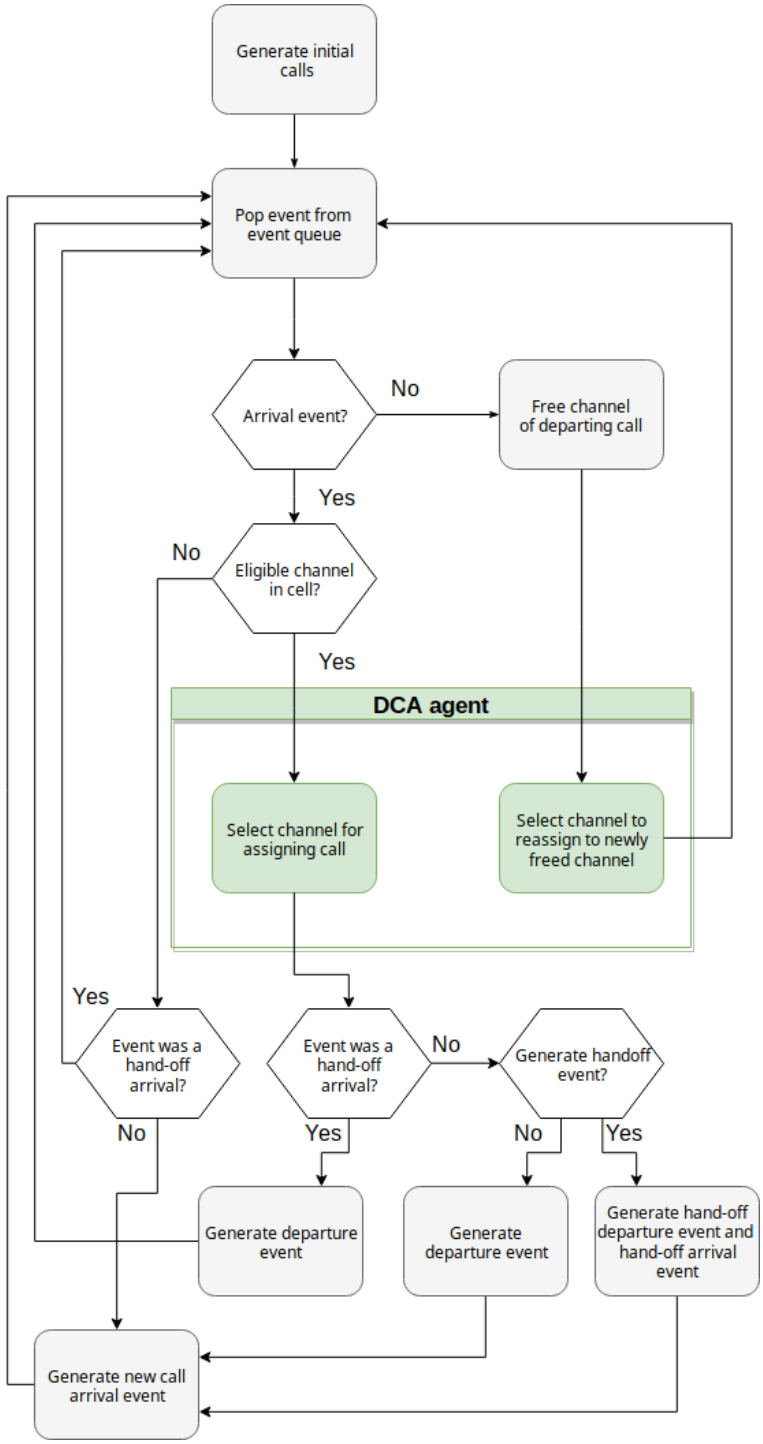


Figure 5.1: Discrete event simulation

Each simulator iteration, an event identifier is popped off the priority queue and its corresponding event is retrieved from the hash table. For arrival events in cells without any eligible channel there is nothing to be done — if the arrival event was a regular call and not a hand-off arrival, then the next new call arrival event is generated and added to the priority queue. For departure events, the channel of the departing call is freed and the DCA agent chooses a channel, if any, to reassign to the newly freed channel. For arrival events that can be accepted, the DCA agent allocates a channel from the set of eligible channels. Provided that the call arrival was not handed off from another cell, there is a chance that the just-accepted call will hand off instead of terminate when its service time has elapsed. With probability p_{handoff}, a hand-off departure and subsequent hand-off arrival is added to the priority queue. The hand-off arrival — if accepted — has expected duration μ_{handoff}. New calls, accepted or not, always generate the next arrival event according to the exponential distribution (Equation 2.2).

The environment grid of cells is modelled as a grid of hexagons, the allocation map of which must be stored as a rectangular array. There are three relevant options for the grid shape: rectangular, rhombus, and hexagonal; all of which have seen use throughout channel assignment literature. In addition, a hexagonal grid can be mapped to a rectangular array through either axial, cube, or offset coordinate systems (Redblob-Games (2015) provide an extensive resource). A rhombus-shaped grid with an axial coordinate system is the only combination which does not result in any wasted space in the array where some array elements do not correspond to cells in the grid, and where the offsets from any cell to its set of n-distance neighbors are the same regardless of the row and column of the cell. The latter property simplifies algorithms on the grid such as labelling and is crucial if a representation of the grid maintaining its spatial structure is used as input to a convolutional network, although that will not be done in this work.

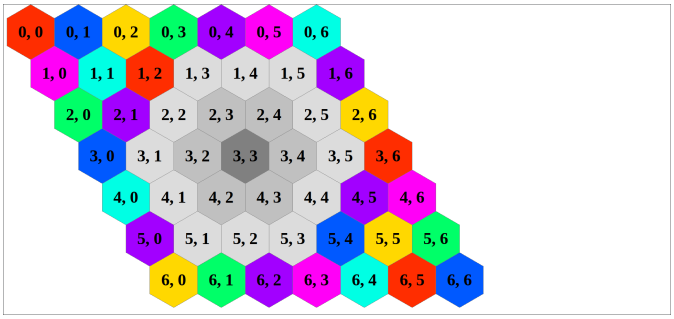


Figure 5.2: A 7×7 rhombus grid with an axial coordinate system, showing the cell $(3, 3)$ in dark gray, its neighbors with a radius of 1 (gray) and its neighbors with a radius of 2 (light gray). These are the interfering neighbors of cell $(3, 3)$ provided the reuse distance is 3.

When labelling the grid for use with FCA (subsection 2.1.1), the shape of the grid must be taken into account to keep results comparable. Both rectangtular and rhombus grid shapes have corner cells with interfering neighbors that do not span the entire range of partitions, and the performance of the FCA strategy can be improved if their set of nominal channels are expanded. This is not the case for the hexagonal grid shape and because some

literature (e.g. Jordan (1996)) uses hexagonal grids, expanding the set of nominal channels should be avoided in order to keep results comparable across grid shapes.

We conclude this section with a simple demonstration of how to efficiently find the set $\mathcal{K}_{\mathrm{EL}}$ of channels that are free and that do not violate the reuse constraint, and thus are eligible for assignment to arriving calls. This operation is invariant of grid shape and coordinate system.

Algorithm 1: Eligible channels — $\mathcal{A}(\mathbf{x}, \mathrm{NEW})$

Input : Allocation map $\mathbf{x}$, event $e = \mathrm{NEW}_{\mathrm{i,t}}$
Output: Eligible channels for cell i: $\mathcal{K}_{\mathrm{EL}}(\mathbf{x}, i)$
$h \leftarrow \mathbf{x}(i, \cdot)$ `// Row vector length` K `(number of channels)`
`// For the cell index of each interfering neighbor`
for $j \in \mathcal{I}_{IF}(i), j \neq i$ **do**
  $h \leftarrow h \mid \mathbf{x}(j, \cdot)$ `// Bitwise OR`
end
$\mathcal{K}_{\mathrm{EL}}(\mathbf{x}, i) \leftarrow where(h)$ `// Indecies of elements that are 1`
return $\mathcal{K}_{\mathrm{EL}}(\mathbf{x}, i)$

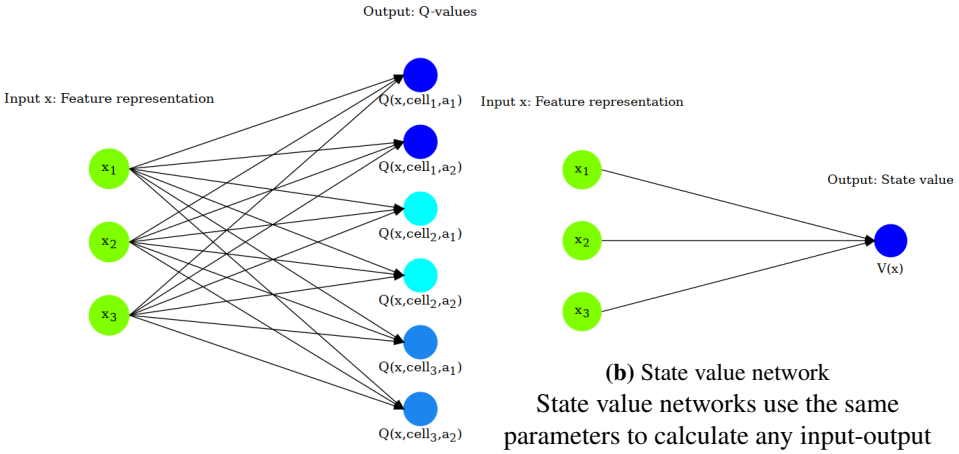
Using the bitwise OR operator, $|$, between the allocation map of a cell and its interfering neighbors $\mathcal{I}_{\mathrm{IF}}(i)$ allows for efficient calculation of the eligible channels. Determining the interfering neighbors will depend upon the coordinate system and reuse distance.

5.2 DCA agent

5.2.1 State value-function approximation

To begin, consider the structure of a 1-layer artificial neural network function approximator. For state value networks, regardless of which state forms the input to the function approximator, all the parameters in the network will affect the output which is the value of the state, unless either parts of the feature representation or a parameter is zero. This is also true for the backward pass of the same network: a value update of a state will affect the estimate of any other state because the same parameters are used for all state inputs (Figure 5.3b).

Table-lookup and 1-layer neural network function approximators for state-action value-functions or policies use a different set of parameters for each state-action pair (Figure 5.3a), and therefore cannot generalize directly across actions. If the value-function or policy uses the same set of actions for assignments as reassignments (as in section 4.3), the size of the action space alone is $I \times K$ for I cells and K channels since we cannot expect a channel to have the same Q-value or action-probability across cells. For a given feature representation, a state-action table or neural network will thus have $I \times K$ times more parameters than a 1-layer network state value-function, and learning time will increase correspondingly since the parameters are not shared.



(a) State-action value network
1-layer state-action value (or policy) networks have disjoint sets of parameters for each state-action pair.

(b) State value network
State value networks use the same parameters to calculate any input-output pair.

Figure 5.3: State-action vs. state networks

This major increase in the number of parameters necessitates a severe reduction in the size of the feature representations, which is why previously discussed feature representations for state-action methods are so small compared to the one introduced by Singh and Bertsekas (1997). Some even go as far as to eliminate the feature representation entirely (Morozs, 2015). For a typical environment with 49 cells and 70 channels, the input for SB-VNet has $7 \times 7 \times (70 + 1) = 3479$ elements while the input for RS-SARSA has two elements — the cell index of the current event and an action.

Thus for 1-layered networks, estimating the state value is clearly preferable to estimating the state-action value. Usually, the use of a state value-function requires a full model of the environment; both the expected reward $\mathcal{R}_{\mathrm{s}}^{\mathrm{a}}$ and state transition probabilities $\mathcal{P}_{\mathrm{ss}\prime}^{\mathrm{a}}$. In the case of DCA, as previously discussed (section 4.1), deterministic afterstates permit the use of a state value-function for selecting actions without requiring the full model of the environment. Using a state value-function in turn allows for a large feature representation while maintaining high convergence rate as measured by the number of value updates because learning generalizes across states and actions. Convergence rate is important for DCA agents that learn, with prior work (Morozs, 2015) specifically done to improve it, because performance during learning (temporal performance) is important in its own right if agents are to be deployed and cannot be pre-trained, and furthermore because it indicates the agent's ability to adapt to dynamic cellular environments.

