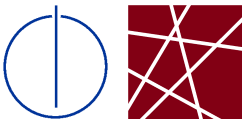
Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste



Machine Learning for Bandwidth Management in Decentralized Networks

Masterarbeit in Informatik

durchgeführt am
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

von
cand. inform.

Fabian Oehlmann

Februar 2014

Technische Universität München
Fakultät für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Maschinelles Lernen für Bandbreitenverwaltung in dezentralisierten Netzwerken

—

Machine Learning for Bandwidth Management in Decentralized Networks

Masterarbeit in Informatik

durchgeführt am
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik
Technische Universität München

von
cand. inform.

Fabian Oehlmann

Aufgabensteller:	Christian Grothoff, PhD (UCLA)
Betreuer:	Dipl.-Inform. Matthias Wachs
Tag der Abgabe:	15. Februar 2014

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this thesis only supported by declared resources.

Garching, den 15. Februar 2014 Fabian Oehlmann

Abstract:

The successful operation of a peer-to-peer network depends on the resilience of its peer's communications. On the Internet, direct connections between peers are often limited by restrictions like NATs and traffic filtering. Addressing such problems is particularly pressing for peer-to-peer networks that do not wish to rely on any trusted infrastructure, which might otherwise help the participants establish communication channels. Modern peer-to-peer networks employ various techniques to address the problem of restricted connectivity on the Internet. One interesting development is that various overlay networks now support multiple communication protocols to improve resilience and counteract service degradation.

The support of multiple protocols causes a number of new challenges. A peer should evaluate which protocols fulfill the communication requirements best. Furthermore, limited resources, such as bandwidth, should be distributed among peers and protocols to match application requirements. Existing approaches to this problem of transport selection and resource allocation are rigid: they calculate the solution only from the current state of the environment, and do not adapt their strategy based on failures or successes of previous allocations.

This thesis explores the feasibility of using machine learning to improve the quality of the transport selection and resource allocation over current approaches. The goal is to improve the solution process by learning selection and allocation strategies from the experience gathered in the course of many iterations of the algorithm. We compare the different approaches in the field of machine learning with respect to their properties and suitability for the problem. Based on this evaluation and an in-depth analysis of the requirements of the underlying problem, the thesis presents a design how reinforcement learning can be used and adapted to the given problem domain.

The design is evaluated with the help of simulation and a realistic implementation in the GNUnet Peer-to-Peer framework. Our experimental results highlight some of the implications of the multitude of implementation choices, key challenges, and possible directions for the use of reinforcement learning in this domain.

Contents

List of Figures

List of Acronyms

ACO Ant Colony Optimization. 22

ANN Artificial Neural Network. 20, 21

API Application Programming Interface. 6

ATS Automatic Transport Selection. 1, 2, 9, 11, 13–15, 17, 19–23, 25–27, 29, 33, 39–41, 43, 44, 47, 53, 54, 65, 66

CDN Content Delivery Network. 4

DP Dynamic Programming. 33, 34

GA Genetic Algorithm. 21–24, 27

HMM Hidden Markov Model. 21

MC Monte Carlo method. 33, 34, 36

MDP Markov Decision Process. 31–33, 40, 41, 47, 48, 53

ML Machine Learning. 1, 2, 17, 19, 20, 22–28, 42

MLP Mixed Integer Linear Programming. 13, 15–17, 23, 25, 27, 28

NAT Network-Address-Translation. 1

P2P Peer-to-Peer. 1–7, 9, 10, 12, 13, 15, 17, 19

RBF Radial Basis Function. 46, 47, 50, 56, 60, 70

RL Reinforcement Learning. 2, 21–23, 25–29, 31–34, 38, 39, 42, 44, 45

SG Stochastic Game. 40, 41

SMDP *Semi*-Markov Decision Process. 48, 50, 53, 54

SVM Support Vector Machine. 20

TD Temporal Difference. 34, 36, 39, 56

1. Introduction

The GNUnet Peer-to-Peer (P2P) Framework is a development platform for an overlay network which is built on top of ordinary networking infrastructure. It has a special focus on the security of its end-users. As a central concept to protect these goals, it relies on the full decentralization of its infrastructure. Ideally, the communication structure an overlay is built upon enables a connectivity between all of the overlay's participants. Unfortunately, this is not always the case in the Internet, due to technical restrictions and limitations, such as Network-Address-Translations (NATs) or firewalls, which prevent certain connections. A further hindrance are governmental efforts to decrease the reachability of certain parts of the Internet for censorship purposes. In order to overcome these obstacles and increase the connectivity, GNUnet incorporates an abstraction layer of transport mechanisms [FGR03]. Due to the offering of a variety of transport protocols for addressing another peer, exists the possibility to switch from one encountering impediments to another protocol, which does not face impediments.

In case several transport mechanisms remain functional a sending node is facing the choice which mechanism to use. The answer to this question might be clear, if there is a single one surpassing the rest in all its properties. The question becomes more intricate, if the choice has to be made e.g. between a transport, which offers a high bandwidth but a high latency and another one, which offers a low latency but only a low bandwidth. Taking into account the requirements for a specific connection enables weighting up different properties against each other. A further increase of complexity is faced, if some transports share a resource, such as a common network interface, where only a limited amount of bandwidth is available. The questions which connections to make, which transport mechanism to use, and which share of the available resources to allocate, are intertwined and have an impact on each other. Finding good answers to these questions is the problem of Automatic Transport Selection (ATS).

An important aspect of the ATS problem, is its dynamic environment, which is why static solutions become outdated fast and need to be recalculated when peers join and leave. Machine Learning (ML) is a subfield of artificial intelligence and studies the employment of methodologies which improve a machine's performance at distinct tasks by gathering experience. Experience can be gathered from past events and data collected during the execution of a task. Hence, it is the topic of this thesis to research the prospects of enhancing the decision making process for ATS by the means of the ML domain.

The thesis begins with an evaluation of the varying approaches ML provides, what the characteristics of the respective approaches are, and in which problem domains the ap-

proaches are most suitable. Based on this evaluation the next target is to apply a particular learning method from the available ML methods to the ATS task. The thesis proposes a design and shows its usability and feasibility. The design is implemented in a simulation prototype, which can be adapted and modified easily to display and evaluate the impacts of various adaptations to the underlying learning technique. Furthermore, a prototypical implementation within the GNUnet P2P Framework was developed to ensure the applicability of the design within the context of a P2P application. Based on these implementations the design decisions, and their advantages and drawbacks are assessed. I conclude by summarizing the suitability of machine learning to improve solving the ATS problem. Limitations and challenges of the approach are disclosed and an outlook on future work to cope with the ATS problem is given.

The structure of the thesis is the following. Chapter 2 introduces to the concept of decentralized and P2P networks and the motivation behind their development. As one representative the GNUnet P2P framework is explained, along with its special focus on security and privacy. Afterwards Chapter 3 specifies the ATS problem in general, deriving from the requirements as they appear in GNUnet and discusses two alternative solution methods. Having the other solutions in mind Chapter 4 researches the methodologies offered by the ML domain and argues about their merit for a further investigation. Consecutively the formal foundation of the chosen approach of Reinforcement Learning (RL) is explained and the substantial modifications for an adaption to the ATS problem are specified in Chapter 5. A short overview of the implementations is then given by Chapter 6. Thereafter, the characteristics of the approach are evaluated empirically in Chapter 7. Eventually the thesis finishes with a conclusion of the gained insights in Chapter 8.

2. Background

In order to put this work into context, this chapter argues for the importance of decentralization in communication systems. An overview of the paradigm of Peer-to-Peer (P2P) is given and GNUnet as an exemplary implementation of such a system and basis in this work is introduced consecutively. By analyzing GNUnet and the communication pattern, the problem to solve can be figured out.

2.1 Decentralized and Peer-to-Peer Networks

Traditionally, the Internet in its early days grew from just very few connected research institutes, whose main goal was to enable resource sharing between those facilities. Of course, the Internet developed further since these days. E-Mail and the World Wide Web are the first applications that come to mind. They incorporate the common client-server architecture, where a usually always available computer provides a service (e.g. delivering the content of a website) to authorized computers posing a request (e.g. a web browser retrieving a website). It is only the server, who shares its resources with its clients. The majority of today's end-users of the Internet only act as a client in most of their communication. However, the ever improving performance and resources within the network, in terms of bandwidth and low-latency, as well as the resources at its endpoints, namely storage and processing-power, make it possible to realize the idea of sharing those at a much smaller scale. Resources can be shared not only among organizations, but between single machines owned by ordinary people everywhere and even in mobile scenarios.

In addition, the decentralization of computation and communication infrastructure has desirable advantages. For instance, databases replicated to different sites can enable shorter access times. Additionally the redundancy can keep their functionality up and running even in the case of failures. Such forms of decentralization come at the cost of coordination.

The term Peer-to-Peer (P2P) evokes thoughts about diverse illegally used file-sharing systems in many people's minds. However, this only reflects one type of application, which has been successfully realized using a whole concept or paradigm, which is summarized under the term P2P. Another application, which got public media attention, following the P2P principle is the digital cryptocurrency Bitcoin[1], which maintains a decentralized database of payment transactions. Less well known to the general public but successful

[1] http://bitcoin.org

applications of the concept include Content Delivery Networks (CDNs) like Coral[2] and streaming applications such as Tribler[3].

What those applications have in common is that they realize a certain functionality, which is not provided by some central instance, but by all of the participants involved. The fact that for example file-sharing and digital currencies are not very similar in their typical usage scenario gives a good insight that the underlying principle is a powerful one. It allows for multifarious use cases and yields a bigger number of aspects to research.

An extensive definition of the P2P paradigm has been made by Wehrle and Steinmetz in [WS05], which is based on the definition of Oram in [Ora01]:

> *A P2P system is a self-organizing system of equal, autonomous entities (peers), which aims for the shared usage of distributed resources in a networked environment avoiding central services.*

In comparison to the common client-server architecture, a peer here fulfills both roles. It is client and server at the same time. Therefore, in a pure P2P system, where every participant is on a par with every other, there is no central instance, which coordinates or controls the operation of the network. As a direct consequence of the lack of central control, the peers have to establish and maintain communication and operation by themselves. Thus, they have to provide self-organizing mechanisms to keep the system intact. Especially in the face of *churn*, which describes the possibility of peers leaving the network at arbitrary times and without coordination, this becomes a necessity.

The peers build an *overlay* network, which is created on top of the regular Internet infrastructure. Many P2P applications follow the end-to-end-argument in systems design described by Saltzer et.al. in [SRC84], which has largely been credited for enabling the Internet's success in general. It states that the Internet as a basis for communication should only serve the purpose of message transfer for the end-user, and all functionality should be provided at the end points.

Wehrle and Steinmetz evaluate in [WS05] three main challenges, which occur in the design of P2P systems. However, these can sometimes also be understood as opportunities to overcome weaknesses of centralized systems:

Scalability: The mechanisms in use have to provide usability even with a large number of participants on a global level. Bottlenecks in system design limit the overall performance and managing mechanisms may not grow disproportionately with the amount of peers. On the other hand, when sharing resources, the rules of economies of scale may apply. Goals may be achieved by the combination of said resources, which would never be feasible for a single peer.

Security and reliability is a vital necessity, as a compromised distributed system, may not only be affected in its own availability, but can also pose a threat to other Internet infrastructure through e.g. distributed denial of service attacks. However, distributed systems do not reveal a single point of failure. Therefore, they are potentially more robust and resilient against outages and attacks.

Flexibility and Quality of Service: Especially considering upcoming phenomena like ubiquitous computing it is relevant how easy new services and technologies can be integrated into the present landscape.

[2] http://www.coralcdn.org/
[3] http://www.tribler.org

An important factor for P2P systems always is the motivation for its participants to take part in its operation, i.e. when sharing local resources, one may want to see a benefit in return. Consequently, the success of a decentralized P2P system is strongly dependent on the right balance between the incentives and the costs its participants have to face.

2.2 The GNUnet Peer-to-Peer Framework

This section introduces to the GNUnet P2P Framework[4]. It is one instance of a completely decentralized P2P network in the sense of the definition in the previous section. First its special focus on the security of its participants is explained, which imposes further requirements on it than those of past P2P networks. Afterwards, a brief overview of the parts of its architecture relevant to the rest of the thesis is given.

2.2.1 Focus

Deriving from its own success Blumenthal and Clark argue in [BC01] that today's Internet draws a different picture, compared to when it started out. Internet related companies, such as the service providers themselves or streaming services are looking for new business models, or try to protect out-dated ones, which are already obsolete from a technical viewpoint. Governments try to seize influence and control to preserve legal or state interests. The biggest change, though, derives from the loss of confidence in the other participants' benevolence. Due to the numerous and diverging interests of the parties connected to and operating on the Internet, it is appropriate to face it with the right amount of cautiousness.

Based on the shift of paradigms described in the previous Section 2.1 the GNUnet P2P Framework aims to become a developmental and operational base for modern Internet applications fully relying on the principle of decentralization. Furthermore, it targets at exploiting the full potential of modern communication technology. Apart from meeting the requirements of P2P systems as described in the previous section, a further emphasis lies on the protection of its end-users' interests, considering Blumenthal and Clarks observations. Ultimately, in GNUnet the security of the communication is valued higher than performance or efficiency. The goal is to preserve what Blumenthal and Clark [BC01] call the conventional Internet philosophy:

> *Freedom of action, user empowerment, end-user responsibility for actions undertaken, and lack of controls "in" the Net that limit or regulate what users can do.*

In order to sustain this philosophy, GNUnet targets at a secure communication. "Secure" in the sense, that it makes guarantees for the authenticity of the communication partners and preserves the integrity and the confidentiality of the messages exchanged. This is achieved by the use of cryptography, in order to confirm the communicating parties' identities and to prevent the unauthorized modification of or access to their exchanged information. Moreover, anonymity is provided by the deniability of who the originator of a message is, by hiding it amongst other traffic.

GNUnet is a developmental platform, which can be extended and reused to grow with new demands of future Internet applications. It supports many different varieties of P2P applications, by providing a secure basis for operation. With this context in mind, GNUnet also offers the chance to study new methodologies and mechanisms in recent P2P networking research.

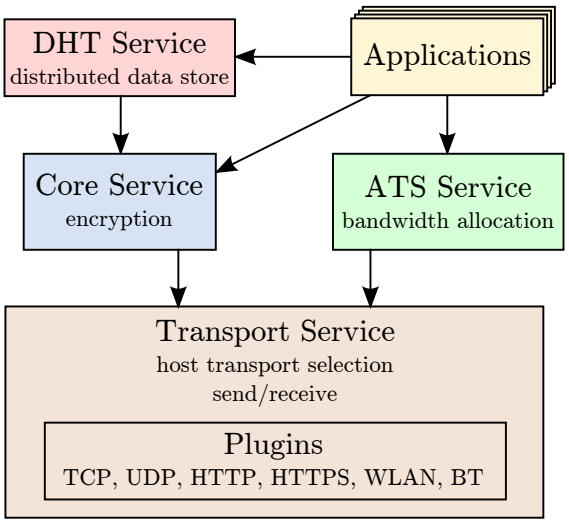


Figure 2.1: The GNUnet Service Architecture

2.2.2 Design and Architecture

The design of GNUnet uses a multi-process architecture, where functionality is encapsulated in modules called *services*, which run in their own process for the sake of fault isolation and easy extensibility. They follow a loosely layered scheme, where higher residing modules make use of or coordinate lower ones. Each service exposes an Application Programming Interface (API) implemented as a library. Via a specific network protocol the library communicates with the main process of its service. Many services rely on the functionality of other services as subcomponents. This modular composition allows the testing of singled out building-blocks and their replacement by new implementations, richer in features, or more suitable for different devices.

The services are controlled by GNUnet's main service the Automatic Restart Manager (`arm`), which takes care of the initialization of the system, reading its configuration and performs other tasks, such as (re-)starting service processes.

Figure 2.1 based on the one found in [Gro11] shows a small excerpt of some of GNUnet's foundational modules, which are relevant to the problem discussed in this thesis.

In the layer-oriented modular architecture at a lower layer is the `transport` service. Its responsibility is the low-level connectivity to peers. Via plugins, which fit a generic transport API, many communication protocols are supported, among which there are currently TCP, UDP, HTTP, HTTPS, WLAN and Bluetooth. The service hides away the implementation of specific protocols from the rest of the system as designed by Ferreira et. al. [FGR03]. The `transport` service may thus use several means of communication and peers can have multiple addresses at the same time. This increases the general connectivity and efficiency of the system in two distinct ways:

On the one hand, a peer can utilize different physical connections to connect to the P2P network, such as a 3G mobile data connection as well as a stationary WLAN with an Internet gateway. In case of failure of one of the two physical accesses, there is still another one left. The redundancy of used interfaces thus increases the resilience of the network.

On the other hand, a variety of transport protocols or mechanisms can be used to overcome NATs or firewalls. In a network, which impedes the usage of certain protocols, using

[4] https://gnunet.org/

traffic shaping or filtering, alternatives can be used. For example, if an operator allows connections outside his network only via an HTTP proxy, a peer could still connect to the network using the HTTP transport plugin.

Thus, connectivity might even be preserved in restricted networking environments, even in scenarios with a lack of infrastructure, protocol discrimination or censorship.

The `core` service is responsible for the provision of link-encrypted P2P communication and peer discovery.

Above those essential services, lie higher-level services like GNUnet's implementation of a distributed hash table, a standard distributed data structure in modern P2P systems. Applications, such as the already present anonymous file-sharing, are located on top of the services infrastructure.

At an intermediate level is the `ats` service, whose task is to allocate bandwidth to connections made by the `transport` service. The target of the `ats` service hereby is to satisfy the requirements of the applications. Its functionality is further discussed in Chapter 3.

3. The Problem of Automatic Transport Selection

For the purpose of an improved connectivity GNUnet supports communication via multiple transport protocols. As a result peers can have several addresses to be communicated with. The communication resources in terms of bandwidth are limited. The decision needs to be made, which address to use and how much bandwidth to allocate for a connection, in order to serve the applications' requirements best. This is called the Automatic Transport Selection (ATS) problem.

This chapter details further the characteristics of this problem. For this purpose first the `ats` service of the GNUnet framework is explained, in order to derive the inputs, outputs, and objectives that are part of the problem. A formalism is given to abstract the problem statement. Afterwards, it is argued about the limits of an automatic solution to the problem, given a realistic adversary model in P2P networks. Finally, two solution methods which have already been in existence prior to this research, are presented and their unique characteristics are compared and open issues are identified.

3.1 The ATS Service

In order to understand the ATS problem and how it is approached in GNUnet it is necessary to know its inputs' and outputs' origins. In GNUnet the `ats` service solving the problem has three interfaces for these. Its direct environment are the `transport` service below in the layered architecture, the applications on top, and the static user configuration. Their roles are as follows:

Transport Service The `transport` service as described in Section 2.2.2 manages the connections with other peers. This is done by the dissemination of *HELLO* messages. These basically contain the cryptographic identification of the peer, as well as all its addresses of the supported transport plugins. The several communication mechanisms are managed by the `transport` service and can each provide several addresses. For instance, a peer might have one WLAN (basically a MAC address), and two TCP/IPv6 addresses. Upon the reception of a HELLO message, the `transport` service cryptographically validates newly learned addresses, in order to check their availability and authenticity. It tries to prevent the chance of man-in-the middle attacks, where there might be a third party intercepting the communication spoofing the other peer's identity or address.

The service's second main task is to establish and maintain these connections. The channels (i.e. the address and thus the underlying mechanism) for these can be chosen by the `ats` service in the outbound direction. The sender of a message decides via which means to communicate with a peer. To support the decision making process of the `ats` service, it is supplied with generic channel *properties* by the `transport` service. Metrics, such as latency, available bandwidth, connection overhead, but also network utilization in the past are available. Further metrics that may be used in the future could be the energy consumption, financial and computational costs, error rates, past availability etc. Lastly, the `transport` service is also responsible for the enforcement of inbound traffic limits, as allocated by the `ats` service, and further restrictions, such as blacklisted clients. Outbound traffic limits are enforced at the interface between the `transport` and the `core` service, via which communication is typically managed from the application's point of view.

Configuration GNUnet is most likely not the only piece of software running on a device, and the user usually does not want to share his entire available resources with the P2P network. Thus, limits for the resource consumption of GNUnet can be set in configuration parameters. For the task of bandwidth allocation the limitations on traffic are of vital importance. In order to give the user more distinct control, GNUnet differentiates between *network scopes*, in which the bandwidth resources reside. In the current implementation the configurable scopes are the ones for loopback, LAN, WAN, WLAN, Bluetooth and connections, where the scope has yet to be determined. The user depicts a *quota* both for inbound and outbound bandwidth, for each of these. The quota in a network scope is the maximum amount of bandwidth the user allows to be consumed there. Therefore, these quotas must not be exceeded in sum by the allocation of the `ats` service in each scope.

Application The applications relying on GNUnet may have differing requirements regarding communication channels. Since GNUnet is a framework the potential variety of use cases is vast. For example, a file sharing scenario prefers a connection with high throughput over one which offers a low round trip time. An audio conferencing scenario on the other hand will demand for a low latency and just a small amount of bandwidth, just enough to guarantee the successful transmission of the audio stream. Therefore, each application wants to influence the bandwidth allocation by stating certain *preferences* towards each peer it wants to communicate with. Preferences are a numerical value, which reflects the importance of a certain property of the channel. Currently, preferences can be issued for the latency and the bandwidth of the connection. A difficulty here is the non-comparability of different preferences, since applications typically have a varying idea of what actually is a "low" latency.

Summing up, Figure 3.1 displays the location of the `ats` service between the just mentioned entities. It shows the inputs and outputs, as they come from the services it interacts with. It combines and coordinates the inputs of those services.

Automatic Transport Selection When there is a connection request for a peer, the service has to provide to the `transport` service the information regarding which address it should use and how much bandwidth is reserved for it. As a result, the `ats` service has to allocate the resources available in each network scope. Those are given by the user configuration and should not be exceeded. The selection of the appropriate address determines in which network scope bandwidth has to be spent.

The goal is to meet the applications' demands in the best possible manner. Distributing as much bandwidth as possible is not the only target. Due to the channel

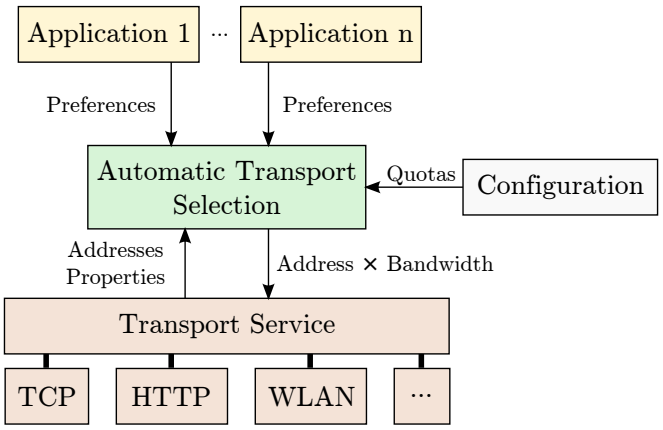


Figure 3.1: The Automatic Transport Selection Problem

properties and application preferences there are multiple objectives to fulfill, which can also dynamically change. Eventually, the decision is not only about a fair balance of the allocation, but also about which connections should not be made anymore.

Furthermore, the `ats` service should take into account requirements, which are not given as direct inputs. For a good user experience, for instance, it should rather not change the channel interrupting it effectively in order to achieve a slightly better allocation. In general the allocation should not oscillate between different ones, which are both good, but the administrative overhead of changing between them impacts the system performance. Therefore, a certain stability in the allocation is required. Another target that should be aimed for is to keep the number of connections made above a distinct level. Sine GNUnet forms an overlay network, which optimally should never be split, it is favorable to maintain at least a minimum amount of connections. Thus, a diversity in the allocation is necessary to keep in mind. For the usability of a connection, at least a certain minimum amount of bandwidth should be allocated for each of them. Eventually, the decision should in its nature make the most efficient and usable composition of the resources at hand, so that the applications are provided with the best performance achievable under the inflicted constraints.

3.2 Problem Summary

In summary a solution to the ATS problem answers three questions:

- Which peers should and can be made a connection to?
- Which address should be used to talk to these peers?
- How much bandwidth should be reserved for each connection?

