Dynamic Computation and Adjustment of Channel Hopping Sequences for Cognitive Radio Networks Based on Quality Metrics

Markus Engel
Technische Universität Kaiserslautern
engel@cs.uni-kl.de

Prof. Dr. Reinhard Gothzein
Technische Universität Kaiserslautern
gotzhein@cs.uni-kl.de

Abstract

In cognitive radio networks (CRNs), secondary users employ spectrum sharing with license holders (primary users) of a frequency band. Whenever a primary user uses his frequency band, all secondary users must vacate the channel and meet on a new one. Many network access schemes for CRNs employ proactive channel hopping, where all or a subset of secondary users switch channels from time to time in order to reduce interference with primary users. This calls for dynamic channel hopping sequences (schedules) shared among and applied by secondary users.

In this paper, we present a new approach for the dynamic computation and adjustment of channel hopping sequences for CRNs. Based on channel quality, we introduce quality metrics for the number of channel utilizations, and for the channel hopping sequence. Based on these metrics, we dynamically compute and adjust optimal schedules, i.e. assignments of channels to time slots. Our approach gives preference to channels of higher quality, by using them more frequently, and yields schedules that are less prone to channel failure, e.g. due to usage by a primary user. Furthermore, dynamic schedule adjustments typically lead to small changes, keeping the operation of secondary users more stable.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication

General Terms

Reliability, Algorithms

Keywords

Channel Hopping, Hopping Sequences, Cognitive Radio

1 Introduction

To enable the usage of licensed spectrum by secondary users, i.e. users without license, a new communication paradigm called cognitive radio (CR) has been put forward. Following [1], CR technology will enable nodes to detect and assess spectrum unused by the license owner (spectrum sensing), to select the best available channels (spectrum management), to coordinate access to these channels with other users (spectrum sharing), and to leave a used channel when a primary user, i.e. a license holder, is detected (spectrum mobility).

CR requires that all network nodes permanently sense the spectrum, to determine the qualities of a set of channels dynamically. While the original idea is that the CR always operates in the best available channel, this idea can be generalized by stipulating that nodes may use a set of channels with good quality by performing channel hopping. Thus, if the quality of a used channel drops, the channel can be left without jeopardizing network operation, as the remaining channels are still usable. Therefore, they can replace the left channel, which results in a modified hopping sequence. Furthermore, if the quality of an unused channel rises, it can be upgraded to a used channel and be integrated into the hopping sequence. In summary, this channel usage scheme provides redundancy, reducing the impact of channel failure, which, e.g., may occur when a primary user is detected.

For channel hopping, several problems are to be solved. First, there is the need for network-wide time synchronization. This enables the structuring of time into consecutive intervals, consisting of a sequence of (time) slots, to which available channels can be assigned. Thus, nodes can switch between channels in a synchronized way. Second, a channel hopping sequence has to be determined and updated dynamically, and to be exchanged among all nodes.

In this paper, we will present our approach for the dynamic computation and adjustment of channel hopping sequences (schedules) for CR networks (CRNs). We assume that spectrum management permanently assesses channel qualities, and provides a list of usable channels and their current qualities. Using this list, we introduce quality metrics Σ and Ω for the number of channel utilizations, and for channel hopping sequences, respectively. We then show that w.r.t. these metrics, optimal solutions exist, and how they can be computed. Further, we show how dynamic changes of channel qualities can be handled by adjusting these solutions dynamically, typically leading to small schedule changes, to keep the operation of secondary users more stable. Schedules are computed from scratch only if an adjustment would

International Conference on Embedded Wireless Systems and Networks (EWSN) 2018
14–16 February, Madrid, Spain

ISBN: 978-0-9949886-2-1

result in solution that is too far from the optimum.

The paper is structured as follows: In Section 2, we introduce our system model. Section 3 and Section 4 present quality metrics for the number of channel utilizations and for channel hopping sequences, respectively, elaborate on optimal solutions, and introduce algorithms for their computation and dynamic adjustment. Section 5 is on related work, Section 6 presents conclusions and future work.

2 System model

In this section, we briefly describe our system model. We assume a non-empty set of channels $\mathcal{C}$ that can be used for communication depending on their current quality. A channel is defined by physical parameters and signal processing techniques, such as center frequency, bandwidth, modulation and coding. In this work, we assume $\mathcal{C}$ to be finite and fixed, but we can also extend our model such that, e.g. the set of channels can change dynamically.

Each channel $c \in \mathcal{C}$ is assigned a quality $\mathfrak{q}_c \in [0,1]$ where higher values denote better quality. The definition of channel quality is not addressed in this paper – we assume it depends on primary user activity on the channel. In addition, this may be combined with static channel properties (e.g. throughput as derived from the channel's physical parameters). We also assure that channels with special value $\mathfrak{q}_c = 0$ won't be used. The qualities of all channels are listed in a vector $\vec{\mathfrak{q}} \in [0,1]^{|\mathcal{C}|}$.

Time is structured into consecutive time intervals, each consisting of a fixed number of n_{slot} slots of equal duration d_{slot}. The idea is that each slot of a time interval will be assigned one channel, and that this pattern is repeated until the schedule changes. Slot duration d_{slot} depends on various aspects: it should be chosen long enough such there is enough usable time for communication, and short enough such that channel failure does not lead to a long period of inactivity.

Our objective is to dynamically compute and adjust schedules $\vec{\mathfrak{s}} \in \mathcal{C}^{n_{slot}}$, assigning each time slot $n \in \{1,\dots,n_{slot}\}$ a channel $\mathfrak{s}_n \in \mathcal{C}$. The schedule is supposed to adhere to an underlying utilization $\vec{\mathfrak{u}} \in \mathbb{N}^{|\mathcal{C}|}$ such that a channel $c \in \mathcal{C}$ is used $\mathfrak{u}_c$ times in $\vec{\mathfrak{s}}$. The utilization corresponds to channel qualities $\vec{\mathfrak{q}}$, such that channels with higher qualities are used more often. As qualities are subject to change over time, we have to adjust the utilization and thus the schedule.

3 Channel Utilization

Our first objective is to use channels according to their quality. This way, we assure that better channels are used more frequently but also take care of the fact that a channel's quality may degrade any time. In this section, we first give some preliminaries to formalize our objectives and introduce a quality metric that qualifies a given utilization by means of the best and worst utilization possible. We then give algorithms to compute an optimal utilization and to converge from any utilization to an optimal one in small steps. Finally, we refine this algorithm for better convergence.

3.1 Preliminaries

Absolute quality values as given by $\vec{\mathfrak{q}}$ are not expressive when considered on their own: a value of $\mathfrak{q}_c = 0.4$ might be considered bad if the qualities of all other channels are much higher – on the other hand, it might be considered good if all other channels yield worse values. Therefore, we relate a channel's quality to the overall quality of all channels by defining a relative quality:

$$\vec{\mathfrak{r}} := \frac{\vec{\mathfrak{q}}}{\sum_{c\in\mathcal{C}} \mathfrak{q}_c}$$

From this, we calculate the fair share (a hypothetical optimum) of slots:

$$\vec{\mathfrak{u}}^\star := \vec{\mathfrak{r}} \cdot n_{slot}$$

Ideally, in an infinite run, channel c should be used $\mathfrak{r}_c$ of the total time. This problem is perfectly solvable, if n_{slot} is not fixed and all qualities are rational (i.e. $\vec{\mathfrak{q}} \in \mathbb{Q}^{|\mathcal{C}|}$): in this case, $\vec{\mathfrak{r}}$ is rational, too, and we can find a cycle length n_{slot}, such that $\vec{\mathfrak{u}}^\star$ consists of natural numbers only and is therefore a valid utilization.

Example 1. Let $\mathcal{C} = \{c_1, c_2, c_3\}$ denote the set of channels and let $\vec{\mathfrak{q}} = \begin{pmatrix} 1 & \frac{3}{8} & \frac{1}{8} \end{pmatrix}$ denote their qualities. From this, we obtain the relative qualities $\vec{\mathfrak{r}} = \begin{pmatrix} \frac{2}{3} & \frac{1}{4} & \frac{1}{12} \end{pmatrix}$. If we were allowed to choose n_{slot} freely, we would find $n_{slot} = \mathrm{lcm}(3,4,12) = 12$ to be a good value, as it is divided by all denominators of $\vec{\mathfrak{r}}$. The fair share $\vec{\mathfrak{u}}^\star = (8 \quad 3 \quad 1)$ is of natural numbers only and thus a perfect solution in this example.