Taking the discussion above into account, the use of a state value-function with a large feature representation, and the use of afterstates to select actions is therefore carried on from SB-VNet.

5.2.2 Optimizing for a better target

The objective of a DCA agent that does not perform CAC is to minimize call blocking probability. If CAC is not performed, the agent cannot preemptively terminate calls. How should the objective be formulated in terms of rewards, and is there a way to incorporate a preference for servicing hand-offs over new calls?

Compare the SMDP reward at time step k, R_k^S, side by side to the MDP reward R_k, and recall that the reward (reward rate for the SMDP) is the call count at the next time step:

$$R_k^S = \frac{1 - e^{-\beta(t_{k+1} - t_k)}}{\beta} c(\mathbf{x}_{k+1}) \qquad\qquad R_k = c(\mathbf{x}_{k+1}). \tag{5.1}$$

In the SMDP formulation, prioritizing hand-offs will result in a larger return because hand-offs have shorter mean duration and $2 \times (1 - e^{-\beta \Delta t}) > (1 - e^{-2\beta \Delta t})$, that is, serving two short calls yields higher reward than serving a single call of twice the duration.

Despite that, in the case of uniform hand-off probabilities, maximizing R_k^S might not be preferable over maximizing R_k in terms of either new call or hand-off blocking probability. If the agent cannot deny call service requests or drop calls in progress, it simply does not have any means of prioritizing hand-offs over new calls, no matter how much hand-offs are prioritized in the reward function. This hypothesis is easily verified by defining a reward function with large hand-off prioritization, e.g. by counting previously handed off calls as 10 when summing the allocation map (Equation 3.14), to see if there is any reduction in hand-off blocking probability, which there is not. Regardless of which reward definition is used, hand-off blocking probability will be lower than new call blocking probability for reasons described in Figure 4.1, but cannot be further reduced by shaping the reward definition alone. As such, there does not seem to be anything to gain in choosing the more complex SMDP reward definition over the MDP reward definition.

Selecting hyperparameters for a SMDP RL agent is also harder because trading off short-term versus long-term rewards affect the magnitude of the value update to a larger degree. Even for small changes in trade-off, the learning rate α must be selected jointly with β. This issue is exasperated by $\beta > 0$ being unbounded compared to $\gamma \in (0, 1)$, which is bounded. Furthermore, integrating the reward rate over time causes a mismatch in the scale of the reward compared to the scale of the feature representation, which counts eligible or in-use channels.

Based on these arguments, we relax the SMDP formulation in Equation 4.1 to a MDP. In addition, due to the objective of minimizing cumulative call blocking probability not being discounted, we optimize for average reward instead of discounted return. These two changes results in the following average-adjusted state value-function update:

$$\bar{V}_{t+1}(\tilde{\mathbf{x}}_t) = \bar{V}_t(\tilde{\mathbf{x}}_t) + \alpha \big[r(\mathbf{x}_\mathrm{t}, e_t, a_t) - \rho_t + V_T - \bar{V}_t(\tilde{\mathbf{x}}_t) \big] = \bar{V}_t(\tilde{\mathbf{x}}_t) + \alpha\, \bar{\delta}_\mathrm{t} \tag{5.2}$$

where the reward is the count of calls in progress (Equation 3.15), $\bar{\delta}_\mathrm{t}$ is the differential TD error and V_T is a partial value target. In general, the full value target may be the one-step return $r_t - \rho_t + \bar{V}(\tilde{\mathbf{x}}_{t+1})$, where $V_T = \bar{V}(\tilde{\mathbf{x}}_{t+1})$; the two-step return $r_t - \rho_t + r_{t+1} - \rho_t + \bar{V}(\tilde{\mathbf{x}}_{t+2})$, where $V_T = r_{t+1} - \rho_t + \bar{V}(\tilde{\mathbf{x}}_{t+2})$; or an arbitrary number of steps. In SB-VNet, no exploration is performed and the partial value target is the bootstrapping value attained from executing the greedy action, i.e. $V_T = \max_{\tilde{\mathbf{x}}'} \bar{V}_t(\tilde{\mathbf{x}}')$. If an exploration policy

is used, the value target need not correspond to the greedy choice of action, and if we are somehow able to look more than one step into the future, the value target need not be a one-step return, hence we have used a generic variable to be specified later.

The average reward ρ_π is not known beforehand and must be estimated during learning. For large t, it is possible to estimate ρ_π with $\bar{V}_{t+1}(\tilde{\mathbf{x}}_t) - \bar{V}_t(\tilde{\mathbf{x}}_t) = \bar{\delta}_\mathrm{t}$ (Gao, 2006):

$$\rho_{t+1} = \rho_t + \alpha^A \, \bar{\delta}_\mathrm{t} \tag{5.3}$$

where $a^A \in (0, 1)$ is the learning rate for the average reward. This approach was found to yield slightly superior results over using an exponentially weighted moving average (Equation 4.26).

5.2.3 Gradient corrections

In general, it is not possible approximate the true value-function v_π with a function approximator $V_{\boldsymbol{\theta}}$ with zero error for all states as measured by the Bellman error (Equation 3.40), unless the function approximator has as many parameters as there are states.

Given that the true value-function v_π is known, the best possible function approximator is the one that minimizes the Mean Squared Value Error, $\overline{VE}$ (Sutton and Barto, 2018):

$$\overline{VE}(\boldsymbol{\theta}) \doteq \sum_{s \in \mathcal{S}} \mu(s) \big[v_\pi(s) - V_{\boldsymbol{\theta}}(s) \big]^2 \tag{5.4}$$

where μ with $\sum_{s \in \mathcal{S}} \mu(s) = 1$ is an arbitrary weighting of states, signifying the relative importance of accurately valuing the different states. When doing control, it is more important to have an accurate estimate of states that are frequently visited over those that are rarely visited, thus μ is usually the on-policy distribution. The best function approximator is then $\min_{\boldsymbol{\theta}} \overline{VE}(\boldsymbol{\theta})$, which is a projection of the true value-function into the subspace of value-functions that are actually representable with $\boldsymbol{\theta}$.

Monte Carlo methods converge to $\min_{\boldsymbol{\theta}} \overline{VE}(\boldsymbol{\theta})$ for the on-policy distribution of states, but are high-variance, sample-inefficient and unsuited for infinite-horizon environments since there are no terminating states. Bootstrapping methods like TD(0) or Q-Learning converge to a different point, if they converge at all. The MDP relaxation of the TD(0) semi-gradient SGD update in Equation 4.6 has an upper bound on its TD error, known as the TD fixed point:

$$\overline{VE}(\boldsymbol{\theta}_\mathrm{TD}) \leq \frac{1}{1-\gamma} \min_{\boldsymbol{\theta}} \overline{VE}(\boldsymbol{\theta}) \tag{5.5}$$

over the on-policy distribution. The TD fixed point $\boldsymbol{\theta}_\mathrm{TD}$ is guaranteed to exist when linear function approximator is used for the value-function and updates are done on-policy. For DCA there is no inherent discount in the penalty of blocking calls, thus we want large γ which in turn implies a large upper error bound. Also, if updates are done off-policy then linear TD(0) can diverge with infinite weights.

Even if dynamic programming is used in conjunction with function approximation, the act of projecting the current estimate of the value-function corrected by the Bellman error back in to subspace of the function approximator parameters causes convergence to the TD fixed-point instead of $\min_{\boldsymbol{\theta}} \overline{VE}(\boldsymbol{\theta})$.

Exact solution methods such as Least-Squares TD converge to the TD fixed point for the value prediction problem (Equation 3.27) and provide excellent temporal performance until convergence. Unlike stochastic approximation methods, Least-squares methods used for prediction weigh old trajectory data the same as new trajectory data and require no learning rate. There exists derivations for control (e.g. Lagoudakis et al. (2002)), which unlike their counterparts for prediction are not exact at each point in time until convergence due to the continuously changing policy. Least-squares methods are of $O(d^2)$ computational complexity where d is the size of the feature representation, and require aggressive reduction of the size of the feature representation even for small caller environments. Furthermore, they rely on the environment being stationary, and we found them unsuited for DCA.

More recently, methods have been developed which are able to consistently converge to the TD fixed point under off-policy linear value-function approximation. TD(0) with gradient correction (TDC) (Sutton et al., 2009) defines a gradient by using the minimum projected Bellman error (PBE) as loss instead of the TD error as is used with semi-gradients. Minimum PBE is achieved at the TD fixed point thus SGD with TDC converges to the same point as semi-gradients.

TDC is originally only defined for the discounted case, where the loss gradient is given by:

$$\nabla_{\boldsymbol{\theta}_t} L = 2\frac{\pi(s_t, a_t)}{b(s_t, a_t)}(\delta_t\, \tilde{\mathbf{x}}_t - \gamma\, \tilde{\mathbf{x}}_{t+1}\, \tilde{\mathbf{x}}_t^T\, \mathbf{v}_t) \tag{5.6}$$

where $\mathbf{v}_t \in \mathbb{R}^d$ is a gradient correction term, which is updated each iteration with a secondary learning rate parameter $\alpha^G > 0$:

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \frac{\pi(s_t, a_t)}{b(s_t, a_t)}\alpha^G(\delta_t - \tilde{\mathbf{x}}_t^T\, \mathbf{v}_t)\, \tilde{\mathbf{x}}_t\,. \tag{5.7}$$

π is the target policy, b is the behavior policy, and $\frac{\pi(s_t,a_t)}{b(s_t,a_t)}$ is the importance sampling ratio, which is greater than 1 if the action is deemed more important by the target policy than the behavior policy and vice versa. Provided the inner vector product $\tilde{\mathbf{x}}_t^T\, \mathbf{v}_t$ is computed first, TDC has $O(d)$ complexity.

As TDC is only defined for discounted MDPs, and since our preferred objective is to optimize for average reward, it needs to be adapted. If we follow the derivation of TDC, but use the differential Bellman equation (Equation 3.37) as starting point instead of the discounted Bellman equation, the result is the same as in Equation 5.6 and 5.7 but with $\gamma = 1$ and the TD error δ_t replaced with the differential TD error $\bar{\delta}_\mathrm{t}$. As it turns out, the performance of the naive TDC derivation for average-reward MDPs is terrible. Instead, we propose the following pseudo-TDC gradient for use with an average-adjusted value-function:

$$\nabla_{\boldsymbol{\theta}_t} L = 2\frac{\pi(s_t, a_t)}{b(s_t, a_t)}(\bar{\delta}_\mathrm{t}\, \tilde{\mathbf{x}}_t + \rho_t - \tilde{\mathbf{x}}_{t+1}\, \tilde{\mathbf{x}}_t^T\, \mathbf{v}_t) \tag{5.8}$$

which is equal to the naive derivation except with the addition of ρ_t. The average reward term appears twice; once in the differential TD error $\bar{\delta}_\mathrm{t}$ and as a standalone term. The gradient correction equals the naive derivation which just has δ_t swapped for $\bar{\delta}_\mathrm{t}$:

$$\mathbf{v}_{t+1} = \mathbf{v}_t + \frac{\pi(s_t, a_t)}{b(s_t, a_t)}\alpha^G(\bar{\delta}_\mathrm{t} - \tilde{\mathbf{x}}_t^T\, \mathbf{v}_t)\, \tilde{\mathbf{x}}_t\,. \tag{5.9}$$

Using an aggressive exploration policy, and thus off-policy updates, is less important in highly stochastic domains because state space exploration is to some degree is unavoidable. This line of argumentation have been used (Silver, 2015) to explain the success of the Backgammon RL agent TD-Gammon (Tesauro, 1995) which always select actions greedily, while state space exploration is ensured by players having to roll a dice to determine their current action space. We found that for both discounted and average-adjusted rewards, an exploratory behaviour policy with greedy off-policy updates did not perform significantly different than using the behaviour policy with on-policy updates. Therefore, we use on-policy updates, where the importance sampling is always 1 since the target and behavior policy is the same, but keep TDC due to its improved convergence properties which might provide some benefit to average-reward methods.