Whether or not to talk to a peer influences if a share of the available bandwidth needs to be allocated for it in one of the network scopes where it has addresses in. The premise is that there are enough resources to satisfy the connection demands in the first place. The choice of the address then specifies the said network scope and what properties the connection request is served with. The share of the bandwidth, which is then assigned, can also be seen as a property of the connection. It can be chosen freely compared to, e.g. the delay of a channel. The allocation should, of course, respect the constraints of the user and correspond to the requirements of the applications.

The answers to those questions, the *address selection* and the *resource allocation*, influence each other. It might make more sense to allocate a smaller amount of bandwidth to many connections, but it might as well be more useful to have less connections, which can then be served with more bandwidth each. Choosing the address for a connection in an already "crowded" scope, might mean the address cannot receive a large amount of bandwidth. It might still be the better choice for the connection, if all other available addresses of the according peer do not fulfill the delay requirements. A mechanism needs to make the according trade-offs taking into account the application preferences and address properties.

A further characteristic of the problem is that it has to be solved in a dynamic environment. A P2P-network such as GNUnet is confronted with churn: The problem of peers connecting to and disconnecting from the network at arbitrary times without prior coordination. In general, connections do not have to be static. An application might change its preferences towards it, or the connection might simply not be requested anymore at all, after communication over it has finished. Therefore, the best possible decision for a resource allocation can change at any time. This means the problem has to be solved either repeatedly or even continuously, so that the change in the environment is reflected in the allocation.

3.3 Formalism

Formally the approach has the following elements:

The **Inputs** are:

- Peers P
- Transport mechanisms M
- Network scopes N
- A bandwidth restriction R, with $r_{n,d} \in R$ for each network $n \in N$ and direction $d \in \{\text{in}, \text{out}\}$
- Address $a \in A_{p,m}$ for peer $p \in P$ using mechanism $m \in M$
- All addresses to connect to peer $p \in P$, $\bigcup_{m \in M} A_{p,m}$
- All addresses for a transport mechanism $m \in M$, $\bigcup_{p \in P} A_{p,m}$
- Quality and cost properties of addresses Q
- Upper layer applications U
- Application $u \in U$'s preference for peer $p \in P$ and property $q \in Q$, $u_{p,q} \in [1, 2]$

The **Outputs** are:

- Active addresses $\hat{a} \in \hat{A} \subseteq A_{p,m}$ to be used for communication
- An amount of bandwidth $b_{\hat{a},d}$ for each address $\hat{a}$ and direction $d \in \{\text{in}, \text{out}\}$

The **Constraints** a solution must adhere to are:

- For all network scopes and directions the amount of bandwidth allocated in sum must not exceed the respective bandwidth restriction for the network

$$\bigwedge_{n \in N, d \in \{\text{in},\text{out}\}} \sum_{\hat{a} \in \hat{A}} b_{\hat{a},d} < r_{n,d}$$

- For each peer at maximum one address may be active

$$\bigwedge_{p \in P} |\hat{A} \cap A_{p,m}| \in \{0, 1\}$$

The **Objectives** a solution should exhibit are:

- Maximize the amount of bandwidth assigned to and utilized by the peers
- Choose addresses with the lowest cost and best quality
- Select an address that corresponds best with the applications' preferences
- Pursue a diversity in connections
- Have a stability in the allocation
- Provide a minimum amount of bandwidth $b_{\min}$ for each active address

3.4 Local vs Global View

The problem as described in the previous sections is based purely on a local view. However, it would also be possible to take a figurative step back and define a global point of view. Seeing all the peers, a local peer might be connected to, there are further ways to optimize the allocation of bandwidth, if they exchange messages. According to this information a consensus could be reached on how bandwidth should be assigned in order to attain the best performance possible.

Due to a lack of trust towards the other peers, this is not considered in this thesis, though. As has been shown in [Dou02], in a decentralized system, there is always the practical possibility of counterfeiting multiple identities. Therefore, performing a so-called Sybil attack, a potential adversary could compromise a disproportional share of the number of nodes in the system. Additionally, due to the results of the byzantine generals problem [LSP82], it is known that it is necessary that at least two-thirds of the peers participating in this calculation are not hostile, in order to reach consensus. As a result, for a malicious participant in the network this always opens the possibility to exploit a peer's resources.

Taking these facts into account, the problem can only be considered from a local viewpoint. The resulting bandwidth allocation should benefit the local peer most. As discussed in Section 2.1 it is vital for a P2P system to give its users an incentive to share their resources. Consequently, it is necessary to protect them from being abused by doing so.

3.5 Existing Solution Methods

In this section the descriptions of two mechanisms are given, which provide solutions to the ATS problem. The first section covers the Proportional Heuristic, which is a fast method to reach a solution. The second one comprises the Mixed Integer Linear Programming (MLP) approach, which uses a mathematical optimization approach based on the means of linear programming. It begins with a short introduction to the mathematical background. Eventually, the last section compares those solutions and explains their characteristics and which issues are not addressed by either of them.

3.5.1 Proportional Heuristic

The proportional heuristic takes into account the applications' peer preferences and the transport properties of the peer's addresses to make a fast informed decision. This is done in the following three steps:

Address Selection For each peer, for whom an address is requested, the method finds the "best" address, which is in a network where at least the minimum bandwidth can be allocated. "Best" is evaluated here in perspective of currently three criteria, which are in strict order:

1. Addresses in use for a connection are kept, in order to not disrupt an ongoing connection.

2. The distance property of the address. This reflects the hop-distance in the network the address resides in. Local addresses are preferred over global addresses this way.
3. The latency property of the address. Among addresses with equal distance, the one with the shortest latency is preferred.

Preference Calculation A total preference is determined for each peer by summing up all its single preference values.

Bandwidth distribution For each network, which contains at least one active address chosen during the address selection, the bandwidth is distributed as follows: First each address gets the minimum bandwidth $b_{\min}$ required for a connection. Then, the remaining bandwidth is divided in proportion to the global preference value calculated previously. This proportional share plus the minimum bandwidth are then allocated for this address. This calculation is the same both for the inbound and the outbound direction.

The question, if resources can still be allocated to a new connection, is determined by whether there is enough bandwidth for each connection to receive at least the required minimum. If this is not the case no bandwidth suggestion for the new request will be made.

3.5.2 Combinatorial Optimization

With the help of advanced mathematical and analytical methods and the computational resources of today's computers, it is possible to find solutions to the ATS problem, which are provably optimal. Linear Programming is a long-established technique, which allows this in an efficient manner in a certain mathematical model. But before it's direct application is explained in Section 3.5.2.2 its theoretical background is clarified next.

3.5.2.1 Mathematical Optimization Overview

The theory around mathematical optimization has its origins in the field of Operations Research. It was developed to improve logistic processes, on the ground of mathematical considerations. Having found applications both in the military and civil industries, the theory is thoroughly developed and has many practical solution methods.

In optimization problems the goal is to find the optimal solution to a problem among all possible ones. Often it is easy to evaluate the quality of a solution (e.g. the length of a path in a graph), but it is not straightforward how to construct the optimal one (e.g. finding the shortest path in a graph, which includes certain nodes). Sometimes analytical means can be used to enhance this process.

The formal definition is as follows according to the general introduction by Chong et al. [CZ13]:

$$\begin{aligned} &\text{minimize/maximize } f(\vec{x}) \\ &\text{subject to } \vec{x} \in \Omega \end{aligned}$$

Hereby the value of the result, which is given by an *objective function* $f : \mathbb{R}^n \to \mathbb{R}$ has to be minimized or maximized. This means for the variables of its input, the vector $\vec{x} \in \mathbb{R}^n$, the variables $x_1, x_2, \ldots, x_n$ have to be found, which yield the minimum/maximum value of f. The variables of x are also called the decision variables. The set $\Omega \subseteq \mathbb{R}^n$ depicts the allowed vectors of x. Ω is therefore named the constraint set or feasible set. If Ω equals

$\mathbb{R}^n$ all possible values are allowed and the problem is said to be unconstrained. In reality this is very seldom the case, though. Consequently, it is then called the constrained case.

The minimization of f and its maximization are equivalent: The maximization can simply be achieved by minimizing $-f$. Typically the constraint set Ω is defined in a functional form such as $\Omega = \{\vec{x} : \sum_{i=1}^{n} x_i \leq y\}$, which for instance means that the sum of all the variables in $\vec{x}$ may not exceed y.

A classical example of an optimization problem is the one of the travelling salesman, who has to visit a number of cities and wants to spend the least amount of time possible for his travels. He knows the distance between all pairs of cities he wants to visit and wants to find an itinerary, which has exactly one stop in every city and takes him back to his hometown. The objective function to minimize here is the sum of the distances in the itinerary. Constraints are, that every city may only be visited once and that the last destination has to be the city of the salesman's origin.

This example is of the same complexity as the ATS problem considered in Section 3, which deals with the allocation of discrete resources. This means that at least one variable in $\vec{x}$ is an integer. As it turns out resource allocation problems with discrete variables are NP-hard and can therefore not be solved in polynomial time complexity. The proof can be found in the book by Ibaraki et al. [IK88].

3.5.2.2 Mixed Integer Linear Programming Solver

The Mixed Integer Linear Programming (MLP) approach uses the theory of linear programming in order to solve the ATS problem. This means it formalizes the problem as a special case of the general constrained optimization problem discussed previously. According to the introduction by Chong et al. [CZ13], the special restrictions here are, that the objective function, one seeks to minimize or maximize, is linear. Additionally, the decision variables are subject to linear constraints. Therefore the set of valid solutions is determined by a number of linear equations and/or inequalities.

Formally a linear program has the following standard form:

$$\begin{aligned} &\text{minimize/maximize } \vec{c}^T \vec{x} \\ &\text{subject to } \boldsymbol{A}\vec{x} = \vec{b} \\ &\vec{x} \geq 0 \end{aligned}$$

where $\vec{c} \in \mathbb{R}^n$, $\vec{b} \in \mathbb{R}^m$, and $\vec{A} \in \mathbb{R}^{m \times n}$. $\vec{x} \geq 0$ means that each component of $\vec{x}$ is nonnegative (e.g. there may be no negative bandwidth assignments). Also inequalities, resulting in constraints such as $\boldsymbol{A}\vec{x} = \vec{b}$ fall under the same category of problems. As shown in the book by Chong et al. it is possible to transform problems with linear inequalities into the above standard form of linear programs.

In order to specify the linear program in the ATS problem domain, the MLP method formulates the following operational constraints, which have to be fulfilled for valid solutions:

- For the sake of a finite solution space, a high maximum amount of bandwidth per connection is set. Transport mechanisms, such as Unix domain sockets, could otherwise theoretically be treated without bandwidth limitations.
- Each peer has only one active address.
- Each requested peer is allocated at least the minimum bandwidth b_{min}
- A minimum number of connections is maintained, i.e. the bandwidth is distributed among a minimum set of addresses. This serves the connectivity of the overlay network established by the P2P system.

- The resource boundaries, which are set for each network scope, may not be exceeded.

The objective function the MLP approach seeks to maximize is the sum of a metric for the connection diversity and a metric for the matching of the upper layer applications' preferences. The connection diversity is hereby the number of active addresses times a constant D, which determines the metric's share in the objective. The metric for a solution's match of preferences is the sum of the product of each quality with the matching preference. This results in:

$$\text{maximize } |\hat{A}| \cdot D + \sum_{q \in Q} \sum_{p \in P} \sum_{m \in M} u_{p,q} \cdot a_{m,q} \cdot n_t$$

where $\hat{A}$ is the set of active addresses, Q the set of quality properties, P the set of peers, M the set of transport mechanisms, $u_{p,q}$ the preference for peer p and quality property q, $a_{m,q}$ the actual quality of property q and transport mechanism m, and n_t a value from $\{0, 1\}$ determining the activity of an address.

The solution of this integer linear program is performed in two steps. First, the problem is treated as if all the variables were continuous and a solution is derived using either the simplex or the interior point method. Then, based on this solution, an MLP solution is found, which takes the integer representation of the addresses into account using e.g. a branch and bound approach. In repeated executions the problem can be solved very efficiently, based on information from the previous execution, if the problem shape has not changed. This means no addresses or peers may have been added or removed in the meantime.

3.5.3 Evaluation and Open Issues

The two methods presented here are different in computational demands and complexity, which has consequences for the quality of the solutions they output:

Proportional Heuristic The advantage of the approach is, that the separation of the subtasks of address selection and resource allocation reduces the complexity of the problem. Consequently, less computational resources are necessary to calculate the solution. This can be expected to decrease execution time significantly.

Because of its separation of address selection and bandwidth distribution, the algorithm does not perform a search in the entire space of valid solutions. Instead, for each peer the "closest" address in terms of distance in the topology of the network is chosen and an appropriate share of the available bandwidth is allocated for it. This yields solutions, that are desirable in a sense of generally avoiding connections, which bridge a higher distance in the network topology than necessary. However, it potentially wastes available bandwidth in a scenario, where all requested peers have two addresses, both in a local and a global scope. The possibility of distributing the bandwidth of the global scope, i.e. trading it in for more bandwidth for each address in the local scope would be neglected. In addition a connection, which has a lower distance in the topology, might as well have a higher latency. Therefore, a better performance is not necessarily achieved due to this metric.

Linear Programming As has been mentioned in Section 3.5.2.1 an optimization problem with discrete variables is NP-hard and therefore not solvable in polynomial time

complexity. Consequently, a complete recalculation using the MLP approach is computationally expensive. With regard to the amount of addresses, which have to be respected, execution time of the algorithm increases rapidly, but may still be in acceptable bounds considering the number of connections in current P2P applications. Another characteristic of this approach, though, is that new solutions may look largely independent from one another. While the theoretical distribution of the resources is optimal considering the objective function, it is hard to formulate requirements, such as the stability of ongoing connections. Furthermore, the formulation of the objective function itself is a non-trivial task. Boiling the evaluation of the allocation down to a single number, the objective function's result, might not be ideal. However, it is a good indicator for such and the linear programming method provably outputs the best solution on its base.

Both approaches, however, do not account for the dynamic nature of the ATS problem. The inputs can change over time. Application preferences, connection requests, measured address properties (e.g. delay and distance), and more change permanently. Due to churn in P2P networks even the topology of the overlay network cannot be assumed to be stable. The proportional heuristic and the linear programming method basically need to perform a complete recalculation when the inputs change, but do not benefit from solving it repeatedly.

Furthermore, it has been observed that there is a difference between the provisioning of resources, i.e. the assignment, and their actual utilization. During operation, connections often do not exploit the full amount of their share of assigned bandwidth, which leads to a utilization below what is allowed. An intentional overallocation could solve this issue, as long as the actual utilization does not exceed the quota. The existing approaches do not make an effort in this direction.

The field of ML offers methods to learn from past events and to improve upon the execution of a task over the course of time. The ML domain therefore lends itself for further investigations, of how the ATS problem can be solved differently or how the decision making process can be further improved. Therefore, the next Chapter will analyze how ATS can benefit from ML methodologies.

4. Machine Learning for Automatic Transport Selection

This chapter elaborates about the possibilities how Machine Learning (ML) methods can improve ATS in decentralized P2P networks. Now that the ATS problem is specified and explained, the open question is how ML can be used solving it. Therefore, this chapter gives an overview of different ML approaches and what problems can be solved in its domain. The next section narrows further down how ML methods can be applied on the ATS problem and gives an overview and comparison of different perspectives.

4.1 Machine Learning Overview

The field of ML methodologies is very wide as it covers a vast number of algorithms and has successful applications in a variety of fields. From language processing and malware detection over control to optimization, ML is used to find solutions for many problems.

Where machines accomplish tasks, they have to be programmed how to do so. However, there are tasks for which it is not easy to write a program. Yet sometimes the programmer has an idea or a good amount of data, which indicates how the program should behave or what it should produce. Typically ML algorithms are involved in the approach of bridging this gap. Thus, a model is defined, which lets the machine make predictions or descriptions or both from a certain input. Learning, then, is the optimization of the parameters of the model with the help of data or past experience or a bit more formal as it is defined in the introductory book by Mitchell [Mit97]:

> *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*

Thus, ML has natural ties with mathematical optimization, which was introduced in Section 3.5.2.1. The challenges lie in finding a model, which fits the task at hand, and in finding an efficient approach to adjust its parameters. The risk is to have a model, which cannot do the task independent of the adjustment of its parameters. Experience in this sense could be anything, that does not let the algorithm start from scratch, e.g. previous executions or data.

4.2 Machine Learning Classification

A common categorization of ML methods splits them into supervised, unsupervised, reinforcement and evolutionary learning algorithms and can be found e.g. in the introductory books by Marsland and Mitchell [Mar09, Mit97]. The first three categories divide them according to the amount of information provided to the learner. The last stems more from having rather separate research origins. Most of the ML algorithms and applications can be found to fit one of these or are at least strongly related.

Supervised Learning is the most prominent category of ML. It is the task of deriving a function from training data. This includes the problem of classification of inputs. A typical example is deciding, whether an incoming e-mail is considered "spam" or "no spam", based on previous e-mails, which have been flagged as such by the user.

The learning algorithm is confronted with a training set (e.g. the emails marked by the user), which consists of input/output pairs (e.g. a specific e-mail and "spam"). This set serves as a sampling of the whole problem space, which the algorithm covers (e.g. all the e-mails).

The first step is to use pairs of the training set to teach the algorithm some correct mappings of the problem space. This is where the "supervision" comes from, the learner is told what the right outcome for the function is for an example and is therefore supervised. Afterwards the algorithm can be confronted with unseen inputs/samples for which it determines the outputs/target, solely based on its experience with the training set. The algorithm is said to generalize over the problem space.

Besides the classification of inputs, another typical ML task is regression. Regression comes very close to interpolation, which is the task of finding a function that is exactly correct in the points given by the training set. Regression adds the assumption that the samples of the training set are affected by noise. Thus, the task is to find an approximation for the function, from which the training set was sampled.

It turns out that the tasks of classification and regression basically solve the same problem and the solution of one can also be used for the other using an appropriate transformation.

Most of the widely known ML techniques belong to this category, such as Artificial Neural Networks (ANNs), Support Vector Machines (SVMs), Bayesian Statistics, Kernel Estimators and Decision Trees.

Applying supervised learning alone to the ATS problem is not realistic. It would be conceivable to set up an ANN or another mechanism in order to learn good allocations from a training set, which gives exemplary ATS inputs and according allocations. However, is the success of such an approach highly questionable for two main reasons. The dimensionality and size of the inputs and outputs of the ATS problem are very high. Application preferences and address properties are numerous and some of these properties are continuous variables. An additional dimension for the problem space would be the space of peer identities, of which combinations would have to be learned as inputs as well. Without a reasonable mean to reduce the complexity and size of the learning problem it is not practical to pursue this. Additionally, there is the difficulty of how to generate a meaningful training set. A manually created training set is unrealistic due to the numerous parameters and because the problem is dynamic. A machine generated training set, bears the question why its generator is not used itself in the first place to solve the ATS problem.

Unsupervised Learning Unlike in supervised learning this category of algorithms is not confronted with input/output pairs, but only with input pairs. The algorithm then finds similarities among the input samples and/or gives them a structure.

A typical example application would be the recommendation system of an online shop. It analyzes which products should be advertised to a customer based on information about products he bought before and products bought by other customers purchasing the same articles beforehand.

It is closely related to density estimation in the field of statistics. Typical approaches are clustering such as with the k-Nearest Neighbor Algorithm, Hidden Markov Models (HMMs) or the self-organization of ANNs.

Even more than supervised learning, does unsupervised learning alone not fit with the ATS problem, simply because its results only give additional information of the inputs. However, it can potentially enhance other solution techniques by reducing the number of inputs to a reasonable size for the respective technique. The difficulties of a missing meaningful training set and of the dynamically changing inputs of the ATS problem are present here as well. First, the goal is, though, to find a suitable solution technique.

Reinforcement Learning (RL) is located between supervised and unsupervised learning. Like in unsupervised learning, there are no correct outputs, which would belong to some inputs. The algorithm receives from an *environment* a numeric value about how well the generated output meets the desired outcome. Through many iterations the algorithm tries many possibilities until it makes a correct prediction about which outputs or decisions will lead to a good overall feedback. Algorithms following this framework are also said to learn with a critic.

A common subproblem is to make correct predictions of the feedback in yet unseen situations. This can be done by generalizing over situations. This subproblem of generalizing over situations is a supervised learning problem.

A typical application would be to teach a computer how to play a game. The feedback it receives can be its score or even just the information, whether it won or lost. By repeated playing the actor learns which (sequence) of situations and actions leads to a better score. An actor can learn playing while doing so (online), or be trained beforehand, e.g. by playing against itself.

Solution techniques in this category consist of Dynamic Programming, Monte Carlo methods and Temporal Difference algorithms.

An RL approach fits better with the ATS problem, because a reward for an address selection or resource allocation could be derived from measuring its quality and/or its difference to previous allocations. A reinforced mechanism can alter the resource allocation sequentially and learn about the consequences of its decisions. The solution can learn to find an allocation as the inputs occur.

Evolutionary Learning This category covers methods, which simulate processes in nature that are perceived as "learning" from experience. Evolution is one such process, that does improve the chance of survival of a species.

This kind of learning is simulated by Genetic Algorithms (GAs). The basic idea is to have a string or array representation for the solution of a problem which serves as the *DNA* of a solution. Each solution has a certain *fitness*, which is a real valued number. Its evaluation is similar to the objective function in mathematical optimization problems. In the first step GAs randomly generate a *population* of solutions of which the fittest are selected for the second step of reproduction, i.e. a new *generation* of

solutions. This second step takes place using the definition of two operators, namely *crossover* and *mutation*. Crossover is the generation of new solutions by combining the DNA of two old ones. Mutation is the alteration of the new solutions. In a third step, the fittest new solutions replace the least fit solutions of the old population. This is then considered the next generation. By repeating steps two and three for a number of iterations, the solution space is effectively sampled in many more or less random points. The fittest one is then used. Through the mutation operator, local solutions are sought to be improved, while the crossover operator is based on the assumption that combinations of existing solutions again result in good solutions.

Ant Colony Optimization (ACO) is a second learning approach inspired by biology. ACO models the behavior of ants searching for food. While first wandering around randomly, an ant which has found food leaves a pheromone trail on its way back to the colony. Other ants are likely to follow a path once they found such a trail. While the pheromones on a path evaporate over time, the pheromones on a shorter path will be reinforced more, because the ants return earlier. The path is more often used as a consequence. This scheme has been found to be applicable to other numerical problems and can be used as a general metaheuristic for mathematical optimization problems as presented in Section 3.5.2.1.

Evolutionary and biologically inspired methods in the field of ML are sometimes used as search heuristics and stochastic optimization techniques. Other techniques, which serve exactly this purpose, but which are not so strongly correlated with machine learning are Tabu Search and Simulated Annealing.

The applicability of these methods to ATS is therefore plainly the same as is the optimization approach using linear programming as discussed in Section 3.5.2.2. The difference would be the non-confirmability of the optimality of the result, because these ML techniques do not search the whole solution space, as is the case with linear programming.

As we can see from the differing categories and applications of ML, the border of what actually is ML and what is not is not always clear. While introductory books to ML like Marsland's [Mar09] and Mitchell's [Mit97] list GAs as learning methods it could also be seen as a heuristic for mathematical optimization. On the other hand, even the framework of RL, which fits very well an intuitive notion of the term "Machine Learning" can be used as a heuristic for mathematical optimization. This is proposed in studies by Miagkikh [MPI99, Mia12].

Viewing at the four basic categories of ML, supervised, unsupervised, reinforcement, and evolutionary learning, only the last two are worthwhile for a further investigation on how to employ them on the ATS problem directly. The main difference between those two is how the problem is perceived. Evolutionary or biologically inspired optimization techniques treat the ATS problem as a typical mathematical optimization problem. Reinforcement learning on the other hand offers possibilities to treat it in the fashion of a control process. Here the learner regulates the allocation while it is used in a closed feedback loop.

4.3 Machine Learning for Automatic Transport Selection

This section further investigates on the usage of metaheuristics and RL on the ATS problem. A third perspective of how ML can be of help is to improve meta- and other heuristics by applying learning techniques on top of them. These efforts are referred to in here as *higher level learning*.

This section therefore highlights the three main perspectives of how the ATS problem can be tackled using ML methodologies. First, the different approaches, namely the use of

evolutionary metaheuristics, higher level learning on heuristics, and RL are explained in how they relate to the ATS problem. Then they are compared in a further subsection.

