Institutsprojekt – A Distributed Q-Learning Approach for Adaptive Sleep Modes in 5G Networks

Im Rahmen des, im 4. Semester des Studiengangs Elektrotechnik, Informationstechnik und Technische Informatik, obligatorischen Institutprojekts. Hatten wir die Möglichkeit uns genauer mit dem Thema „A_Distributed Q-Learning Approach for Adaptive Sleep Modes in 5G Networks“ am Institut für intelligente Regelungssystem der RWTH auseinander zu setzten. Das Projekt wurde von uns, einem sechsköpfigen Team bearbeitet und von Oberingenieur Dr. ing. Michael Reyer und von Universitätsprofessor Dr.-Ing. Christian Ebenbauer betreut. Es basiert auf einem von Ali El-Amine*, Mauricio Iturralde*, Hussein Al Haj Hassan† and Loutfi Nuaymi* im IEEE publizierten Paper, welches den Namen „A_Distributed Q-Learning Approach for Adaptive Sleep Modes in 5G Networks“ trägt. Im Folgenden schildern wir unsere Erwartungen, Erfahrungen/Schwierigkeiten sowie Erkenntnisse, welche wir im Laufe des Projektes erlangt haben.

Ziele welche wir uns für das Institutsprojekt setzten

Zu Beginn sei zu erwähnen, dass wir uns zusammen mit unserem Betreuer Dr. Michael Reyer darauf verständigt haben, in einer größeren Gruppe (sechs Personen) zu arbeiten. Grund hierfür war es, dass eines der für uns wichtigsten Ziele es war Einblicke in die Arbeit als Ingenieure zu erlangen. Schon während unseres bisherigen Studiums stellte sich heraus, dass man auf Hilfe und Unterstützung von Kommilitonen angewiesen ist. Diese Erkenntnis lässt sich auch auf den Arbeitsalltag von Ingenieuren und Ingenieurinnen übertragen. Auch hier ist es essenziell, das Arbeiten in einem Team zu erlernen, unterschiedliche Erfahrungen zu machen sowie Schwierigkeiten als auch Vorteile festzustellen und zu erlernen. Daher standen wir alle dem Vorschlag, in einer größeren Gruppe zu arbeiten, positiv gegenüber.
Des Weiteren hielten wir es schon zu Beginn des Projekts für wichtig, nicht lediglich einen Programmcode für ein gegebenes Problem zu schreiben, sondern setzten auch einen Schwerpunkt darauf, das gegebene Problem wirklich zu verstehen, um uns mit der Problemstellung intensiv beschäftigen zu können. Wir planten daher schon anfangs Zeit dafür ein das gegebene Paper zu durchdringen, Sekundärliteratur zu recherchieren als auch ausführlich das Paper im kollektiv zu besprechen und unterschiedliche Lösungsansätze zu diskutieren.
Ferner war es für uns von größter Wichtigkeit, nicht nur die Problemstellung ausführlich zu verstehen, sondern ebenfalls uns intensiv mit dem maschinellen Lernen, speziell dem Q-Learning zu beschäftigen.

Zudem war natürlich die Implementation einer Simulationsumgebung in MATLAB ausschlaggebend, um unsere Überlegungen und Thesen zu testen, zu überprüfen und zu dokumentieren. Hier sahen wir es ebenfalls als Ziel an, auch abseits vom Thema unsere Fähigkeiten in MATLAB zu verbessern, da wir es als wichtigen Skill für das spätere Arbeitsleben als Ingenieur ansehen.

Abschließend war es uns auch wichtig, im Rahmen des Projekts viel Spaß miteinander zu haben und neue Freundschaften zu knüpfen, was sich im Nachhinein auch bestätigte.

Problemstellung

Jede Basisstation hat einen gewissen Energieverbrauch, welcher reduziert werden soll, möglichst ohne Performanz zu verlieren. Wir machen uns zu nutze, dass es auch Zeitpunkte gibt, an denen eine Basisstation keinen Consumer bedienen muss. Während diesen Perioden ist es somit nicht nötig dauerhaft im Ruhemodus zu bleiben, da dies Energie verbraucht. Wir führen daher sogenannte Schlafmodi ein, in welche jede Basisstation wechseln kann, um den Energieverbrauch runterzuschrauben. Unsere Simulation modelliert drei Schlafmodi, wobei die Energieersparnis vom ersten zum dritten Schlafmodus zunimmt. Auf der anderen Seite benötigen die Basisstationen nun eine gewisse Zeit, um aus diesem Energiesparmodus wieder in den aktiven Zustand zu wechseln und Consumer zu bedienen. Dies führt zu einer Zeitverzögerung der Bedienung (Delay).