5.2.4 Hand-off look-ahead

When defining a caller environment as a (S)MDP, it is convenient to model a hand-off as a departure event $\text{END}_{i,\cdot,\cdot}$ in cell i and an arrival event $\text{NEW}_{j,\cdot}$ in another cell j. To distinguish hand-off departures from regular departures, let

HOFF$_{\text{i,j,k,t}}$ be the departure of a call on channel k in cell i, to be handed off the next time step to another cell j,

such that a hand-off consists of the departure event $\text{HOFF}_{\text{i,j,k,t}}$ immediately followed by the arrival event $\text{NEW}_{j,t+1}$ for $i \neq j$, and call departures that are not handed off are labelled END as before. The allocation map transition function z behaves identically for HOFF as END events: it marks the channel specified by the action as free. The action space is also the same for HOFF as END events: we choose must between channels in use (Equation 3.11).

If the RL algorithm does not distinguish regular departure events from those followed by a hand-off, we forfeit useful information when reassigning channels. In this particular instance, the next event is known to be a hand-off arrival in an immediate neighbor cell. The reassignment action on hand-off departure is likely to have an effect on which channels are eligible for the hand-off arrival precisely because the cells are neighbors (Figure 5.4 and 5.5). When selecting the reassignment, we can look ahead an additional time step to see if the choice of action allows the hand-off to be accepted, furthermore, we can select the reassignment that allows for the highest joint value of reassignment action and hand-off assignment action. Reassignments that allow the hand-off to be accepted are likely more desirable than reassignments that do not leave any eligible channels for the hand-off arrival cell, at least if we prioritize servicing hand-offs over regular new calls.

Formally, if $e_t = \text{HOFF}_{\text{i,j,k,t}}$ then e_{t+1} is an event of type $\text{NEW}_{j,t+1}$ and is fully known at time step t. Then both one-step and two-step afterstates $\mathbf{x}'$ and $\mathbf{x}''$ can be deterministically determined for each possible sequence of actions a, a' by using the allocation map transition function; $\mathbf{x}' = z(\mathbf{x}_\text{t}, a, e_t)$ and $\mathbf{x}'' = z(\mathbf{x}', a', e_{t+1})$. Reassignment actions on hand-off departures (i.e. $e_t = \text{HOFF}_{\text{i,j,k,t}}$) can then be chosen greedily as follows using

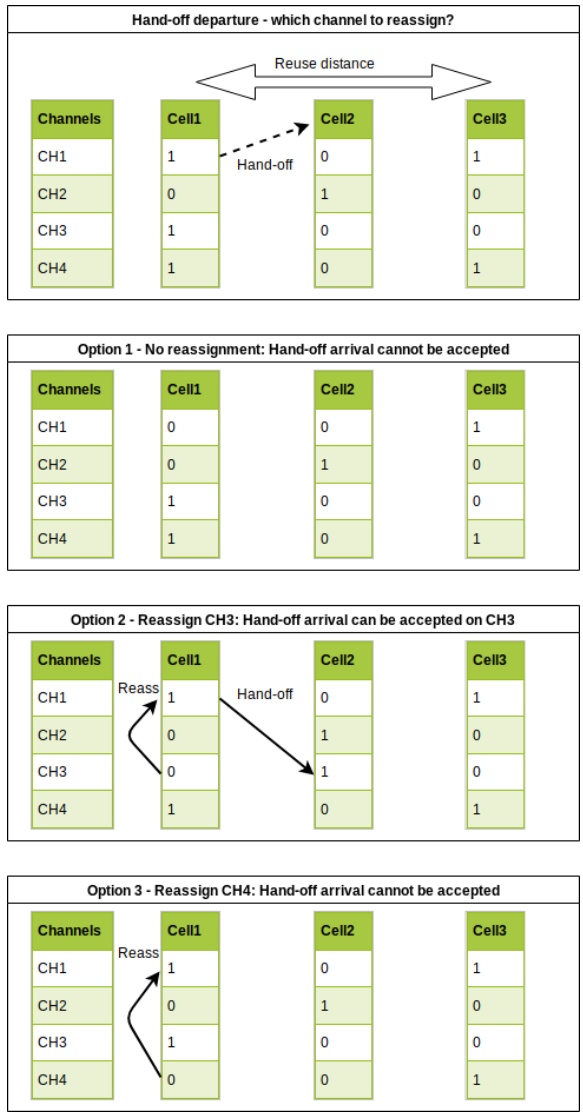


Figure 5.4: Reassignment on hand-off departure: A three-cell system with four channels. The reuse distance is 2: cell 1 has cell 2 as interfering neighbor, but not cell 3. On hand-off departure in cell 1, we can reassign any call in progress to the channel of the departing call, or not do any reassignment at all. Of the three possible actions, only one allows the hand-off arrival to be accepted in cell 2. The optimal reassignment creates an eligible channel for the hand-off arrival cell. While simple, an RL agent cannot perform the optimal reassignment for 2D grids unless the hand-off arrival cell is made known to the agent, because there is no way of knowing which cell to create an eligible channel for when doing the hand-off departure reassignment.

the hand-off look-ahead (HLA) policy improvement operator:

$$\pi^{\text{HLA}}(\mathbf{x}_\text{t}, \text{HOFF}_{\text{i,j,k,t}}) = \begin{cases} \underset{a \in \mathcal{A}(\mathbf{x}_\text{t}, e_t)}{\arg\max} \underset{a' \in \mathcal{A}(\mathbf{x}', e_{t+1})}{\max} V_t(\tilde{\mathbf{x}}'') & \text{if } \exists \mathbf{x}' : |\mathcal{A}(\mathbf{x}', e_{t+1})| > 0 \\ \underset{a \in \mathcal{A}(\mathbf{x}_\text{t}, e_t)}{\arg\max} V_t(\tilde{\mathbf{x}}') & \text{otherwise.} \end{cases} \tag{5.10}$$

The first condition checks if any reassignment allows the hand-off arrival to be accepted. If not, that is, if (using explicit notation):

$$|\mathcal{A}(z(\mathbf{x}_\text{t}, a, \text{HOFF}_{\text{i,j,k,t}}), \text{NEW}_{j,t+1})| = 0, \forall a \in \mathcal{A}(\mathbf{x}_\text{t}, \text{HOFF}_{\text{i,j,k,t}}), \tag{5.11}$$

then actions are chosen using the usual one-step afterstate as before (second case statement). HLA can be thought of as a policy improvement operator — in the above example it is applied to and improves the greedy policy. Look-ahead might in theory be applied to an exploration policy, although in practice it is best to avoid performing exploratory reassignments, in particular on hand-off departures as we know these actions to be important for reducing hand-off blocking probability.

On hand-off departure, the two-step hand-off look-ahead afterstate — i.e. the allocation map where both the hand-off departure reassignment action and subsequent hand-off arrival assignment actions are executed — can be used as bootstrapping target when updating the value function, although it is of minor importance.

Unlike the usual afterstate technique, HLA is also applicable to state-action value-functions and explicit policies. On hand-off departures, the HLA-Q policy for state-action value-functions maximizes over hand-off arrival channel assignments (if any are possible) instead of minimizing over hand-off departure reassignments:

$$\pi^{\text{HLA-Q}}(\mathbf{x}_\text{t}, \text{HOFF}_{\text{i,j,k,t}}) = \begin{cases} \underset{a \in \mathcal{A}(\mathbf{x}_\text{t}, e_t)}{\arg\max} \underset{a' \in \mathcal{A}(\mathbf{x}', e_{t+1})}{\max} Q_t(\tilde{\mathbf{x}}', a') & \text{if } \exists \mathbf{x}' : |\mathcal{A}(\mathbf{x}', e_{t+1})| > 0 \\ \underset{a \in \mathcal{A}(\mathbf{x}_\text{t}, e_t)}{\arg\min} Q_t(\tilde{\mathbf{x}}, a) & \text{otherwise.} \end{cases} \tag{5.12}$$

In the HLA policies above, reassignments that allow the hand-off to be accepted are always preferred over those that do not. The policies select the hand-off departure reassignment that allows for the best *joint* hand-off departure reassignment and hand-off arrival assignment.

An example HLA algorithm using a state value-function is written out below (Algorithm 2). As implemented, it returns the bootstrapping value of the two-step afterstate in addition to the selected action. Without significant sacrifice, one-step value targets of the reassignment afterstate can be used instead and the multi-purpose pseudo-policy function $\hat{\pi}$ turned into a regular (deterministic) policy function $\pi : \mathcal{S} \to \mathcal{A}$.

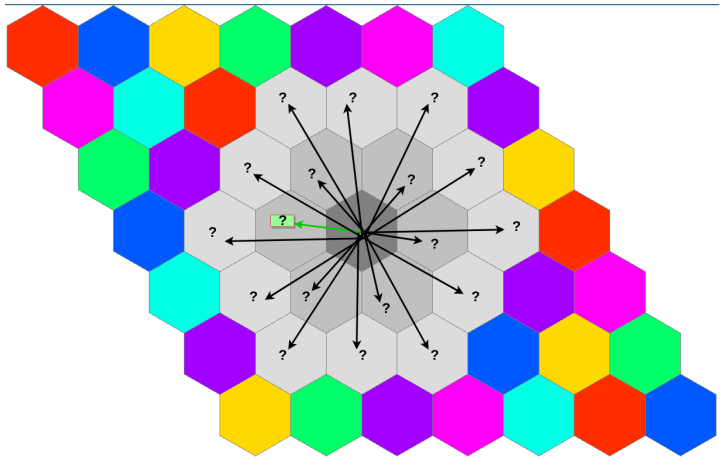


Figure 5.5: When performing a reassignment in the center (dark gray) cell, the agent may learn to choose the reassignment as to create an eligible channel for the event-cell or any of its interfering cells, as indicated by the arrows, if at all possible. If the reassignment is triggered by a hand-off departure, it is known for sure that one of the immediate neighbors (medium gray) will receive a hand-off arrival the next time step, though this information cannot be taken advantage of unless the policy treats hand-off departures differently from regular departures. Hand-off look-ahead chooses the reassignment in the departure cell as to create an eligible channel for the actual hand-off arrival cell (green arrow), if at all possible.

Algorithm 2: Hand-off look-ahead for state value-functions — $\hat{\pi}^{\text{HLA}}(\mathbf{x}, \text{HOFF})$

Input : Allocation map $\mathbf{x}_t$
Event $e_t = \text{HOFF}_{\text{i,j,k,t}}$
Output: Action a_t
Two-step afterstate value (bootstrapping value) $V(\tilde{\mathbf{x}}_{t+2})$

$a_t \leftarrow None$
$V(\tilde{\mathbf{x}}_{t+2}) \leftarrow -\infty$
// For every channel in use at departure cell
for $a \in \mathcal{A}(\mathbf{x}_t, HOFF_{\text{i,j,k,t}})$ **do**
  $\mathbf{x}' \leftarrow z(\mathbf{x}_t, \text{HOFF}_{\text{i,j,k,t}}, a)$ // Flip $\mathbf{x}_t(i,a)$ to 0
  // For every eligible channel at arrival cell
  for $a' \in \mathcal{A}(\mathbf{x}', NEW_{j,t+1})$ **do**
    $\mathbf{x}'' \leftarrow z(\mathbf{x}', \text{NEW}_{j,t+1}, a')$ // Flip $\mathbf{x}'(j,a')$ to 1
    // Greedy action selection
    if $V(\phi(\mathbf{x}'')) > V(\tilde{\mathbf{x}}_{t+2})$ **then**
      $a_t \leftarrow a$
      $V(\tilde{\mathbf{x}}_{t+2}) \leftarrow V(\phi(\mathbf{x}''))$
    end
  end
end
return a_t, $V(\tilde{\mathbf{x}}_{t+2})$

5.2.5 Incremental feature representation calculation

We borrow the feature representation from SB-VNet, and decide on the necessary assumptions left out in the paper (Singh and Bertsekas, 1997). For each cell-channel pair, the feature representation counts the number of times the channel is used by cells with distance 4 or less, not including the cell itself. For each cell, it counts the number of eligible channels. Formally, the feature representation $\tilde{\mathbf{x}} \in \mathbb{N}^{I \times (K+1)}$ is given by:

$$\tilde{\mathbf{x}}(i,m) = \phi(\mathbf{x})(i,m) = \begin{cases} |\{j \in \mathcal{I} : 1 \leq d(i,j) \leq 4, x(j,m) = 1\}| & \text{if } m \leq K \\ |\,\mathcal{K}_{\mathrm{EL}}(\mathbf{x}, i)| & \text{otherwise.} \end{cases} \tag{5.13}$$

To decrease the computational cost of constructing the feature representation, although practically feasible, the feature representation can be constructed incrementally. When constructing the incremental representation, it is assumed that assignment events have at least one eligible channel. If they do not, neither the allocation map nor its feature representation will change. Let:

$$b(e_t) = \begin{cases} 1 & \text{if } e_t = \mathrm{NEW_{i,t}} \\ -1 & \text{otherwise,} \end{cases} \tag{5.14}$$

such that b is the change in the number of calls in progress if an action is executed in response to event e.