4.3.1 (Meta)heuristics

Heuristics are approximations of optimization techniques. A new solution, which gains a significant advantage e.g. over the optimization as it is performed by the MLP approach presented in Section 3.5.2.2 would therefore be one, which massively reduces computational demands, while finding almost optimal solutions. They do not perform an exhaustive search in the solution space, yet try to sample it enough to have a good chance of finding an appropriate solution.

As an example shall serve Genetic Algorithms (GAs). They are particularly interesting here, since they have both been used in a similar manner to non-ML based metaheuristics. In the book by Pétrowski et al. [PT06], for example, they are compared to methods such as Simulated Annealing or Tabu Search. However, they have also been used to improve the rule set in ML applications, such as the "classifier systems" described in the book about GAs by Goldberg [Gol89].

As discussed before the method is inspired by the theory of evolution and maps its concepts to the search of solutions in a problem space. This concept could be used to solve the ATS problem in the following manner: DNA of a solution could be represented as an array of triples with one entry for each requested peer. The triple would consist of an address of the respective peer and the associated bandwidth allocated inbound and outbound. The fitness function could be exactly the same as the objective function of the MLP solver. Solutions that break its constraints, would have to result in some minimum fitness value. The crossover operation may be defined by using the randomly chosen first n elements of the array of the first solution and the rest of the second solution. This is a very typical definition for the crossover operator. It can be expected that it is not very effective, when searching for solutions to ATS, because the combination of two good solutions is likely to just break the resource constraints. The mutation of the DNA could mean a random or heuristic increase or decrease of some bandwidth for a peer or switching to another of its address.

Such a procedure to solve the ATS problem could potentially find convenient solutions, which are "good enough", i.e. finding a reasonable allocation, which does not waste resources and respects the application requirements. Compared to the MLP approach, the results might be achieved faster, since the search in the solution space is not exhaustive. This is not necessarily the case, though, because the simulated evolution in GAs might have more computational overhead than the analytic solution of the MLP approach. However, the quality of the solution would stand back against the ones found by MLP, which finds the optimum with regard to the objective function. To a certain extent the approach can also reuse existing solutions, even when peers join or leave. In case of a leaving peer, a new population could be generated by equally distributing that peer's bandwidth over the addresses in the same scope used by other peers. In this manner new array entries for the remaining peers could be generated. As a consequence, the quality of the solution could potentially improve over repeated executions of the calculation. Aside from that, the quality of the solution found in each new calculation can of course not exceed that of the MLP solver. In this regard an approach using a metaheuristic is not inherently different from the MLP approach. At best it saves execution time in the repeated calculations. At worst it neither saves execution time, nor does it find solutions superior to that of the proportional solver.

However, there is a difficulty about metaheuristics, which often lies in the good selection of their parameters. In the GA example of the previous section such parameters would be

the number of generations to produce, the size of a population, probabilities of when and how to use the crossover and mutation operators, and so on. Taking one step-back, even the selection of the right metaheuristic can be seen as some sort of a parameter in itself. The mechanisms might have very different characteristics, which are not obvious in their impact on their ability to solve the problem well. Pétrowski and Taillard therefore write in their book [PT06] that the choice and adjustment of a heuristic now rather relies on expert knowledge instead of established and well-founded rules. In fact, Wolpert and Macready delivered theoretical proof in their "No Free Lunch" theorems [WM97], that there is no such thing as a superior solver for all combinatorial optimization problems. Any algorithm that performs better than random on a set of problems, must necessarily perform worse on all the others. However, this only holds if there are no assumptions made about the problem itself. For general purpose metaheuristics each parameter configuration is one instance of an algorithm. Consequently follows that there is no perfect setting for their parameters, but their well adjustment is problem dependent.

4.3.2 Higher Level Learning

Once again from the field of ML, there are efforts towards solving the parameter configuration issue. Works, such as Birattari's PhD thesis [Bir04] have determined the right choice of parameters or heuristics as a ML task. Birattari builds on the problem of model selection, which occurs in the supervised learning domain. The task here is to pick the statistical model, which generalizes over the training set best. This is transformed into the question, which configuration of a metaheuristic derives the best results on past instances of a combinatorial optimization problem. Birattari's approach, since it derives from supervised learning, however, does this in an off-line fashion. This means there is a training set of problem instances, with which the final parameters are tuned without any interaction with the system. Afterwards, the best deemed configuration is left unchanged during "production". Similar approaches addressing the necessity of automatic parameter tuning are surveyed in a paper by Dobslaw [Dob10]. The developments mentioned therein are mostly motivated from a statistical viewpoint and not from a ML one, even though the fields overlap in their methodologies here. However, the general pattern of an off-line calculation of the best deemed parameter configuration is the same. On-line approaches towards a parameter control during operation exist in the research area of evolutionary algorithms. This direction is less a learning approach, though, but rather an attempt at a better algorithmic design, with the help of dynamic control parameters, such as a varying population size in a GA. While the motivation of the research addressing the postulations of the no free lunch theorems is a viable effort, Dobslaw also raises questions about the usability of the efforts. The additional computational resources spent by parameter tuning need to make up for it, resulting in a significant advantage during production.

Related is the research regarding so called *hyper-heuristics*. In a similar manner they operate on a higher level on top of heuristics, in order to combine them or find new ones. These low-level heuristics are then used to solve the actual problem. The PhD thesis of Misir [Mis12] contains a very recent and exhaustive survey of this methodology. Hyper-heuristics are categorized along two dimensions. The first is the difference between those, which *select* heuristics and those, which *generate* them. Selection algorithms choose the best performing one from a set of heuristics to solve a specific instance of a problem. Frequently used in this domain are standard metaheuristics, such as GAs, Tabu Search, and Simulated Annealing. Generating algorithms, on the other hand, are ones that find and combine new heuristics from a set of low-level manipulations. Here can be found a strong link towards the genetic programming domain, which is the main instrument for the generation process. The second dimension along which hyper-heuristics are typically distinguished are, once again, online and offline approaches. The former ones try to improve

their findings during the search/optimization of one or many problem instances, while the latter ones are trained beforehand or search for a suitable combination in advance and solve the problem subsequently in a static manner. The continuous improvement effort made here often times relies on the typical RL framework as presented in Section 4.1, but may also rely on global search techniques.

Clearly, these high-level approaches offer an even bigger number of ways on how to incorporate ML in the solution of the ATS problem. While (meta)heuristics offer the chance for a faster search for solutions, on top of them can be incorporated a plethora of mechanisms, which increase their effectiveness. Especially online methods for parameter adjustment and heuristic combination promise an ever increasing probability of finding high quality solutions, since the ATS problem has to be solved repeatedly. Higher-level learning does not hold fundamentally different concepts as is already the case with the use of metaheuristics. The solutions found may neither exceed in optimality those from the MLP solver. In that sense, the mentioned high-level approaches are always just a search, mitigating the risk of using a wrong heuristic adjustment for the problem, or augmenting its flexibility or effectivness to a certain extent. The question to ask here is how much this extra effort in implementation and computation pays off and if it is even necessary. Additionally, higher learning methods are focused on static problems and their adaptability to the dynamic environment is to be questioned.

4.3.3 Reinforcement Learning

Accomplishing the task of ATS (see Section 3) directly through the means of RL is most promising, because it enables the system to learn how to perform the bandwidth management. Such an approach sets itself apart from the previous two sections as it is less fixated on the notion of viewing ATS purely as a problem of combinatorial optimization.

The learner, which is referred to as *agent* makes decisions within its *environment*. It performs *actions* depending on the situation in which the environment is called *state*. By receiving information in the form of a numerical reward, the agent receives feedback about how good his action was. By repeatedly performing actions the agent learns which action is followed by positive feedback, depending on the situation. The agent effectively learns how to find a suitable solution by itself. The approach is inspired by nature, just like a newborn learns what it needs to do by the feedback of its environment. However, similar to nature, where not every animal can learn how to count, there are theoretical and practical boundaries to what tasks an algorithm can learn.

The RL framework has a well studied theoretical background and has been applied in very different domains, such as trading, game playing, routing, and robotics. The most impressive application is probably Tesauro's Backgammon agent, which managed to reach a level of world class human players and even found formerly unknown but very successful strategies of playing [Tes95]. This self-learning property suggests that it is suitable to apply a similar approach to the ATS problem. Especially the fact that it is difficult to formulate requirements that are sought for in a good allocation leaves the opportunity for a reinforced mechanism to figure this out by itself.

Contrary to metaheuristics, this yields an online learning and control process, in which the "calculation" of the solution happens dynamically in response to the system. Therefore, such a solution to the ATS problem is less one like in combinatorial optimization, but also incorporates a view of classical control theory.

The applicability of reinforcement learning depends strongly on the complexity of the learning task. It is defined by the magnitude of the possible situations the agent can be in, the number of actions available to it, and the informative value of the reward.

The known approaches to learning in this framework have four elements the agent's behaviour is based on:

Policy An agent's policy determines its way of behaving, based on its perception of the surroundings. It is a mapping from the state of the environment to the actions to take.

Reward function The reward function is not influenced by the agent itself, but is returned to it along with the current environment state. It is a single real valued number and gives the agent a sense of the "usefulness" of the state it is in and how good the last action was. The reward signalizes to the agent what has to be achieved, but not how to do so.

Value function The value function is closely related to the reward function as it also evaluates the benefit of a state. Contrary to the reward function, though, it does not indicate the immediate reward connected to a state or a state-action pair, but a long-term value. It also takes into account the rewards of future states, which can be reached from this one. Most RL algorithms evolved around the estimation of a value function, on top of which the policy is derived to simply take the action with the highest expected value.

Optional: Model Some algorithms make use of a model of some sort of the environment. This model is adjusted by the rewarded experience and used to calculate the outcomes of certain actions and considers possible future states. These algorithms relate to a kind of planning the course of actions taken by the agent.

In fact, RL has already been studied on problems not too distant from the ATS problem. Examples are Tesauro's approach to managing the resources of a data center in order to perform load balancing [TJDB06]. The resources, though, could not be split arbitrarily, but the task is rather to assign servers to application. The chosen approach was not to introduce one agent, which centrally manages this assignment, but to have several learning agents for each application. Similar ideas have been followed in domains, such as job scheduling in grid computing [GCL04], general load-balancing in distributed systems [SST95], or resource allocation in local systems [PSC+13]. The resource allocations performed in these works differ significantly. Some are done under realistic circumstances, some are just performed on models. Some tasks involve homogeneous resources, like computational power, some involve resources of different nature, such as memory and processing power that are needed. Also the learning processes themselves vary. There are model-based and model-free, single-agent and multi-agent solutions. On the bottom line, the solutions found are always adapted very much to the specific problem at hand. Yet, promising results have been achieved, which point out that a similar self-organizing RL based resource allocation mechanism for the ATS problem is feasible. The design of such is a non-trivial task, though.

4.3.4 Comparison of Different Machine Learning Approaches

The ML domain offers a wide range of methodologies. As the definition in Section 4.1 suggests, it is neither restricted to a set of tasks that are fulfilled by a distinct set of algorithms, nor is it bounded by a common formalism. The idea of learning as improving by experience can be applied manifold. Therefore, there are many levels at which it can happen while solving the ATS problem, even using the same idea such as RL. Wauters determines in his PhD Thesis [Wau12] levels where RL may be used to solve problems of combinatorial optimization. Based on this, Figure 4.1 similarly shows the perspectives attributed to the three preceding sections of how learning can be incorporated into a solution of the ATS problem.

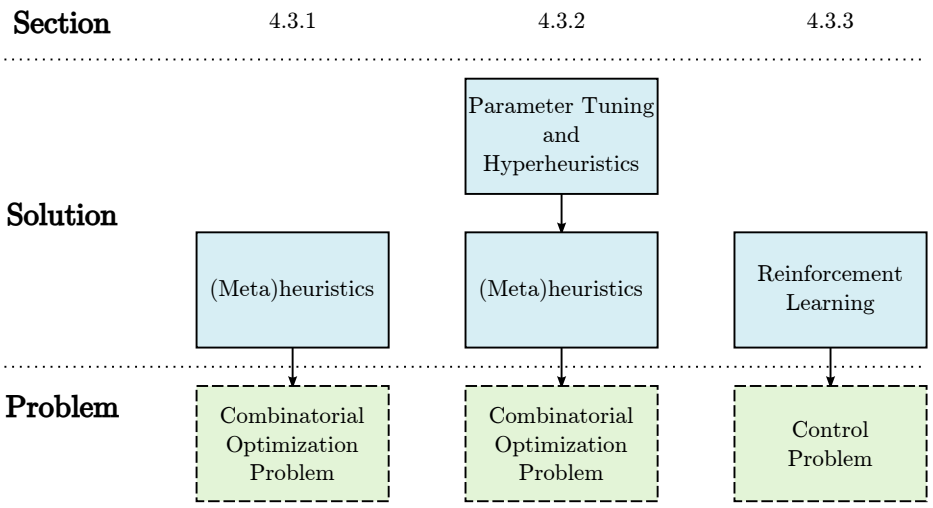


Figure 4.1: A Schematic Diagram of Different Learning Approaches

(Meta)Heuristics and higher-level learning generally do not share a different perspective on the ATS problem. Their perspective is the formulation of ATS as a problem of combinatorial optimization. (Meta)Heuristics are general purpose suboptimal solutions to decision making on very complex questions. ML algorithms often rely on such solutions. Specifically GAs [Gol89] is such a universal ML method next to non-ML methods like Simulated Annealing. What they can learn is how to find "good" non-optimal solutions. The search would have to take place whenever the inputs to the `ats` service change and result in a new allocation. The potential advantage over existing solutions is a reduced computational cost compared to the MLP solver and an improvement in allocation quality compared to the proportional solver. Additionally, the search can gradually improve its search technique, since the ML based mechanisms resort to a model formed during their search.

Higher-level approaches would, of course, benefit from the same advantages as the underlying (Meta)Heuristics. What can be learned are parameter configurations or combinations of those heuristics. On the one hand, it is possible using recorded performance data and statistics collected from GNUnet in order to generate a solution and then statically embed the result in the ATS service afterwards. On the other hand, collecting such data live and altering the way the service works is also possible during operation. However, apart from optimizing the service's strategy, this level of learning does not improve its capabilities.

When solving ATS on the background of combinatorial optimization, the quality of the result is bounded by the formulation of the objective function. By the means of MLP, however, a solution method has already been implemented. It is provably optimal regarding the solution quality, while already achieving reasonable execution times.

The RL framework developed in the ML domain, on the other hand, offers a different approach. In GNUnet's `ats` service an agent can be embedded, which manipulates the bandwidth allocation. Due to the reward signal it receives, it can learn how to manage the available resources effectively. The reward signal could be the same as the result of an objective function, but it can also be based on performance metrics of the system. A repeated measurement of these, thus, changes the perspective onto the problem more towards one of control. Theoretically, ATS, as a control problem, does not necessarily need to be solved using ML techniques. The literature of control theory offers a wide range of methodologies. A rule-based mechanism, such as fuzzy logic is thinkable to fulfill this role as well. The additional advantage of ML here is that it offers mechanisms to learn the rules for control by itself.

The potential advantages derive from the formulation of the reward signal. Based on live system performance metrics it might be easier to formulate than an objective function for classical optimization. Moreover, the self-optimizing characteristic of the approach suggests to autonomously find the correct decisions. The theory offers answers to the question of how a change in the resource allocation affects the performance of the system. It is potentially powerful enough to avoid unwanted behaviors such as oscillating allocations. Additionally, it can learn to assign more bandwidth than is available, as long as the actual utilization is kept below the given quotas.

The disadvantage is the complexity of the learning framework itself. It has to be well-adapted to the problem domain to be effective. Only very-small scale problems can be handled with provable convergence of the learning algorithm. With a growing complexity of the problem more and more trade-offs regarding the memory and computational effort have to be made in order to keep them reasonably low. They affect the speed, the general convergence, and the boundaries of what can be learned. Furthermore, the inherent different approach needs to be designed in a way for it to be reactive enough, so that a new input is processed fast enough.

Taking these considerations into account, it is more beneficial to explore the problems and boundaries of a direct RL solution, while aiming for its potential benefits. Building upon the view of combinatorial optimization, ML methods may be applied as well, but seem less “interesting” from a research oriented point of view. From an implementational point of view it is also less worthwhile, as there is already the MLP approach in existence.

5. Solving Automatic Transport Selection using Reinforcement Learning

This chapter describes how the Reinforcement Learning problem is defined in the literature and comprises both the foundations of how it is solved theoretically and how the practical algorithms SARSA(λ) and Q(λ) are developed from the theoretical approaches.

The second section concerns the necessary adaptations when applying it to the ATS problem. It explains how the RL framework is embedded into the ATS question by dividing the problem structure and how further mathematical means are used to fit the specific requirements.

Most of the information in the first two sections of this chapter stems from Sutton and Barto's absolute standard book [SB98], if not stated otherwise the formulas in this Chapter are taken or directly derived from this source. Another not as exhaustive but often cited source is the survey by Kaelbling et al. [KLM96], although it does not draw a complete different picture. Another self-contained survey is provided by Keerthi and Ravindran [KR94].

5.1 The Reinforcement Learning Problem

In RL there is an actor performing actions. In the abstraction the actor is called *agent*. What the agent influences with its actions is called the *environment*. While performing actions, the agent is trained to take on a behavior, which determines the actions it performs in the future. The actions depend on the situation the environment is in, which is referred to as its *state*. The training is done by giving the agent a numerical feedback called *reward*. This results in a closed feedback loop depicted in Figure 5.1.

The model discretizes this learning process in discrete time steps $t = 1, 2, 3 \ldots$. At a time t the agent is provided with a *state* s_t. Based on this, it chooses an *action* a_t. At the next point in time $t + 1$, along with s_{t+1}, the agent additionally receives a positive or negative *reward* r_{t+1} for its last action a_t. Depending on s_{t+1} the agent chooses a_{t+1} and so on.

RL tasks can be distinguished into *episodic* and *continuing* tasks. Episodic tasks come to a final state: Everything is afterwards reset to a starting condition again except for the agent's experience. Continuing tasks opposing to that do not have a natural end and in principle go on until eternity.

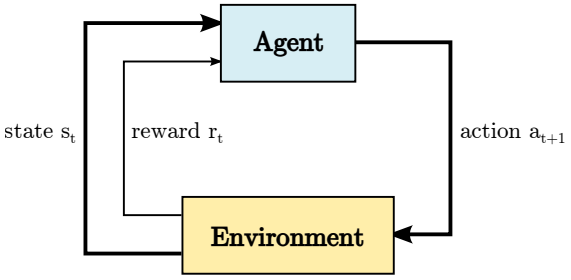


Figure 5.1: The General Reinforcement Learning Framework

The goal for the agent is to find the best way to "behave" doing so by maximizing the rewards it gets. This is expressed in formulas, which assume knowledge about the future and are therefore acausal. Later on the future is predicted using the experience in the past.

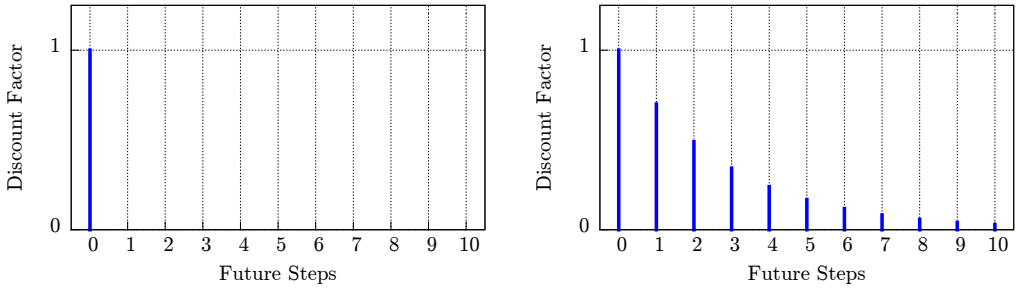
It's behavior is expressed by its *policy* $\pi(s, a)$ defining the probability to take action a in state s. This policy depends on whether the agent's more specific target is to maximize the short-term reward of the next step or the long-term reward of the next T steps. This is called the *return* R_t, defined in Equation 5.1

$$R_t = \sum_{k=0}^{T} r_{t+k+1} \tag{5.1}$$

Potentially, a reward received now might be less valuable in the long-term as the action that led to it might have prohibited the agent from receiving a greater reward in a later step. Additionally, in the case of continuing tasks, it is not possible to maximize the return for an infinite amount of time-steps, as then the return R_t would always approach infinity. As a compromise, the literature uses the concept of *discounted future rewards*, where $\gamma \in [0, 1[$ is a discount rate:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{5.2}$$

When $\gamma = 0$, the discounted return amounts exactly to the one-step reward. The closer γ is to 1, the closer the return is accounting for rewards in the far future. Figure 5.2 shows the difference between those two cases at the examples of $\gamma = 0$ and $\gamma = 0.7$. This goal definition grants the algorithms solving the reinforcement learning problem their ability to deal with delayed rewards.



(a) $\gamma = 0$ (b) $\gamma = 0.7$

Figure 5.2: The Future Discounted Return for Different γ Values

In summary, the reinforcement learning task consists of finding the policy $\pi(s, a)$, which maximizes the discounted return R_t. Only for the special case of $\gamma = 0$ the agent will seek to maximize the immediate reward. For the rest of the thesis the term *return* refers to the discounted future reward defined in Equation 5.2 and *reward* to the feedback the agent receives in one step.

5.2 Markov Decision Processes

For a proper understanding of almost all RL methods the formalism of the Markov Decision Process (MDP) is of vital importance. It serves as a foundation of the later explained algorithms and has allowed to formally prove the convergence of the learning process to the optimal policy under the correct assumptions. Furthermore, it inherently deals with the problem of delayed rewards.

An MDP is defined as a tuple $(S, A, \mathscr{P}^a_{ss'}, \mathscr{R}^a_{ss'})$ consisting of the following elements:

- A finite set of states S
- A finite set of actions A
- A transition probability function defining the probability of getting from state s to state s' by taking action a

$$\mathscr{P}^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$$

- An expected reward function defining the expected reward value for the transition from state s to state s' taking action a

$$\mathscr{R}^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$$

The model is named after the so-called *Markov* property, which implies that the state signal s, which is presented towards the agent, is independent from any preceding states. A system, which is markovian is also called memoryless, because its future only depends on its current status and not how it got there. It is possible to model non-markovian systems, which do not hold this property, with an MDP. There the whole trace of states, which were visited so far have to be included in the state signal s.

Episodic tasks, which come to an end give the MDP a final state, from which it cannot leave anymore and where the agent receives a reward of 0. The MDP has to be started anew. Continuing tasks on the other hand can be perceived as a task with one never-ending episode.

Based on the notion of the discounted future reward, it is possible to estimate the *state-value* of a certain state s as the expected value E of the return achieved starting from s. Since the future rewards also depend on the policy π followed by the agent, the *state-value function for* π is defined as:

$$V^\pi(s) = E_\pi\{R_t | s_t = s\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s \right\} \tag{5.3}$$

Similarly, the *action-value function for* π is defined for the expected return if the agent takes action a in state s:

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a \right\} \tag{5.4}$$

As a very important property, value functions have a recursive relationship regarding the value of state s, $V(s)$ and the value of its successor state s', $V(s')$. Thanks to this, it is possible to rewrite Equation 5.3 as:

$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathscr{P}^{a}_{ss'} \left[\mathscr{R}^{a}_{ss'} + \gamma V^{\pi}(s') \right] \tag{5.5}$$

Equation 5.5 is called the *Bellman equation for* V^{π}.