Es stellt sich somit die Frage welchen Modus jeden Basisstation wählen soll, um bei möglichst guter Performanz (geringer Delay) möglichst wenig Energie zu verbrauchen.

Maschinelles Lernen (Q-Learning)

Einen Verbesserungsansatz, welchen wir im Rahmen des Projekts untersucht haben, um das gegebene Problem zu optimieren, ist die Optimierung mithilfe von maschinellem Lernen, genauer Q-Learning.

Maschinelles Lernen bzw. Reinforcement Learning ist ein Konzept zur Verbesserung von Algorithmen. Hierbei wird ein Agent ohne Kenntnisse seiner Umgebung und den Einfluss bzw. die Folgen seiner Aktionen mit dem Problem konfrontiert. Dieser Agent soll im Folgenden sein Handeln darauf trainieren, eine möglichst große Belohnung zu erhalten.

Zu erwähnen sei, dass es mehrere Arten gibt, das Reinforcement Learning System zu trainieren, wir uns im Rahmen des Projekts schwerpunktmäßig jedoch auf einen fundamentalen, relativ simplen Algorithmus konzentriert haben, dem Q-Learning.

Ziel des Q-Learning Ansatzes ist es eine möglichst optimale Policy zu erhalten. Hierunter versteht man das gelernte Verhalten des Agenten, welches im sagt, welche Aktion er in einem bestimmten Zustand in einer bestimmten Umgebung ausführen soll. Diese Policy wird mittels sogenannter Q-Values (Formel 3) in einer Matrix gespeichert, welche die zu erwartenden Belohnungen darstellen. Der Index m steht hierbei für die den jeweiligen Agenten (hier: Basisstation) und der Index t stellt den Zeitschritt dar.

Diese Matrix ist so aufgebaut, dass jede Zeile für eine bestimmte Beobachtung steht und jede Spalte für eine auszuführende Aktion. Diese Matrix bzw. ihre Q-Values werden während der Durchführung der Simulation stetig aktualisiert, um, wie bereits oben erwähnt, eine möglichst optimale Lösung eines gegebenen Problems zu erlangen. Die Aktualisierung geschieht wie in Formel 3 zu sehen, anhand des vorherigen Q-Values und den neu gemachten Erfahrungen bei der Erkundung der Umgebung des Agenten. Hierbei regelt der Parameter Alpha ($\alpha \in [0,1]$; Learning Rate), inwieweit bzw. wie schnell der Algorithmus aus der aktuellen Aktion Schlüsse zieht und der Diskontinuitätsfaktor Gamma ($\gamma \in [0,1]$) regelt, wie stark die Zukunft in unsere Bewertung einfließen soll. Je höher der Wert des Parameters desto größer ist der Einfluss der zukünftigen Werte.

Zu Beginn wird unsere Q-Matrix als Null-Matrix initialisiert und der Algorithmus springt in eine Schleife, welche bei dem Zeitparameter t gleich Null startet. Es wird gemäß Formel 4 eine aktuelle Aktion ausgewählt. Dies geschieht zunächst mittels der epsilon-greedy Methode.

Nachdem nun die aktuelle Aktion ermittelt wurde, wird diese ausgeführt. Es wird somit anhand Formel 3 einem Tupel s_m^t und a_m^t zum Zeitpunkt t ein zukünftiges s_m^{t+1} ermittelt. Das kleine m steht hier für die Basisstation.

Schließlich werden die Q-Values anhand Formel 3 aktualisiert.

$$Q(s_m^t, a_m^t) = Q(s_m^t, a_m^t) + \alpha \left[r_m^t + \gamma \max_a Q(s_m^{t+1}, a) - Q(s_m^t, a_m^t) \right] \quad (3)$$

Des Weiteren hat der Agent beim Q-Learning stets zwei Möglichkeiten des Lernens. Entweder folgt unser Agent den bereits erkundeten und für besten identifizierten Pfad weiter oder er beschreitet einen neuen Weg, um entweder noch nicht optimalen oder sich ändernden Bedingungen folgen zu können. Dieser Ansatz nennt sich Exploration.