Let $f(\mathbf{x}, e)$ be a function that frees the channel specified by a departure event:

$$f(\mathbf{x}_\mathrm{t}, e_t)(h,l) \begin{cases} 0 & \text{if } e_t \in \{\mathrm{END_{i,k,t}}, \mathrm{HOFF_{i,j,k,t}}\} \text{ and } h = i \text{ and } l = k \\ \mathbf{x}_\mathrm{t}(h,l) & \text{otherwise.} \end{cases} \tag{5.15}$$

Then the incremental feature representation is given by:

$$\begin{aligned} &\tilde{\mathbf{x}}_{t+1}(i,k) = z_\phi(\mathbf{x}_\mathrm{t}, \tilde{\mathbf{x}}_t, e_t, a_t)(i,k) = \\ &\quad \begin{cases} \tilde{\mathbf{x}}_t(i,k) + b(e_t) & \text{if } k \leq K \text{ and } 1 \leq d(i, cell(e_t)) \leq 4 \\ \tilde{\mathbf{x}}_t(i,k) - b(e_t) & \text{if } k = K+1 \text{ and } a_t \in \mathcal{K}_{\mathrm{EL}}(f(\mathbf{x}_\mathrm{t}, e_t), i) \\ \tilde{\mathbf{x}}_t(i,k) & \text{otherwise.} \end{cases} \end{aligned} \tag{5.16}$$

where $cell(e_t)$ is the event-cell. Intuitively, the next feature representation may only change within a 4-cell radius of the event-cell. If the action is to assign channel i, then the neighbors with distance of 4 or less will increase their count of how many times channel i is used within their 4-distance neighborhood. When a call departs, the number of eligible channels must increase by 1 in the departure cell. The same is not necessarily true of the interfering neighbors of the event-cell, since they in turn may have other interfering neighbors which continue to use the channel. Conversely, the number of eligible channels does not necessarily decrease at the interfering cells when a channel is assigned, because the channel to be assigned may already be ineligible in any cell but the event-cell.

5.2.6 The AA-VNet DCA agent

We construct a DCA agent by stitching together the aforementioned pieces. The aim of the agent is to maximize the differential MDP return (Equation 3.24), represented as an

average-adjusted state value-function (Equation 5.2). The value-function is approximated using a linear neural network with parameters $\boldsymbol{\theta}$ as function approximator, which is updated by SGD using TDC gradients for average-adjusted value-functions (Equation 5.8 and 5.9). The agent is referred to as the afterstate average-reward value network (AA-VNet) DCA agent and is shown in Algorithm 3. For simplicity, the periodic decay of the learning rates is not shown. Also not shown is the transitioning of the feature representation in feature space, i.e. $\tilde{\mathbf{x}}_{t+1} = z_\phi(\mathbf{x}_t, \tilde{\mathbf{x}}_t, e_t, a_t)$, instead of creating it from scratch each iteration as is indicated below.

Algorithm 3: AA-VNet DCA agent

Input: Learning rates α, α^G and α^A
$\mathbf{x} \leftarrow 0_{\mathbb{R}^{I \times K}}$ // Allocation map with I cells and K channels
$\boldsymbol{\theta} \leftarrow 0_{\mathbb{R}^d}$ // Neural net parameters
$\mathbf{v} \leftarrow 0_{\mathbb{R}^d}$ // Gradient correction weights
$\rho \leftarrow 0$ // Average reward
while *True* **do**
  $e \sim \epsilon$ // Sample call event from environment
  $a, V_T \leftarrow \hat{\pi}_{\boldsymbol{\theta}}(\mathbf{x}, e, \rho)$ // Get action and value target (Alg. 4)
  $\mathbf{x}' \leftarrow z(\mathbf{x}, e, a)$ // Execute action on grid
  $r \leftarrow c(\mathbf{x}')$ // Reward is number of calls in progress
  $\tilde{\mathbf{x}} \leftarrow \phi(\mathbf{x})$ // Create feature representations
  $\tilde{\mathbf{x}}' \leftarrow \phi(\mathbf{x}')$
  $\bar{\delta} \leftarrow r - \rho + V_T - \boldsymbol{\theta}^T \tilde{\mathbf{x}}$ // Differential TD error
  $d \leftarrow \tilde{\mathbf{x}}^T \mathbf{v}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + 2\alpha(\bar{\delta}\,\tilde{\mathbf{x}} + \rho - d\,\tilde{\mathbf{x}}')$ // SGD with TDC gradient
  $\mathbf{v} \leftarrow \mathbf{v} + \alpha^G(\bar{\delta} - d)\,\tilde{\mathbf{x}}$ // Update grad. correction weights
  $\rho \leftarrow \rho + \alpha^A\,\bar{\delta}$ // Update avg. reward
  $\mathbf{x} \leftarrow \mathbf{x}'$
end

The three learning rates are decayed exponentially, e.g.:

$$\alpha_{t+1} = \alpha_t * \alpha_\nu \tag{5.17}$$

with $\alpha_{t=0}$ as the initial value given by α in the table of hyperparameters (subsection 7.0.3), and $\alpha_\nu \in (0, 1)$ as the decay factor. The learning rate for the neural net is decayed every 10,000 events while the gradient and average reward learning rate is decayed every event.

Afterstates allow action selection using the state value-function. We found that, as done in SB-VNet, strictly greedy action selection performed on par with or better than a wide range of exploration policies and amounts. Therefore we continue to pick actions greedily, using HLA for reassignments when applicable. The action selection algorithm (Algorithm 4) returns the partial value target in addition to the greedy action, which is practical because a two-step target can be used on HLA while the standard one-step target is used otherwise. Not shown is the calculation of the state value, which for a linear value network is the inner vector product $\boldsymbol{\theta}^T \tilde{\mathbf{x}}$, as any state value-function may be used.

Algorithm 4: Action and value target selection — $\hat{\pi}(\mathbf{x}, e, \rho)$

Input : Allocation map $\mathbf{x}$
Event e
Average reward ρ
Output: Action a
Afterstate partial value target V_T

```
a ← None
if e == HOFF then
    a, V̂ ← π̂^HLA(x, e)        // Greedy HLA reassignment (Alg. 2)
    V_T ← c(x) − ρ + V̂          // Look-ahead partial value target
if a == None then
    V_T ← −∞
    // Greedily choose (re)assignment
    for â ∈ A(x, e) do
        x' ← z(x, e, â)
        if V(φ(x')) > V_T then
            a ← â
            V_T ← V(φ(x'))  // For regular one-step value target
        end
    end
return a, V_T
```

When the event is a hand-off departure, actions are selected greedily using the HLA policy (Algorithm 2), which also returns a partial value target. The two-step differential TD error on hand-off look-ahead is then:

$$\bar{\delta}_t = r(\mathbf{x}_t, e_t, a_t) - \rho_t + V_T - V_t(\tilde{\mathbf{x}}_t) \tag{5.18}$$
$$= r(\mathbf{x}_t, e_t, a_t) - \rho_t + r(\mathbf{x}_{t+1}, e_{t+1}, a_{t+1}) - \rho_t + V_t(\tilde{\mathbf{x}}_{t+2}) - V_t(\tilde{\mathbf{x}}_t) \tag{5.19}$$
$$= c(\mathbf{x}_{t+1}) + c(\mathbf{x}_t) - 2\rho_t + V_t(\tilde{\mathbf{x}}_{t+2}) - V_t(\tilde{\mathbf{x}}_t) \tag{5.20}$$

because rewards are defined as calls in progress, and a completed hand-off does not change the number of calls in progress (i.e. $c(\mathbf{x}_{t+2}) = c(\mathbf{x}_t)$ if e_t is a hand-off departure).

If the hand-off arrival cannot be accepted, then the HLA policy will not return an action and actions are selected using the regular one-step afterstate.

Source code (Python) for AA-VNet agent is available online[1].

5.2.7 Policy with nominal channel preference

At last, we introduce a policy for doing channel assignments with a preference for nominal channels. This policy will not be used by our DCA agent, but is a useful tool for analyzing agent performance.

We define the nominal channel preference (NCP) policy which for assignment events always selects the highest numbered eligible nominal channel as given by the partitioning

[1] https://github.com/tsoernes/dca

done for use with FCA, while reassignments are handled by another policy, e.g. HLA for hand-off departures and greedy action selection otherwise. The order in which the nominal channels are picked does not matter, so long as it remains fixed.

An example NCP policy for Q-functions is given in Algorithm 5, where $\mathcal{K}_{\text{NC}}(i)$ is the set of nominal channels for cell i. This policy is only defined for arrival events and selects actions greedily if no nominal channel is eligible.

Algorithm 5: Nominal channel preference state-action value-functions — $\pi^{\text{NCP-Q}}(\mathbf{x}, \text{NEW})$

Input : Allocation map $\mathbf{x}_\text{t}$
Event $e_t = \text{NEW}_{\text{i,t}}$
Output: Action a_t
$a_t \leftarrow None$
// For every nominal channel at arrival cell
for $n \in \mathcal{K}_{NC}(i)$ **do**
  // Pick highest numbered eligible nominal channel
  if $n \in \mathcal{A}(\mathbf{x}_\text{t}, NEW_{\text{i,t}})$ **then**
    $a_t \leftarrow n$
  end
end
// No nominal channel is eligible
if $a_t == None$ **then**
  // Greedy action selection
  $q \leftarrow -\infty$
  for $a \in \mathcal{A}(\mathbf{x}_\text{t}, NEW_{\text{i,t}})$ **do**
    if $Q(\mathbf{x}_\text{t}, a) > q$ **then**
      $a_t \leftarrow a$
      $q \leftarrow Q(\mathbf{x}_\text{t}, a)$
    end
  end
end
return a_t

The thought behind the NCP policy is to improve channel packing by utilizing the partitioning done in FCA. It might seem counterproductive to fully disregard the value-function on arrival events and use a fixed, non-learning policy instead. As FCA is optimal in the limit of infinite call traffic, the partitioning pattern might serve as a good baseline, which is deviated from by using the RL policy to select actions when no nominal channels are eligible and for reassignments. While the value-function is not consulted for picking actions on arrival events if there is an eligible nominal channel, it is still trained on the result of executing these actions and thus learns the values of the nominal channels.

5.3 Summary

In building the AA-VNet agent, we started off by making a case for the approach behind SB-VNet. We have seen that the use of a state value-function instead of a state-action value-function allows a 1-layer neural network function approximator to generalize. While the use of a state value-function usually requires a full environment model, afterstates allow action selection with a partial model.

We have shown the construction of the AA-VNet agent and the reasoning behind the design decisions. The use of afterstates has been extended to looking ahead on hand-off departures to find a reassignment that allows the hand-off to be accepted. In addition, the AA-VNet agent is made to maximize the differential return of an average-reward MDP, which we argue is a better target than the discounted return of a SMDP or MDP, because minimizing call blocking should not be discounted in time. The agent updates its linear neural network by using a variant of TDC gradients proposed for average-reward MDPs.

Chapter 6

Results and Analysis

In this chapter, each design decision of the AA-VNet (subsection 5.2.6) agent is inspected incrementally. To start, the SB-VNet agent (section 4.1) is modified to maximize discounted MDP returns and average-reward MDP returns. Then, continuing optimizing for average reward, the choice between TDC, residual gradients and semi-gradients is compared. The last step of the incremental analysis verifies the efficacy of hand-off look-ahead (subsection 5.2.4) for both the AA-VNet agent and RS-SARSA (section 4.3). Then, we analyze how these two agents react to the use of a policy with nominal channel preference. Finally, these two agents are compared to Random Channel Allocation and FCA over a range of traffic loads. The configuration of all value-net agents are listed in Table 6.1.