5.2.1 Solving a Markov Decision Process

Solving an MDP and therefore the RL task is to find the best policy, which maximizes the return as defined previously. Value functions are beneficial in this process, as they define a partial ordering over policies. A policy π is better or equal to another policy π', if and only if in all states its expected return is better or equal to that of all other policies: $\forall s \in S$: $V^{\pi}(s) \geq V^{\pi'}(s)$. The optimal policy is called π^* and is not necessarily unique. Optimal policies have both an optimal state-value function $V^*(s)$ as well as an optimal action-value function $Q^*(s,a)$, which are defined as Equation 5.6 and 5.7 respectively.

$$V^*(s) = \max_{\pi} V^{\pi}(s) \tag{5.6}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{5.7}$$

The optimal state-value function can also be interpreted in relation to the optimal action-value function. The reasoning is that only the selection of the optimal action yields the optimal return. This results in the following equation, where $A(s)$ determines the set of actions, which is possible to take in state s:

$$V^*(s) = \max_{a \in A(s)} Q^*(s,a) \ , \tag{5.8}$$

The optimal policy π^* should be the one, which chooses in each state s the action with the highest action-value:

$$\pi^*(s) = \arg\max_{a \in A(s)} Q^*(s,a) \tag{5.9}$$

This term still depends on maximizing for the best policy, since it contains Equation 5.7. It is not realistic to search in the space of policies for the optimal policy π^*. Luckily, similar to non-optimal policies exists a Bellman equation for $V^*(s)$ and similarly for $Q^*(s,a)$ not referring to any specific policy. Those equations are called the *Bellman optimality equations*. For the action-value function the equation is:

$$Q^*(s,a) = \sum_{s'} \mathscr{P}^{a}_{ss'} \left[\mathscr{R}^{a}_{ss'} + \gamma \max_{a'} Q^*(s',a') \right] \tag{5.10}$$

As a result, in a finite MDP, i.e. one with a finite state space S and a finite action space A, it is possible to calculate all three: the optimal state-value function $V^*(s)$, the optimal action-value function $Q^*(s,a)$, and also the optimal policy π^*. The only precondition is the knowledge of the dynamics of the environment modeled by the MDP. That means $\mathscr{R}^{a}_{ss'}$ and $\mathscr{P}^{a}_{ss'}$ have to be known.

5.2.2 Theoretic Solution Methods

This section shows two theoretical approaches to finding the optimal policy for a finite MDP. They are no candidates for a real implementation, in order to solve ATS, because they do not account for the boundedness of computing resources or assume knowledge about the MDP, which is not available in reality. Yet, they play an important role for the understanding of the practical RL algorithms described in the consecutive section.

5.2.2.1 Dynamic Programming

One way to solve the RL problem is to use the means of Dynamic Programming (DP). Here complex problems are simplified by dividing them into subproblems, which can be combined to form an optimal solution. The basic idea is to take advantage of the recursive consistency of the value functions constituted by the Bellman equations. They are a necessity for Bellman's principle of optimality. This allows two iterative solutions methods:

Value Iteration A first approach is to take an arbitrary state-value function $V(s)$ and repeatedly loop through all states $s \in S$ and improve the value function in each state to a better one by performing the following update:

$$V(s) \leftarrow \max_a \sum_{s'} \mathscr{P}^a_{ss'} \left[\mathscr{R}^a_{ss'} + \gamma V(s')\right]$$

Note that this update simply yields from putting Equation 5.8 in 5.10 for the next state s' in the sum and putting the result again in Equation 5.8 for the current state. Eventually, the computation converges towards the optimal state-value function. The optimal policy can then be derived from Equation 5.9.

Policy Iteration This starts from a random policy π and uses its Bellman Equation 5.5 to calculate the state-value $V^\pi(s)$ in each state. This is a calculation which converges in infinity and has to be aborted when the approximation is accurate enough. Afterwards the resulting value function is used again to derive a better policy π' by changing π in every state to the action yielding the best value. If the policy does not change, the optimal policy has been found.

These algorithms presume the availability of a model, which gives an approximation for the expected reward function $\mathscr{R}^a_{ss'}$ and the transition probability function $\mathscr{P}^a_{ss'}$. In real RL problems, these are not available, though, but have to be acquired.

5.2.2.2 Monte Carlo Algorithms

A second theoretic approach is the usage of Monte Carlo methods (MCs). An advantage of them is that they do not require complete knowledge or a model of the environment. Their approach is the very simple one to just sample the return for each action in each state. For episodic tasks, in order to obtain a correct sample of an action-value it is necessary to record the rewards until the end of an episode. For continuing tasks these methods are not well-defined, because there the episode does not end. One episode is generated using a certain policy π. At the end of the episode the average action-value for each state-action pair is used to generate a new policy, the next episode is started with. This process can be repeated until there is enough statistical confidence that the best policy is approximated. For this approach to work, at the start of each episode a random state and action have to be chosen or it has to be ensured that each state-action pair is eventually visited infinitely often. In the case of non-random starts, for a guaranteed convergence to the optimal policy π^* a minimum exploration of the state-action space has to be performed.

MCs do not need to know $\mathscr{R}^a_{ss'}$ or $\mathscr{P}^a_{ss'}$. They become unnecessary through the direct sampling of the action-value. This comes at the cost of an increased memory consumption, by the recording of all the samples, as well as a runtime approaching infinity with an increasing probability of finding the optimal policy.

5.2.3 Temporal Difference Learning Algorithms

Temporal Difference (TD) learning algorithms lie at the heart of the approach of this thesis and they are what is called in Sutton's book the central and novel idea of RL. They do not need a model of the environment, since they learn from sample experience, similar to MCs. Additionally, they improve their value functions on the basis of already existing estimates of their value functions as in DP. To some extent, they are a combination of the two theoretical approaches, and have proven to work well in practice, especially on small problems.

TD algorithms maintain an estimation of the action-value function, deriving the estimated best policy by choosing the action with the highest estimated return. The estimations get more accurate with the number of steps they perform. They perform an incremental update of the estimation per-step. The memory consumption for the estimation stays constant.

During the steps, the agent has to fulfill two tasks. First, the value-prediction of the current policy needs to become more accurate, in terms of generating enough samples. Second, the policy needs to be improved, so the return is maximized. As a consequence TD algorithms need to diverge from the action deemed best in some steps, in order to find better policies. A step, in which the next action taken is the one with the highest value, is called an *exploitation* step, while a step, which tries another action is called an *exploration* step. The agent explores the space of policies.

The more an agent explores, the higher is the possibility for it to find the optimal policy. However, if an agent does not exploit it does not maximize its return. There is a trade-off between trying out the unknown and taking advantage of the already gained knowledge. The right balance between exploration and exploitation is solved by heuristics, of which two are presented later on.

As long as there is a non-zero probability for a state-action pair to be chosen, the agent's estimation of the action-value function will eventually converge towards the one for the optimal policy.

5.2.3.1 SARSA

One of the two most popular model-free temporal difference learning algorithms is the so-called *SARSA* algorithm, named after the one-step learning tuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$.

When a new learning tuple is determined, the old estimation of the action-value in the previous state $Q(s_t, a_t)$ is updated with a new one $\hat{Q}(s_t, a_t)$:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) \cdot Q(s_t, a_t) + \alpha \cdot \hat{Q}(s_t, a_t)$$

The new estimation influences the updated value with a weight of $\alpha \in [0, 1]$. This weight is called the learning rate. A lower value of alpha leads to a longer learning period but a better stochastic sampling, a higher value shortens the learning phase, but leads to a less accurate estimation of $Q(s, a)$.

For the new estimation SARSA uses $\hat{Q}(s_t, a_t) = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$. Here r_{t+1} is the reward received for the last action. Therefore, $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$ is an estimation of

the future discounted return for the last action. $Q(s_{t+1}, a_{t+1})$ is taken from SARSA's past experience.

Most often in the literature the update is written in the following form:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

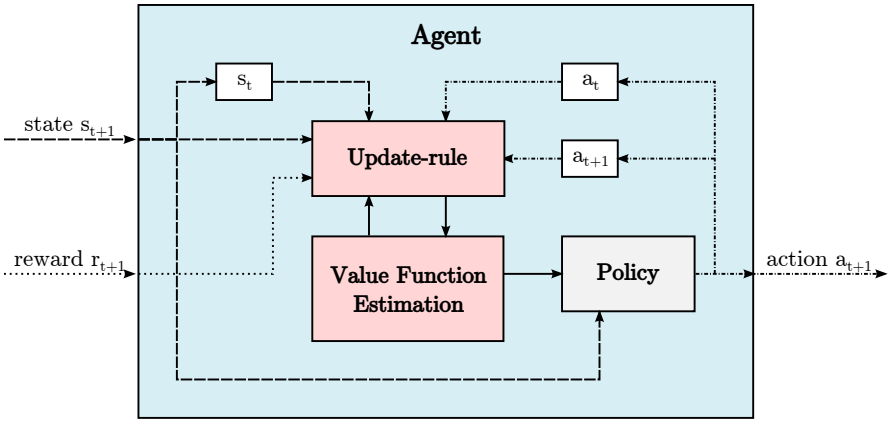


Figure 5.3: Information Flow - On-Policy Update in the SARSA Learning Algorithm

Figure 5.3 shows the information flow in an agent, which implements the SARSA algorithm to derive the next action from the received reward and state signal. There and from the update rule itself can be seen, that it is only necessary to memorize the last state-action pair and do the update after the current policy has derived the next action. This means, the algorithm first chooses the next action a_{t+1} according to its current policy. Then it updates the action-value for the previous state-action pair (s_t, a_t) using the received reward r_{t+1} and its currently saved estimation for the future discounted return from the next state-action pair (s_{t+1}, a_{t+1}). Then the action a_{t+1} is taken to influence the environment.

Because the update rule performs the update, using the discounted action-value of the next state-action pair, the SARSA algorithm is said to be an *on-policy* algorithm.

5.2.3.2 Q-learning

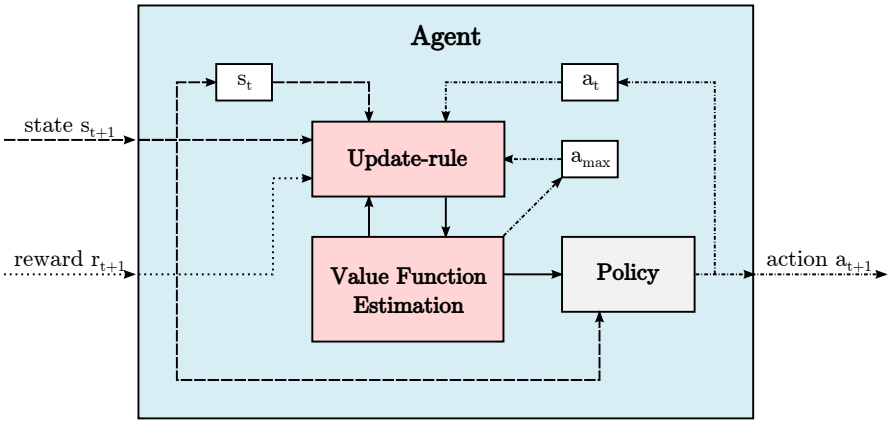


Figure 5.4: Information Flow - Off-Policy Update in the Q-Learning Algorithm

A different approach is taken by the so-called *Q-learning* algorithm, which is on the first view very similar, since its update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

The difference to the SARSA algorithm is that its update rule does not depend on the action it derives from its current policy. Instead, it uses $r_{t+1}+\gamma \max_a Q(s_{t+1}, a)$ as the estimator $\hat{Q}(s, a)$. This means the Q-learning algorithm searches for the optimal future action a in the current state s_{t+1} and uses its currently estimated state-action-value $Q(s_{t+1}, a)$ in the update of $Q(s_t, a_t)$, independent of whether or not this action a is taken afterwards.

As can also be seen in the information flow of a Q-learning step in Figure 5.4, therefore the update of $Q(s, a)$ does not improve the action-value estimation based on the policy it follows, but on the best policy the agent currently knows. As a consequence, as opposed to SARSA, Q-learning first performs its update and then chooses its next action.

The update rule performs the update, using the discounted action-value of the action currently estimated to be optimal, although it is not taken in exploratory steps. Therefore, the Q-learning algorithm is said to be an *off-policy* algorithm. This serves to speed up convergence and therefore increases the effectiveness per learning-step.

Due to the similarity of the SARSA and Q-algorithm it is a minor implementational cost to make use of both of them for the purpose of comparison. SARSA has the advantage of estimating the values of the policy, which is actually being followed. Q-learning on the other hand has a better convergence.

5.2.4 Eligibility Traces

The just mentioned TD algorithms perform a *one-step* update. Upon a step, they change the estimated value of the last state-action pair. MC methods, however, update not only the last one, but all of the already seen state-actions pairs. This idea can be used to enhance the mentioned algorithms in their learning efficiency. Sutton and Barto [SB98] therefore describe the method of *eligibility traces*.

It is based on the idea of an *n-step* update, which performs an update on the last n state-action pairs in the past. The bigger the number n is, the closer the method becomes to an MC method. Actually, in an episodic task, this would exactly yield an MC method, if n is bigger than the number of steps taken in an episode. Unfortunately, this is not always practical and it is difficult to argue how n is chosen. It has been found, though, that it is also possible to perform the update as a weighted average of many n-step returns, while maintaining the algorithm's convergence properties.

To be able to update states in the past, the definition of the return is extended as the sum of weighted n-step returns. Similar to the discounted future return, the closer a step is, the more weight it gets. This results in the future λ-*return* defined as follows:

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} \tag{5.11}$$

$R_t^{(n)}$ describes the future n-step return and $\lambda \in [0, 1]$ is a weight parameter. Each n-step return accordingly has a weight of $\lambda^{(n-1)}$. The sum is then normalized by a factor of $1-\lambda$ so that it results in 1. Instead of choosing n directly this definition allows a more general control. A λ value of 0 corresponds to the one-step return, while a λ value of 1 corresponds to an ∞-step return.

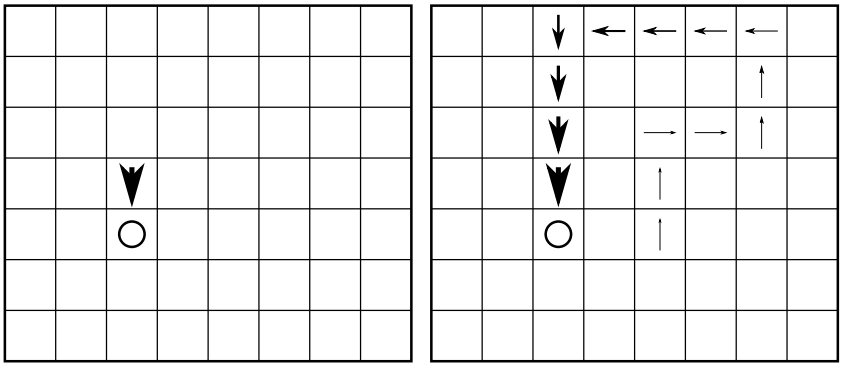
Of course, the future λ-return is not known at time step t. It has to be approximated for the past state-action pairs, which can then be updated accordingly. For this purpose, for each pair an *eligibility value* is memorized as in the following equation:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \wedge a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases} \tag{5.12}$$

For each past action, a decreasing value is maintained, which denotes how eligible or responsible it was for the current reinforcement. The sequence of the visited state-action pairs together with the eligibility values is called the *eligibility trace*. How far the updates reach back depends on λ, which is also called the *trace-decay* parameter. Applying these traces to the SARSA algorithm is done by performing the algorithm's update in every time step for every state-action pair, multiplying the new Q-value with the pair's eligibility:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)\right] e_t(s,a)$$

This approach is known as the SARSA(λ) algorithm. The one-step version of SARSA is still contained in the algorithm, as $\lambda = 0$ yields the same behavior.



(a) $\lambda = 0$ (b) $\lambda = 0.8$

Figure 5.5: Eligibility Traces

Figure 5.5 shows how the two algorithms compare similar to the illustration in [SB98]. The states of a task are denoted by a square, while the eligibility values of the actions are denoted by arrows. The thicker an arrow is, the more does the reward have an impact on its update. Since now much more state-action pairs are updated than in the one-step version, the learning speed benefits significantly. The problem of the increased computational complexity is solved by not updating all already visited state-action pairs. Instead only those, with an eligibility value above a certain threshold are updated. Since the eligibility value decreases exponentially the amount remains reasonable.

The application of eligibility traces to Q-learning is similar. The Q-learning algorithm learns an action-value function different from the policy it actually follows. It learns the action-value function without the exploratory steps it takes. Therefore, it cannot hold actions responsible which did not follow this policy. To account for this, Watkins proposed to reset the eligibility trace to zero whenever an exploratory step is taken. This version of the algorithm is therefore called Watkins' Q(λ). Otherwise, the eligibility traces are applied in the same manner as in SARSA(λ):

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right] e_t(s,a)$$

Eligibility traces as defined before are called *accumulating* traces. For state-action pairs, which are seen frequently, their eligibility can result in a value greater than 1. This means

an overestimation of the eligibility. Therefore, according to Sutton [SB98] an improved performance can be achieved by avoiding this behavior by using *replacing* traces instead. Whenever a state-action pair is seen, then 1 is not added to its eligibility, but it is just reset to 1. The update of replacing eligibility values result in Equation 5.13. This form of eligibility traces is used throughout the rest of this thesis.

$$e_t(s,a) = \begin{cases} 1 & \text{if } s = s_t \wedge a = a_t \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} \end{cases} \tag{5.13}$$

5.2.5 Exploration-Exploitation Balancing Heuristics

In Section 5.2.3 it was already mentioned that RL algorithms have to do both exploratory as well as exploitative steps. Exploratory ones in order to find out about the best way to behave, and exploitative ones in order to actually behave the best way. Since the agent wants to maximize its reward, it cannot avoid to do both in a balanced way. This is called the *action-selection* strategy.

Practical RL applications use some more or less sophisticated heuristics for this task, out of which two popular ones are presented here: *ϵ-greedy* and *softmax*.

5.2.5.1 ϵ-Greedy

The approach is to use the best known action so far. Thus, it chooses the "greedy" action in all of its exploitative steps, i.e. the action with the highest action-value $Q(s,a)$. With a small probability of $\epsilon \in [0,1]$ it performs an exploratory step instead. Then it selects an action at random.

Its disadvantage is that it never stops exploring, even though it might have learned the optimal policy already. Furthermore situations could appear in which the learner gets stuck in some local optimum of policies, if ϵ is very small and it is unlikely for the agent to perform many exploratory steps consecutively.

An idea to mitigate this behavior is to gradually decrease ϵ over time, in order to shift the balance more and more towards exploitation. This works only for completely stationary problems, because as a consequence the learner would eventually not be able to adapt to the environment anymore.

5.2.5.2 Softmax

Softmax action-selection strategies introduce a probability to choose action a_t given the state s_t depending on the current value estimates, such as:

$$P(a|s_t) = \frac{e^{Q_t(s_t,a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(s_t,b)/\tau}} \tag{5.14}$$

As a consequence the learner avoids to take actions he already knows to lead to a bad return. The action with the best value will still be chosen with the highest probability. It is defined in Equation 5.14 and is derived from the Boltzmann distribution, which is why this particular strategy is also called Boltzmann selection. Coming from the area of thermodynamics, the parameter $\tau > 0$ is called the *temperature*, and has some similarities to the ϵ value in the previous section. A bigger action-value yields a higher probability that the action will be taken. The bigger τ is, the more the action-values will be ignored and the probability to select an action is distributed more equally. Every action-selection, not corresponding to the greedy action is considered an exploratory step. The more knowledge

the agent has gained, the more exploitative its actions will be. The temperature value is a bit less intuitive to set, though, as is the case with ϵ, because its size depends on the action values that will occur. Naturally, it is also possible to decrease the temperature over the course of time, in order to reach a solely exploitative policy eventually.

5.3 Adaption to the Problem Domain

The algorithms SARSA(λ) and Q(λ), which were explained in the previous section, are two of the most popular and widely used to solve the reinforcement learning problem. In order to apply them to the ATS problem, an agent-environment interface has to be defined, which is explained in the next section. The difficulties and challenges arising are discussed afterwards. The actual multi-agent based design is then specified and mathematical adaptations and tools for a successful application are explained thereafter.

5.3.1 The Agent-Environment Interface

In order to map the RL framework to a problem, the agent-environment interface has to be defined. It is located where the absolute control of the agent ends.

For the ATS problem the agent has to answer the questions of which peer a connection should be made to, which one of its addresses should be used, and how much bandwidth for it should be allocated. Thus, the actions available to an agent need to provide answers to these. More precisely, the actions have to determine the outputs of the ATS problem: The set of active addresses among the available ones $\hat{a} \in \hat{A} \subseteq A_{p,m}$, as well as the allocated bandwidth $b_{\hat{a},d}$ for address $\hat{a}$ and direction d.

The environment determines the state and the reward signal the agent's decisions are based upon.

Potentially all of the inputs and outputs of the ATS problem are candidates to be included in the state signal. It has to be taken care to not include more information than necessary, because the size of the state-space has an impact on the complexity of the learning problem. The minimum information, which should be conveyed to the learning agent, is what is necessary to distinguish different resource allocations.

Since the reward consists solely of one real number, a function has to be defined, which maps all relevant considerations to one value. This can include every measure, which is of interest to meet the objectives and constraints of the ATS problem described in Chapter 3. The reward function teaches the agent the desired behavior. Therefore, desired actions, such as ones pleasing the applications, must entail a positive impact on the reward, undesired actions, such as ones leading to an overutilization, must entail a negative impact on the reward.

The last issue to address is when an agent actually performs an action. The ATS problem does not have natural cycles, which enforce this. Learning/action steps can be performed at more or less arbitrary times. Furthermore, solving the ATS problem by reinforcement in this thesis is not considered as an episodic problem. Steps are performed with a dynamic frequency, controlling and optimizing the resource allocation.

5.3.2 Challenges

In their standard form, as described before, the TD algorithms save the current estimation of the action-function $Q(s, a)$ as a simple look-up table. This turns out to be rather disadvantageous, looking at the ATS problem. Potentially, there are a lot of states, the system can be in. The agent's view of the system could comprise of all the inputs, mentioned in Chapter 3, namely the amount of bandwidth allocated outbound and inbound

per peer, the properties of each of the addresses of a peer, the preferences of each of the applications, the current quota for each of the scopes, and recent applications' feedbacks about the resource allocation. Taking all of these into account will result in a very high dimensional and continuous state-space. While the state-space could be discretized, it will still be very big, which leads to two severe problems for standard SARSA(λ) and Q(λ). Since they save their experience in a look-up table, they only learn for a particular action, taken in a particular state. First of all, this is not practical, as it would simply consume a vast amount of memory. But furthermore, it would slow down the learning process unnecessarily. The algorithms only make an informed decision for precise state-action pairs, they have already seen. With a high-dimensional state and action space and a limited amount of computation it is unlikely for the agent to have traversed a significant area of the space of possibilities.

This problem is treated for example in the work by van Hasselt [vH12], but also Sutton and Barto provide some ideas to overcome these problems in their book [SB98]. The obvious method is to generalize over the MDP. Effectively, an agent should not only learn from a reinforcement for a particular state, but to some extent should also draw conclusions for the surrounding states, which have similar characteristics. For the two algorithms chosen for this thesis, this means that instead of saving the action-value function $Q(s, a)$ in a simple look-up table, they are saved in a parameterized form, where the parameters have to be updated in each time-step.

This approach is further explained in the later Section 5.3.4. It is possible to do this both for the set of possible states, as well as for the set of possible actions to take. For tackling the ATS problem, this thesis does not do the latter, though, for two reasons.

First, in a given time-step the state of the system is given. The task of the agent is to perform a search on all the available actions, and (at least when performing an exploitation step) choosing the one which promises the most return. For a large or continuous action-space, this search may be quite exhaustive. Therefore, this is still a domain of ongoing research in the field of reinforcement learning. At the time of the writing of this thesis it, thus, seems to be reasonable to experiment using the more well-studied and successfully applied methods with a small and discrete action space. The usage of an algorithm designed for large or continuous action-spaces such as van Hasselt's and Wiering's [vHW07] Continuous Actor Critic Learning Automaton (CACLA) might be a worthwhile investigation, though.

Second, inspired by different works mentioned in Section 4.3.3 solution is to break down the action-space and have it handled by multiple-agents. The actions of a single agent managing everything would have to comprise possibilities for setting an amount of bandwidth in two directions for each of the peers a connection is requested for and choosing the according address to use. Instead, the decision was made, to incorporate one agent, which allocates the bandwidth for each of these peers. Therefore remains a small set of possible actions, which would mainly be, to switch to a particular address of that peer, or to increase or decrease the allocated bandwidth for the address currently proposed. This divides the ATS problem in a natural manner. Unfortunately, it also introduces some difficulties. This approach and its consequences are further discussed in the next Section.