However, using the approach given in the example, n_{slot} could grow big (especially if the denominators in $\vec{\mathfrak{r}}$ are coprime), and n_{slot} is considered fixed in our system model in Section 2. For this reason, we have to expect that $\vec{\mathfrak{u}}^\star$ is real valued in general. Our goal therefore is to synthesize a utilization $\vec{\mathfrak{u}} \in \mathbb{N}^{|\mathcal{C}|}$ that sums up to n_{slot} and approximates the fair share $\vec{\mathfrak{u}}^\star$. This problem is related to the Apportionment Problem [3], where n_{slot} seats[1] in the House of Parliament are apportioned among a set of states $\mathcal{C}$, according to the relative population $\mathfrak{r}_c$ of each state. As restriction, seats can only be apportioned as a whole (i.e. a natural number) whereas the relative population of a state is considered rational. There have been many solutions for this problem, all with different properties or goals on how to approximate proportionality in a fair way.

According to our objective, we use the apportionment method by Hamilton [3], which minimizes the error sum:

$$\Phi(\vec{\mathfrak{u}}) := \sum_{c\in\mathcal{C}} |\mathfrak{u}_c - \mathfrak{u}_c^\star| \tag{1}$$

Instead of Hamilton's Method, other apportionment methods can be used, such as the ρ-rounding method [7, 11], which can unproportionally prefer channels with higher quality.

3.2 Quality Metric

To assess the quality of the numbers of channel utilizations $\vec{\mathfrak{u}}$, we introduce a quality metric $\Sigma(\vec{\mathfrak{u}})$ based on the error sum $\Phi(\vec{\mathfrak{u}})$. Given this metric, the objective then is to determine optimal vectors $\vec{\mathfrak{u}}^+$, i.e. vectors minimizing the error sum. This metric then can also be used to determine how far the currently used vector $\vec{\mathfrak{u}}$ is away from any optimal vector $\vec{\mathfrak{u}}^+$ and to define a threshold when the currently used vector is to be adjusted.

The problem with the error sum as given in (1) is that it depends on the fair share $\vec{\mathfrak{u}}^\star$ and thus on the qualities $\vec{\mathfrak{q}}$.

[1] We use our notions when referring to the Apportionment Problem.

Using only that notion, we cannot decide whether some utilization $\vec{u}$ is the best one possible for the given qualities or more particular how far $\vec{u}$ is away from an optimal utilization. We therefore define the following metric $\Sigma(\vec{u})$, which classifies a given utilization between the best and worst utilization possible for the given channel qualities and normalizes these extremes to the interval $[0,1]$.

$$\Sigma(\vec{u}) = \begin{cases} 1 & \text{if } \Phi^{min} = \Phi^{max} \\ 1 - \frac{\Phi(\vec{u}) - \Phi^{min}}{\Phi^{max} - \Phi^{min}} & \text{otherwise} \end{cases} \tag{2}$$

To determine $\Sigma(\vec{u})$, we need the error sum of the best and worst possible utilization, denoted by Φ^{min} and Φ^{max}, respectively. Instead of obtaining them by enumeration, we use efficient algorithms.

The worst possible utilization is obtained by assigning the worst channel to all slots and calculating the error sum over that utilization. From this idea, we can deduce $u_{c^-} = n_{slot}$ for a channel $c^- \in \mathcal{C}$ which has the lowest quality and consequently $u_c = 0$ for all other channels. Note that this is not only valid for Hamilton's Method, but can be applied to any apportionment method described in [11] to obtain the worst possible utilization. For our chosen objective function $\Phi(\vec{u})$, the error sum of such a utilization is

$$\Phi^{max} = 2 \cdot \left(n_{slot} - \min_{c \in \mathcal{C}} u_c^\star \right) \tag{3}$$

For Φ^{min}, we calculate an optimal solution $\vec{u}^+$ as given in the next section and have $\Phi^{min} = \Phi(\vec{u}^+)$.

3.3 Computation of Optimal Solutions

The authors of [11] focus upon objective functions of apportionment methods and developed an algorithm for their optimization. In their work, they require an error function $\varphi_c \colon \mathbb{N} \to \mathbb{R}$ for each channel $c \in \mathcal{C}$, such that the sequence $H_c(u) := \varphi_c(u) - \varphi_c(u-1)$ for each positive $u \in \mathbb{N}$ is non-decreasing. Then the objective function consisting of the sum over all channels is minimized by their algorithm MINIMALSOLUTIONS.

We therefore choose $\varphi_c(u) := |u - u_c^\star|$ and, by summing, obtain the error function $\Phi(\vec{u})$ as in (1) and thus can apply Hamilton's Method using their algorithm. Note that $\varphi_c(u)$ denotes the error of using channel c u times and $H_c(u)$ denotes the change in the error when using channel c u times instead of $u-1$ times.

The algorithm MINIMALSOLUTIONS works as follows:

1. setup the matrix $\mathcal{H} = (h_{u,c})$ with $h_{u,c} = H_c(u)$.
2. select the n_{slot} smallest entries of $\mathcal{H}$.
3. for each channel c, let u_c be the number of selected entries in the column vector for c in $\mathcal{H}$.

Example 2. Let $\mathcal{C} = \{c_1, c_2, c_3\}$ be the set of channels. The qualities of these channels shall be $\vec{q} = \left(\frac{19}{50} \quad \frac{13}{100} \quad \frac{69}{20}\right)$. If we apportion $n_{slot} = 6$ slots using MINIMALSOLUTIONS, we obtain matrix $\mathcal{H}$:

$$\mathcal{H} = \begin{pmatrix} -1 & -0.3 & -1 \\ \underline{-0.8} & 1 & -1 \\ 1 & 1 & \underline{-1} \\ 1 & 1 & 0.1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

The 6 smallest numbers of $\mathcal{H}$ are above the line, and we obtain $\vec{u}^+ = (2 \quad 1 \quad 3)$ as an optimal utilization.

Note that Hamilton's Method assures $u_c = 0$ for $q_c = 0$, i.e. unusable channels won't be used. This is, however, not necessarily true for other Apportionment Methods, or their error function $\varphi_c(u)$ might be undefined upon $q_c = 0$. This can be fixed by extending MINIMALSOLUTIONS by a prefiltering step in which $u_c = 0$ is set for all channels having q_c and then run MINIMALSOLUTIONS afterwards on the remaining channels. By doing so, we can also define a minimum absolute threshold q^T (or a relative threshold[2] r^T) such that only channels having $q_c \geq q^T$ (respectively $r_c \geq r^T$) are considered.

Note also that any Apportionment Method fails, if there is no usable channel. In this case, however, no communication is possible, so we don't need to calculate any utilization.

3.4 Dynamic Adjustments

In the original context of the Apportionment Problem, the optimal solution is always calculated after a census and stays valid until the next one. The old apportionment is discarded then and the new results of the census are used to calculate a new optimum. Likewise, the quality of channels is constantly in flux, and we seek to always have an optimal utilization. Therefore, we could recalculate the optimal utilization every time the quality of channels changes and use this new solution until quality changes again. However, we face the problem that the optimal utilization (and consequently the schedule) may change frequently. The new schedule must be disseminated in the network and if a node misses one update, its schedule might be completely different if we discarded the old and recalculated a new optimal one. We therefore allow the utilization to be suboptimal temporarily and improve it towards the new optimum stepwise. In this section, we define an algorithm to produce a sequence of utilizations that steadily converges to the new optimum.

When improving the utilization, we aim for small updates that increase the utilization's quality. The smallest modification of a utilization is when one channel is used one time more often and another channel is used one time less. This is formalized in the next definition:

Definition 1. Given a utilization $\vec{u}$, an *atomic repair* of $\vec{u}$ is given by a pair of channels $(c^-, c^+) \in \mathcal{C}^2$. The *application* of an atomic repair (c^-, c^+) to $\vec{u}$ is denoted as $\vec{u}[c^-, c^+]$ and defined as follows:

$$\vec{u}[c^-, c^+] := \vec{u} + \vec{b}_{c^+} - \vec{b}_{c^-}$$

where $\vec{b}_c$ denotes the vector $(0 \ \ldots \ 0 \ 1 \ 0 \ \ldots \ 0)$ that is 1 at the position corresponding to channel c and 0 otherwise.

