

Study of algorithms to optimize frequency assignment for autonomous IEEE 802.11 access points

Michael Finsterbusch and Patrick Schmidt

Dept. of Communication and Computer Science, Hochschule für Telekommunikation Leipzig (FH), University of Applied Sciences (HfTL)
Leipzig, Saxony, Germany

Abstract

This study points to an automatic channel assignment for unadministrated, chaotic WLANs to take advantage on the given capacity of the IEEE 802.11 frequency spectrum and to enhance the quality of the entire WLAN sphere.

This paper determines four public channel assignment algorithms for IEEE 802.11 networks. We show the problem of channel assignment in unadministrated WLANs and describe each algorithms functional principles. We implemented each one and simulated them on a huge amount of random topologies. The results show the timing behavior, the used iterations and the error statistics. Based on these data we determined problems in each algorithm and found graphs were they failed to find a collision free solution. We also implemented some improvements and finally a modified algorithm is presented that shows best results.

Keywords: *Wireless LAN, Channel Selection, Heuristic, Optimization.*

1. Introduction

The use of IEEE 802.11 wireless LANs (WLAN) has been grown rapidly for the last ten years, promoted by inexpensive IEEE 802.11 capable PDAs, mobile phones, laptops and WLAN routers. WLAN is used for business and private networks. In the early days of 802.11 WLANs were mainly installed and administrated by experts. Today's WLANs are often operated by end users regardless of their possible influence on other wireless networks in immediate vicinity. In urban areas often exist up to 30 WLANs in a place. Mostly they use factory default settings with the same channel and maximum transmit power. That results in a bunch of badly working wireless networks.

To increase the performance of WLANs in such an environment, automatic lerning and self organized algorithms are needed to optimize the channel load and take action on changing network topologies.

This study deals with two kinds of algorithms, that we classify into *distributed* and *centralized* algorithms. A distributed algorithm runs on any single Access Point (AP), gathering information from its environment and choosing its channel configuration with respect to other APs within reach. A centralized algorithm collects information about the whole network topology and the disturbing environment from all APs in its domain. Then one node in the domain calculates the best channel assignment for the whole topology and distributes the results to all APs in its domain.

This paper is organized as follows. First we shortly describe the problematic nature of channel use and channel access in 802.11. Then we describe the aims of automatic channel assignment. In section four the different channel assignment algorithms are presented, in section five the performance of the several algorithms is compared. We summarize our research findings in section six.

2. Principles of IEEE 802.11 data exchange

The air interface in IEEE 802.11 networks is a shared medium. Both data- and control-frames and up- and down-link share the same channel, in contrast to other wireless networks like GSM, EDGE or UMTS. Furthermore the IEEE 802.11 frequencies are license free, so everyone can provide its own WLAN network.

The next paragraphs show in short the channel division and the channel access method.

2.1 Channels in IEEE 802.11

IEEE 802.11 wireless LANs can use two different frequency ranges in 2.4GHz (IEEE 802.11b/g) and 5GHz (IEEE 802.11a/h) [1]. The frequency ranges are split into different channels. Every channel has a bandwidth of 20MHz. Between the center frequencies of neighboring channels is a frequency gap of 5MHz. Channels 1 to 11 in the 2.4GHz range may be used in the USA, whereas in most parts of Europe channels 1 to 13 are available. This means there are only three non-overlapping channels within 2.4GHz range. The 5GHz band ranges from

5.0GHz to 5.725GHz. The first available channel in Europe is channel 36 (5180MHz), the last is channel 140. In 5GHz range only channel numbers that are a multiple of four (e.g. 36, 40, 44) may be used. This means all available channels in 5GHz band are non-overlapping. Nevertheless the 2.4GHz range (IEEE 802.11b/g) is used mostly.

2.2 Access to the wireless network

The access to the wireless interface in IEEE 802.11 is coordinated by the *Carrier Sense Multiple Access* with *Collision Avoidance* (CSMA/CA) protocol. Any transceiver must sense for some microseconds (*backoff time*) for a free channel. This avoids simultaneous access to a channel. The backoff time is calculate before a frame will be sent. The backoff time is a multiple of a time slot and a random number within the range of 0 to *Contention Window* (CW). The contention window ranges from

$$CW_{\min} \leq CW \leq CW_{\max} \qquad (1)$$

At startup, CW is initialized with $CW_{\min}$. After collision CW will be increased by a power of 2, but with maximum of $CW_{\max}$:

$$CW = \min(2^{init+collisions} - 1, CW_{\max}) \qquad (2)$$

If a frame was send successfully CW is reset to $CW_{\min}$. Default values for $CW_{\min}$ and $CW_{\max}$ are 31 and 1023. A time slot is 9µs or 20µs long, depending on modulation used [1]. Overall the CSMA/CA algorithm shares the medium fair between all transmitting stations.