5.3.3 Multiple Agents

The dynamics of reinforcement learning with multiple agents is an active field of research of which Busoniu et al. give an overview in [BBDS08]. Formally multi-agent learning extends the MDP to a Stochastic Game (SG), which is similarly described as a tuple $(S, A_1, \ldots, A_n, \mathscr{P}^{\vec{a}}_{ss'}, \mathscr{R}^{\vec{a}}_{ss',1}, \ldots, \mathscr{R}^{\vec{a}}_{ss',n})$ with the following elements:

- A number of participating agents n
- A finite set of environment states S
- A finite set of actions A_i for each agent
- A transition probability function defining the probability of getting from state s to state s' by taking the joint action $\vec{a}$, which results from the combination of each agent's action $a_i \in A_i$:

$$\mathscr{P}^{\vec{a}}_{ss'} = Pr\{s_{t+1} = s' | s_t = s, \vec{a}_t = \vec{a}\},$$

- An expected reward function for each agent i defining the expected reward value for the transition from state s to state s' taking joint action $\vec{a}$:

$$\mathscr{R}^{\vec{a}}_{ss',i} = E\{r_{t+1,i} | s_t = s, \vec{a}_t = \vec{a}, s_{t+1} = s'\},$$

Notably, the definition comprises a regular MDP as an SG with just one agent. However, it increases the complexity of the learning problem.

Multi-agent settings are necessarily non-stationary, because the actions of the other agents effectively change the environment in each step. For instance, an agent, who allocates more bandwidth for his peer, may either get a positive reward for increasing his utility, but might as well get a negative reward for exceeding the bandwidth limit. Both outcomes might happen depending on whether another agent similarly allocates more bandwidth or not.

Would be all of the agents but one stationary, the remaining one can be expected to converge to a policy, where he can max out his utility. With the number of other agents this becomes increasingly difficult and needs more exploration. The convergence of independent learners in a multi-agent environment has been observed first by Sen et. al. [SSSH94]. Efforts are made in the field of multi-agent reinforcement learning, to improve the learning process by introducing forms of coordination or communication between agents. The success of these efforts depends much on the properties and assumptions which can be made on the SG and e.g. allow to make use of game theoretical insights. Especially for dynamic tasks it is unclear how to incorporate useful coordination mechanisms in the multi-agent setting. A study conducted by Schaerf et al. [SST95] describes a similar setting to our idea of mapping agents to the clients of resources in a load-balancing application. The study has found that the use of communication between agents in their setting even has a negative impact on the efficiency of the system. On this background the decision was made not to introduce communication between learning agents about their state spaces for solving the ATS problem. Depending on future findings in the field of multi-agent reinforcement learning in this direction there might be potential to improve the efficiency of the approach.

5.3.3.1 Action- and State-Space

Each learning agent is responsible for the resource provisioning for the connection to one peer. Using the reinforcement learning framework it tries to maximize the utility, which can be achieved given the application preferences and feedbacks, address properties, and available bandwidth. The action-space is kept small, enabling the agent to:

SWITCH_A	Switch to address number a of peer p
INC_D_X	Increase the amount of provisioned bandwidth for the currently suggested address in direction d by x kilobytes/s
DEC_D_X	Decrease the amount of provisioned bandwidth for the currently suggested address in direction d by x kilobytes/s
NOP	Do nothing and leave the allocation unchanged

This results in $5 + n$ different actions for a peer with n addresses. Given a peer might have 50 addresses in all of its transport mechanisms combined, this would result in 55 different actions. A small action-set like this defines along which dimensions an agent can manipulate the allocation for its respective peer. Defining the state-space accordingly results in three dimensions:

1. The chosen address
2. The amount of bandwidth allocated outbound
3. The amount of bandwidth allocated inbound

Typically for ML methods, also RL suffers from the curse of dimensionality. With the number of dimensions of the action- and state-space increases the computational complexity of the problem. The presented state-space definition is as big as necessary to distinguish the agents' control decisions. Thus, the computational complexity is kept as small as possible.

The question of whether a peer should be talked to or not is answered by whether a peer gets bandwidth allocated or not. Decreasing both the inbound and outbound bandwidth to a value of zero leads to the disconnection from the respective peer.

5.3.3.2 Reward Function

A reinforcement learning agent maximizes its return, which derives from the short- or sometimes also long-term reward. Therefore, in order to maximize the utility of the connection for each peer, it should get a positive or negative reward, according to whether the utility of the connection was increased or decreased. The utility of a connection could be thought of a numerical value determined by the application, which indicates how well it sees its requirements fulfilled.

Additionally, the reward should be negative, if the agents, which chose an address in one network scope, exceed the quota there. As a result the agents learn for which address and which amounts of bandwidth the utility is best without exceeding the quota.

A relatively important property for the convergence of the allocation, i.e. finding an allocation which is not changed anymore, is its steadiness. An agent should not learn to continuously maximize its return by de- and increasing the allocation. The considerations about the reward do not reward this behavior yet. Choosing to stay with the current allocation leads to no change in the utility, which means a reward of zero. Instead a bonus should be given for choosing to stay with the current allocation, which should always be below what an increase in utility would mean. This way a steady allocation is encouraged, but not more than further increasing the utility when possible.

The last important aspect not considered yet is the fairness of the resulting allocation. There must be an incentive for the agents to not allocate all of the bandwidth in a scope for their own connection, but also leave some for the others to take. Separate from the problem of learning by reinforcement, the research field of multi-agent resource allocation knows this as a problem of social welfare. Chevaleyre at al. give in a summary paper [CDE+06] a comprehensive summary of solutions with roots in welfare economics and social choice theory. Two useful measures of social welfare to optimize for, in order to reduce the inequality of an allocation, are:

The Egalitarian Social Welfare of an allocation P is defined by the minimum of the utilities u of all of the agents i in a set of agents $\mathscr{A}$:

$$\mathrm{sw}_e(P) = \min\{u_i(P) \mid i \in \mathscr{A}\} \tag{5.15}$$

When all resources are already allocated an increase in utility is only possible by giving resources from one agent to another, i.e. decreasing another one's utility. The egalitarian goal enforces an allocation, in which no agent gets an advantage over the others.

The Nash Product is defined by the multiplication of the agents' utilities:

$$\mathrm{sw}_N(P) = \prod_{i \in \mathscr{A}} u_i(P) \tag{5.16}$$

This measure reduces inequality as well, since a low utility for one agent affects the overall social welfare negatively. Yet, it may lead to different results, if for instance an increased share of resources for agent a gives it a significant better utility than the same share would increase the utility of another agent b. The Nash product therefore has a different concept of fairness, which respects the usefulness of the resource for an agent. It is only valid, when all the utilities are positive. Compared to the egalitarian social welfare its numerical value is of course bigger, due to the product operator and the direct dependence of the result on the amount of agents. This has to be accounted for when it is used in a reward function.

The foregoing aspects motivate the reward calculation depicted in Algorithm 1 at time step t for agent i:

In lines 2 to 14 it is checked, whether an overutilization has happened. Overutilization is when in the network scope n, where the agent's currently chosen address is in, the utilized bandwidth b exceeded the quota $r_{n,d}$ or not. This is done for both directions d, inbound and outbound, and yields a *penalty*, which is the negative of the amount of bandwidth overutilized. The penalty is doubled per direction, if the last action of the agent was to increase the bandwidth. Thus, in case of overutilization, the agents get more punished, the more they overutilize.

Otherwise they maximize their *objective*, which is both to increase their own utility value, as well as one of the social welfare measures as taken from above. The social welfare only makes sense to calculate among the addresses which are in the same network scope as well. In case of the Nash product it should be numerically adapted to fit the same value range, e.g. by calculating its $|\hat{A}_n|$-th root. Additionally the agent receives a bonus as an incentive to eventually stay with a configuration. The bonus is decayed over time.

5.3.3.3 Decay of the Exploration Ratio

As written in the description of the reward function calculation, a desirable property of the allocation process is that an allocation does not continuously change its value. While this is the motivation for increasing the value of the **NOP** action, there is another impediment for reaching a steadiness in the allocation: In case of an exploratory step, the agent's decision does not depend on the learned action-values. Therefore, one either has to eliminate exploration altogether or decrease the ratio of exploratory steps over time. Due to the possibility of changed inputs in the environment of the ATS problem, the system is non-stationary. Consequently, there remains the problem of learning the new dynamics after a change in the environment, while having the objective to come to a completely exploitative policy.

In order to account for both sides in this dilemma, the approach of this thesis compromises in the following way: In order to reach a steady allocation, in which the **NOP** action will be in favor, the exploration ratio (or the temperature in the case of the softmax action-selection strategy) can be decayed by a factor < 1 each time an exploratory step is taken.

Algorithm 1 Procedural reward function

function REWARD
  if $\sum_{\hat{a} \in \hat{A}_n} b_{\hat{a},\text{out}} > r_{n,\text{out}}$ **then** ▷ Calculate penalty
    $\text{over\_out}_t \leftarrow r_{n,\text{out}} - \sum_{\hat{a} \in \hat{A}_n} b_{\hat{a},\text{out}}$
    if $a_{t-1} = \text{INC\_OUT\_X}$ **then**
      $\text{over\_out}_t \leftarrow 2 \cdot \text{over\_out}_t$
    end if
  end if
  if $\sum_{\hat{a} \in \hat{A}_n} b_{\hat{a},\text{in}} > r_{n,\text{in}}$ **then**
    $\text{over\_in}_t \leftarrow r_{n,\text{in}} - \sum_{\hat{a} \in \hat{A}_n} b_{\hat{a},\text{in}}$
    if $a_{t-1} = \text{INC\_IN\_X}$ **then**
      $\text{over\_in}_t \leftarrow 2 \cdot \text{over\_in}_t$
    end if
  end if
  $\text{penalty}_t \leftarrow \text{over\_out} + \text{over\_in}$

  $\text{objective}_t \leftarrow \frac{1}{2}(u_i(P) + \textit{social welfare}_n)$ ▷ Calculate change in objective
  $\text{delta}_t \leftarrow \text{objective}_t - \text{objective}_{t-1}$

  if $\text{delta}_t \neq 0$ **and** $\text{penalty}_t = 0$ **then** ▷ Calculate bonus
    $\text{bonus}_t \leftarrow \text{delta}_t \cdot 0.5$
  else
    $\text{bonus}_t \leftarrow \text{bonus}_{t-1} \cdot 0.5$
  end if
  if $a_{t-1} = \text{NOP}$ **then**
    $\text{steady}_t \leftarrow \text{bonus}_t$
  else
    $\text{steady}_t \leftarrow 0$
  end if

  if $\text{penalty}_t \neq 0$ **then**
    return penalty_t
  end if
  return $\text{delta}_t + \text{steady}_t$
end function

This leads to an exponential decrease in exploratory steps and the policy converges to a fully exploitative one. Whenever the inputs to the ATS problem change significantly, though, the exploration ratio is reset to its initial value. This improves the learning agent's chance to learn a new policy.

5.3.4 Function Approximation

As discussed before, for a useful application of SARSA(λ) and Q(λ) on the ATS problem, it is necessary to employ an approximation of the action-value function $Q(s, a)$. The task is to generate meaningful outputs of the function for yet unseen inputs. Therefore, this is a supervised learning task, like it is discussed in the most prominent area of the machine learning domain. However, in RL, special requirements exist for the method of choice: Learning has to be incremental and on-line, since this is how updates of the action-value

are generated. Furthermore, typical supervised learning algorithms are naturally coined to be as precise as possible. In RL a further requirement is, though, to be able to adapt to a change of the function over time. Calculations, which are rough in their estimations, but are faster in their adaption abilities to a change in environment are thus the primary choice.

While more sophisticated methods, such as Support Vector Machines and multi-layered Artificial Neural Networks could be used for this task, they are not entirely corresponding to those requirements. Instead a linear function approximation in connection with gradient descent is chosen, which form the standard approach in RL applications for their mathematical compactness.

Instead of saving $Q(s, a)$ as a lookup-table of the action-values for each particular state, it is realized as a linear function of a parameter vector $\vec{\theta}$. The state is represented by a feature vector $\vec{\phi}_s$, which is extracted from the real state. The selection of the features is a delicate task, which strongly impacts the generalization ability of the approximation. A generalization over different actions is not necessary, since they were cut down to a small discrete set for each agent in the previous section. The action-value estimation maintained by each agent then is the following vector-matrix-vector multiplication:

$$Q(s, a) = \vec{\phi}_s^T \cdot \boldsymbol{W} \cdot \vec{a} \tag{5.17}$$

Matrix $\boldsymbol{W}$ is a simple sequence of parameter vectors $\vec{\theta}_a$, which has as many rows as the feature vector $\vec{\phi}_s$ has components. One such parameter vector is maintained for each action. The vector $\vec{a}$ is an action selection vector, which is zero in all but its a-th component. Assuming the available actions are enumerated, it determines which one of $\boldsymbol{W}$'s columns is to be multiplied with $\vec{\phi}_s$. Ergo, in order to perform an exploitation step an agent needs to multiply $\vec{\phi}_s^T$ with $\boldsymbol{W}$. The number of the action to take results from the index of the maximum component of the result.

After having received the reward for the last step, the agent needs to update the parameter vector $\vec{\theta}_a$ of the last used action. For this purpose Gradient Descent methods are the dominant solution, as they allow a favorable incremental update, which fits very well with the TD-updates.

This can be done using the following correction:

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \frac{1}{2}\alpha \nabla_{\vec{\theta}_t} \left[Q^\pi(s_t, a_t) - Q_t(s_t, a_t)\right]^2 \tag{5.18}$$

$$= \vec{\theta}_t + \alpha \left[Q^\pi(s_t, a_t) - Q_t(s_t, a_t)\right] \nabla_{\vec{\theta}_t} Q_t(s_t, a_t) \tag{5.19}$$

Here $\nabla_{\vec{\theta}_t}$ is the vector of partial derivatives with respect to the parameter vector. It is also called the gradient. It points in the direction where the derived function has the greatest rate of increase. In this case this function is the squared error of the action-value estimation. This means that Equation 5.18 adjusts the parameter vector, by taking a step of the size $\alpha/2$ times the squared error in the opposite direction. Consequently, the error is reduced proportionally to the magnitude of the squared error of the last step.

It is not possible for the agent to know the real value $Q^\pi(s_t, a_t)$ of the true action-value function it approximates. Luckily, it is possible to use the same estimators $\hat{Q}(s_t, a_t)$ of the return as in the case of look-up tables. For SARSA this is $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$ and for Q-learning $r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$.

Additionally, the linear approximation of Equation 5.17 has the mathematical elegancy, that its derivative with respect to the parameter vector $\nabla_{\vec{\theta}_t} Q_t(s_t, a_t)$ results in the vector of state features $\vec{\phi}_s$. From putting this into Equation 5.19 results the new SARSA and Q-learning update rules:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha \left[\hat{Q}(s_t, a_t) - Q_t(s_t, a_t) \right] \vec{\phi}_s \tag{5.20}$$

5.3.4.1 Eligibility Traces with Function Approximation

Also the concept of eligibility traces can be mapped onto the usage of function approximation. Instead of maintaining an eligibility value for each state-action pair, it is possible to maintain an eligibility value for each feature-action pair. This requires the allocation of a second matrix $\boldsymbol{E}$, which has the same size as matrix $\boldsymbol{W}$. It contains one eligibility vector $\vec{e}_i$ corresponding to each parameter vector $\vec{\theta}_i$. Similarly to the accumulation of eligibility on the lookup case, $\vec{e}_i$ can be updated by adding the degree of activation of each feature.

$$\vec{e}_t = \gamma\lambda\vec{e}_{t-1} + \vec{\phi}_s \tag{5.21}$$

Unfortunately, the concept of replacing traces does not fit this idea. It does not make sense to replace the eligibility of a feature-action pair with a new degree of activation, since if it is below the old value it would counter-productively eliminate the eligibility trace. One solution first proposed by Främling [Frä07] to overcome this could be to replace the eligibility value with the new degree of activation only if it is a bigger value. I.e. the i-th component of $\vec{e}_t$ in the present design is updated with:

$$\vec{e}_t(i) = \begin{cases} \gamma\lambda e_{t-1}(i) & \text{if } \gamma\lambda e_{t-1}(i) > \phi_s(i) \\ \phi_s(i) & \text{otherwise} \end{cases} \tag{5.22}$$

5.3.4.2 State-Feature Extraction

The question remains of how the features of the current state of an agent can be extracted. The guidelines given by Sutton and Barto are to choose them "appropriate" or "natural to the task". This task is an intricate one, since it very much influences the generalization and approximation abilities of the result. The feature extraction answers the question of how similar states in the state space are subsumed into one feature in $\vec{\phi}_s$. Saving the action-values $Q(s, a)$ in a lookup table is a special case where every single state has one feature (i.e. a component of $\vec{\phi}_s$) for itself, which is the only *active* one, when the precise state occurs.

Instead a more complex mechanism is necessary to generalize over the state-space, i.e. as already mentioned in Section 5.3.3.1 the settings an agent has control over: The active address as well as the out- and inbound bandwidth.

The concept of a Radial Basis Function (RBF) network as described by Sutton [SB98] distributes certain *prototype features* over the complete state-space. Here, each component of the feature vector $\phi_s(i)$ belongs to one of the prototypes and is determined by its degree of activation, which is a number between 0 and 1. This activation is computed using the distance between the actual state s and the center position of the prototype c_i in the possible state-space:

$$\phi_s(i) = e^{-\frac{\|s - c_i\|^2}{2\sigma_i^2}} \tag{5.23}$$

Equation 5.23 describes a Gaussian activation, where the width of feature i is determined by the parameter σ_i. Technically this is a single layer Artificial Neural Network using RBFs as activation functions. Figure 5.6 demonstrates this principle for the one-dimensional case using four prototypical states. Here a one-dimensional state space which reaches from c_0 to c_3 is approximated. The bell shaped functions show the activation of each prototype for arbitrary values on the x-axis. For x values, which are precisely at a prototype, almost only the prototype is active. x values in-between the prototypes generate a mixed activation of neighboring prototypes. Effectively, for states which do not directly correspond to a prototype the approximation is achieved by an interpolation between those neighboring prototypes.

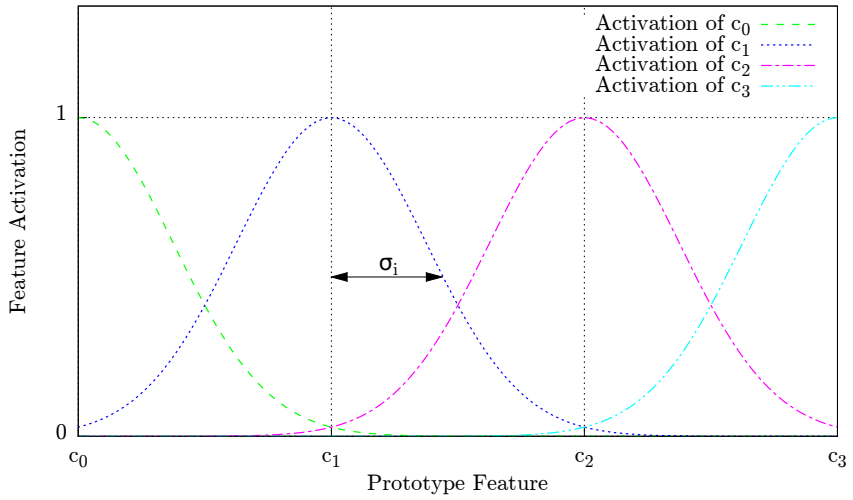


Figure 5.6: One-Dimensional Radial Basis Functions

In order to apply this on our agent's state-space and to make the approach flexible, a configuration parameter is introduced, called the *RBF-divisor*. It denotes in how many intervals between two prototypical states one of the two bandwidth dimensions should be divided. E.g. a divisor of 3 would mean that $4 \cdot 4 = 16$ features per address would be used to generalize over the possible combinations of out- and inbound bandwidth. The distance is calculated using the Euclidean Norm for the features of the currently selected address. For the width of the features

$$\sigma = \sqrt{2} \cdot \frac{m}{2(d+1)} \tag{5.24}$$

was chosen, where m is the maximum bandwidth possible and d the RBF-divisor. Using this method the resolution of the function-approximation can be adjusted. A higher resolution leads to an increase in computational and memory demands, but also yields a more accurate distinction between real states. Decreasing the number of prototype features results in a more rough approximation, which makes learning less fine-grained, but faster.

5.3.5 Adaptive Stepping

The formalism of the MDP as stated before only allows time-steps in discrete constant time intervals. This discrete treatment fits the approach's principle to treat ATS like a control-problem. The agents "measure" the perceived quality of the allocation, i.e. their utility, and system state every τ seconds and decide on what action to take in order to optimize the operation. Furthermore, the actions taken have an almost instant effect, neglecting possible flow control by the underlying transport mechanism. The solver so

far formally deals with the problem of delayed rewards, due to the maximization of the future discounted return. It is not very flexible though, because of the rigid equidistance of the time steps. Considering the case, in which no bandwidth is allocated at all yet, it is desirable that the agents perform several subsequent steps faster until the resources are taken. Additionally, it is advantageous to be able to perform learning/control steps, when the problem changes. E.g. in case a connection request is withdrawn and the corresponding agent is disabled for the decision making process, a further step should be triggered.

5.3.5.1 Learning in Continuous Time

This problem can be resolved by the generalization of MDPs to *Semi*-Markov Decision Processes (SMDPs), which take the time into account that has passed between two steps. For the continuous case, in which the time interval is real valued, this has been studied by Bradtke and Duff [BD95], but is also described in Sutton's book [SB98]. Recalling Equation 5.2 from Section 5.2 the return is the discounted sum of future rewards:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Assuming infinitesimal interval lengths an SMDP generalizes this to a discounted integral of future rewards:

$$R_t = \int_0^{\infty} e^{-\beta\tau} r_{t+\tau} d\tau \tag{5.25}$$

In the Equation 5.25 $r_{t+\tau}$ is the instantaneous reward given at time $t+\tau$. Similar to the discount-rate parameter $\gamma \in [0,1[$ in the case of MDPs, the parameter $\beta > 0$ has an influence on how strong the discount shall be.

Whenever an agent gets to decide a new action, this is a discrete-event transition, since from the agent's point of view just the decisions matter. Consider the transition over a time-step with length $\tau = t_2 - t_1$. If at time t_1 the system is in state s and the agent decides for action a and at time t_2 the system is in state s' and the agent decides for action a' having received reward r_τ. The SARSA update rule is adapted to:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[\frac{1 - e^{-\beta\tau}}{\beta} r_\tau + e^{-\beta\tau} Q(s',a') - Q(s,a) \right]$$

Accordingly, the update for the Q-learning algorithm is adapted to:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[\frac{1 - e^{-\beta\tau}}{\beta} r_\tau + e^{-\beta\tau} \max_{a''} Q(s',a'') - Q(s,a) \right]$$

The modified formalism to the use of SMDPs now allows to introduce a mechanism on top of the learning algorithm, which changes the length of the time interval between learning steps dynamically. Furthermore, learning steps can be triggered at arbitrary times.

5.3.5.2 Adaptive Stepping Function

The action space of each agent is small with just one action which increases the allocated bandwidth and one which decreases it per direction. Therefore, if the optimal allocation, e.g. due to a significant change in the environment, is very different from the current one, many steps are necessary to reach it. In order to speed up the allocation process a function is proposed, which influences the length of the intervals between learning steps. It's underlying heuristic is to increase the number of learning steps, if the available bandwidth is not used.