[2] In the Apportionment Problem, this is known as the n-percent hurdle.

We show now that we obtain an optimal $\vec{u}^+$ from an arbitrary $\vec{u}$ by repeatedly using atomic repairs:

1: **choose** c^- **such that** $H_{c^-}(u_{c^-}) = \max_{c\in\mathcal{C}} H_c(u_c)$
2: **choose** c^+ **such that** $H_{c^+}(u_{c^+}+1) = \min_{c\in\mathcal{C}} H_c(u_c+1)$
3: **if** $H_{c^-}(u_{c^-}) \le H_{c^+}(u_{c^+}+1)$ **then**
4: **stop** ▷ $\vec{u}$ is optimal
5: **else**
6: $\vec{u} \leftarrow \vec{u}[c^-, c^+]$ ▷ update utilization
7: **end if**

THEOREM 1. *Let $(\vec{u}^0, \ldots, \vec{u}^k)$ be a sequence of utilizations, obtained by repeatedly applying the above algorithm. Let also $\vec{u}^k$ be the first utilization which hits line 4. When needed, let (c_i^-, c_i^+) denote the atomic repair (c^-, c^+) chosen in step $0 \le i < k$. Then*

1. *$\vec{u}^k$ is optimal.*
2. *$\forall 0 \le i, j < k.\ c_i^- \neq c_j^+$, i.e. no channels' utilization will be increased and decreased while converging.*
3. *$\forall 0 \le i < j \le k.\ \Sigma(\vec{u}^j) > \Sigma(\vec{u}^i)$, i.e. the quality of the obtained utilizations is strictly increasing.*
4. *$\forall 0 \le i < k.\ \forall (c^-, c^+) \in \mathcal{C}^2.\Sigma(\vec{u}^{i+1}) \ge \Sigma(\vec{u}^i[c^-, c^+])$, i.e. of all possible atomic updates, the chosen update maximizes the quality change in each step.*

PROOF. 1. First, have $H_c(u_c+1) \ge H_{c'}(u_{c'})$ for all $c, c' \in \mathcal{C}$ if and only if $\vec{u}$ is optimal from [11]. This is equivalent to $\min_c H_c(u_c+1) \ge \max_c H_c(u_c)$, which is exactly the condition in line 3, thus $\vec{u}^k$ is optimal.

2. For all previous steps $i < k$, we thus also have that $\vec{u}^i$ is not optimal. We examine 3 cases:
   a) For any step $0 \le i < k$, we have $c_i^- \neq c_i^+$ because otherwise $\vec{u}^i$ was optimal.
   b) Assume that for $0 \le i < j < k$ we would have $c_i^- = c_j^+$. For better readability we denote that channel by $c^\star := c_i^-$.
   We w.l.o.g. assume that $j - i$ is minimal, i.e. step i was the last decrease before the increase in step j for channel $c^\star$. From that, we have $u_{c^\star}^i = u_{c^\star}^j + 1$. Furthermore, by induction we have $\max H_c(u_c^i) \ge \max H_c(u_c^j)$. Then $\min H_c(u_c^j + 1) = H_{c^\star}(u_{c^\star}^j + 1) = H_{c^\star}(u_{c^\star}^i) = \max H_c(u_c^i) \ge \max H_c(u_c^j)$. Then $\vec{u}^j$ is optimal, contradicting $j < k$, thus $c_i^- \neq c_j^+$.
   c) By analogous reasoning we also have $c_i^+ \neq c_j^-$.
   Together, we conclude $c_i^- \neq c_j^+$ for all $0 \le i, j < k$.

3. Note again, that $\vec{u}^i$ is not optimal for $i < k$ and thus $H_{c^-}(u_{c^-}^i) > H_{c^+}(u_{c^+}^i + 1)$. Then have
$$\Phi(\vec{u}^i) - \Phi(\vec{u}^{i+1}) = H_{c^-}(u_{c^-}^i) - H_{c^+}(u_{c^+}^i + 1) > 0$$
   While this holds for any $0 \le i < k$, we generalize $\Phi(\vec{u}^i) > \Phi(\vec{u}^j)$ for $0 \le i < j \le k$ by induction. From that and by substituting into (2) we conclude $\Sigma(\vec{u}^j) > \Sigma(\vec{u}^i)$.

4. From the previous proof and the choice of c^- and c^+ we have
$$\begin{aligned}\Phi(\vec{u}^i) - \Phi(\vec{u}^{i+1}) &= H_{c^-}(u_{c^-}^i) - H_{c^+}(u_{c^+}^i + 1) \\ &= \max H_c(u_c^i) - \min H_c(u_c^i + 1)\end{aligned}$$
   From that and by substituting into (2) we conclude that $\Sigma(\vec{u}^{i+1}) - \Sigma(\vec{u}^i)$ is maximized by the choice of c^- and c^+.

□

Example 3. Let $n_{slot} = 6$ and $\mathcal{C} = \{c_1, c_2, c_3\}$ be the set of channels and let their quality be as in Example 2. Now consider that channels c_1 and c_2 gain in quality while c_3's quality decreases, such that their new quality vector is $\vec{q}' = \left(\frac{29}{50} \quad \frac{33}{100} \quad \frac{29}{100}\right)$. We obtain the new matrix $\mathcal{H}'$ as follows:

$$\mathcal{H}' = \begin{pmatrix} -1 & -1 & -1 \\ -1 & -0.3 & 0.1 \\ -0.8 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

The solid line in the matrix depicts the optimal solution $\vec{u}^+ = (3 \quad 2 \quad 1)$ for $\vec{q}'$. The dashed line gives us the old (i.e. our current) utilization as in Example 2. If we run the algorithm, it will select $c^- = c_3$ since it has the highest number (marked blue) just above the dashed line and it will select $c^+ = c_1$ since it has the lowest number (marked red) just below the dashed line. After running the algorithm again (this time having $c^- = c_3$ and $c^+ = c_2$) we will actually obtain the minimum solution.

LEMMA 1. *Let $\vec{u}$ be any utilization and $\vec{u}^+$ be the optimal utilization for a given channel quality vector $\vec{q}$ and n_{slot}. Then we converge from $\vec{u}$ towards $\vec{u}^+$ in exactly*

$$n_{runs}(\vec{u}) := \frac{\sum_{c\in\mathcal{C}} |u_c^+ - u_c|}{2} \tag{4}$$

runs. Furthermore, $n_{runs}(\vec{u})$ is bound by n_{slot} for any $\vec{u}$.

PROOF. From the second proposition in Theorem 1 we know that each channel's utilization is only increased or decreased in all runs. Thus, for a channel $c \in C$ to converge from u_c to u_c^+, c must be chosen exactly $|u_c^+ - u_c|$ as c^- (or c^+, respectively) times. Since in every run, a pair (c^-, c^+) of channels is chosen, the overall sum is divided by 2 and we obtain (4).

In the worst case, there is a channel c which is used in all slots (i.e. $u_c = n_{slot}$) but should never be used (i.e. $u_c^+ = 0$) in the optimal utilization. Since we only transfer one utilization in each run, we'd have to run the algorithm $u_c - u_c^+ = n_{slot}$ times to finally reach the optimum. □

It is worth noticing that the algorithm converges to the new optimal utilization from *any* utilization. This way, if the optimal utilization changes while converging, the algorithm adapts to the new situation and converges to the new optimum.

3.5 Refinements

In the previous section, we introduced the concept of atomic repairs. Whenever we run one loop of the algorithm to improve the utilization, we get a pair of channels (c^-, c^+) whose utilization will be decreased (respectively increased) by one. Consider that channel quality may suddenly drop, e.g. by detecting a primary user. We will now address two problems arising in this situation by giving an example:

Example 4. Let $\mathcal{C} = \{c_1, \ldots, c_4\}$ be the set of channels and let $n_{slot} = 12$ be the number of slots to be assigned. Let the current utilization be $\vec{u} = (2 \quad 2 \quad 4 \quad 4)$. Now consider that suddenly the quality of channel c_4 drops, and we get a new optimal utilization $\vec{u}' = (4 \quad 5 \quad 3 \quad 0)$ from matrix $\mathcal{H}'$:

$$\mathcal{H}' = \begin{pmatrix} -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & 1 \\ -1 & -1 & -0.7 & 1 \\ -1 & -1 & 1 & 1 \\ 0.3 & -0.6 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

We can see that channel c_1 and c_2 should be used more often, while c_3 and c_4 should be used less often in the new optimum. When we calculate an atomic repair (c^-, c^+), we see $H_{c_3}(u_{c_3}) = H_{c_4}(u_{c_4}) = 1$, and thus, both c_3 and c_4 are equally suited when we choose c^-. Intuitively we want c_4 to be the only candidate in this case since that channel's utilization is further away from its new optimum.