Name	Target	Gradient	HLA	Figure
SB-VNet	Discounted SMDP	Semi	No	6.1
	Discounted MDP	Semi	No	6.1
	Average-reward MDP	Semi	No	6.1 and 6.2 (no hand-offs)
	Average-reward MDP	Residual	No	6.2 (no hand-offs)
	Discounted MDP	TDC	No	6.2 (no hand-offs)
	Average-reward MDP	TDC	No	6.2[1](no hand-offs) and 6.3
AA-VNet	Average-reward MDP	TDC	Yes	6.3 and 6.5[2]and 6.6

Table 6.1: Value-net agents

All results are on a simulated caller environment with a uniform traffic pattern using the parameters in Table 6.2. Hand-offs are only performed in simulations where the agents being compared might have a different delta between hand-off and new call blocking probability. All simulations, besides the last where it is stated otherwise, use a call rate of 200 calls per

[1] also corresponds to "AA-VNet" though it does not employ HLA, because the simulation is without hand-offs thus HLA has no effect

[2] some use a non-greedy policy on arrival events

Parameter	Description	Value
I	Number of cells	$7 \times 7 = 49$
K	Number of channels	70
	Number of simulated events	470,000
μ	Call duration, minutes	3
$\mu_{handoff}$	Hand-off duration, minutes	1
$p_{handoff}$	Hand-off probability	15%

Table 6.2: Caller environment parameters

hour, i.e. $\lambda = 3.33$ calls per minute. A traffic pattern with mean call rate of 200 calls per hours is a high-traffic simulation, which is precisely the situation where FCA performs at its best, due to being optimal in the limit of infinite offered traffic. Keep in mind that for a given mean (new) call arrival rate and call duration, a simulation with hand-offs will have higher new call blocking probability than one without because the effective offered traffic has increased.

$470,000$ events corresponds to roughly 24 hours in simulator time for simulations with hand-offs and 26 hours for simulations without. Comparing the temporal performance of agents by the number events instead of simulator time is more accurate because learning happens on the experience of events, and the number of events for a given time frame is inherently stochastic.

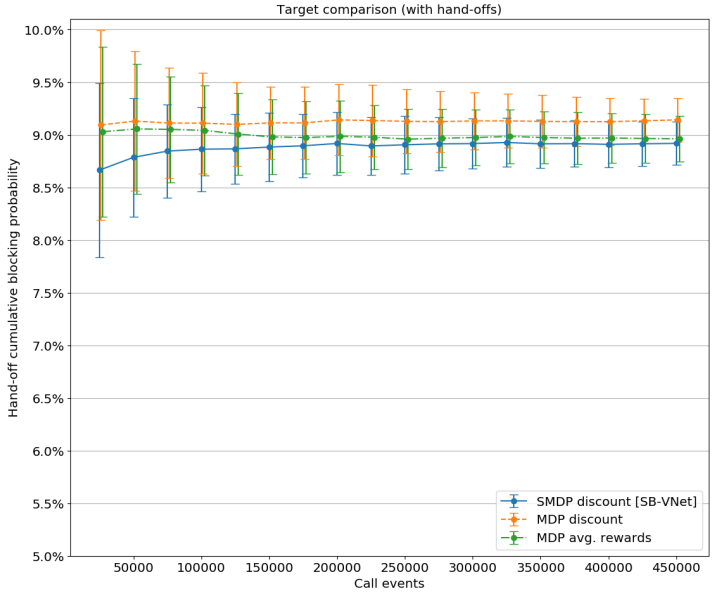
All plots show cumulative blocking probability for either new calls, hand-offs, or both (total), with error bars showing standard deviation over 80 independent runs with different random seeds. The value networks are initialized with the weight vector set to zeros, and the same is true of the Q-tables, thus the stochasticity stems from call events and the exploration policy when used. Plot points are slightly shifted on the x-axis to avoid overlapping error-bars; cumulative blocking probability measurements occur simultaneously on the earliest instance of each period, with a period lasting $25,000$ events.

Hyperparameters such as learning rates for each of the state value-based agents are listed in subsection 7.0.3, and were found using LIPO (Malherbe and Vayatis, 2017) which constructs an upper bounding model of the hyperparameter versus loss (blocking probability) space by estimating the Lipschitz constant of the RL agent, in addition to local trust region optimization for fine tuning (King, 2009). Hyperparameters for RS-SARSA are the same as used by the authors of the system, listed in section 4.6.

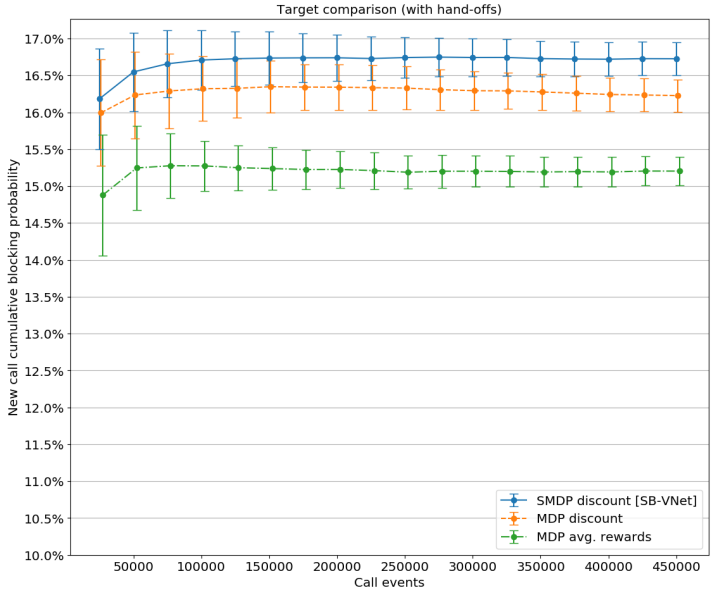
For simulations where hand-offs are performed but only a subset of the graphs are shown, the remaining graphs are to be found in subsection 7.0.4.

6.1 Choice of returns

We test the hypothesis that the MDP formulation is no worse than the SMDP formulation, and that optimizing the average reward might be preferable over optimizing the discounted return due to the true objective of minimizing call blocking probability not being discounted.



(a) Hand-off blocking probability



(b) New call blocking probability

Figure 6.1: Return comparison (with hand-offs)

The agents are all value networks based on SB-VNet and differ only in their definition of returns. The reward (rate) is defined as the number calls in progress, in the case of the SMDP integrated over time, and in the case of differential returns (A-MDP) subtracted by the average reward. All agents use semi-gradients without any exploration. The hyperparameters are selected individually for each agent. For the SMDP, the learning rate α is selected jointly with the discount factor β and for the MDP, α is jointly tuned with γ. For the A-MDP, the average reward learning rate α^A is selected jointly with α because the average reward is a function of the neural network output (Equation 5.3), although in practice these turn out to be nearly independent.

There is an insignificant difference in the hand-off blocking probability between the SMDP (8.92%), MDP (9.14%), and average-reward MDP (8.96%), while as for new call blocking probability it seems to confirm the hypothesis that using differential returns is superior. Using differential returns reduces blocking probability by about 1.03% over the discounted MDP which in turn reduces it by about 0.50% over the SMDP. The difference between the MDP and SMDP can possibly be explained by the different scaling of the rewards, which is known to have a significant effect in RL in general (Henderson et al., 2017), or it could be caused by a worse choice of hyperparameters for the SMDP.

6.2 Gradients

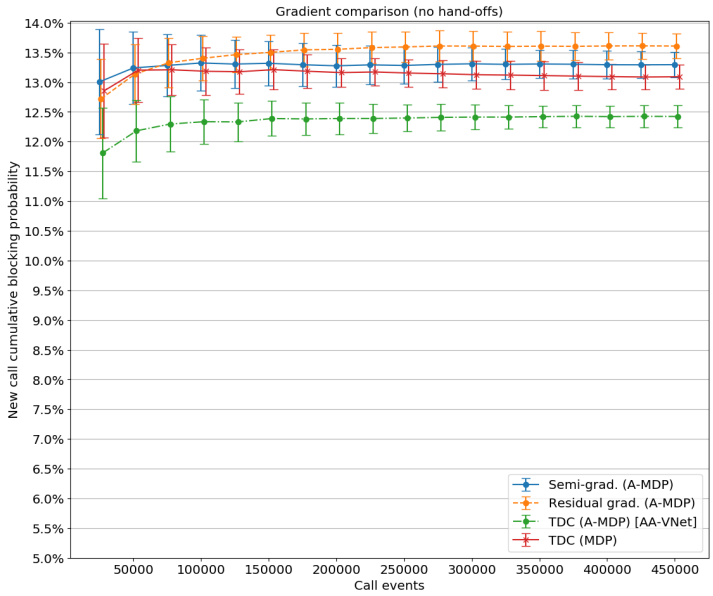


Figure 6.2: Gradient comparison (without hand-offs)

In this section, we compare gradients for the average-reward MDP agent from the previous section. The most commonly used semi-gradient is compared to residual gradients and TDC. To discover if gradient corrections are applicable and provide any benefit to the

average-reward case, TDC is applied to both the average-reward MDP formulation (TDC A-MDP, Equation 5.8) and the discounted MDP formulation (TDC MDP, Equation 5.6). The residual and semi-gradients are as defined earlier (section 4.1), but with differential TD errors corresponding to an average-reward MDP formulation:

$$\nabla_{\boldsymbol{\theta}_t} L = 2\bar{\delta}_t(\tilde{\mathbf{x}}_{t+1} - \tilde{\mathbf{x}}_t) \qquad \text{(Residual gradient)}$$
$$\nabla_{\boldsymbol{\theta}_t} L = 2\bar{\delta}_t(-\tilde{\mathbf{x}}_t) \qquad \text{(Semi-gradient)}$$

The learning rates have been tuned individually for each gradient method; for the TDC gradient methods jointly with the gradient correction learning rate α^G.

While in general, residual gradients converge to a less desirable point and are to be avoided, for the special case of deterministic environments, residual gradients converge to a more desirable point (Sutton et al., 2009). Transition dynamics are not deterministic in DCA, but independent Bellman errors (Equation 3.40) of the value-function for the MDP are deterministic due to deterministic rewards and allocation map transitions. Thus, ignoring exploration, the Bellman error of a single transition equals the TD error of the same transition, while a series of either Bellman errors or TD errors is not deterministic due to stochastic events determining the action space of the next time steps. It is therefore of interest to compare the residual gradient to the much more widely used semi-gradient.

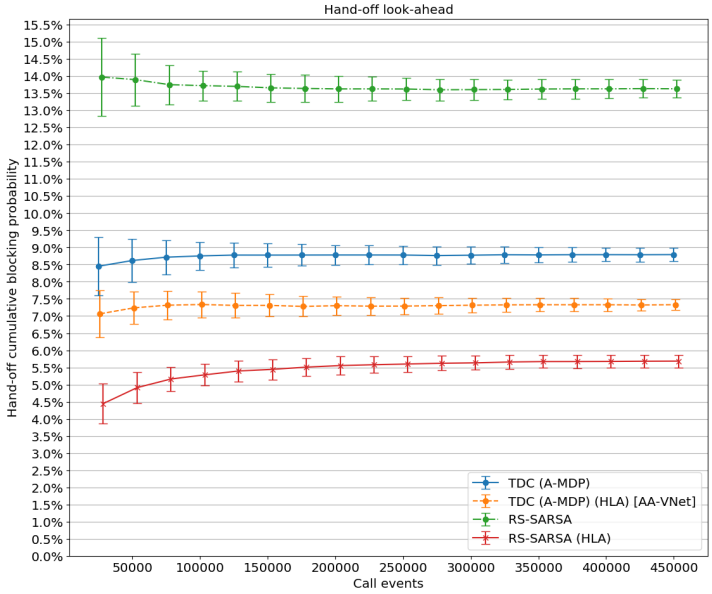
Figure 6.2 reveals that residual gradients perform about 0.31% worse than semi-gradients. TDC gradients for the A-MDP outperform semi-gradients by approximately 0.87%, and with an average cumulative new call blocking probability of 12.42% nearly matches the reported results of $\approx 12.3\%$ of SB-VNet. We reiterate that all updates are done on-policy. Convergence is only known to be guaranteed for the MDP, as average-reward RL methods using linear neural networks have no known convergence guarantee (Tadepalli, 2017), though none of the agents ever diverged in practice. The most important property of TDC gradients are their guaranteed convergence under off-policy updates for MDPs with discounted returns, but these findings show that not only is TDC applicable to differential returns, but also provide improved steady-state performance over semi-gradients.