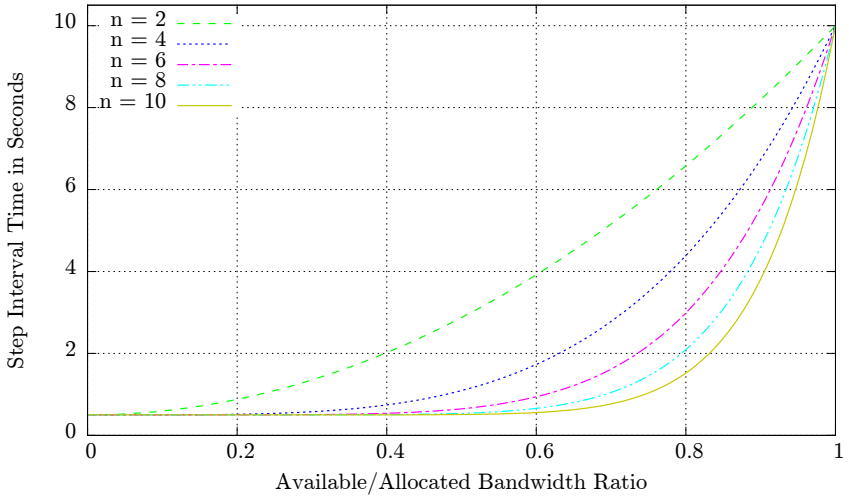


Figure 5.7: Function to Determine the Step Interval Length

Whether or not there is bandwidth yet to be allocated or not is decided in the following way: For each network scope, which contains at least one active address, the utilized and the assigned bandwidth (both inbound and outbound) are summed up to two values $b_{\text{available}}$ and b_{utilized}. Their ratio $b_{\text{assigned}}/b_{\text{utilized}}$ gives an indicator about how much of the quota has already been spent. In case of overutilization a ratio of 1 is used. In the configuration of the implemented solver a minimum and a maximum interval length, $t_{\min}$ and $t_{\max}$, can be set. Depending on how much bandwidth is used, the adaptive interval mechanism sets the next interval length according to the function from Figure 5.7, where n is a positive integer.

$$(t_{\max} - t_{\min}) \cdot \left(\frac{b_{\text{utilized}}}{b_{\text{available}}} \right)^n + t_{\min} \tag{5.26}$$

The values in the figure are set to $t_{\min} = 0.5\text{s}$ and $t_{\max} = 10\text{s}$. It may also be set in the configuration of the solver. The idea behind it is to have a non-linear mapping, which only increases the interval length, if most of the bandwidth is used already. If that is the case, there is less reason to change the allocation fast and there is more time for the measurements of the quality properties to take place. In effect the solver is more reactive, e.g. if an address was deleted and its resources are free to be distributed again.

The share of used bandwidth is not the only possible metric, which is conceivable to determine the frequency of the steps. E.g. one could calculate the standard deviation of the utility of the agents per network scope, which is expected to equalize over the learning process. Nevertheless, the adaptive stepping formalism allows to decrease the computational resource demands, in case the allocation is deemed good and less control over it is needed.

5.4 Summary

In order to conclude this chapter and to give a better insight of how the previous ideas are assembled, the resulting algorithms are now given in procedural form as pseudocode. The general idea is to have a set of agents, where each of them represents one peer for whom a connection request is present. One agent has a set of actions, which it can perform in steps in order to alter the current allocation for the respective peer. The steps are triggered by the agents' environment. During each step they receive a reward for their last action, update their experience (i.e. their estimation of the action-value function), and choose the next action to perform.

Therefore, the inputs are: First, the current allocation for each agent, determining the state s described in Section 5.3.3.1 of which the state-feature vector $\vec{\phi}(s)$ is calculated using the RBF-network described in Section 5.3.4. Second, the reward described in Section 5.3.3.2, which is discounted by γ', depending on the time τ, which has passed since the last step. The discounting is following the SMDP concept of Section 5.3.5.

The outputs are the actions of each of the agents. The actions are enumerated so that each action a has a corresponding parameter vector $\vec{\theta}(a)$ in the matrix $\boldsymbol{W}$ as defined for the approximation of the action-value function described in Section 5.3.4. Similarly, each action has a corresponding eligibility vector $\vec{e}(a)$ in the eligibility matrix $\boldsymbol{E}$, as defined in the same section. The concept of eligibility traces themselves is explained in Section 5.2.4. Matrix $\boldsymbol{W}$ can be referred to as the experience of the agent, since it determines the output of the agent's action-value estimation. Matrix $\boldsymbol{E}$ is a helper structure to perform the update. It determines how much each component of $\boldsymbol{W}$ should be affected by the gradient-descent update. The update depends on the difference between the estimated action-value of the last step and that of the current step δ.

The configuration parameters are: β, influencing the discount of future rewards, α, the step-size of the gradient descent/the learning rate, and ϵ/the temperature of the action-selection strategy.

Algorithm 2 shows the approach based on the SARSA algorithm described in Section 5.2.3.1, with both ϵ-greedy and softmax action-selection strategy, as well as replacing eligibility traces.

As initialization in the beginning the matrix $\boldsymbol{W}$ can be set to arbitrary numbers. The eligibility matrix $\boldsymbol{E}$ has to be set to zero. An initial action a can be chosen at random or depending on the random numbers in $\boldsymbol{W}$.

Then the basic control loop starts.

The discount values γ and γ' are the same for all of the agents and can be calculated only once depending on the length of the time interval τ between two steps. At the beginning of each step, the reward function is evaluated, as well as the feature vector of the current state. Afterwards the eligibility for the current step is decayed for all feature-action pairs. For the eligibility vector of the last chosen action the values are replaced if they increase the eligibility. Consecutively, the first component of the update δ of the parameter vectors is given just before the next action is selected. This is done either using the ϵ-greedy strategy or the softmax strategy. Knowing what the next action is in the current policy, the rest of δ can be calculated and the update can be performed on $\boldsymbol{W}$'s parameter vectors with the eligibility given by the corresponding vectors in $\boldsymbol{E}$. Finally, the selected action is taken.

Algorithm 3 displays the combination of the preceding ideas based on Watkins' Q-learning algorithm described in Sections 5.2.3.2 and 5.3.4. Basically, it does the same procedure as the SARSA-version, but it has some subtle differences. The first is the eligibility traces

Algorithm 2 Gradient descent SARSA(λ) with continuous time and function approximation

for all agents **do** ▷ Initialization
  initialize $\boldsymbol{W}$ randomly
  $\boldsymbol{E} \leftarrow 0$
  select and take initial action a
end for
repeat for each step ▷ Main loop
  measure τ
  $\gamma \leftarrow e^{-\beta\tau}$ ▷ Calculate discount of the future return
  $\gamma' \leftarrow \frac{1}{\beta}(1-\gamma)$ ▷ Calculate discount of the current reward
  for all agents **do**
    observe reward r and next state-features $\vec{\phi}(s)$
    $\boldsymbol{E} \leftarrow \gamma\lambda\boldsymbol{E}$ ▷ Reset eligibility
    for all features i in $\vec{\phi}$ **do** ▷ Increase eligibility of the last state-action pair
      $\vec{e}(i,a) = \max\left[\vec{e}(i,a), \vec{\phi}\right]$
    end for
    $\delta \leftarrow \gamma' r - \vec{\phi}^T\vec{\theta}(a)$ ▷ Difference calculation

    ▷ Action-selection:
    $n \leftarrow$ random number $\in [0,1]$
    if $n < \epsilon$ **then** ▷ In case of ϵ-greedy
      $a \leftarrow$ random action
      decay ϵ
    else
      $a \leftarrow \arg\max_{a'} \vec{\phi}^T\vec{\theta}(a')$
    end if
    for all possible actions a' **do** ▷ In case of softmax
      if $n < P(a'|s)$ **then**
        $a \leftarrow a'$
        if a is non-optimal **then**
          decay temperature
        end if
      end if
    end for
    $\delta \leftarrow \delta + \gamma\vec{\phi}^T\vec{\theta}(a)$ ▷ Difference calculation
    for all column vectors $\vec{\theta}$ and $\vec{e}$ in $\boldsymbol{W}$ and $\boldsymbol{E}$ **do** ▷ Gradient descent
      $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    end for
    take action a
  end for
until forever

being deleted, if the last action was an exploratory one; or in other words, if it was not the one with the highest estimated value. As mentioned before, this is necessary, because the received reward does not result from the optimal policy, which the Q-learning algorithm intends to approximate. The second difference is that Q-learning does not need to know which action is chosen next for its update of the action-value estimation. Thus, it performs the update already before choosing the next action.

Algorithm 3 Gradient descent Watkins' Q(λ) with continuous time and function approximation

for all agents **do** ▷ Initialization
  initialize $\boldsymbol{W}$ randomly
  $\boldsymbol{E} \leftarrow 0$
  select and take initial action a
end for
repeat for each step ▷ Main loop
  measure τ
  $\gamma \leftarrow e^{-\beta\tau}$ ▷ Calculate discount of the future return
  $\gamma' \leftarrow \frac{1}{\beta}(1-\gamma)$ ▷ Calculate discount of the current reward
  for all agents **do**
    observe reward r and next state-feature vector $\vec{\phi}(s)$
    if a was exploratory **then**
      $\boldsymbol{E} \leftarrow 0$ ▷ Reset eligibility
      decay temperature/ϵ
    else
      $\boldsymbol{E} \leftarrow \gamma\lambda\boldsymbol{E}$ ▷ Decay eligibility
    end if
    for all features i in $\vec{\phi}$ **do** ▷ Increase eligibility of the last state-action pair
      $\vec{e}(i,a) = \max\left[\vec{e}(i,a), \vec{\phi}\right]$
    end for
    $\delta \leftarrow \gamma' r - \vec{\phi}^T\vec{\theta}(a)$ ▷ Action-value difference calculation
    $\delta \leftarrow \delta + \gamma \max_{a'}\left[\vec{\phi}^T\vec{\theta}(a')\right]$
    for all column vectors $\vec{\theta}$ and $\vec{e}$ in $\boldsymbol{W}$ and $\boldsymbol{E}$ **do** ▷ Gradient descent
      $\vec{\theta} \leftarrow \vec{\theta} + \alpha\delta\vec{e}$
    end for

    ▷ Action-selection:
    $n \leftarrow$ random number $\in [0,1]$
    if $n < \epsilon$ **then** ▷ In case of ϵ-greedy
      $a \leftarrow$ random action
    else
      $a \leftarrow \arg\max_{a'} \vec{\phi}^T\vec{\theta}(a')$
    end if
    for all possible actions a' **do** ▷ In case of softmax
      if $n < P(a'|s)$ **then**
        $a \leftarrow a'$
      end if
    end for
    take action a
  end for
until forever

6. Implementation

This section explains the implementations of the proposed design based on the ATS problem and the findings of the last chapter. In an analysis of requirements for reinforcement learning libraries [KE11] Kovacs and Egginton point out how difficult it is to account for the complexity of the domain given the variety of environments. A reinforcement learning library with a wide scope of potential use cases has the tendency to become less easy to use than implementing the desired techniques. Furthermore, it proves beneficial for a good understanding of common reinforcement learning techniques to implement, test, and adapt them. As a consequence, there is no library, on which the reinforcement learning community has reached consensus. The existing ones either lack integratability into present systems, such as GNUnet, ease of use, expressiveness for the desired scenario or project activity. Taking this prospect into account it was more reasonable to implement the algorithms as designed in the previous chapter specifically for this thesis.

More precisely there are two implementations. One is a simulation prototype, which was used during the design phase to investigate on the effects of changes in the approach, described hereafter. The other is the implementation of a solver plugin for the GNUnet framework. The implementation sources are publicly available online[1].

6.1 Simulation Prototype

In order to conduct experiments and tweak the reinforcement learning mechanism, a prototypical implementation of the previously described approach was developed. This was done using the scripting language Python, as its high level nature allowed fast tryouts of different ideas.

The main goal of this prototype was to evaluate the general applicability of the algorithms and to gain experience with the dynamics of reinforcement learning. Therefore, the simulations have several simplifications, as opposed to the implementation considered for production use:

- The environment represents only one network scope, whose dynamics are examined.
- The learning steps are performed with equal time-intervals, which means the model is reduced to that of the MDP instead of the SMDP. Thus, the discount rate of the future return is set by γ and not β. Since the main motives for the introduction of

[1] GNUnet: https://gnunet.org/svn/gnunet Prototype: http://home.in.tum.de/~oehlmann/ril.tar.gz

the SMDP formalism is to be able to schedule learning steps at arbitrary times, this simplification is not grave in its effect. In the simulation the steps are discrete and it does not make a difference for the learning agent, whether the reward has been discounted according to the length of the time-interval between steps or not.

- An agent, which is allocated for one peer, manages the resource allocation for only one address. In the implementation for productive use, once an address suggestion is accepted and a connection is established over it, an address change is prohibited. An ongoing communication process should not be interrupted. The simulation examines the learning process for when an address is actually used for communication, because only then it will gain meaningful experience for learning the utility of the address.
- The allocated bandwidth for agent i inbound $b_{i,\text{in}}$ and outbound $b_{i,\text{out}}$ together with a preference value for agent i, u_i, is used as a measure of the agent's utility. It is defined as: $u_i \cdot \sqrt{b_{i,\text{in}} \cdot b_{i,\text{out}}}$.

Despite these simplifications, the simulation prototype serves as a valuable tool to gain insight into the complicated allocation process and gives an impression of how effective different settings of algorithms and parameters are for the learning progress.

It is easy to use, as it simply performs a given amount of learning steps with a specified number of peers, which share a scope. The configuration parameters are all given as command-line arguments. Outputs are log-files of the allocation of each peer and their sum, which can be processed and visualized afterwards.

6.2 GNUnet's ATS Service

In order to show the usability and feasibility also an implementation in the targeted framework is provided.

As discussed in Chapter 3, the ATS problem in the GNUnet framework is solved by the `ats` service. It keeps track of the relevant peers, addresses, transport mechanisms, address properties, application preferences and the user configuration.

The solving mechanism is realized as an exchangeable plugin. This plugin carries out the decision making process. It is further detailed in Section 6.3.

Outside the plugin the `ats` service takes care of the "administrative" tasks, which are necessary for the operation of a GNUnet service. It handles the messages received from the `transport` service below, and the applications above in the layered architecture. It extracts and prepares the information it receives. I.e. it maintains data structures, which contain the inputs of the ATS problem.

Concerning the information received from the `transport` service this is mainly the address management. This includes the knowledge about which addresses of a peer are valid and what properties they have, such as delay, distance, and current utilization.

The information communicated by the applications is the preferences they have for the connections to each peer. This includes a value for the importance of each of the properties a connection can have, which should be fulfilled by the address selection. Because of the multitude of applications, the preference values are already accumulated in one normalized value in the range between 1 and 2 for each peer and each desired property of the connection.

The only output of the `ats` service is the response to connection requests, in which it suggests which address to use for a connection, and how much bandwidth is to be allocated for it.

6.3 ATS Service Solver Plugin

In order to abstract the task of finding a resource allocation from the inputs of the problem and to be able to implement different approaches, a plugin mechanism has been designed. During the startup of a GNUnet peer and its `ats` service, a plugin is loaded, which carries out the decision making process. While the rest of the `ats` service communicates with the other services and applications, and maintains the necessary information, the solving functionality is solely consisted in the *solver plugin*. This alleviates exchanging the decision process for one, which is better suited in different use cases. Moreover, research into novel approaches can be done fast, since there is just a minimal integration effort for new implementations. All of the solution methods described in this thesis are realized as a solver plugin of their own. These are the proportional method, and the linear programming approach from Section 3.5 as well as the reinforcement learning solver, as it is designed in Chapter 5.

6.3.1 ATS Solver API

When a plugin is loaded its initialization function is called, so that it can prepare its data structures and preprocess the user configuration. Afterwards, it has to be ready to handle all the necessary requests.

The plugin is informed about the addition of new addresses, which can be taken into account for the decision making process. Addresses, which are not valid anymore, are communicated to the solver as well. Such an address may not be suggested afterwards, and an alternative has to be found in case it was used before, if possible. Whenever the properties of an address change, the solver plugin is notified about it and can act accordingly. Address properties can be looked up by the solver at any time.

As mentioned previously, the `ats` service performs an accumulation of the application preferences. Accordingly, the solver is agnostic about the number and kind of applications operating on top of GNUnet. Just as the address properties, the application preferences are looked up at arbitrary times and the solver is notified about their changes.

Furthermore, the solver has to keep track of connection requests, which are valid for a period of time. In case it decides to change the allocation or the proposed address, due to a change in preferences, properties, or other additional requests, it has to notify the `transport` service about this. This includes the cases where an address request was made for a peer, of which `ats` does not know an address initially, but learns it later, and where a solver decides that for a connection no resources should be allocated anymore. This will cause the `transport` service to destroy an ongoing connection. This is also the case, if the solver decides to change the address to a different one. If the solver wants to avoid such a situation, it can do so by taking information about the session and usage status of an address into account. The solver is notified if a connection request is revoked and does not have to be served anymore.

Additionally, it is possible to block the solver's operations, if it is known that many consecutive notifications of input changes could trigger unnecessary recalculations.

6.3.2 The Reinforcement Learning Plugin

The essential information that the implemented reinforcement learning solver keeps is a list of learning agents and an array of network scopes. These in turn keep the following information:

Parameter	Meaning
Algorithm	Whether to use the SARSA or the Q-learning update rule
α	The gradient descent step size
β	The reward discount factor for continuous time-steps
Social Welfare Measure	Whether to use the Nash or the egalitarian metric in the reward function
Action-Selection Strategy	Whether to use ϵ-greedy or softmax
λ	The eligibility-trace decay factor
ϵ	The exploration Ratio in the ϵ-greedy action-selection strategy
ϵ decay	The decay factor of ϵ when an exploratory step was taken
Temperature	The temperature in the softmax action-selection strategy
Temperature decay	The decay factor of the temperature when an exploratory step was taken
RBF divisor	The density of the RBF network
$t_{\min}$	The minimum step-interval time
$t_{\max}$	The maximum step-interval time

Table 6.1: Plugin Configuration Parameters Loaded at Startup

Agents:

- A list of available addresses for the respective peer.
- The currently chosen address for the respective peer.
- The amount of bandwidth currently allocated for the connection to the peer.
- Its activity status, i.e. whether a connection request is currently pending or not.
- The matrix $\boldsymbol{W}$ used for the action-value estimation with one column for each action, and as many rows as state-features are specified by the density of the prototype states in the RBF network. This is the $(rbf\_divisor + 1)^2$ times the number of addresses known for the peer.
- Information necessary for the Temporal Difference (TD) algorithms. The matrix $\boldsymbol{E}$ for the eligibility, which has the same size as $\boldsymbol{W}$. Furthermore, it memorizes the last action taken and the last state-feature vector.

Network Scopes:

- The inbound and outbound quotas.
- The number of active agents, which currently have an address selected in the scope.
- The sum of the assigned and utilized bandwidth of the peers controlled by active agents.

During the initialization of the plugin the configuration parameters from Table 6.1 are loaded and the quotas for the network scopes are set.

The solver plugin is in a continuous loop of controlling the current resource allocation. Therefore, as soon as it receives the first requests, global steps are triggered, which in turn schedule the next step in the future according to the time calculated by the adaptive stepping mechanism. Upon an incoming request or notification, a step is triggered immediately, which cancels the already scheduled task.

The agents' actions are enumerated. The first five numbers correspond to the actions of increasing or decreasing the amount of bandwidth and the action to do nothing. All the

actions with higher numbers are to switch to one of the addresses from the agents' list of available addresses for their respective peer.

When the solver receives a connection request for a peer, the respective agent is set *active* and therefore takes part in the allocation as described by the algorithms in Section 5.4. Inactive agents do not allocate any resources. In each global step, the solver triggers a learning and action step of each active agent, for whose peer at least one address is known. An exception is the request of a new connection to a peer when all of its addresses lie in network scopes, which are considered *full*. This is the case when the number of active addresses in a network scope times a fixed minimum amount of bandwidth already exceeds the quota. Assuming a connection needs at least this minimum bandwidth, a further connection can not be served anymore. As long as this is the case an agent is blocked and does not take part in the allocation game.

The values for the current utilization obtained from the `ats` service's information of each address and the social welfare within a network scope are updated at the beginning and at the end of each global step. Similarly, the discount factor for a step is only calculated once before triggering the agent's steps, as it depends on the time, which has passed in-between.

7. Evaluation of the Approach

This chapter evaluates the approach and its underlying mechanisms and parameters. In order to do so, simulation scenarios were executed to empirically evaluate the functioning of the approach, as well as to gain an insight on the impact of different settings. Afterwards the implementation of the GNUnet solver plugin is evaluated in terms of runtime.

7.1 Simulation

This section puts emphasis on the results obtained using the simulation script. The focus is to see if and how well the approach works in a first, simplified scenario, and how the parameters impact the allocation procedure. As the most significant settings we identified: the choice of the action-selection strategy, whether or not to decay the exploration ratio, and which update rule should be used. Furthermore, we evaluate the social welfare model in the reward function. Before the adjustment of the numerical parameters of the approach, such as the learning rate, or the reward discount factor, it is important to understand the consequences of the settings which change the actual procedure of the approach.

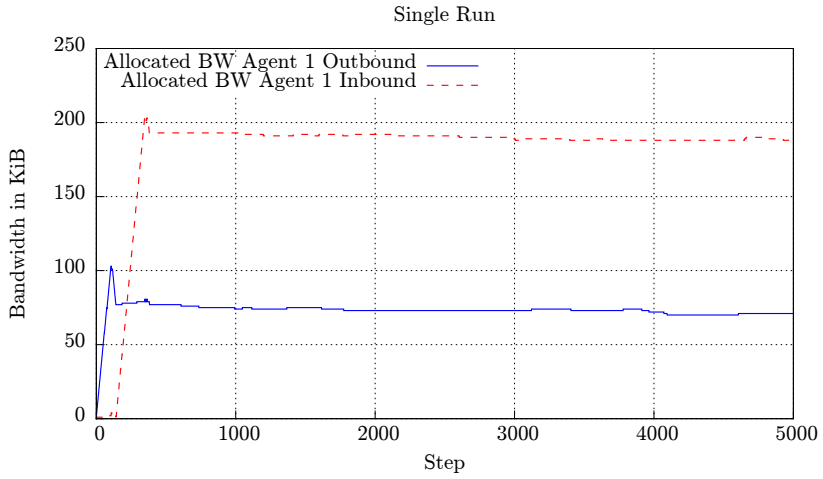
The first three experiments highlight the actions determined by the algorithm in the case of just a single agent with different configurations. Some learning parameters are equal for all of the results conducted from the experiments. These are on display in Table 7.1. The learning rate α is set to 0.01. Therefore, the learning experience of a step will change the current estimation by 1%. The discount factor for the future return γ and the eligibility trace decay factor are both set to 0.5. This means that half of the value an action is estimated at is the reward of the next step. The eligibility trace is set to update past state-action pairs accordingly with the same weight.

Parameter	**Value**	**Range**
Learning Rate α	0.01	[0,1]
Discount Factor γ	0.5	[0,1[
Eligibility Trace Decay Factor λ	0.5	[0,1]
Outbound Quota	100 KiB	$\mathbb{N}$
Inbound Quota	200 KiB	$\mathbb{N}$
Maximum Bandwidth	300 KiB	$\mathbb{N}$
RBF Divisor	3	$\mathbb{N}$

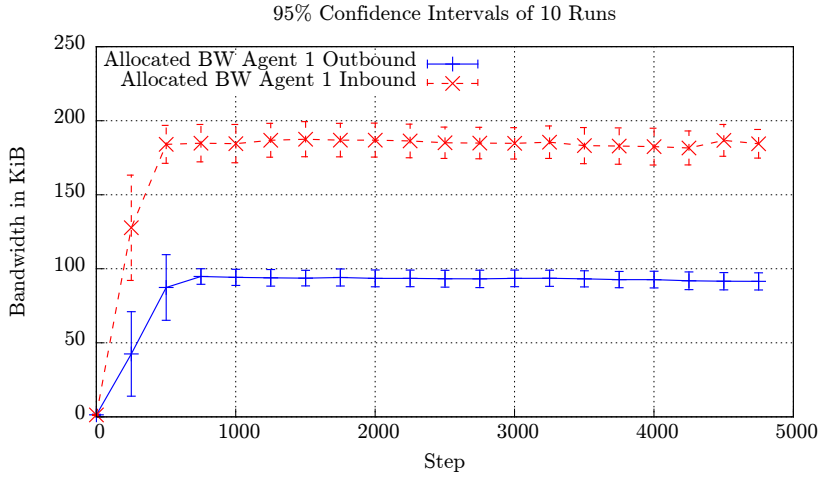
Table 7.1: Common Parameter Configuration in the Single Agent Experiments

The outbound and inbound quota in the scenarios are set to 100 KiB and 200 KiB respectively. These are the bandwidth limits in the simulated network scope, which may not be exceeded. A new parameter is the maximum bandwidth value. It is the maximum bandwidth in KiB which can be allocated by the agent inbound or outbound. The value sets a boundary to the state space at a small scale. With a value of 300 KiB and an RBF divisor of 3 follows, that there are prototype states at every 100 KiB both for the inbound and outbound dimension. The actions available to the agent are to de- or increase the inbound and outbound bandwidth by 1 KiB. The initial allocation is 1 KiB in each direction. The simulated agent uses all bandwidth allocated for it, which increases its utility.