The source of this problem is the choice of the objective function Φ as in (1): no matter which channel we choose for c^-, the error will decrease by $H_{c^-}(u_{c^-}) = 1$. A way to fix this issue is choosing another objective function which gives us an $H_c(u_c)$ which *strictly* increases in u_c. From [11], we know that Hamilton's Method does not only minimize the sum of the absolute value of $u_c - u_c^\star$, but it is also the unique method which minimizes any ℓ_p-norm (for $p \geq 1$) of the error vector $(\vec{u} - \vec{u}^\star)$. Thus, we can also choose

$$\Phi^p(\vec{u}) := \sum_{c \in \mathcal{C}} |u_c - u_c^\star|^p \tag{5}$$

for any $p > 1$ as objective function and still have Hamilton's Method, i.e. we still follow our objective: that each channel c should be used $\mathfrak{r}_c$ of the total time. However, when it comes to improving the utilization, we see that distances of a channel's current and optimal utilization have a linear[3] influence on $H_c(u_c)$ and thus, channels which have a higher loss in utilization are preferred when it comes to choosing c^- (and analogously c^+).

Example 5. Consider the same problem as above, but now use $\Phi^2(\vec{u}) = \sum_{c \in \mathcal{C}} (u_c - u_c^\star)^2$ to calculate matrix $\mathcal{H}'$:

$$\mathcal{H}' = \begin{pmatrix} -7.7 & -8.6 & -4.7 & 1 \\ -5.7 & -6.6 & -2.7 & 3 \\ -1.7 & -2.6 & 1.3 & 7 \\ 0.3 & -0.6 & 3.3 & 9 \\ 2.3 & 1.4 & 5.3 & 11 \\ 4.3 & 3.4 & 5.3 & 11 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

We can see that our algorithm now chooses $c^- = c_4$ and $c^+ = c_2$ and thus complies with our intuition.

A second problem is that the use of atomic repairs – although motivated by us for having smaller changes in utilizations – may be obstructive in a scenario as described above: Even in Example 5 where we used $\Phi^2(\vec{u})$ as objective function, we'd still need 5 updates until the utilization of channel c_4 finally reaches 0. This means that in between, this channel would still be scheduled. So far, when doing an improvement, we only applied one atomic repair to obtain a new intermediate utilization, which now results in slow convergence. To get around this, we advice to apply several atomic repairs per improvement and let the number of repairs be determined by means of metric Σ or the by n_{runs}, i.e. number of atomic repairs needed to converge as given in (4).

[3] for p=2

4 Channel Hopping Sequence

Our second objective is to compute channel hopping sequences (schedules) that distribute channel usage uniformly, based on the optimal number of channel utilizations obtained for the metric Σ. This way, channel hopping sequences are less prone to channel quality changes, e.g. channel failures due to detection of a primary user. In this section, we start with preliminaries, motivate the objective function, introduce a quality metric Ω, elaborate on optimal solutions, conceive two heuristics for their computation, and experimentally assess them.

4.1 Preliminaries

In this section, we introduce some formal foundations to finally give an objective function to assess the quality of a schedule.

4.1.1 Optimal Channel Reuse Distances

A simple approach is to use channels in blocks as in the following example:

Example 6. Let $n_{slot} = 6$ and $\mathcal{C} = \{c_1, c_2, c_3\}$ be the set of channels, and $\vec{u} = (2 \quad 1 \quad 3)$ their utilization (cf. Section 3). Then one possible schedule is

slot i	1	2	3	4	5	6
$\mathfrak{s}_i$	c_1	c_1	c_2	c_3	c_3	c_3

Although such schedules obviously adhere to the underlying utilization $\vec{u}$, we discourage them: Consider, that in the 4th slot a primary user (PU) appears. Then the nodes, detecting the PU must wait to inform the other nodes for three consecutive slots. Moreover, all nodes, that do not detect the presence of the PU might interfere with him as the schedule is not updated. If, on the other hand, slot 5 was assigned a channel other than c_3, we could use that slot to announce the new situation (i.e. disseminate a new schedule), thus having shorter reaction times and less interference with the PU.

Our objective here is to distribute reuses of one channel equally over all slots. For any used channel c, the optimal distance (a hypothetical optimum) between two uses would be

$$\delta_c^\star := \frac{n_{slot}}{u_c} \tag{6}$$

We face two problems from that: first, $\delta_c^\star$ might not be a natural number for each channel. Second, even if all $\delta_c^\star$ are natural numbers, we might not be able to synthesize a schedule that assigns each channel with distance $\delta_c^\star$, as shown in the following example:

Example 7. Let $n_{slot} = 6$, $\mathcal{C} = \{c_1, c_2, c_3\}$ be the set of channels, and $\vec{u} = (2 \quad 1 \quad 3)$ their utilization, then $\vec{\delta}^\star =$

$(3 \quad 6 \quad 2)$ is their optimal distance. We could start our schedule $\vec{\mathfrak{s}}$ like this:

slot i	1	2	3	4	5	6
$\mathfrak{s}_i$	c_1	c_3	c_2	x		

We face a problem in slot 4: according to $\vec{\delta}^\star$, we would have to schedule c_1 as well as c_3 simultaneously. Even if we started our schedule in another way, we would face the same problem in another slot. From the Chinese Remainder Theorem, we can conclude that in this example, there is always a conflict: since $\delta_1^\star = 2$ and $\delta_3^\star = 3$ are coprime, there is a conflicting slot, no matter in which slots we schedule c_1 and c_3 first.
In conclusion, there are utilizations for which no schedule exists that meets equal distances of reuses.

4.1.2 Formalization of Reuse Distances

We now formally define the notion of reuse distance. Given a set of channels $\mathcal{C}$, a utilization $\vec{\mathfrak{u}}$, and a schedule $\vec{\mathfrak{s}}$, we consider, for each channel c, the slots where c is assigned. Then, for $c \in \mathcal{C}$ and $1 \leq i \leq \mathfrak{u}_c$, the function $\delta(c,i)$ denotes the distance of the i^{th} and $(i+1)^{\text{th}}$ assignment, given in number of slots. Note that for any $c \in \mathcal{C}$, $\delta(c,\mathfrak{u}_c)$ is the distance of the last assignment of the current interval and the first assignment of the next interval.

We explain this with an example:
Example 8. Let $n_{slot} = 6$, $\mathcal{C} = \{1,2,3\}$ be a set of channels, $\vec{\mathfrak{u}} = (2 \quad 3 \quad 1)$ a utilization of $\mathcal{C}$, and let $\vec{\mathfrak{s}} = (1 \quad 2 \quad 2 \quad 3 \quad 1 \quad 2)$ denote a schedule, complying with $\vec{\mathfrak{u}}$.

We extend the schedule by the next cycle, up to until each channel has been first scheduled in that cycle:

slot i	1	2	3	4	5	6	1	2	3	4
$\mathfrak{s}_i$	1	2	2	3	1	2	1	2	2	3

Channel 1 — 4 — 2
Channel 2 — 1 — 3 — 2
Channel 3 — 6

We can read distance $\delta(c,i)$ for each channel $c \in \mathcal{C}$ and each reuse $1 \leq i \leq \mathfrak{u}_c$ from the diagram:

i	1	2	3
$\delta(1,i)$	4	2	
$\delta(2,i)$	1	3	2
$\delta(3,i)$	6		

$\delta(c,i)$ could be formalized as follows:

$$S(c,i) := \{n \mid i \leq n \leq n_{slot} \wedge \mathfrak{s}_n = c\} \tag{7}$$

$$s(c,i) := \begin{cases} \min S(c,1) & \text{if } i = 1 \\ \min S(c, s(c,i-1)+1) & \text{if } 2 < i \leq \mathfrak{u}_c \\ s(c,1) + n_{slot} & \text{if } i = \mathfrak{u}_c + 1 \end{cases} \tag{8}$$

$$\delta(c,i) := s(c,i+1) - s(c,i) \quad \text{for } 1 \leq i \leq \mathfrak{u}_c \tag{9}$$

$S(c,i)$ yields the set of slots to which channel c is assigned, starting with usage i. Then $s(c,i)$ defines an order on this set, i.e. $s(c,i)$ yields the slot, in which c is scheduled the i^{th} time. Finally, $\delta(c,i)$ is the distance between usage i and usage $i+1$ of channel c, for $1 \leq i \leq \mathfrak{u}_c$.