6.3 Hand-off look-ahead

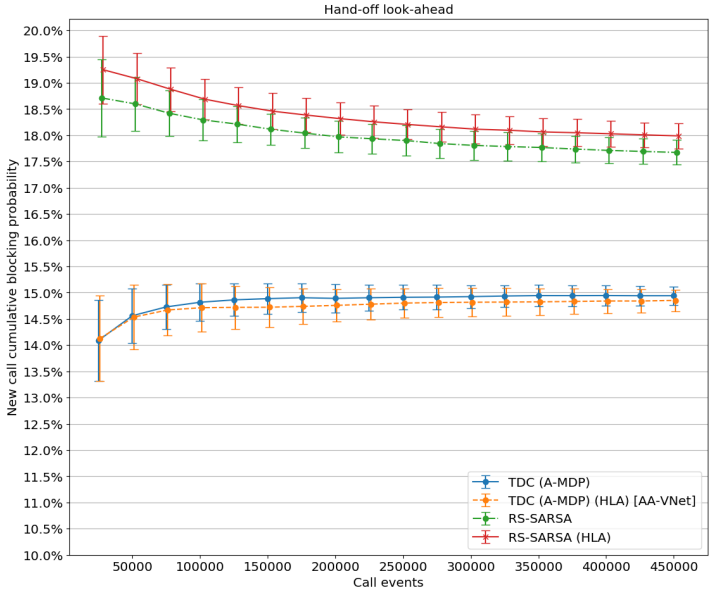
Figure 6.3 compares the use of hand-off look-ahead (HLA) with regular, non-look-ahead reassignments. Enabling HLA for the state value-network agent (Equation 5.10) improves hand-off blocking probability by about 1.46% in addition to an insignificant improvement for new call blocking probability. With HLA enabled, this agent corresponds fully to AA-VNet (subsection 5.2.6).

For the state-action table-lookup agent RS-SARSA (section 4.3), HLA (Equation 5.12) is used for hand-off departures while arrival events are handled by a Boltzmann exploration policy (Equation 4.13) and regular departures by greedy action selection, the latter two the same as in the original system. In RS-SARSA, HLA yields a decrease of 7.95% in the blocking probability for hand-offs and 0.31% increase for new calls; a total decrease of 0.59% (Figure 6.4).

HLA is likely more impactful in terms of hand-off blocking probability for RS-SARSA than AA-VNet because the total blocking probability of RS-SARSA is significantly higher.



(a) Hand-off blocking probability



(b) New call blocking probability

Figure 6.3: Hand-off look-ahead

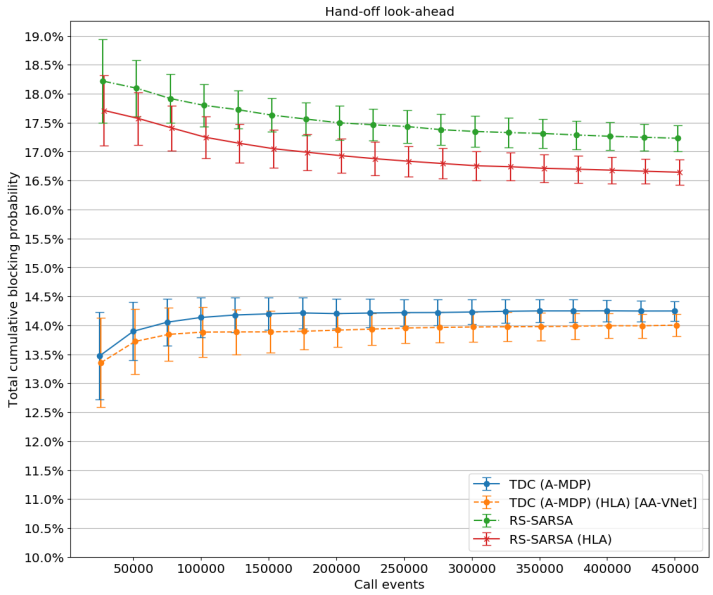


Figure 6.4: HLA total blocking probability

New call blocking probability is approximately 3% worse for RS-SARSA, making HLA able to take advantage of the relatively poor channel packing when reassigning channels before hand-offs, causing the significantly improved hand-off blocking probability.

As demonstrated in Figure 6.4, there is a significant difference in the temporal behavior between the agents. The Q-table in RS-SARSA has nearly the same total amount of parameters as the function approximator in AA-VNet — $(I \times K)$ vs. $(I \times (K+1))$ — yet the state value approach shows near-perfect temporal performance, while the state-action value approach undergoes significant learning during the first 12 hours and takes roughly 24 hours of simulation time before converging. As discussed in subsection 5.2.1, improving the estimate of one state-action pair in a Q-table does not directly generalize to any other. The same would be true of a state value table and a 1-layer state-action neural net. However the state value-function never shows worse than steady-state performance while the initially empty caller environment fills up with calls.

These results are not directly comparable with results from CAC systems because the latter usually model multiple classes of calls, with the objective of prioritizing hand-offs as well as new calls of high-priority call classes. As a result, new calls of low-priority call classes see a major increase in call blocking probability and so does the total blocking probability. In Lilith and Dogançay (2005), a CAC agent was tested in an identical environment as used in this work (Table 6.2) with traffic split evenly into a low- and high-revenue class. Hand-offs for either class had approximately 3% chance of being blocked while new calls were blocked with 45% and 4% chance respectively. Compared to the RS-SARSA agent using HLA, which blocks hand-offs with probability 5.7% and new calls with probability 18.0%, the small decrease in hand-off blocking probability achieved by

using CAC might not be worth the large increase in total blocking probability.

6.4 Exploration

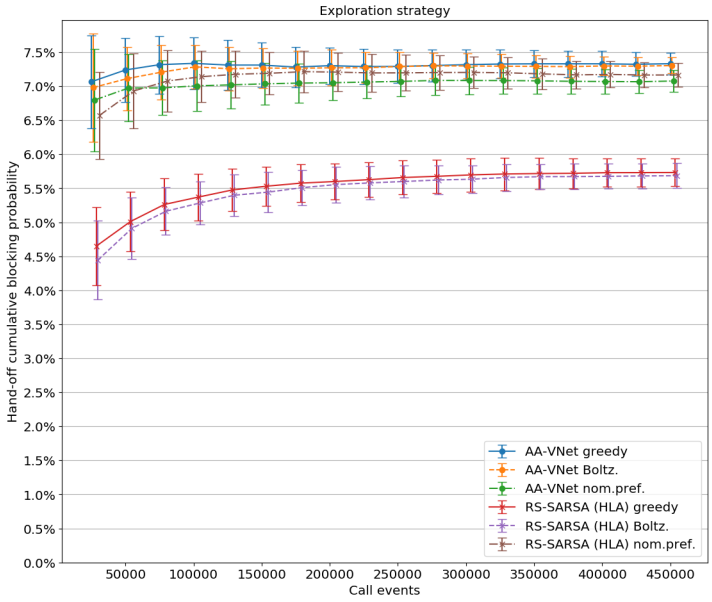
In this section, different 'exploration' strategies are compared for both AA-VNet and RS-SARSA. In previous simulations, AA-VNet has used greedy action selection on arrival events, where RS-SARSA has used a Boltzmann exploration policy. Here, we apply Boltzmann exploration with the same parameters in the same manner to AA-VNet. Conversely, greedy action selection on call arrivals is also tested for RS-SARSA. Looking at the results for hand-offs (Figure 6.5a) or new calls (Figure 6.5b), we see that neither change results in any significant difference. As earlier (subsection 5.2.3) discussed, exploration is less important in highly stochastic domains, because the unavoidable stochasticity inherent in state transitions ensure state space exploration.

More interestingly is how the two agents respond when using the policy with nominal channel preference (NCP) (subsection 5.2.7), where, for either agent, call arrivals are assigned to an eligible nominal channel, hand-off departures use HLA and actions are selected greedily otherwise. NCP reduces new call blocking probability by approximately 1.53% for RS-SARSA while having only a very slight negative effect for AA-VNet. The use of NCP also eliminates the advantage RS-SARSA with HLA has shown over AA-VNet in terms hand-off blocking probability. For RS-SARSA, the use of NCP improves channel packing as is evident by the significant reduction of total blocking probability (Figure 7.1b). These findings give substantial credence to the hypothesis discussed in the previous section that RS-SARSA with a Boltzmann policy or greedy action selection exhibits poor channel packing and that HLA causes greater improvement in hand-off blocking probability because of it.

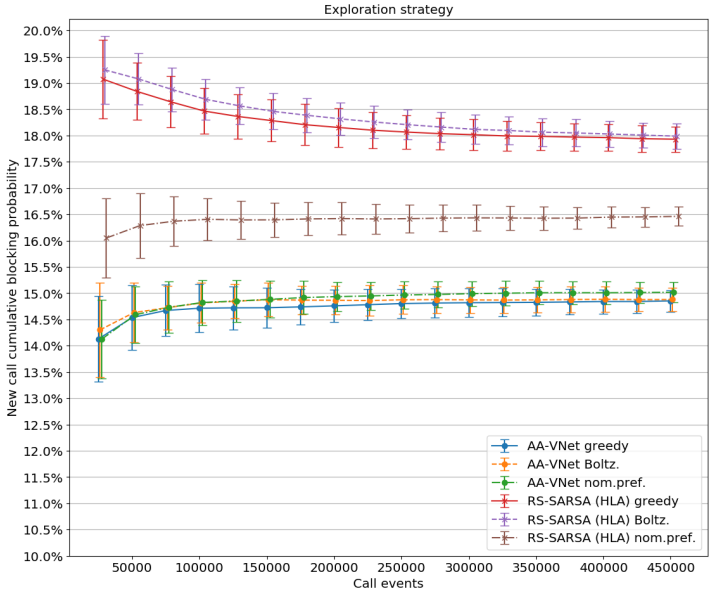
At last, NCP significantly improves the temporal performance of RS-SARSA. Knowing optimal channel partitions for a given traffic pattern allows the use of a NCP policy, and can therefore be useful for some DCA agents if HLA cannot be performed or if servicing hand-offs is not preferred over servicing new calls.

6.5 Comparison to non-learning agents

In Figure 6.6, we compare FCA and random assignment (subsection 2.1.2) with our previous RL agents. The simulations include hand-offs and have (new call) offered traffic ranging from 5 to 10 Erlangs, corresponding to 100 to 200 calls per hour (per cell) with call duration of 3 minutes. Perhaps surprisingly, random assignment outperforms FCA over all traffic conditions. Interestingly, the difference is much greater for hand-offs than new calls. In high-traffic conditions, hand-off blocking probability is approximately 5% lower. In FCA, the departure of a call has no effect on the amount of eligible channels in neighboring cells. Any free nominal channel is eligible — which is the point of assigning nominal channels in the first place — thus on hand-off departure, the probability of blocking the subsequent hand-off arrival is the same as for blocking a new call anywhere on the grid.