7.1.1 Comparison of Action-Selection Strategies



(a) $\epsilon = 0.01$, ϵ decay $= 1$

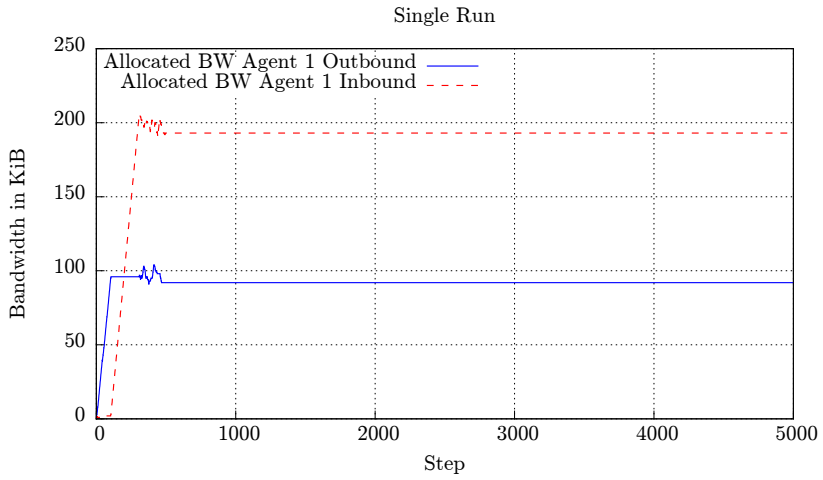


(b) $\epsilon = 0.01$, ϵ decay $= 1$

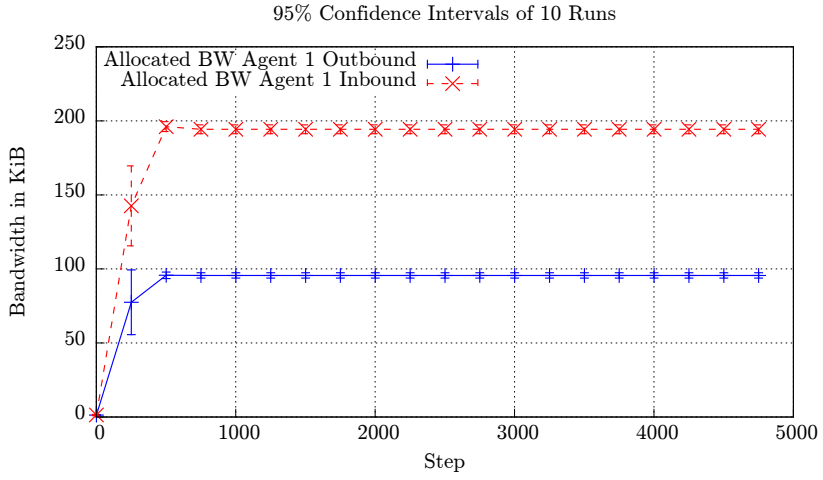
Figure 7.1: Convergence of Single Agent SARSA with ϵ-Greedy Action-Selection in the Exploitative Setting

The goal of this section is to compare the action-selection strategies ϵ-greedy and softmax. The question is what impact an increased exploration ratio or temperature has on each of the selection strategies. Therefore, two settings are researched: In the first setting both strategies have a configuration which leads to a behavior of the agent that includes only a small amount of exploratory steps. Then the agent is very exploitative. In the second setting the respective configuration values are set to increase the amount of exploratory

steps. Both settings are examined with the SARSA algorithm. The simulations in the two settings have been run for both action-selection strategies with a constant exploration ratio and a constant softmax temperature respectively.



(a) temperature = 0.01, temperature decay = 1



(b) temperature = 0.01, temperature decay = 1

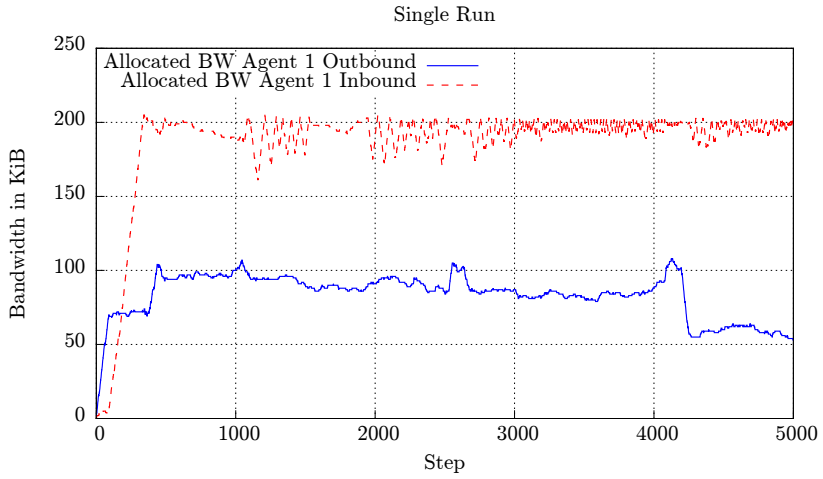
Figure 7.2: Convergence of Single Agent SARSA with Softmax Action-Selection in the Exploitative Setting

Figure 7.1 shows the result for an ϵ value of 0.01. This means a random action is performed on average in 1% of the steps. Opposing to that Figure 7.2 shows the results with the softmax selection strategy at a temperature of 0.01. This yields a low probability of diverting from the optimal action once meaningful action values have been learned. Both Figures consist of two Subfigures each. Subfigure (a) shows the behavior of a single run. Subfigure (b) displays the 95% confidence intervals and the average of the allocation after every 250 steps of 10 runs.

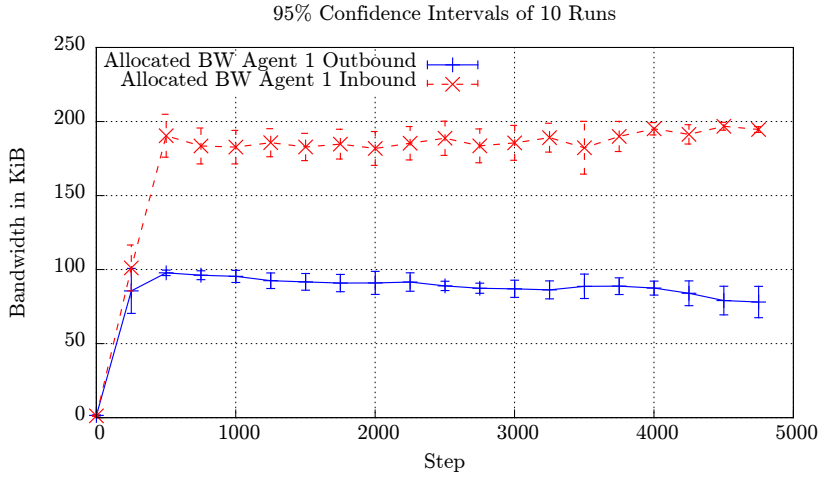
Using these parameters both action-selection strategies find an allocation close to the quotas after about 500 steps. Both strategies show a constant behavior afterwards.

The ϵ-greedy strategy shows few exploratory steps, which cannot be avoided, if the ϵ value is constant. In the single run can be seen that after having exceeded the quotas once, the best policy is estimated to stay below the optimum. The few exploratory steps do not lead to a change in this estimation.

The softmax strategy shows a similar behavior of shortly exceeding the quotas and staying constant afterwards. Different from the ϵ-greedy strategy it does not divert from the action estimated optimal anymore. This means then the agent effectively does not explore at all. The distance the agent keeps from the quotas is lower than is the case with the ϵ-greedy strategy.



(a) $\epsilon = 0.2$, ϵ decay $= 1$

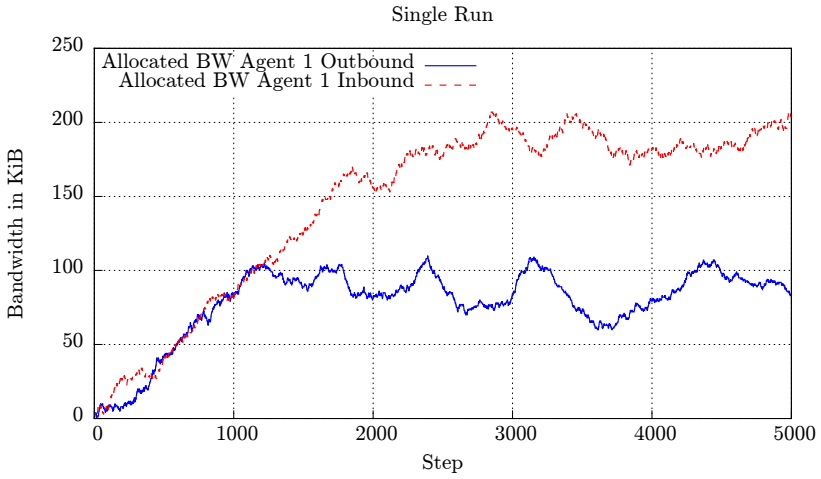


(b) $\epsilon = 0.2$, ϵ decay $= 1$

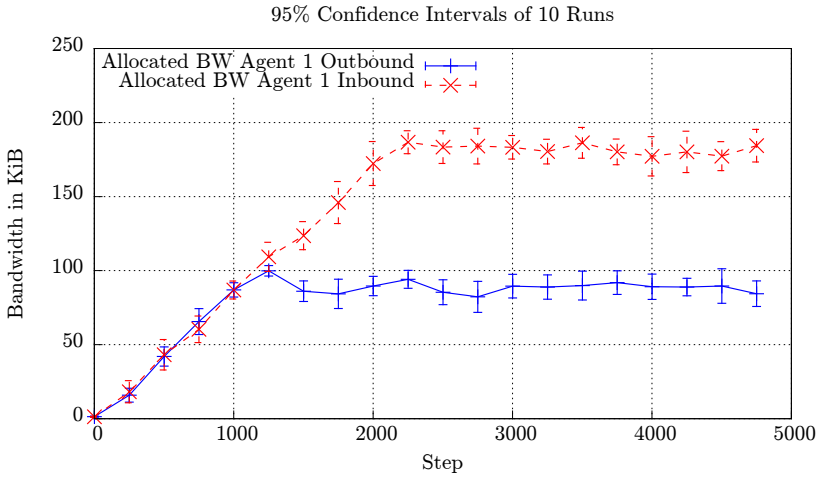
Figure 7.3: Convergence of Single Agent SARSA with ϵ-Greedy Action-Selection in the Exploratory Setting

In the exploratory setting the ϵ-greedy strategy was performed with an ϵ value of 0.2. Thus the agent takes an exploratory step on average every 5 steps. The resulting behavior is shown in Figure 7.3. As before in the exploitative setting, the quotas are exceeded for the first time after approximately 500 steps. The frequent exploitative steps lead to a an oscillation of the allocation around the inbound quota. The amplitude of the oscillation becomes smaller. The agent exceeds the outbound quota less often and keeps a higher distance after receiving a higher penalty.

The softmax action-selection strategy was run in the second setting at a temperature of 0.2. The result is displayed in Figure 7.4. Different from the first setting and different from the ϵ-greedy strategy it takes the agent more than 2000 steps to reach both quotas on average. The single run reveals that the strategy exposes a pattern of repeatedly exceeding the quotas as well.



(a) temperature = 0.2, temperature decay = 1



(b) temperature = 0.2, temperature decay = 1

Figure 7.4: Convergence of Single Agent SARSA with Softmax Action-Selection in the Exploratory Setting

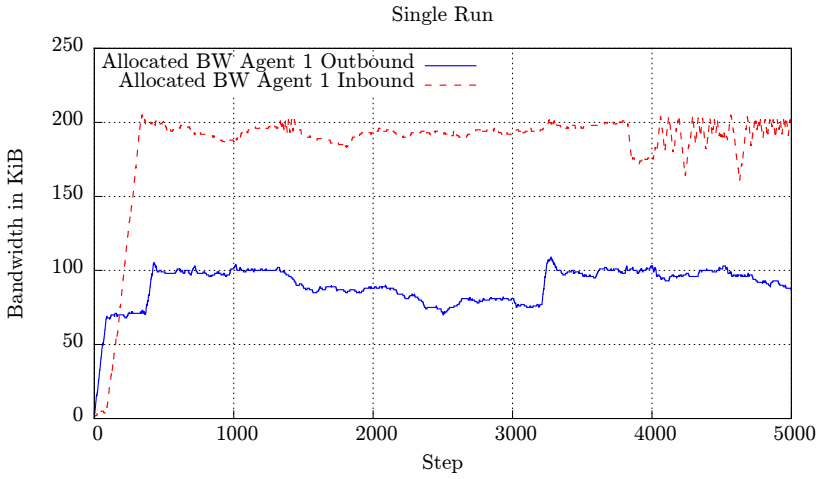
The simulation shows that both action-selection strategies are able to find a constant allocation near the quotas if the number of exploratory steps is small. Increasing the configuration to a more exploratory behavior leads to repeatedly exceeding the quotas in both strategies as well.

Reducing the exploration ratio or temperature over time is an alternative to minimizing the number of exploratory steps to begin with, in order to reach a constant allocation. This is tested in the next experiments.

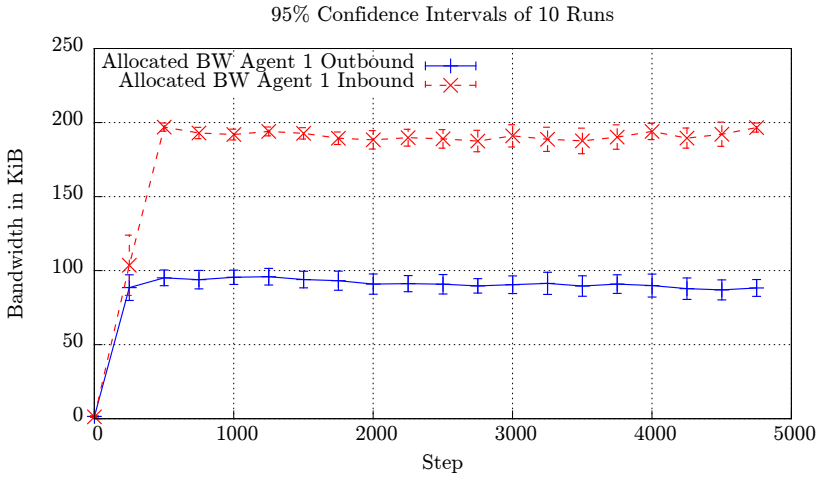
7.1.2 Decaying Exploration in Q-Learning

Since the Q-learning algorithm does not estimate the value of the policy it follows, as long as it does exploratory steps, it can be assumed to profit from decaying the exploration ratio until it only performs exploitative steps.

Again, two experiments have been conducted. Once the ϵ value was set to 0.2 without decreasing it over time, which means that on average every fifth action is a random one. This leads to the results in Figure 7.5. The second configuration has an initial ϵ value of 1, which is decreased by a decay factor of 0.95, every time an exploratory action was taken.



(a) $\epsilon = 0.2$, ϵ decay $= 1$



(b) $\epsilon = 0.2$, ϵ decay $= 1$

Figure 7.5: Convergence of Single Agent Q-Learning without Exploration Decay

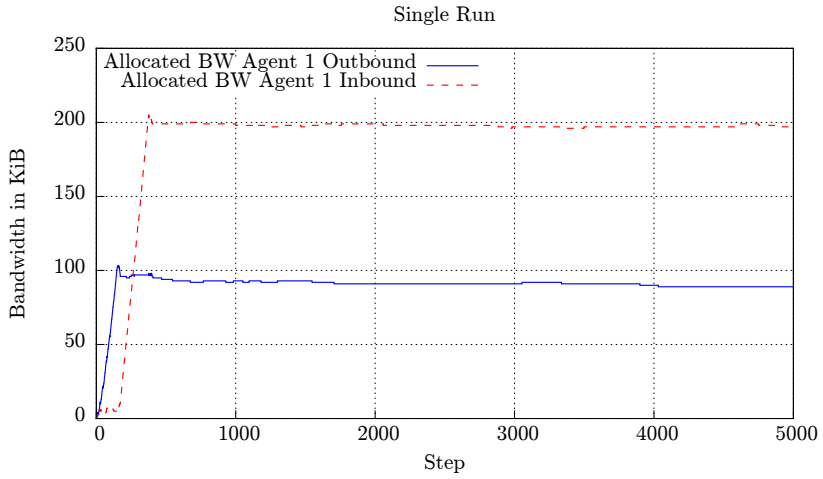
Therefore after about 76 exploratory steps, the algorithm has an exploration ratio of 0.2 as well, but further increasing afterwards. The allocation can be seen in Figure 7.6.

From the Figure 7.5 can be seen, that the Q-learning algorithm finds out the maximum quota allowed, but its many exploratory steps lead to an oscillation of the allocation. It aims more at being close to the quota at the inbound bandwidth. Introducing the decay of the exploration ratio improves the situation: once the quota has been detected and the policy is very exploitative, the allocation remains steadily constant below the quota. Figure 7.6 (b) reveals, though, that this learning process is not always successful after just 500 steps. While the averages of the runs are constant after an initial phase of upwards movement, the quota is not always reached immediately. Instead, it takes several thousand steps to increase the allocation to a level close to the quota.

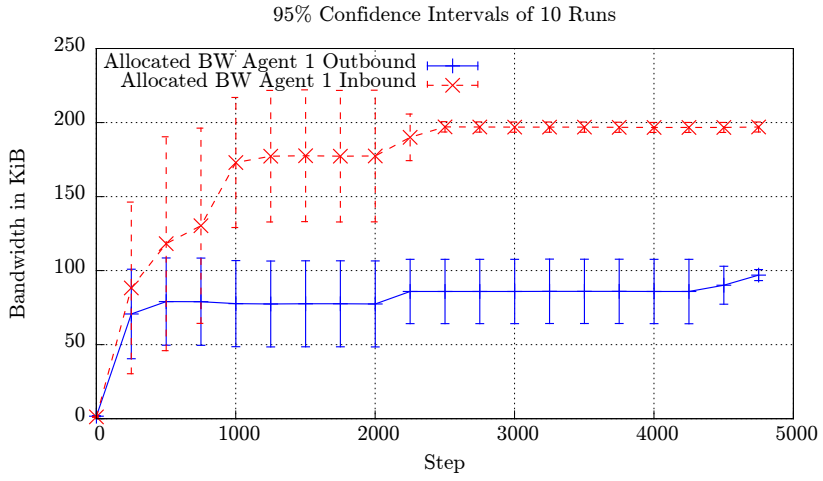
Taking these results into account, it is only possible to have a steady allocation without oscillation, if also the Q-algorithm does not do any exploratory steps eventually. In the latter case, it will have trouble learning the optimal allocation though.

7.1.3 Comparison of SARSA and Q-learning

The aim of this section is to compare the two update rules of SARSA and Q-learning. On the one hand, SARSA is on-policy and estimates the values of the policy it follows. On the



(a) initial $\epsilon = 1$, ϵ decay $= 0.95$



(b) initial $\epsilon = 1$, ϵ decay $= 0.95$

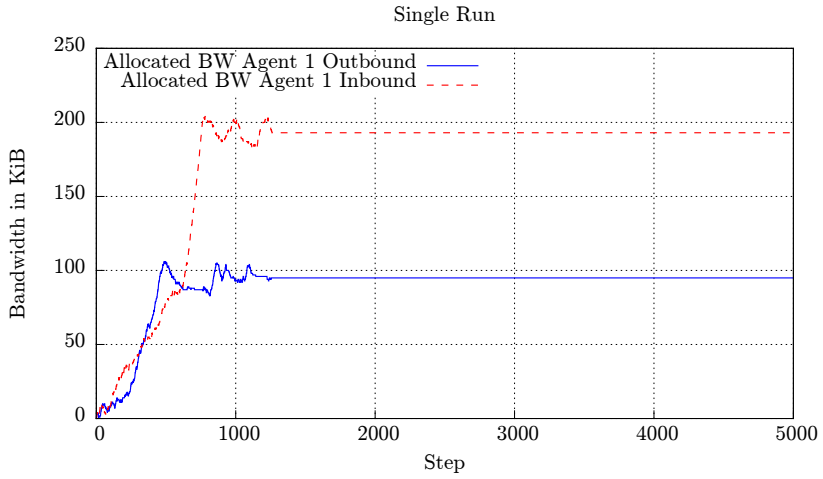
Figure 7.6: Convergence of Single Agent Q-Learning with Exploration Decay

other hand, Q-learning estimates the action-values of the optimal policy, even if it follows a different policy. The question is, which property makes the algorithms more suitable for the ATS task.

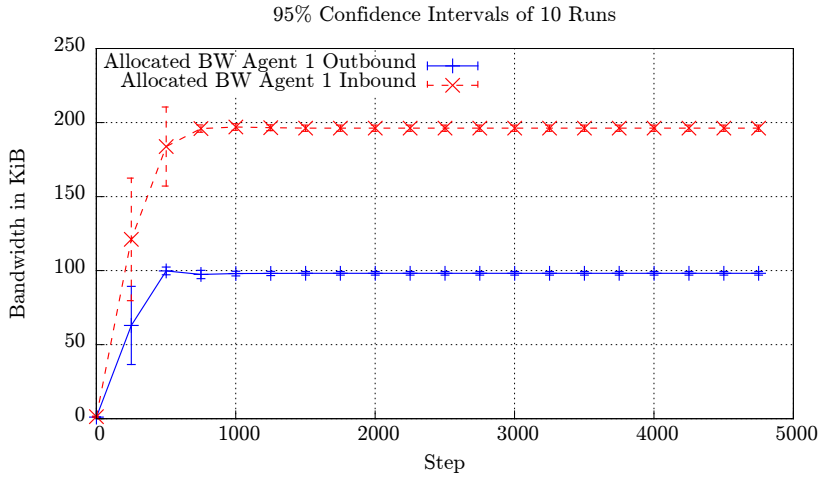
As has been seen in the two preceding sections, both SARSA and Q-learning are able to find an allocation close to the quota, if the number of exploratory steps is low. In fact, if the number of exploratory steps tends to zero both algorithms are the same. They both follow the fully exploitative policy then. The difference reveals itself when both are combined e.g. with the softmax action-selection strategy and a decreasing softmax temperature in this experiment. Both algorithms were run as before with an initial temperature of 1 as well as a temperature decay factor of 0.995. Figure 7.7 shows the allocation process of Q-learning, while Figure 7.8 shows SARSA.

What can be seen in these Figures, is that Q-learning finds a good allocation after roughly 1200 steps and remains constant with it. SARSA on the other hand does not reach a steady allocation within the first 5000 steps.

The SARSA algorithm faces a difficulty. Its principle is to estimate the action-values of the current policy. The decreasing of the temperature value is a change of the current policy, though, which renders the initial learning experience useless later on. After the



(a) initial temperature = 1, temperature decay = 0.995



(b) initial temperature = 1, temperature decay = 0.995

Figure 7.7: Convergence of Single Agent Q-Learning with Softmax Action-Selection and Temperature Decay

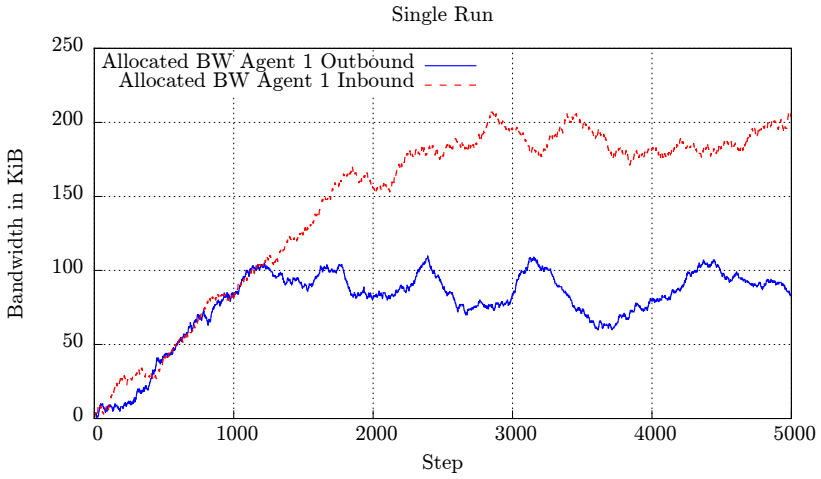
temperature has decreased and the number of exploratory steps tends to zero, the SARSA algorithm has to relearn the action values for the new policy.