4.1.3 Objective Function

We now formally define an objective function, which will then be used to obtain the quality metric for schedules. Our first attempt was to use an error function $\Psi^1(\vec{\mathfrak{s}})$ that sums up the absolute differences between each $\delta(c,i)$ and the hypothetical optimum $\delta_c^\star$ over all channels and their assignments:

$$\Psi^1(\vec{\mathfrak{s}}) = \sum_{c \in \mathcal{C}} \sum_{i=1}^{\mathfrak{u}_c} |\delta(c,i) - \delta_c^\star| \tag{10}$$

However, this error function yields undesirable results, as shown by the following example:
Example 9. Let $n_{slot} = 14$, $\mathcal{C} = \{1,2,3,4\}$ and $\vec{\mathfrak{u}} = (7 \quad 3 \quad 2 \quad 2)$, yielding $\vec{\delta}^\star = \left(2 \quad 4.\overline{6} \quad 7 \quad 7\right)$ as optimal distances.

Then, under the objective function (10), the following schedules $\vec{\mathfrak{s}}_A^+$ and $\vec{\mathfrak{s}}_B^+$ are both optimal:

$$\vec{\mathfrak{s}}_A^+ = (3 \quad 1 \quad 1 \quad 1 \quad 4 \quad 2 \quad 1 \quad 3 \quad 1 \quad 2 \quad 1 \quad 4 \quad 1 \quad 2)$$
$$\vec{\mathfrak{s}}_B^+ = (3 \quad 1 \quad 2 \quad 1 \quad 4 \quad 1 \quad 2 \quad 1 \quad 3 \quad 1 \quad 2 \quad 1 \quad 4 \quad 1)$$

Looking at the distances, we have

i	1	2	3	4	5	6	7
$\delta_A(1,i)$	1	1	3	2	2	2	3
$\delta_A(2,i)$	4	4	6				
$\delta_A(3,i)$	7	7					
$\delta_A(4,i)$	7	7					

and

i	1	2	3	4	5	6	7
$\delta_B(1,i)$	2	2	2	2	2	2	2
$\delta_B(2,i)$	4	4	6				
$\delta_B(3,i)$	8	6					
$\delta_B(4,i)$	8	6					

We notice that in $\vec{\mathfrak{s}}_A^+$, channels 3 and 4 keep the optimal distance for each assignment, while distances of channel 1 deviate from the optimum in several positions. In $\vec{\mathfrak{s}}_B^+$ this is reversed: now the distances of channel 1 are always optimal, while the distances of channels 3 and 4 deviate. Intuitively, $\vec{\mathfrak{s}}_B^+$ is the better schedule, because there is no channel used in consecutive slots.

To remedy this problem, we modify our objective function, by adding more weight to bigger errors (by squaring the difference) as well as to channels with higher utilization (by dividing by the optimal distance):

$$\Psi^2(\vec{\mathfrak{s}}) := \sum_{c \in \mathcal{C}} \sum_{i=1}^{\mathfrak{u}_c} \frac{(\delta(c,i) - \delta_c^\star))^2}{\delta_c^\star} \tag{11}$$

Using $\Psi^2(\vec{\mathfrak{s}})$, schedule $\vec{\mathfrak{s}}_B^+$ of Example 9 is still optimal, whereas schedule $\vec{\mathfrak{s}}_A^+$ is not.

4.2 Quality Metric

To assess the quality of a schedule $\vec{\mathfrak{s}}$, we introduce a quality metric $\Omega(\vec{\mathfrak{s}})$ based on the error sum $\Psi^2(\vec{\mathfrak{s}})$. Given this metric, the objective then is to determine optimal schedules $\vec{\mathfrak{s}}^+$, i.e. vectors minimizing the error sum. Similar to $\Sigma(\vec{\mathfrak{s}})$, this metric can then be used to determine how far the currently used vector $\vec{\mathfrak{s}}$ is away from any optimal vector $\vec{\mathfrak{s}}^+$, and to define a threshold when the currently used schedule is to be adjusted. The structure of $\Omega(\vec{\mathfrak{s}})$ is analogous to $\Sigma(\vec{\mathfrak{u}})$: for

the given utilization $\vec{\mathfrak{u}}$, it classifies a given schedule $\vec{\mathfrak{s}}$ between the best and the worst schedule and normalizes these extremes to the interval $[0,1]$.

$$\Omega(\vec{\mathfrak{s}}) := \begin{cases} 1 & \text{if } \Psi^{2,min} = \Psi^{2,max} \\ 1 - \frac{\Psi^2(\vec{\mathfrak{s}}) - \Psi^{2,min}}{\Psi^{2,max} - \Psi^{2,min}} & \text{otherwise} \end{cases} \tag{12}$$

To compute $\Omega(\vec{\mathfrak{s}})$, we need the error sum of the best and worst possible schedule, denoted by $\Psi^{2,min}$ and $\Psi^{2,max}$, respectively.

The worst possible schedule is obtained by following the principles of Example 6: we place all reuses of a channel in consecutive slots. Then, for each channel c and for each reuse $1 \leq i < \mathfrak{u}_c$, we have distance $\delta(c,i) = 1$, and $\delta(c,\mathfrak{u}_c) = n_{slot} - (\mathfrak{u}_c - 1)$. From that, we conclude the error sum

$$\Psi^{2,max} = \frac{1}{n_{slot}} \sum_{c \in \mathcal{C}} \left((\mathfrak{u}_c - 1) \cdot (n_{slot} - \mathfrak{u}_c)^2 \right) \tag{13}$$

For a true $\Psi^{2,min}$ we can enumerate all possible schedules and determine the minimum of their error sums. Given a utilization $\vec{\mathfrak{u}}$, the number of possible schedules can be determined by using the multinomial coefficient:

$$\binom{n_{slot}}{\mathfrak{u}_1; \mathfrak{u}_2; \ldots; \mathfrak{u}_{|\mathcal{C}|}} = \frac{n_{slot}!}{\prod_{c \in \mathcal{C}} \mathfrak{u}_c!} \tag{14}$$

The number of schedules increases fast with the number of slots and channels because of the growth of the factorial function.

Alternatively, we can give a lower bound on the minimum error or use different heuristics to derive good upper bounds as shown in the next sections.

4.3 Lower Bound for Minimal Error Sum

Following our objective, we try to distribute reuses of channels over all slots, such that $\delta(c,i) \approx \delta_c^\star$ for each channel c and all reuses $1 \leq i \leq \mathfrak{u}_c$. From the previous section, we know that there is a problem if $\delta_c^\star$ is not a natural number.