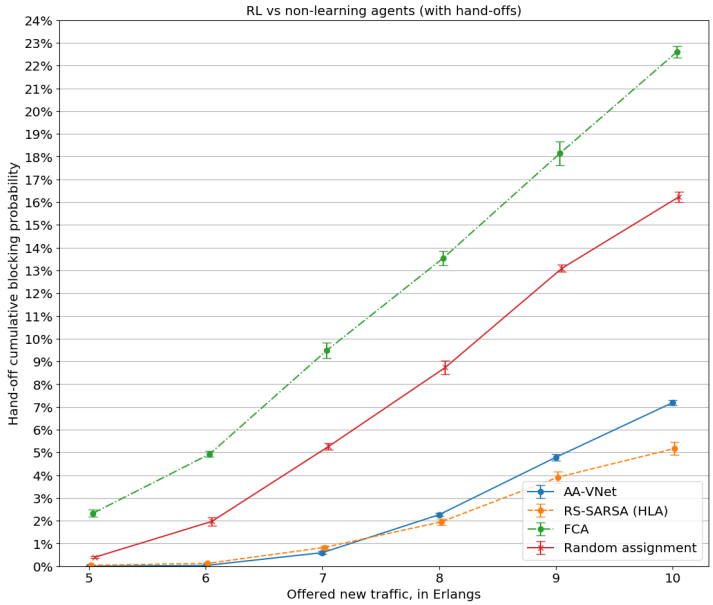


(a) Hand-off blocking probability

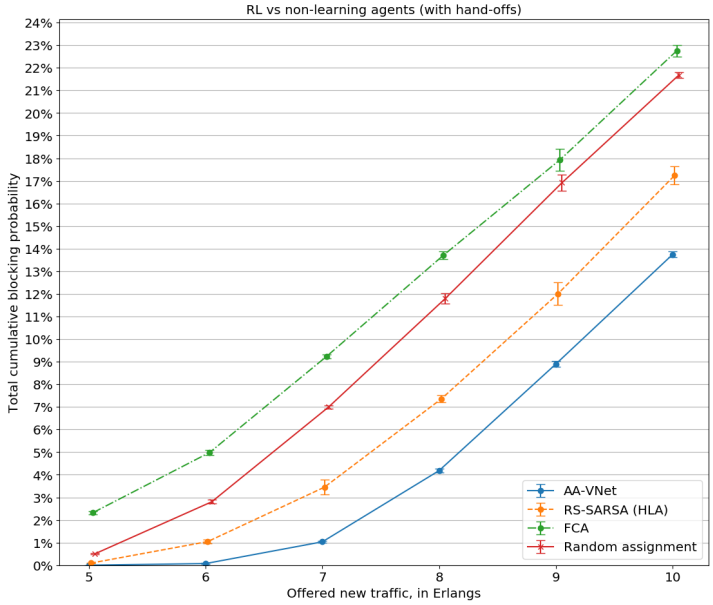


(b) New call blocking probability

Figure 6.5: Exploration strategies (with hand-offs)



(a) Hand-off blocking probability



(b) Total blocking probability

Figure 6.6: RL vs. non-learning agents (with hand-offs)

That is not the case for random assignment. As described earlier in Figure 4.1, the channel to be freed on hand-off departure is guaranteed to be free for a large portion of the interfering cells of the arrival cell. This guarantee cannot be made anywhere on the grid, nor is it applicable to FCA because the departure channel cannot be a nominal channel in the arrival cell, hence random assignment has lower hand-off blocking probability than FCA.

As expected, both RL agents significantly outperform the non-learning agents. At low traffic conditions, the AA-VNet agent is able to maintain near-zero blocking probability for either call type up to 6 Erlangs, whilst FCA has a 5% chance of blocking calls at that traffic load. At 10 Erlangs, the total (Figure 6.6b) blocking probability is 9.13% lower for AA-VNet and 6.49% lower for RS-SARSA compared to FCA. For hand-offs, both RL agents use HLA and perform well compared to the non-learning agents. As seen previously, RS-SARSA with HLA is clearly the best performing with hand-off blocking probability of 5.68% at 10 Erlangs.

6.6 Summary

Throughout this chapter, we have compared each design decision of the AA-VNet agent with the most obvious alternatives. In our results, we found that optimizing for the differential return of an average-reward MDP formulation of the environment yielded lower call blocking probabilities than optimizing the discounted return of either a MDP or SMDP formulation. The TDC gradient proposed for average-reward MDPs (Equation 5.8) was found to perform substantially better than both TDC gradients for discounted MDPs and semi-gradients for average-reward MDPs.

We found that the use of the hand-off look-ahead policy improvement operator significantly decreased hand-off blocking probability, particularly for RS-SARSA where look-ahead was able to exploit the rather poor channel packing. RS-SARSA with HLA showed, to the best of our knowledge, state-of-the-art hand-off blocking probability for an agent not performing call admission. In terms of temporal performance and total call blocking performance, the AA-VNet agent is superior to RS-SARSA at the price of a roughly 2% increase in hand-off blocking probability at 10 Erlangs.

Chapter 7

Conclusion

In this work, we have focused on the task of channel allocation in centralized cellular telephone networks carrying voice traffic. Fixed Channel Allocation, which is the most widely used channel allocation policy in deployed systems, performs badly for hand-offs and in light traffic conditions. Reinforcement learning have been successfully used to create agents for Dynamic Channel Allocation and related tasks in the caller environment domain. Common to nearly all proposed RL systems is the use of state-action methods, in particular schemes like Q-learning paired with table-lookup, with a tiny feature representation of the full state.

In DCA, state transition dynamics are partly deterministic. The notion of afterstates allow us to look at the consequences of potential actions before they are executed. This in turn makes it possible to use a state based RL method as opposed to a state-action based method. With a linear neural network as state value function approximator, learning by stochastic gradient descent on a loss function generalizes across states and actions, which is not the case for either table-based methods or linear state-action networks. This effect is so drastic that it enables the use of a much larger feature representation of the state while maintaining superior temporal performance. The use of afterstates and a state value-function is applicable to a wide range of tasks in the caller environment domain.

By exploiting the fact that hand-off departures are always succeeded by perfectly predictable hand-off arrivals, afterstates can be used to look ahead on hand-off departure to create eligible channels for the subsequent hand-off arrival. The hand-off look-ahead policy improvement operator reassigns channels to create a desirable configuration for the next known event. The result is a significant decrease in hand-off blocking probability without, as is the case when doing call admission control, incurring increased total blocking probability. For RS-SARSA, HLA had a significant positive impact on total blocking probability as well which we ascribe to poor channel packing which HLA is able to take advantage of.

If call admission control is not performed, then the reward function cannot be shaped to freely trade off prioritization of hand-offs over new calls. We did find however that the choice of return definition had a large impact on total blocking probability; optimizing

for average reward yielded better results than optimizing the discounted return of either a MDP or SMDP since cumulative call blocking probability is not a discounted objective.

In addition, our proposed TDC gradient for average-reward MDPs performed very well although it is backed only by an empirical performance demonstration and almost surely does not carry the off-policy convergence guarantee of regular TDC gradients for discounted MDPs.

7.0.1 Hand-off look-ahead in distributed systems

In distributed systems, knowledge of the state of neighboring base stations may be limited. In order to perform HLA, the hand-off departure cell must have access to the allocation maps of the interfering neighbors of the hand-off arrival cell, in order to ascertain whether a reassignment will create an eligible channel. The total amount of information that needs to be communicated to the departure cell is $18 \times K$ bits for K channels if the reuse distance is 3, because a cell has 18 neighbors with distance of 2 or less. In section 6.3, we concluded that the more inefficient the agent is, the greater improvement can be obtained by using HLA. Thus, if the transfer of the necessary information proves realistic in distributed systems, HLA should prove fruitful since distributed agents in general perform worse than their centralized counterparts due having less information on which to act.

If computational power in the base station is very limited, the value of any successor state $V(\tilde{\mathbf{x}}_{t+1})$ may be calculated incrementally given the current allocation map $\mathbf{x}_\mathrm{t}$ and its value $V(\tilde{\mathbf{x}}_t)$. The difference between any two sequential allocation maps $\mathbf{x}_\mathrm{t}$ and $\mathbf{x}_{\mathrm{t}+1}$ is at most 1 bit, and since the value of linear state value is an inner vector product, constructing an incremental state value is as easy as adding or subtracting the value of the parameter corresponding to the changed bit.

7.0.2 Model-based new call look-ahead

If a departure event is not succeeded by a hand-off arrival, look-ahead on channel reassignment is still possible if a model of the environment is known. Upon reassignment, the look-ahead policy should prioritize creating eligible channels for neighbor cells according to both their offered traffic and current number of eligible channels. If a high-traffic neighbor within reuse distance does not have a single eligible channel, choosing a reassignment that creates one is arguably a highly valued action.

Look-ahead on reassignments followed by regular new calls is only beneficial for environments with non-uniform call traffic. In uniform traffic, the probability of a cell being the recipient of the next new call service request is the same for all cells, thus looking ahead using a probabilistic model cannot result in the prioritization of a cell.

The benefit will be greater in non-stationary call traffic environments since a model should be able to adapt to changes in the call traffic distribution faster than any prioritization in the value-function because it can explicitly estimate the (relative) offered traffic of cells which can be used in the look-ahead policy, while any RL algorithm must shift potentially large state or state-action values to accommodate the shift of which cells to prioritize.

For a call environment with exponentially distributed call duration and inter-arrival times (section 2.5), constructing an environment model is particularly easy. If the call traf-

fic is stationary and uniform, we need only to estimate two parameters, namely the average call duration and the average inter-arrival time. If it is non-uniform, two parameters per cell will be necessary and for the non-stationary case, more recent samples needs to weigh more heavily than old samples when estimating the parameters. A stationary call distribution can be modelled simply by forming a running average of call rate and duration. For a non-stationary environment more advanced models are necessary, such as exponential smoothing or perhaps a time-adaptive drift diffusion model (Rivest et al., 2014).

Bibliography

Baird, L., 1995. Residual algorithms: Reinforcement learning with function approximation. Machine Learning Proceedings, 30–37.

Battiti, R., Bertossi, A. A., Brunato, M., Dec. 2001. Cellular channel assignment: a new localized and distributed strategy. Mobile Networks and Applications 6, 493–500.

Bernardo, F., Agust, R., Prez-Romero, J., Sallent, O., 07 2009. A self-organized spectrum assignment strategy in next generation ofdma networks providing secondary spectrum access. IEEE Communications Conference, 1 – 5.

Bertsekas, D. P., 2005. Dynamic Programming and Optimal Control, Vol 1, 3rd edition. Athena Scientific.

Biggelaar, O. V. D., Dricot, J.-M., Doncker, P. D., Horlin, F., 2012. Power allocation in cognitive radio networks using distributed machine learning. IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications.

Brunato, M., 1999. Channel assignment algorithms in cellular networks. Ph.D. thesis.

Calamoneri, T., 2011. The l(h, k)-labelling problem: An updated survey and annotated bibliography. The Computer Journal 54 (8), 1344–1371.

Chen, Y., Jia, C., 12 2009. An improved call admission control scheme based on reinforcement learning for multimedia wireless networks. International Conference on Wireless Networks and Information Systems 0, 322–325.

Dutta, J., Chakraborty, S., Barma, P. S., Kar, S., Jan 2016. An efficient approach to dynamic channel assignment problem using genetic algorithm. In: 2016 International Conference on Microelectronics, Computing and Communications (MicroCom). pp. 1–6.

El-Alfy, E. S., Yao, Y.-D., Heffes, H., 2001. A model-based q-learning scheme for wireless channel allocation with prioritized handoff. Global Telecommunications Conference 6, 3668–3672.

Gao, Y., 2006. Research on average reward reinforcement learning algorithms (mla06). http://lamda.nju.edu.cn/conf/MLA06/files/Gao.Y.pdf.

Goodfellow, I., Bengio, Y., Courville, A., 2017. Deep Learning. MIT Press.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D., 2017. Deep reinforcement learning that matters. CoRR abs/1709.06560.
URL http://arxiv.org/abs/1709.06560

Hillier, F. S., Lieberman, G. J., 2010. Introduction to Operations Research, 9ed. International. McGraw Hill.

Hong, D., Rappaport, S. S., 1986. Traffic model and performance analysis for cellular mobile radio telephone systems with prioritized and nonprioritized handoff procedures. IEEE Trans. Veh. Tech 35, 77–92.

Jordan, S., 1996. Resource allocation in wireless networks. Journal of High Speed Networks 5 (1), 23–34.

Katzela, I., Naghshineh, M., 1996. Channel assignment schemes for cellular mobile telecommunication systems: A comprehensive survey. IEEE personal communications 3 (3), 10–31.

King, D. E., 2009. Dlib-ml: A machine learning toolkit. Journal of Machine Learning Research 10, 1755–1758.

Kunz, D., 03 1991. Channel assignment for cellular radio using neural networks. IEEE Transactions on Vehicular Technology VT-40, 188 – 193.

Lagoudakis, M. G., Parr, R., Littman, M. L., 2002. Least-squares methods in reinforcement learning for control. Methods and Applications of Artificial Intelligence, 249–260.

Lilith, N., 2005. Reinforcement learning-based resource allocation in cellular telecommunications systems. Ph.D. thesis.

Lilith, N., Dogançay, K., 2004. Reduced-state sarsa with channel reassignment for dynamic channel allocation in cellular mobile networks. Proceedings of 11th International Conference on Telecommunications, 1327–1336.