Der Trade-Off zwischen diesen beiden Möglichkeiten ist in unserem Fall mittels der „epsilon-greedy“-Methode implementiert. Hierbei justiert ein Parameter Epsilon ($\epsilon \in [0,1]$) den Trade-Off zwischen dem weiteren Handeln nach der verfolgten Strategie und einer zufälligen Aktion.

$$a_m^t = \begin{cases} \underset{a}{\operatorname{argmax}}\, Q(s_m^t,\ a\ ), & \text{if } y > \epsilon \\ rand(\mathcal{A}), & \text{otherwise} \end{cases} \quad m = 1, \ldots, M. \tag{4}$$

Unser Epsilon Parameter gibt somit an, mit welcher Wahrscheinlichkeit der Algorithmus eine zufällige Aktion wählt oder mit welcher Wahrscheinlichkeit er einen bereits beschrittenen Pfad weiter folgt (4). Dabei wird y∈[0,1] jedes Mal, gleichverteilt zufällig, neu bestimmt. Somit sollten Epsilon (in Prozent) der Aktionen zufällig, also explorativ, sein. Die rand()-Funktion haben wir ebenfalls als gleichverteilte Zufallsfunktion implementiert.

Gestaltung der Simulation

Orientiert am Paper und in Absprache mit unserem Betreuer Herrn Reyer haben wir uns auf folgende Struktur und Funktionsweise unserer Simulation geeinigt.

Wir verwenden eine zweidimensionale Karte, auf welcher Consumer und Basisstationen platziert werden. Diese werden jeweils durch ein der Klasse spezifisches Icon angezeigt und geben Auskunft über deren Status.

Die Basisstationen bekommen im Vorhinein eine Position zugewiesen und sind in jeder unserer Simulationen gleich. Sie haben eine feste Bandbreite, welche gleich auf die Consumer aufgeteilt wird, dabei wird die Beanspruchung in Prozent gerechnet.
Die Schlafmodi werden wie im Paper angegeben übernommen. Somit haben wir drei verschiedene Schlafmodi, welche die auf der folgenden Abbildung beschriebenen Eigenschaften besitzen. Der Vierte bleibt ungenutzt, da er, wie im Paper auch festgestellt wurde, zu lang ist, als dass die Basisstation noch als aktiv von den Consumern angesehen werden kann. Außerdem hat dieser nur wenig zusätzlichen Energiegewinn.

Sleep level	Deactivation duration	Minimum sleep duration	Activation duration
SM 1	35.5 μs	71 μs	35.5 μs
SM 2	0.5 ms	1 ms	0.5 ms
SM 3	5 ms	10 ms	5 ms
SM 4	0.5 s	1 s	0.5 s

Hierbei entscheidet dann der Agent, falls die Basisstation im Ruhezustand ist, wie sie fortfahren sollte anhand des Q-Learning Algorithmus.

Die Anzahl der Consumer wird vorher festgelegt und diese anschließend über Ort und Zeit gleichverteilt. Beim Erscheinen werden sie dann automatisch der Basisstation mit kleinstem Abstand zugeordnet.
Jeder Benutzer fragt eine gewisse Menge Daten an, welche dann von der Basisstation mit der lieferbaren Datenrate übertragen werden. Die Übertragungsrate wird aber nicht nur durch die Menge der Consumer pro Basisstation skaliert, sondern auch durch die Übertragungsqualität, welche mit Hilfe der Signal-to-Inferference-plus-Noise Ratio (SINR) berechnet wird.

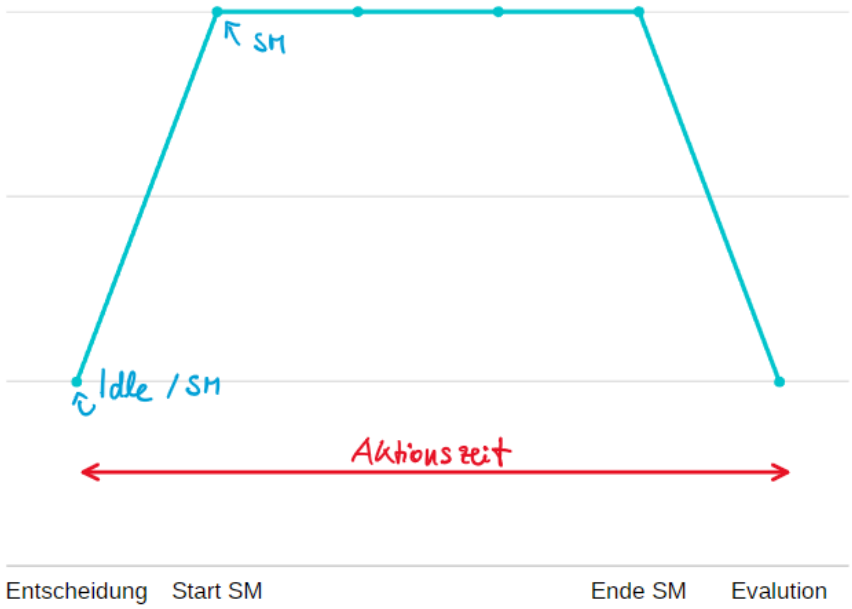
$$\mathrm{SINR}_m(k) = \frac{P_m^{\mathrm{Tx}} h_m(k)}{\sigma^2 + \sum_{m' \in \mathcal{S}, m' \neq m} P_{m'} h_{m'}(k)}$$