We isolate now our view to only one fixed channel c and solve that problem. In the optimal case, all $\delta(c,i)$ would be equal to $\delta_c^\star$, thus we would have $\mathfrak{u}_c$ times the distance $\delta_c^\star$. In the actual case, we have distances $\delta(c,1), \ldots, \delta(c,\mathfrak{u}_c)$, summing up to $\mathfrak{u}_c \cdot \vec{\delta}_c^\star = n_{slot}$ and all being most proportional to $\delta_c^\star$ where proportionality is formalized by means of the inner sum of (11), i.e.

$$E(c) = \sum_{i=1}^{\mathfrak{u}_c} \frac{(\delta(c,i) - \delta_c^\star)^2}{\delta_c^\star} \tag{15}$$

We can, in fact, interpret this as another Apportionment Problem: the House size is now the sum of all optimal distances (i.e. n_{slot}) and the population of each state is denoted by $\delta_c^\star$. We see that this is a special case of the Apportionment Problem in which each state has the same population. The Apportionment Method which actually minimizes the inner error sum $E(c)$ is the Method of Webster [12], which can be calculated by means of the algorithm MINIMALSOLUTIONS again – but in this special case, it turns out that all Apportionment Methods described in [11] (including Webster and Hamilton) calculate the same solutions, i.e. (up to order) the same distances $\delta(c,i)$. Even more, each distance of channel c will either be $\lfloor \delta_c^\star \rfloor$ or $\lceil \delta_c^\star \rceil$, i.e. the optimal distance is only round down for $down_c$ times and round up for up_c times for all $\delta(c,i)$. The numbers $down_c$ and up_c can be easily calculated without actually running any Apportionment Method at all:

$$up_c := \mod(n_{slot}, \mathfrak{u}_c) \tag{16}$$

$$down_c := \mathfrak{u}_c - up_c \tag{17}$$

The meaning of these numbers can now be explained: In order to minimize the inner sum $E(c)$ for a channel c, $down_c$ distances of all $\delta(c,i)$ must be equal to $\lfloor \delta_c^\star \rfloor$ and up_c distances must be equal to $\lceil \delta_c^\star \rceil$. From that rule, we can now calculate the minimal inner error sum for c:

$$E^{min}(c) = \frac{up_c \cdot (1 - \delta_c^\star + \lfloor \delta_c^\star \rfloor)}{\delta_c^\star} \tag{18}$$

Now we assume that this minimization is possible for all channels simultaneously. For such schedules, the outer sum would also be minimal and from that we infer a minimum:

$$\Psi^{2,min,lower} = \sum_{c \in \mathcal{C}} E^{min}(c) \tag{19}$$

However, in Example 7 we have already seen that there are utilizations for which no schedule exists in which the previous assumption applies. Thus, for these utilizations, $\Psi^{2,min,lower}$ does not serve as a true minimum: it is only a lower bound, which cannot be reached in these cases. When we use $\Psi^{2,min,lower}$ for normalization in $\Omega(\vec{\mathfrak{s}})$, we might therefore get values below 1, even if $\vec{\mathfrak{s}}$ is the best schedule according to our objective.

4.4 Heuristic Computation of Optimal Solutions

In this section, we introduce two heuristics which synthesize a high quality schedule for a given utilization.

4.4.1 Heuristic H_1

The idea of heuristic H_1 is founded in the most inner term of error sum (11): If we decide to use some channel c at some slot n, we calculate a local error

$$L(c,n) := \frac{((n - last_c) - \delta_c^\star)^2}{\delta_c^\star}$$

where $last_c$ denotes the slot in which c was last used.

In this heuristic, at each slot n we essentially choose the channel c which minimizes the local error $L(c,n)$. There are, however, three things to consider:

First, we must be aware that the local error has a parabolic shape, especially the error will decrease up to some slot and increase from that point. If we always choose the channel minimizing $L(c,n)$, the local error for a channel whose error is increasing will further increase in the upcoming slots and will perhaps no more be scheduled. We solve that issue by first determining which channels local error is increasing and of those, we choose the one with the maximum local error. We have also found out that it is advisable to take the local error $L(c,n+1)$ *of the next slot* here, because all local errors of channels which already have an increasing local error (except for the one chosen), increase further in the next slot.

Finally, and only if there are no channels with an increasing local error, we take all remaining channels into account and choose the one with the minimum local error.

Second, this procedure does not guarantee that the resulting schedule $\vec{\mathfrak{s}}$ actually adheres to the given utilization $\vec{\mathfrak{u}}$. We resolve this by counting the number of reuses of each channel at each slot and denote these numbers by the vector $\vec{\mathfrak{u}}^{cur}$. This way, when it comes to decide for a channel, we skip all the channels which already reached their given utilization, i.e. channels for which $\mathfrak{u}_c^{cur} = \mathfrak{u}_c$ holds. By doing so, we can guarantee that each channel c appears exactly $\mathfrak{u}_c$ times in $\vec{\mathfrak{s}}$.

Third, the local error is undefined as long as a channel has not been scheduled once. Since schedules are cyclic, it actually doesn't matter when a channel is scheduled first, thus our idea is to set the local error to zero as long as it is not yet scheduled. This will make sure that is has a good chance to be scheduled in an early slot. We will revisit this decision in a refinement in Section 4.6.

The heuristic H_1 is formalized in the following algorithm:

1: $\vec{\mathfrak{u}}^{cur} \leftarrow (0 \quad \cdots \quad 0)$ ▷ initialize usage count
2: **for** $n = 1 \ldots n_{slot}$ **do**
3: **for** $c \in \mathcal{C}$ with $\mathfrak{u}_c^{cur} = 0$ **do**
4: $last_c \leftarrow n - \delta_c^\star$ ▷ reset unused channels
5: **end for**
6: $\mathcal{C}_{\text{increasing}} \leftarrow \{c \in \mathcal{C} \mid (n - last_c) \geq \delta_c^\star \wedge \mathfrak{u}_c^{cur} < \mathfrak{u}_c\}$
7: $\mathcal{C}_{\text{decreasing}} \leftarrow \{c \in \mathcal{C} \mid (n - last_c) < \delta_c^\star \wedge \mathfrak{u}_c^{cur} < \mathfrak{u}_c\}$
8: **if** $\mathcal{C}_{\text{increasing}} \neq \{\}$ **then**
9: **choose** $c \in \operatorname{argmax}_{c \in \mathcal{C}_{\text{increasing}}} L(c, n+1)$
10: **else**
11: **choose** $c \in \operatorname{argmin}_{c \in \mathcal{C}_{\text{decreasing}}} L(c, n)$
12: **end if**
13: $\mathfrak{s}_n \leftarrow c$
14: $last_c \leftarrow n$
15: $\mathfrak{u}_c^{cur} \leftarrow \mathfrak{u}_c^{cur} + 1$
16: **end for**

In the loop beginning in line 3, we reset the last usage $last_c$ of each channel that has not been scheduled yet, such that $L(c,n) = 0$ holds for them. In lines 6 and 7, we determine the channels with an increasing (respectively decreasing) local error. We also filter those channels which reached their targeted utilization. Then, we choose a suitable channel as described above. Finally, from line 13 until the end of the loop, we schedule the channel and update the state.

4.4.2 Heuristic 2

The second heuristic H_2 is motivated by an example where the heuristic H_1 gives a suboptimal schedule:

Example 10. In Figure 1 we see the local error $L(c,n)$ for two channels c_1 and c_2. At slot 1, the local errors of all channels are decreasing. According to Heuristic H_1, we choose c_1 since it has the least local error $L(c_1,1) = 0.5$. In the next slot, we will choose c_2 having a local error of $L(c_2,2) \approx 0.07$, thus, together we have an error of about 0.57 for these two slots.

If we choose c_2 for slot 1 and c_1 for slot 2, we would have local errors $L(c_2,1) \approx 0.53$ and $L(c_1,2) = 0$, thus an error of about 0.53 for these two slots, which would be less.

From that example, we came up with a criterion for a pair of channels, which decides which of the two channels to take first, such that the local error sum of both channels (scheduled in consecutive slots) is guaranteed to be minimal. Essentially, in slot n, we shall decide for channel c_i over channel c_j whenever

$$L(c_i,n) + L(c_j,n+1) \leq L(c_j,n) + L(c_i,n+1) \tag{20}$$

holds. This can be rewritten and we get

$$L(c_i,n) - L(c_i,n+1) \leq L(c_j,n) - L(c_j,n+1) \tag{21}$$

Note that the numbers left and right of the inequation depend on n and c_i (c_j respectively) only. Therefore, we can calculate $L(c,n) - L(c,n+1)$ for each channel c and choose the channel that minimizes this expression.

In this heuristic, we also do not distinguish channels with an increasing (respectively decreasing) local error sum, since channels with an increasing local error implicitly have priority over channels with a decreasing local error.