The learning experience of Q-learning, on the other hand, is quite independent of the changed policy. Due to its on-policy characteristic it makes an estimation of the best policy right to begin with, independent of how random its policy actually is. As soon as the temperature has decreased so much that it is mostly exploitative, it has an estimation of the best policy and sticks to it.

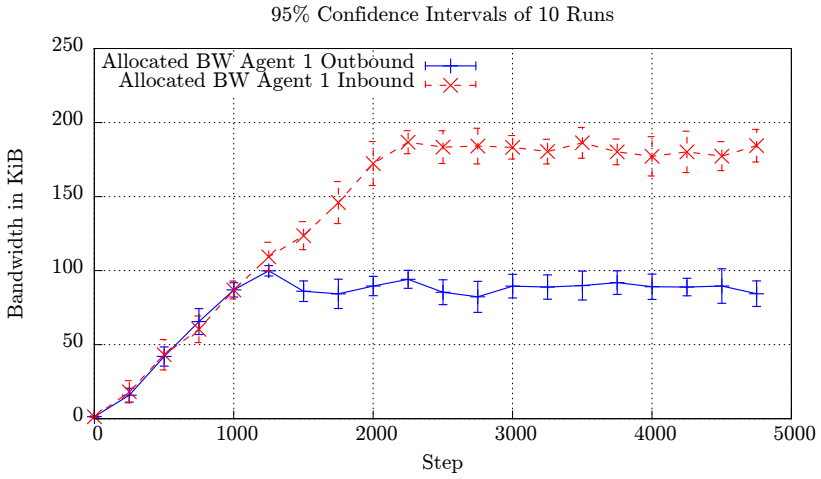
It can be concluded that with the same parameter settings the SARSA algorithm converges slower to a constant allocation than is the case with Q-learning if the amount of exploratory steps is reduced over time. This has an impact on the idea of resetting the amount of exploration when the environment changes in the ATS problem as described in Section 5.3.3.3. If a faster convergence is of demand, it is more beneficial to use the Q-learning update rule.

7.1.4 Comparing Measures of Social Welfare

The target of this experiment is to find out, how multiple agents perform in the allocation. Furthermore, this makes the question relevant which measure of social welfare is used in



(a) initial temperature = 1, temperature decay = 0.995



(b) initial temperature = 1, temperature decay = 0.995

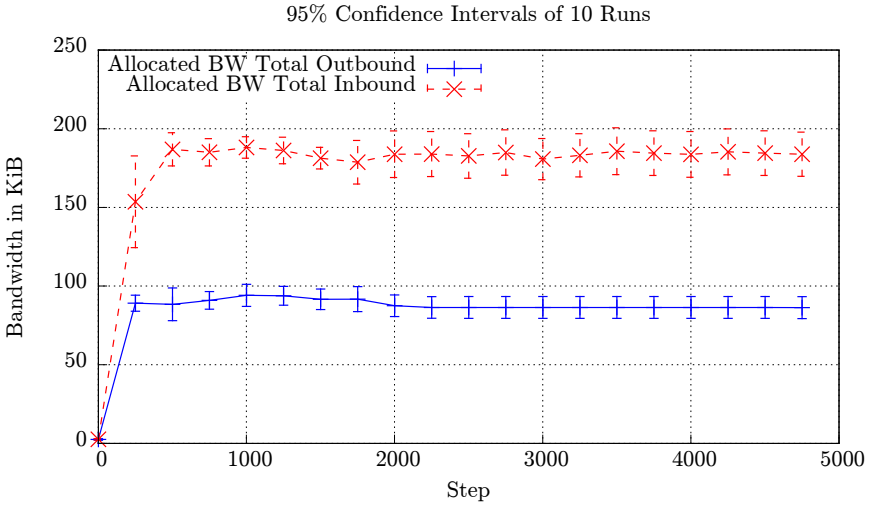
Figure 7.8: Convergence of Single Agent SARSA with Softmax Action-Selection and Temperature Decay

the reward function described in Section 5.3.3.2. Since they have a quite different semantic, this is expected to have an impact on the resulting allocations.

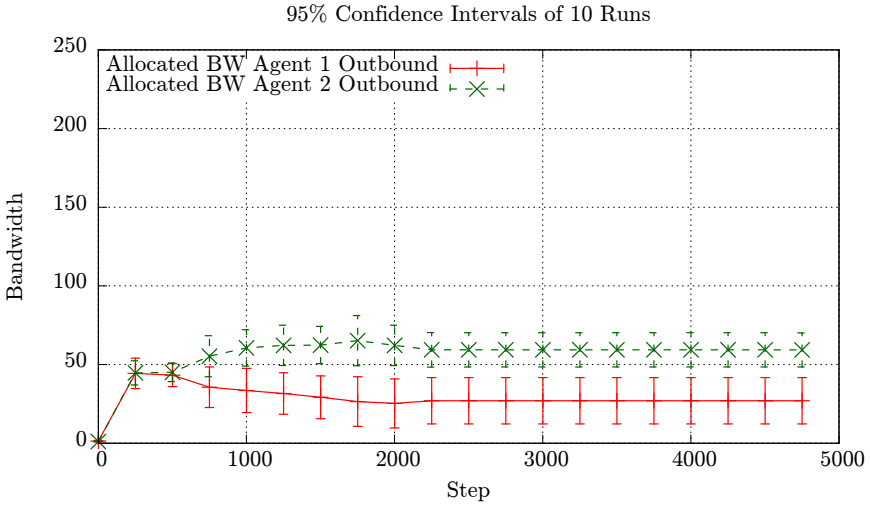
For the purpose of comparison the same simulation was done with both measures of social welfare. SARSA was picked as algorithm and softmax as action-selection strategy with a temperature of 0.1 and without a decay of the exploration ratio. For the social welfare measure to take effect, two agents now share the same network scope and need to find an allocation together. The peers the respective agents act for have preferences, in order to influence the allocation. Hereby, agent 1 has a preference of 1 and agent 2 has a preference of 2. This corresponds to the minimum and maximum normalized preferences calculated in the `ats` service of GNUnet.

Figure 7.9 shows the results, when the Nash measure is used to achieve a fair distribution. Subfigure (a) shows the sum of the allocated bandwidth, while Subfigures (b) and (c) show the outbound and the inbound bandwidth respectively. In each figure again the 95% confidence intervals of the values at every 250th step after 10 runs are drawn.

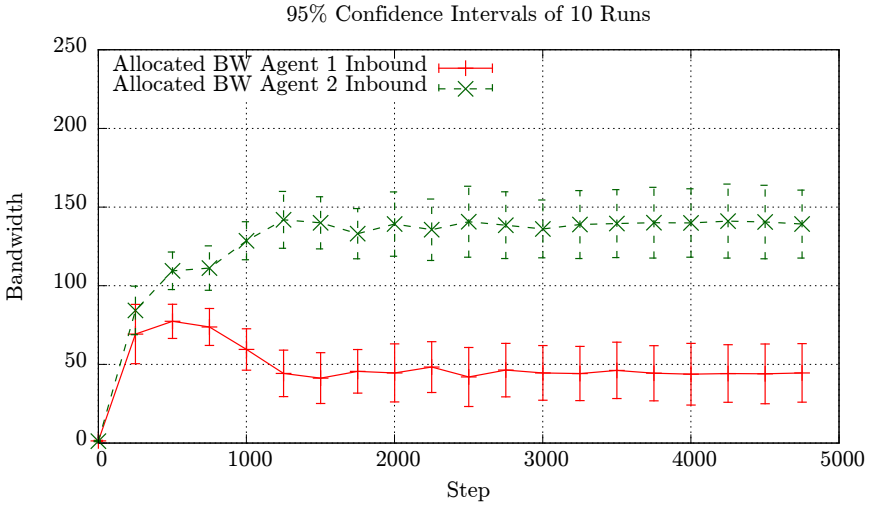
Just as in the single agent case, there is no overutilization above the quotas of 100 and 200. Both allocations, inbound and outbound, are roughly shared at a 2:1 ratio according



(a) Summed Up Bandwidth, Nash Social Welfare

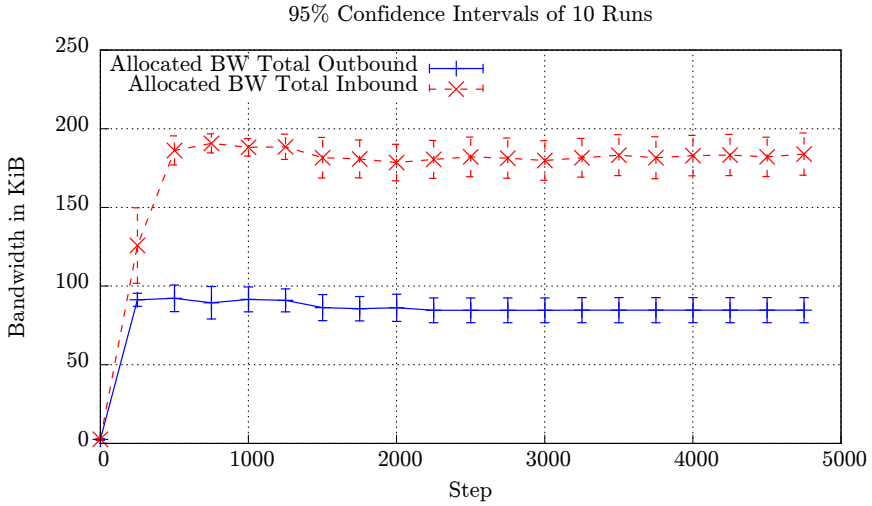


(b) Outbound Bandwidth, Nash Social Welfare

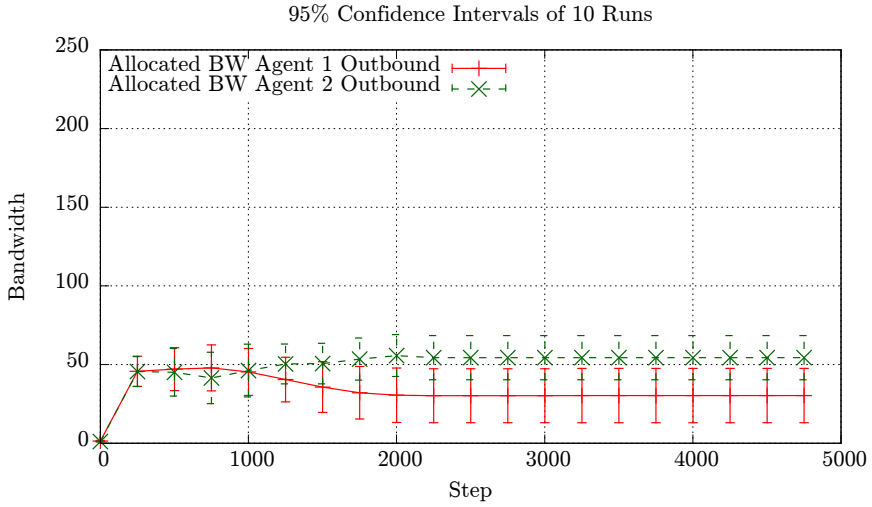


(c) Inbound Bandwidth, Nash Social Welfare

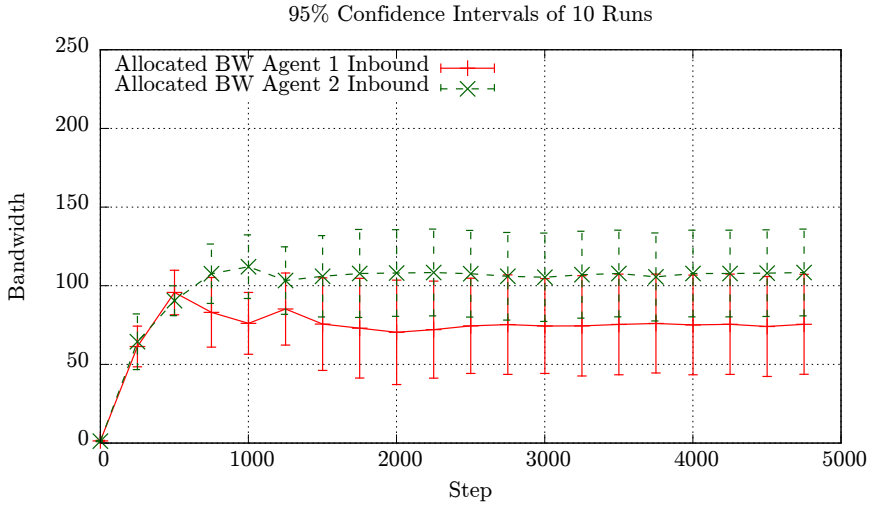
Figure 7.9: Convergence of Two Agent SARSA with Nash Social Welfare



(a) Summed Up Bandwidth, Egalitarian Social Welfare



(b) Outbound Bandwidth, Egalitarian Social Welfare



(c) Inbound Bandwidth, Egalitarian Social Welfare

Figure 7.10: Convergence of Two Agent SARSA with Egalitarian Social Welfare

to the preferences of the peers. The intervals of confidence are smaller for the total sum of allocated bandwidth compared to those for each direction alone. Thus, the resulting allocation has a tendency to be off the exact 2:1 ratio, but is likely to even out in total.

The results of the same experiment, but with the egalitarian social welfare measure instead can be seen in Figure 7.10. Here can be observed as well, that the agents successfully cooperate in distributing the biggest share of the quotas. Different from the Nash social welfare, the egalitarian one does not implement the preferences as radical. The average allocation is in favor of the peer with the high preference, but the difference between the values for agent 1 and agent 2 is less than with the Nash social welfare. Additionally, the likelihood for the agents to reach this allocation is slightly worse than is the case with the Nash measure. Interestingly, also here the agents have a good probability of evening out in sum.

7.2 Implementation Runtime

The runtime of the algorithm is linear. The factors influencing it are threefold. First of all is the number of peers, for which an agent is actively taking part in the allocation. This number determines how many single agent steps are performed in one global step. Second is the number of addresses each peer has, since this number influences the size of the matrices of parameter vectors and eligibility vectors, which are iterated in each step. Also the third factor, the RBF divisor influences the size of these matrices and the amount of state-features, which have to be calculated.

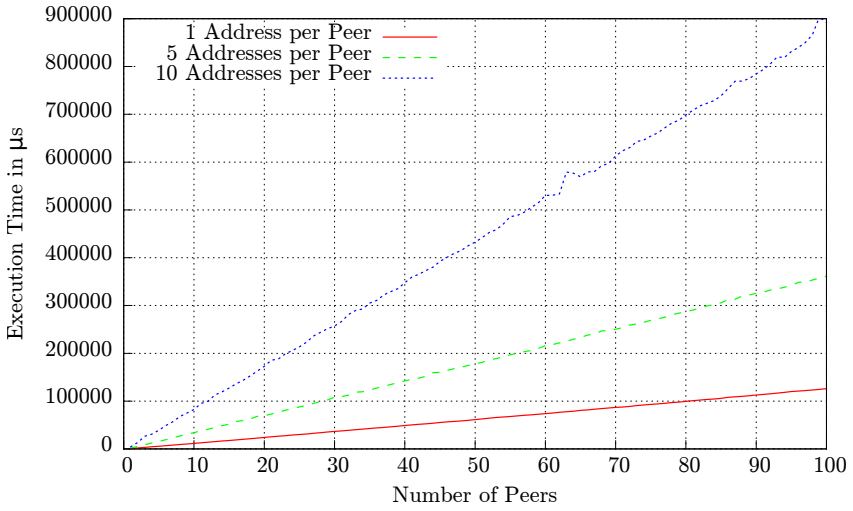


Figure 7.11: The Execution Time of One Global Step

In Figure 7.11 can be seen the execution time of one step with regard to the number of active peers on an Intel Core 2 Duo P8400 CPU with 2.26 Ghz running Ubuntu 13.10. The peers have 1, 5 and 10 addresses each. The RBF divisor is set to 50. The graph shows precisely the linear runtime. For a problem size of 60 peers with 10 addresses each the execution time is above half a second. For a practical employment of the approach it is necessary to find a good trade-off in the settings regarding the minimum step interval time and the RBF divisor. More frequent calculations with a higher resolution of the function approximation improve the approach.

Future work is to find boundaries of how these values can be set in favor of lower computational demands while still yielding useful results. The implementation of the approach was not done yet with an optimization of the runtime in mind. Thus, there is likely room for improvement in terms of efficiency.

8. Conclusion and Future Work

The thesis has evaluated how machine learning methodologies can be applied to the problem of automatic transport selection and resource allocation in decentralized peer-to-peer networks. Based on an evaluation of the different approaches, reinforcement learning stood out as most suitable, due to its distinct property to self-adaptively control the allocation.

It is based on the idea to divide the complicated state and action space by incorporating multiple agents, each managing a small part of it. The design supports both the well-known Q-learning and SARSA algorithms with eligibility traces. For enhanced generalization capabilities, the value function both algorithms estimate is approximated using a radial basis function network. An adaptive mechanism was introduced that allows to change the computational demands when appropriate.

The approach has been implemented in a real application, GNUnet, and the most important configuration parameters have been evaluated through the help of simulations.

A limitation of the approach is the dilemma of balancing exploration and exploitation inherent to reinforcement learning. The dynamic environment requires the capability of the approach to adapt to new situations. This can only be done by trying out new experiences, which can only be facilitated effectively through exploration. A steady allocation, though, is not possible when random manipulations to it are performed. Therefore, both steady allocations and fast reactions to a new environment exclude each other.

A further difficulty is the non-stationarity of the approach introduced by learning with multiple agents. The more agents are involved, the less significant is a single agent's action in the learning procedure. Therefore, the approach will have a practical upper bound to the number of agents with which it yields meaningful results.

The success of the approach depends on the computational resources, which are available to be spent on the control process. The approach offers trade-offs which can be made here. A less fine-granular approximation of the value function decreases the learning accuracy of the approach, but speeds up calculations. A longer step interval time decreases the speed of the approach to adjust the allocation, but will result in less computational demands.

Future work is to find parameter configurations which maximize the performance of the approach. The best configuration is likely not unique for all situations. Thus, heuristics might be adequate to perform the adjustment during operation and to decrease the number of parameters to set manually.

Apart from resetting and decaying the parameters for the exploration ratio, further adaptions of the approach to cope with the dynamics of the environment would be desirable. A first idea would be an adjustment of the learning rate α, depending on the changes in the environment. Thus, the agents could "forget" outdated experiences faster.

The efficiency of the approach might further be improved with substantial changes to the core algorithm, depending on future findings in the field of reinforcement learning. Worthwhile investigations are the use of learning mechanisms for continuous action spaces, as well as a sophisticated use of communication between the learning agents.

Finally, the extensive testing of the mechanism in the GNUnet Peer-to-Peer Framework is a key task to explore the practical limits of the approach.

Bibliography

[BBDS08] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics (SMC), Part C: Applications and Reviews*, 38(2):156–172, 2008.

[BC01] Marjory S Blumenthal and David D Clark. Rethinking the design of the internet: the end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology (TOIT)*, 1(1):70–109, 2001.

[BD95] Steven J Bradtke and Michael O Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Advances in Neural Information Processing Systems 7 (NIPS 7)*, volume 7, pages 393–401. The MIT Press, Cambridge, MA, USA, 1995.

[Bir04] Mauro Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, December 2004.

[CDE+06] Yann Chevaleyre, Paul E Dunne, Ulle Endriss, Jérôme Lang, Michel Lemaitre, Nicolas Maudet, Julian Padget, Steve Phelps, Juan A Rodriguez-Aguilar, and Paulo Sousa. Issues in multiagent resource allocation. *Informatica (Slovenia)*, 30(1):3–31, 2006.

[CZ13] Edwin KP Chong and Stanislaw H Zak. *An introduction to optimization*, volume 76. John Wiley & Sons, New York, NY, USA, 2013.

[Dob10] Felix Dobslaw. Recent development in automatic parameter tuning for metaheuristics. In *Proceedings of the 19th Annual Conference of Doctoral Students (WDS 2010)*, pages 54–63, 2010.

[Dou02] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems (IPTPS 2001)*, pages 251–260. 2002.

[FGR03] Ronaldo A Ferreira, Christian Grothoff, and Paul Ruth. A transport layer abstraction for peer-to-peer networks. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003 (CCGrid 2003)*, pages 398–405, 2003.

[Frä07] Kary Främling. Replacing eligibility trace for action-value learning with function approximation. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN 2007)*, pages 313–318, 2007.

[GCL04] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource allocation in the grid using reinforcement learning. In *Proceedings of the Third International*

Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004) - Volume 3, pages 1314–1315, 2004.

[Gol89] David Edward Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Boston, MA, USA, 1989.

[Gro11] Christian Grothoff. The free secure network systems group: Secure peer-to-peer networking and beyond. In *Proceedings of the First SysSec Workshop 2011 (SysSec 2011)*, pages 57–58, 2011.

[IK88] Toshihide Ibaraki and Naoki Katoh. *Resource allocation problems: algorithmic approaches*. The MIT Press, Cambridge, MA, USA, 1988.

[KE11] Tim Kovacs and Robert Egginton. On the analysis and design of software for reinforcement learning, with a survey of existing systems. *Machine learning*, 84(1-2):7–49, 2011.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research (JAIR)*, 4:237–285, 1996.

[KR94] S Sathiya Keerthi and B Ravindran. A tutorial survey of reinforcement learning. *Sadhana*, 19(6):851–889, 1994.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[Mar09] Stephen Marsland. *Machine Learning: An Algorithmic Perspective*. Chapman and Hall/CRC, 2009.

[Mia12] Victor Miagkikh. A survey of reinforcement learning and agent-based approaches to combinatorial optimization. http://www2.hawaii.edu/˜miagkikh/RLSurvey.pdf, May 2012.

[Mis12] Mustafa Misir. *Intelligent hyper-heuristics: a tool for solving generic optimisation problems*. PhD thesis, Katholieke Universiteit Leuven, September 2012.

[Mit97] Tom Mitchell. *Machine Learning*. McGraw-Hill, New York, NY, USA, 1997.

[MPI99] Victor V Miagkikh and William F Punch III. An approach to solving combinatorial optimization problems using a population of reinforcement learning agents. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, volume 2, pages 1358–1365, 1999.

[Ora01] Andrew Oram. *Peer-to-peer: Harnessing the Benefits of Disruptive Technologies*. O'Reilly Media, Inc., 2001.

[PSC+13] J. Panerati, F. Sironi, M. Carminati, M. Maggio, G. Beltrame, P.J. Gmytrasiewicz, D. Sciuto, and M.D. Santambrogio. On self-adaptive resource allocation through reinforcement learning. In *Proceedings of the 2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2013)*, pages 23–30, 2013.

[PT06] Alain Pétrowski and Eric Taillard. *Metaheuristics for hard optimization*. Springer, Berlin, Germany, 2006.

[SB98] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. The MIT Press, Cambridge, MA, USA, 1998.

[SRC84] Jerome H Saltzer, David P Reed, and David D Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

[SSSH94] Sandip Sen, Ip Sen, Mahendra Sekaran, and John Hale. Learning to coordinate without sharing information. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI 1994)*, pages 426–431, 1994.

[SST95] Andrea Schaerf, Yoav Shoham, and Moshe Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research (JAIR)*, 2:475–500, 1995.

[Tes95] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[TJDB06] Gerald Tesauro, Nicholas K Jong, Rajarshi Das, and Mohamed N Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Autonomic Computing, 2006. IEEE International Conference on*, pages 65–73, 2006.

[vH12] Hado van Hasselt. Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning*, pages 207–251. Springer, Berlin, Germany, 2012.

[vHW07] Hado van Hasselt and Marco A Wiering. Reinforcement learning in continuous action spaces. In *Proceedings of the 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)*, pages 272–279, 2007.

[Wau12] Tony Wauters. *Reinforcement Learning Enhanced Heuristic Search for Combinatorial Optimization.* PhD thesis, Katholieke Universiteit Leuven, November 2012.

[WM97] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[WS05] Klaus Wehrle and Ralf Steinmetz. *Peer-to-Peer systems and applications.* Springer, Berlin, Germany, 2005.