Lilith, N., Dogançay, K., 2005. Distributed dynamic call admission control and channel allocation using sarsa. Asia-Pacific Conference on Communications, Perth, Western Australia, 376–380.

Lin, W., Shen, C., 02 2018. Channel assignment problem and n-fold t-separated $l(j_1, j_2, \ldots, j_m)$-labeling of graphs. Journal of Combinatorial Optimization.

Malherbe, C., Vayatis, N., 2017. Global optimization of Lipschitz functions. ArXiv e-prints.

McEliece, R. J., Sivarajan, K. N., 1994. Performance limits for channelized cellular telephone systems. IEEE Transactions on Information Theory 40 (1), 21–34.

Ming, Z., Yum, T.-S., 1989. Comparisons of channel-assignment strategies in cellular mobile telephone systems. IEEE Transactions on Vehicular Technology 38 (4), 211–215.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing Atari with Deep Reinforcement Learning. Arxiv.

Morozs, N., 09 2015. Accelerating reinforcement learning for dynamic spectrum access in cognitive wireless networks. Ph.D. thesis.

Morozs, N., Clarke, T., Grace, D., 2016. Cognitive spectrum management in dynamic cellular environments: a case-based q-learning approach. Engineering Applications of Artificial Intelligence 5, 239–246.

Nie, J., Haykin, S., 1999. A q-learning-based dynamic channel assignment technique for mobile communication systems. IEEE Transactions on Vehicular Technology 48 (5), 1676–1687.

Ofcom, Februrary 2013. Winners of the 4g mobile auction. https://www.ofcom.org.uk/about-ofcom/latest/media/media-releases/2013/winners-of-the-4g-mobile-auction.

Paxson, V., Floyd, S., 1995. Wide area traffic: the failure of poisson modeling. IEEE/ACM Transactions on Networking 3 (3), 226–244.

Pietrabissa, A., 12 2011. A reinforcement learning approach to call admission and call dropping control in links with variable capacity. European Journal of Control 17, 89–103.

Puterman, M. L., 2005. Markov Decision Processes: Discrete Stochastic Dynami Programming. Wiley-Interscience.

Redblob-Games, 2015. Hexagonal grids. https://www.redblobgames.com/grids/hexagons/, [Online; accessed 10-Dec-2017].

Rivest, F., Kohar, R., Amadou, N., 12 2014. Learning to predict events on-line: A semi-markov model for reinforcement learning.

Robbins, H., Monro, S., 1951. A stochastic approximation method. The Annals of Mathematical Statistics 22 (3), 400.

Rummery, G. A., Niranjan, M., September 1994. On-line Q-learning using connectionist systems. Tech. Rep. 166, Cambridge University Engineering Department.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., Abbeel, P., 2015. High-dimensional continuous control using generalized advantage estimation. CoRR abs/1506.02438. URL http://arxiv.org/abs/1506.02438

Silver, D., 2015. Lectures on reinforcement learning (compm050/compgi13). http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529 (7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., Hassabis, D., Dec. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. ArXiv e-prints.

Singh, S., Bertsekas, D., 1997. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference, 974–980.

Sivakumaran, M., Iacopino, P., 2018. The mobile economy 2018. https://www.gsmaintelligence.com/research/2018/02/the-mobile-economy-2018/660/.

Smith, K., Palaniswami, M., 1997. Static and dynamic channel assignment using neural networks. IEEE Journal On Selected Areas In Communications 15 (2), 238–249.

Sutton, R. S., Aug 1988. Learning to predict by the methods of temporal differences. Machine Learning 3 (1), 9–44.

Sutton, R. S., Barto, A. G., 1998. Reinforcement Learning: An Introduction (1st Edition). MIT Press.

Sutton, R. S., Barto, A. G., 2018. Reinforcement Learning: An Introduction (2nd Edition, Feb. 2018 draft). MIT Press.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., Wiewiora, E., 2009. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In: Proceedings of the 26th Annual International Conference on Machine Learning. ICML '09. ACM, pp. 993–1000.

Tadepalli, P., 2017. Average-reward reinforcement learning. Encyclopedia of Machine Learning and Data Mining, 87–92.

Tesauro, G., 1995. Temporal difference learning and td-gammon. Commun. ACM 38 (3), 58–68.

Usaha, W., Barria, J., 07 2007. Reinforcement learning for resource allocation in leo satellite networks. IEEE transactions on systems, man, and cybernetics (Part B) 37, 515–27.

van Hasselt, H., 2016. Lectures on advanced topics in machine learning). https://hadovanhasselt.com/2016/01/12/ucl-course/.

Watkins, C., Dayan, P., 1992. Q-learning. Machine Learning 8, 272–292.

Williams, R. J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 229–256.

Wong, S. H., 2003. Channel allocation for broadband fixed wireless access networks. Ph.D. thesis.

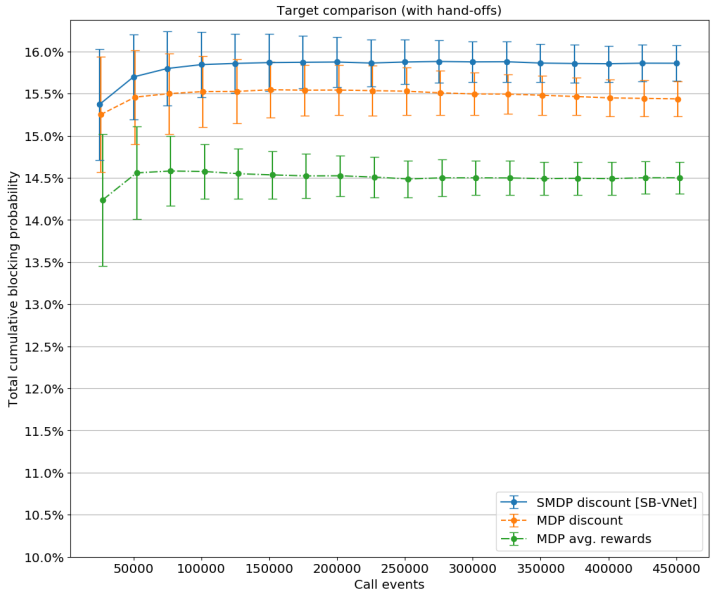
Appendix

7.0.3 Hyperparameters

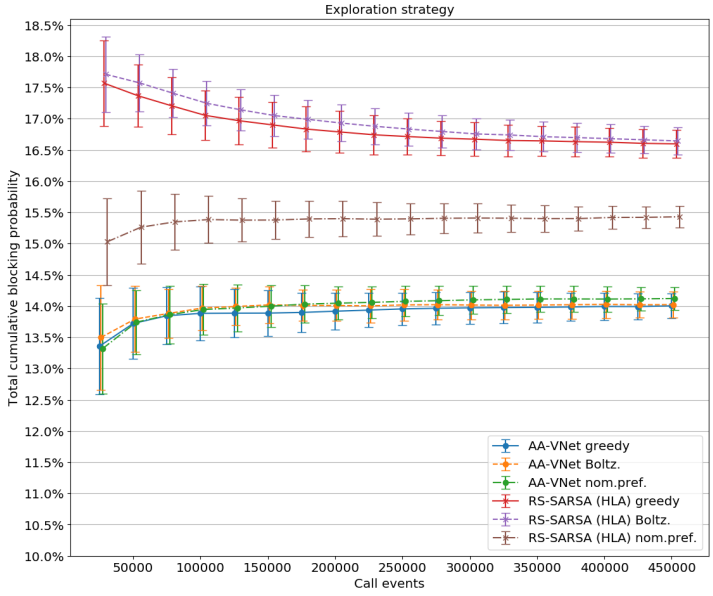
Hyperparameters common to all VNet agents are listed at the top. See subsection 5.2.6 and Equation 5.17 for details of how the learning rate decay parameters are applied, and Equation 4.13 for the Boltzmann temperature.

Parameter	Description	Value
VNet (general)		
α_ν	Learning rate exp. decay (per 10k events), ANN	0.78
SMDP Discount		
α	Learning rate, ANN	5.1×10^{-6}
β	Discount factor	21
MDP Discount		
α	Learning rate, ANN	2.02×10^{-7}
γ	Discount factor	0.845
MDP Average		
α	Learning rate, ANN	3.43×10^{-6}
α^A	Learning rate, avg. reward	3.68×10^{-3}
α^A_ν	Learning rate exp. decay, avg. reward	$1 - 4.75 \times 10^{-5}$
Semi-gradients (A-MDP): See MDP Average		
Residual gradients (A-MDP)		
α	Learning rate, ANN	1.6×10^{-5}
α^A	Learning rate, avg. reward	3.68×10^{-3}
α^A_ν	Learning rate exp. decay, avg. reward	$1 - 4.75 \times 10^{-5}$
TDC gradients (A-MDP)		
α	Learning rate, ANN	2.52×10^{-6}
α^A	Learning rate, avg. reward	6×10^{-2}
α^A_ν	Learning rate exp. decay, avg. reward	$1 - 4.75 \times 10^{-5}$
α^G	Learning rate, grad. corr.	5×10^{-6}
α^G_ν	Learning rate exp. decay, avg. reward	$1 - 9 \times 10^{-4}$
TDC gradients (MDP)		
α	Learning rate, ANN	1.91×10^{-6}
γ	Discount factor	0.845
α^G	Learning rate, grad. corr.	5×10^{-9}
α^G_ν	Learning rate exp. decay, avg. reward	$1 - 9 \times 10^{-4}$
TDC gradients (A-MDP): Boltzmann exploration		
τ_0	Boltzmann temperature	5
s	Boltzmann temperature log decay param.	256

7.0.4 Remaining graphs

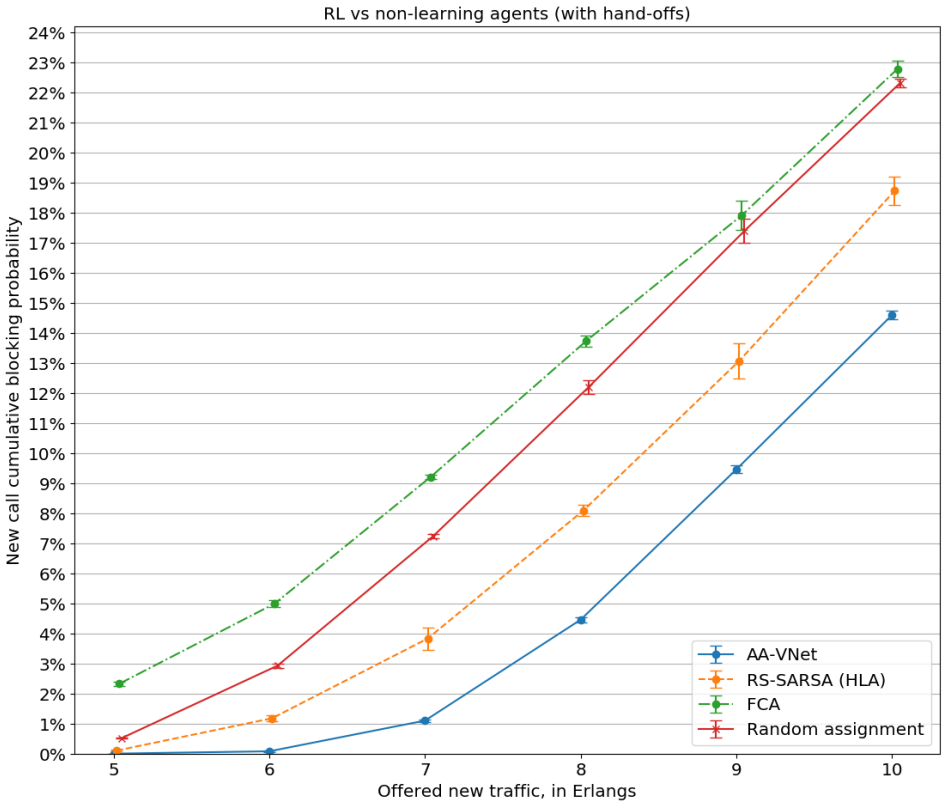


(a) Return comparison (with hand-offs): Total call blocking probability



(b) Exploration comparison (with hand-offs): Total blocking probability

Figure 7.1: Return comparison and Exploration strategies



(a) RL vs. non-learning agents (with hand-offs): New call blocking probability

Figure 7.2: RL vs. non-learning agents