The algorithm itself is the same as H_1, except that lines 6 through 12 are substituted by

6: $\mathcal{C}_{\text{notAtLimit}} \leftarrow \{c \in \mathcal{C} \mid \mathfrak{u}_c^{cur} < \mathfrak{u}_c\}$
7: **choose** $c \in \operatorname{argmin}_{c \in \mathcal{C}_{\text{notAtLimit}}} L(c,n) - L(c,n+1)$

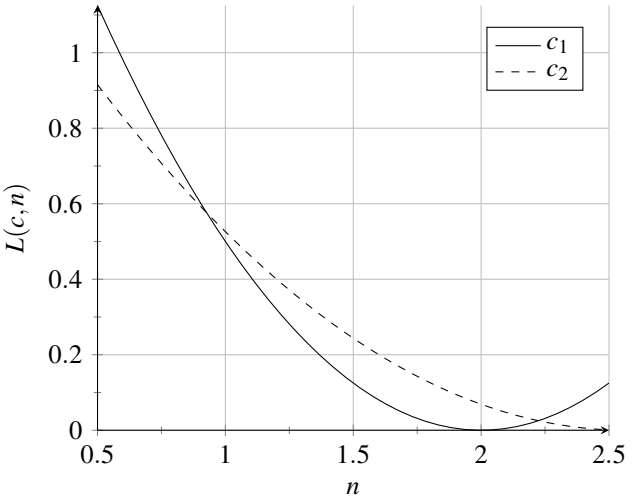

Figure 1. Local error $L(c,n)$ of two channels c_1 and c_2.

4.5 Evaluation of Lower Bound and Heuristics

In order to assess the quality of our heuristics, we first created a test set T_1, consisting of all utilizations with up to 50 slots and up to 10 channels. Since the set of possible schedules for any utilization $\vec{\mathfrak{u}}$ does not depend on the order of the elements of $\vec{\mathfrak{u}}$, we further restricted T_1 to a subset T_1^o in which we imposed an order on the utilizations. This is formalized in the following equations:

$$T_1 := \{(\mathfrak{u}_1 \quad \cdots \quad \mathfrak{u}_n) \in \mathbb{N}^n \mid n \leq 10 \wedge \sum_{i=1}^{n} \mathfrak{u}_i \leq 50\}$$

$$T_1^o := \{(\mathfrak{u}_1 \quad \cdots \quad \mathfrak{u}_n) \in T_1 \mid \forall_{1 \leq i < j \leq n} \mathfrak{u}_i \leq \mathfrak{u}_j\}$$

There are about 500 000 utilizations in T_1^o with a total of 10^{44} schedules, making it computational infeasible to find a true minimum by enumeration. Because of this, we first

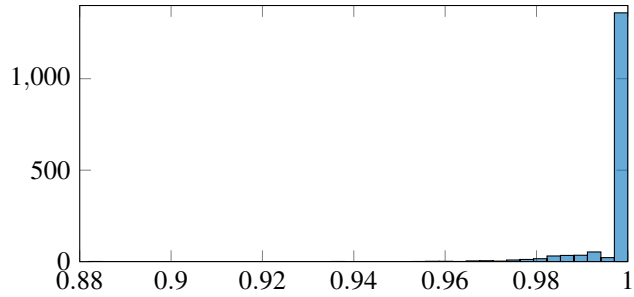

Figure 2. Histogram of true minimum normalized to the Lower Bound.

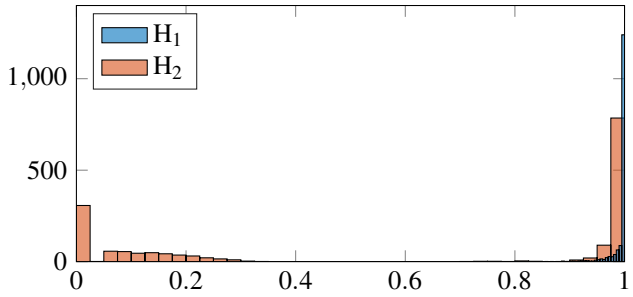

Figure 3. Histogram of heuristics H_1 and H_2 normalized to true minimum and maximum.

created a subset of T_1^o, consisting of all utilizations with $1 \leq n_{slot} \leq 14$ slots and $1 \leq |\mathcal{C}| \leq 10$ channels. We added another subset of T_1^o including all utilizations with up to 1 000 000 schedules. This resulted in a test set we name T_2, with about 1600 utilizations. With this set, we conducted several different evaluations in order to assess the quality of the lower bound and of our heuristics.

4.5.1 Quality of the Lower Bound

For our test set T_2, for which we know the exact minimum by enumeration, we determined the lower bound of the minimum $\Psi^{2,min,lower}$ by means of (19). In this evaluation we examine what happens if we used the lower bound $\Psi^{2,min,lower}$ instead of the true minimum in the normalization, i.e. if we assumed $\Psi^{2,min} = \Psi^{2,min,lower}$ in (12). This way, we get a value of exactly 1 if and only if the lower bound meets the true minimum, which means that there is a schedule for which the assumptions made in Section 4.3 are satisfied. Otherwise, we get a value lower than 1, meaning that $\Omega(\vec{s})$ indicates that there is a better schedule for the underlying utilization $\vec{u}$, although we know that $\vec{s}$ is the best schedule.

The results are plotted in a histogram shown in Figure 2. We can see that about 85% of the calculated lower bounds are exact. We can also see that nearly 99% have a quality more than or equal to 0.97. The biggest outlier (with quality 0.88) was found for the utilization $(2 \quad 3 \quad 6)$, which we already discussed in Example 7.

From that, we conclude that it is quite likely that all channels can be scheduled under the assumptions in Section 4.3 and also that the lower bound is quite close to the true minimum.

4.5.2 Quality of the Heuristics

In this evaluation, we examine the quality of both heuristics H_1 and H_2. For each utilization of our test set T_2, we create 2 schedules by running both heuristics and calculate the error sums of these schedules. We normalize them by means of (12), so schedules of different utilizations become comparable.

The results are plotted in a histogram shown in Figure 3. We see that H_1 generally performs better than H_2: it found an optimal schedule for 70% of the utilizations, 97% of all schedules had a normalized error of 0.95 or better and the error of the worst schedule is at 0.8 for one schedule.

In comparison, H_2 found an optimal schedule for only 35% of the utilizations, and 55% had a normalized error of 0.95 or better. In addition, the schedules calculated by H_2 had a normalized error of 0.2 or worse for 37% of all tested utilizations. However, heuristic H_2 found a better schedule for 5% of the utilizations. We conclude that the problem which motivated H_2 does not occur often.

4.6 Refinements

For both heuristics, we have seen that our choice of when to schedule any channel first is not expedient in all cases: The first use of a channel might be delayed too long, up to the point when it is only scheduled because other channels reached their limit. In the problematic cases, we reset the local error of these channels to 0, which is not sufficient to finally schedule the channel.

We therefore tried two other strategies (called NoReset and Iterative), which can also be combined. Since all of the strategies can improve but also worsen the original heuristics (i.e. there are utilizations for which the modified versions finds worse schedules), we advice to calculate the original heuristics as well as the modified versions and take the better schedule. Because of this, we will now only characterize the improvements made when applying the strategies. Note also that the heuristics have a linear runtime in $n_{slot} \cdot |\mathcal{C}|$, thus calculating several heuristics is still a leightweight computation.

In the NoReset strategy, we do not reset the local error but initially pretend that each channel was scheduled in slot 0, i.e. the slot before the first slot. Then in each slot, the error of yet unscheduled channels rise and we increase chances that they are finally chosen.

When applying NoReset on H_1 (we denote the modified algorithm by H_1^{NoReset}), we could not really see an improvement: Of all tested utilizations, it found a better schedule only on 1.7%. In H_2^{NoReset} we have seen better results: In 36% of all test utilizations, a better schedule could be found.

In the Iterative strategy, we first calculated a schedule using H_1 (respectively H_2) and then recorded the latest reuses of all channels in that schedule. We then run the heuristic again and initialize $last_d$ to the corresponding slot in the previous cycle. Note that this iteration can also be repeated, but on our test set, the schedule was no more improved after the first iteration.

When comparing $H_1^{\text{Iterative}}$ to H_1, we found a better schedule in 9.7%. Doing the same comparison for $H_2^{\text{Iterative}}$ vs. H_2, we found a better schedule in 42%.

Note that the two strategies can also be combined, i.e. we

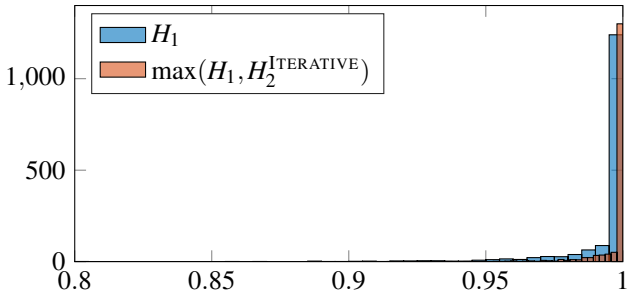

Figure 4. Histogram of H_1 and the better of H_1 and $H_2^{\text{ITERATIVE}}$.

first calculate a schedule by using the NORESET strategy and calculate another iteration by using the ITERATIVE strategy.