Hierbei ist P die Übertragungsleistung der jeweiligen Basisstation, bei uns sind diese alle gleich, h(k) ist die Kanalverstärkung, welche den Pfadverlust und „shadowing-effect“ repräsentiert. Zuletzt ist σ^2 die additive Rauschleistung.
Somit kommen wir zur gelieferten Übertragungsrate durch das Shannon-Hartley Theorem, welches wie folgt lautet:

$$R_m(k) = \alpha \times W \times \log_2(1 + \mathrm{SINR}_m(k))$$

dabei repräsentiert W die Bandbreite und α skaliert diese auf den Anteil, der genutzt wird, um die Daten zu übertragen.

In unserer Simulation haben wir schließlich den Q-Learning-Algorithmus wie oben beschrieben implementiert.
Jede Basisstation wird von einem individuellen, von den anderen Basisstationen unabhängigen Agenten verwaltet. Somit spiegelt sich die Anzahl der Basisstationen im Parameter m wider. Die Aktionen, welche der Agent durchführt, sind in unserer Simulation durch die Wechsel der Schlafmodi repräsentiert, während die Zustände durch den aktuellen Modus dargestellt werden. Je größer der Q-Wert einer Aktion in dem aktuellen Zustand, desto eher wird die Basisstation in diesen Schlafmodus tendieren zu wechseln. Ziel jeder Basisstation ist es den Reward, welcher nach jeder abgeschlossenen Aktion berechnet wird (Abb.), zu maximieren.



Wir definieren die Belohnung, analog wie im Paper, als die gewichtete Summe des Energiegewinns G und der zusätzlichen Verzögerung D, die beide aus dem während einer Episode gewählten Schlafmodus resultieren. (Abb.5)

$$r = (1 - \eta)G - \eta D \qquad (5)$$

Anhand des Parameters η wird der Trade-Off zwischen Energieverbrauch und Delay justiert. Der Delay ist hier die Zeit, die ein Consumer warten muss, bis seine Basisstation aus dem Schlafmodus erwacht ist und ihn wieder bedienen kann.
Abschließend sei zu erwähnen, dass wir uns dazu entschieden haben die Parameter, bis auf das Shadowing, für die Simulation analog zu denen aus dem gestellten Paper zu verwenden, siehe Tabelle (6).

Parameter	value
Antenna height	30 m
BS Tx Power	45 dBm
Bandwidth	20 MHz
Thermal noise	-174 dBm/Hz
Pathloss	128.1 +37.6 $\log_{10}$(d) dB
Shadowing	Log-normal (6 dBm)
User's arrival	Log-normal, λ_a = 1, υ= $\lambda_a/10$
Service type	file with mean=4 Mb
Scale parameter	$\lambda = 441.305$
Shape parameter	$k = 0.8$

(6)

Um den zeitlichen Ablauf zu simulieren, jedoch nicht diese Zeit warten zu müssen, entschieden wir uns, diese durch Events zu modellieren. Dies wird durch das triggern von Events realisiert. Ein Event ist dabei das Eintreffen eines Consumers, der Beginn und das Ablaufen seiner Übertragungszeit, aber auch das Aufwachen und Einschlafen einer Basisstation. Diese sorgen dann für eine Neuevaluierung der Situation und passt die Daten, zum Beispiel wann ein Consumer mit seiner Datenübertragung fertig wird, an, falls nötig. Basisstationen überprüfen dabei immer beim Ende einer Übertragung, ob es die Möglichkeit zu schlafen gibt und wenn ja, wie lange.

Der Weg zur Implementierung in MATLAB

Hier wurde uns die Wahl gelassen, welche Umgebung wir genau benutzen wollen, um unsere Simulation zu implementieren. Nachdem wir uns ausgetauscht hatten, wer welche Vorkenntnisse in der Programmierung besitzt und welche Sprache welche Vorteile bietet, wurde uns schnell klar, dass es auf die MATLAB, hinauslaufen wird. Ein zusätzlicher Vorteil, welcher die Benutzung von MATLAB unserer Meinung nach hat, ist, dass es sich um einen weit verbreiteten Industriestandard handelt, wodurch sich die Vertiefung/Verbesserung unserer MATLAB-Fähigkeiten doppelt lohnt.