This results in a total of 8 heuristics, but for practicality, we advice to use a combination of H_1 and $H_2^{\text{ITERATIVE}}$ which achieves a total improvement of 11.5% compared to H_1 alone. This comparison is also depicted in the histogram in Figure 4. With these two algorithms, we found the optimal schedule in 79% of all test cases and 99.6% of the test cases had a normalized error of 0.95 or better.

4.7 Dynamic Adjustments

In Section 3.4, we addressed the fact that channel qualities are constantly changing over time. This also changes the optimal utilization, and we gave an algorithm that calculates an atomic update, i.e. a pair of channels (c^-, c^+) whose utilization has to be decreased (respectively increased) in order to converge to the new optimum.

Changing the current utilization has of course an impact on the current schedule. When we apply an update to improve the current utilization, we have two goals: First, the schedule must adhere to the new utilization again. This can easily be assured: In an atomic update (c^-, c^+) we know that c^- has been used at least once, i.e. there is at least one position n in $\vec{\mathfrak{s}}$ such that $\mathfrak{s}_n = c^-$ holds. If we exchange that position by c^+, we implicitly decrease the utilization of c^- and increase the utilization of c^+ by 1, thus, the resulting schedule will comply with the updated utilization.

Second, we also want the new schedule to be optimal by means of metric Ω. The problem is that for channels c^- and c^+, also the optimal distance $\delta^\star_{c^-}$ and $\delta^\star_{c^+}$ changes. We use the degree of freedom left by the previously described action: Whenever there are several positions in which c^- is used (i.e. whenever $\mathfrak{u}_{c^-} > 2$), we may choose any of them. Our strategy here is to choose the position, for which the new error sum $\Psi^2(\vec{\mathfrak{s}}')$ is least.

This strategy can still decrease the quality of the schedule. There are two ways to counteract on this: First, we can swap two channels in the schedule and thereby increase the quality of the schedule, but so far we have not come up with a satisfying strategy to choose them. Second, we can define a threshold on metric Ω and calculate a new schedule when the threshold is reached.

5 Related Work

The authors of [9] give an overview of common control channel (CCC) design in cognitive radio networks. As the name suggests, a CCC is a channel primarily used for exchanging control messages between nodes of a CRN. This is, however, not a limitation: some access schemes use the CCC only for exchanging control messages, while some others also use it for data communication – either way, there must be some common channel for the communicating nodes. In their survey, they group the access schemes by different design decisions.

First, they give some examples of a *sequence based* approach: Each node hops on its own schedule, and for communication, nodes must first find each other. The schedules are therefore optimized to reduce the expected time to rendezvous (TTR), i.e. the time until two stations finally meet on the same channel. Most designs [4, 6, 2] do not take channel qualities into account when creating schedules, and therefore there is also no need to change the schedule. At least, the authors of [2] give an outlook on that issue in their future work. Another sequence based approach is given in [5], where a ranking of channels, based on primary user activity, defines how often channels are used. However, there are three major differences to our scheduling approach: First, they do not use a channel's quality but its ranking which can unproportionally distort the utilizations of channels: for example, when having 5 channels, the best one is used in 45% of all 55 slots[4], even if its quality is just slightly better than that of the seconds best channel, which is used in 29% of all slots. Second, although bad channels are used rarely in their schedules, there is no chance that those are not scheduled at all. Third, the order of channels in their schedules is drawn randomly, so they do not care for consecutive reuse. However, they adapt the schedule whenever the ranking changes. In this case, they replace the whole schedule instead of making small changes – on the other hand, each station hops on its own schedule, thus making only small changes is not an advantage here.

Besides these sequence based approaches, the authors of [9] also give some examples on *neighborhood coordination* and *clustering* approaches. In these approaches, a set of nodes share the same CCC, and, provided that channel hopping is employed, use the same schedule. In [8], a channel hopping scheme is applied within a cluster of nodes. The hopping sequence is a random order of all available channels which is then repeated – the quality of channels has thus no influence on the utilization. If a primary user appears, the schedule is adapted by removing the channel from the schedule, which also causes the schedule length to shorten. In [10], the authors use a hopping sequence while discovering neighbors. Very much as in [5], they use a ranking of channels based on their quality. They use the ranking not to determine the utilization exactly, but use it as a probability when choosing from the set of channels randomly in order to create a schedule. The expected utilization of a schedule is thus based on the channel ranking. When it comes to reuse, the quality of the schedules very much depends on the quality of the random generator.

[4] In their work, the number of slots depends on the number of channels only.

6 Conclusion and Future Work

We have presented a new approach for the dynamic computation and adjustment of channel hopping sequences for CRNs. The first metric $\Sigma(\vec{u})$ defines, for a given number of time slots, the optimal number of channel utilizations, giving preference to channels of higher quality by using them more frequently. The second metric $\Omega(\vec{s})$ defines, for a given number of channel utilizations, optimal channel hopping sequences, yielding schedules that are less prone to channel quality changes, e.g. channel failures due to usage by a primary user. We have conceived algorithms to compute (nearly) optimal solutions with low complexity in terms of computation and memory, and have introduced incremental dynamic schedule adjustments to keep the operation of secondary users more stable.

In our future work, we will devise a protocol for the operation of CRNs, covering all required functionalities: spectrum sensing, management, sharing, and mobility. An integral part of this protocol will be the computation of schedules as presented in this paper. Further, time synchronization, which is required for channel hopping, will be incorporated. The protocol will support the computation and exchange of channel quality reports, based on local channel sensing, and the dissemination of schedules. In particular, it will cover phases of stabilization to deal with situations where the CRN is powered up, or where primary users are detected.

7 References

[1] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty. Next generation/dynamic spectrum access/cognitive radio wireless networks: A survey. *Computer Networks*, 50(13):2127 – 2159, 2006.

[2] N. Baldo, A. Asterjadhi, and M. Zorzi. Dynamic spectrum access using a network coded cognitive control channel. *IEEE Transactions on Wireless Communications*, 9(8):2575–2587, August 2010.

[3] M. L. Balinski and H. P. Young. *Fair Representation: Meeting the Ideal of One Man, One Vote*. Brookings Institution Press, 2nd edition, Aug. 2001.

[4] K. Bian, J.-M. Park, and R. Chen. A quorum-based framework for establishing control channels in dynamic spectrum access networks. In *Proceedings of the 15th Annual International Conference on Mobile Computing and Networking*, MobiCom '09, pages 25–36, New York, NY, USA, 2009. ACM.

[5] C. Cormio and K. R. Chowdhury. Common control channel design for cognitive radio wireless ad hoc networks using adaptive frequency hopping. *Ad Hoc Netw.*, 8(4):430–438, June 2010.

[6] L. A. DaSilva and I. Guerreiro. Sequence-based rendezvous for dynamic spectrum access. In *2008 3rd IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks*, pages 1–7, Oct 2008.

[7] K. Kopfermann. *Mathematische Aspekte der Wahlverfahren: Mandatsverteilung bei Abstimmungen*. BI Wissenschaftsverlag, 1991.

[8] L. Lazos, S. Liu, and M. Krunz. Spectrum opportunity-based control channel assignment in cognitive radio networks. In *2009 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pages 1–9, June 2009.

[9] B. F. Lo. A survey of common control channel design in cognitive radio networks. *Phys. Commun.*, 4(1):26–39, Mar. 2011.

[10] B. F. Lo, I. F. Akyildiz, and A. M. Al-Dhelaan. Efficient recovery control channel design in cognitive radio ad hoc networks. *IEEE Transactions on Vehicular Technology*, 59(9):4513–4526, Nov 2010.

[11] H. F. Niemeyer and A. C. Niemeyer. Apportionment methods. *Mathematical Social Sciences*, 2(56):240–253, Oct. 2008.

[12] F. W. Owens. On the apportionment of representatives. *Quarterly Publications of the American Statistical Association*, 17(136):958, dec 1921.