Simulationsevents

Die Events sind wie folgt klassifiziert. Die Struktur dafür enthält 4 Schlüsselwörter: „name“ für den Namen des Events, „type“ für den Typ, „ind“ für den Index des durchgeführten Objekts sowie „time“ für den Zeitpunkt für den dieses Event getriggert wird. Alle Events werden anhand ihrer Zeitpunkte in einem Heap der Klasse „Map“ gespeichert, sodass diese nacheinander abgearbeitet werden.

Basisstation

Für die Klasse Basisstation gibt es 5 Events: „**deact“** für die Deaktivierung, „**sleep“** für den Schlafzyklus, „**act“** für die Aktivierung, und „**idle“** für den aktivierten Zustand ohne Benutzer und „**active“** für den Zustand mit Benutzern.

Bei „**deact**“ wird zunächst mit ε-greedy (Formel (1)) die Anzahl der darauffolgenden Schlafzyklen entschieden. Danach wird der Q-Wert von der letzten Episode mit Formel (2) aktualisiert und dann das Event „**sleep**“ nach der Deaktivierungsdauer vom aktuellen Schlafmodus getriggert.

Bei „**sleep**“ wird geprüft, ob es schon Consumer in der Bufferlist gibt, oder der Schlafzyklus zu Ende ist. Wenn dies der Fall ist, dann wird „**act**“ für diesen Zeitpunkt getriggert. Sonst wird der Zähler für die Anzahl der Schlafzyklen um 1 dekrementiert und ein Event „**sleep**“ nach einer Runde dieses Schlafmodus getriggert.

Bei „**act**“ wird zunächst entschieden, ob die Basisstation aufwacht, wenn die Bufferlist nicht leer ist, oder im niedrigeren Schlafmodus bleibt. Danach wird mit ε-greedy (Formel (1)) die Anzahl der nächsten Schlafzyklen bzw. Idlezyklus wie oben entschieden und der Q-Wert aktualisiert. Wenn die Bufferlist nicht leer ist, wechselt die Basisstation zu „**active**“, sonst wenn der aktuelle Schlafmodus eins ist, dann wechselt die Basisstation zu „**idle**“, sonst bleibt die Basisstation bei „**sleep**“.

Bei „**idle**“ wird nichts gemacht, falls die Basisstation schon aktiviert wurde, sonst wird analog zu „**sleep**“ jedes Mal der Zähler um 1 dekrementiert.

Bei „**active**“ wird zunächst die Verzögerung und der Energiegewinn berechnet, um den Q-Wert zu aktualisieren. Danach wird „**obs**“ von **Map** getriggert, wobei alle Basisstationen und Consumer beobachtet und aktualisiert werden.

Consumer

Für Consumer gibt es nur „**arrive**“ für die Ankunft eines Consumers. Zunächst wird dieser Consumer zu der nächste Basisstation zugeordnet. Wenn die Basisstation noch schläft, wird der Consumer zur Bufferlist hinzugefügt, sonst wird er zur Servelist hinzugefügt. Danach wird ein Beobachtungsevent für die Karte getriggert.

Map

„**obs**“, dass Beobachtungsevent für die ganze Karte, ist das einzige Event für Map, wobei alle Consumer und Basisstationen einmal beobachtet werden. Die Datennachfrage und die Datenrate wird neu berechnet und das Übertragungsende jedes Consumers wird hier eingeschätzt, zu welchem Zeitpunkt die Beobachtung dieser Karte wieder getriggert wird. Außerdem werden alle Basisstationen deaktiviert, bei denen es keine Nutzer mehr gibt.

Hier was irgendwas zu Ergebnissen

Um den Einfluss unseres Algorithmus bewerten zu können haben wir einerseits unsere Simulation mit dem Q-Learning und andererseits ohne unseren Algorithmus durchgeführt und schließlich

Referenzen:

(1) Ali El-Amine*, Mauricio Iturralde*, Hussein Al Haj Hassan† and Loutfi Nuaymi* A Distributed Q-Learning Approach for Adaptive Sleep Modes in 5G Networks, 2019 IEEE Wireless Communications and Networking Conference (WCNC)

(2) Fatma Ezzahra Sale Management of advanced sleep modes for energy-efficient 5G networks