Therefore high channel load, collisions and interferences decrease the throughput per channel and increase the latency.

3. Aims of optimized frequency assignment

An optimized frequency assignment improves the exploitation of the air interface. This decreases the interference, between neighboring wireless stations, channel load and collisions. So the overall quality (throughput, delay and packet loss) of the entire IEEE 802.11 environment will increase. An automatic frequency assignment has the additional advantage that it can be used also in non-administrative domains ('chaotic' networks).

Therefore, an optimal channel assignment algorithm for 802.11 networks will find a channel configuration for each node and each radio device of each node in such a way that a minimum of radio interference occurs during communication.

4. Algorithms

4.1 Channel selection in chaotic wireless networks

Matthias Ihmig and Peter Steenkiste have published a channel selection method [7] discussed in this section. Their method tends to optimize the channel selection in *chaotic* wireless networks. Chaotic wireless networks are a group of single WLAN Access Points including its clients (wireless stations), within different administration domains and without coordination between those. Due to this scenario this method depends only on locally measurements and without communication. We will call this method CHAOTIC afterwards.

The CHAOTIC channel selection procedure is divided in three modules: *monitoring module*, *evaluation module* and channel switching module.

The *monitoring module* permanently collects information about the channel load on a single dedicated channel. The AP collects data for at least t_{hold} seconds. Then it switches to the next channel if it is necessary. For t_{hold} Ihmig et al proposed a 10 seconds interval. To determine the channel load the so called *MAC delay* is used as metric in [7]. During the measurement the channel load can fluctuate significantly, so they use an exponentially weighted moving average to smooth the measured load value:

$$\bar{x}_k = \alpha \bar{x}_{k-1} + (1-\alpha) x_{k,} \qquad \alpha = \frac{n}{n+1} \qquad (3)$$

For each channel the weighted channel load value is saved in the *load table*.

The *evaluation module* takes the decision of channel switching. This is done by comparing the current channel load against a threshold $thresh_{current}$. On startup $thresh_{current}$ is set to minimum threshold $thresh_{\min}$. Ihmig et al used a value of 50% channel load for $thresh_{\min}$. In the case the channel load is higher than the threshold determined for that channel ($channel_{current} > thresh_{current}$), the channel switch is triggered. The flow chart in figure 1 shows the algorithm of the evaluation module.

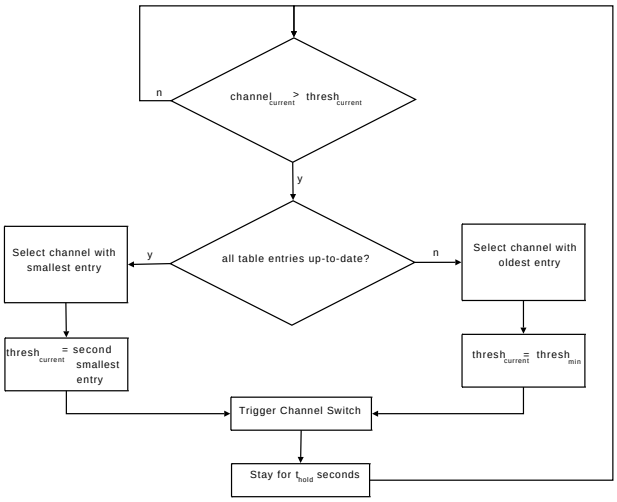

Fig. 1 Program flow chart of channel selection [7].

The *channel switching module*'s job is to switch to the assigned channel together with all connected wireless stations.

Rating of the CHAOTIC algorithm

The CHAOTIC algorithm seems easy to implement. The simulation results in the paper of Matthias Ihmig and Peter Steenkiste showed that at least 95% of the throughput compared to hand-optimized can be reached.

However the CHAOTIC algorithm has a disadvantage that we have pointed out during implementation and testing. There exist topology scenarios for that the algorithm fails. Demonstrating that behavior on a minimal setting (figure 2), assume 4 APs that use the same channel (channel 0) on

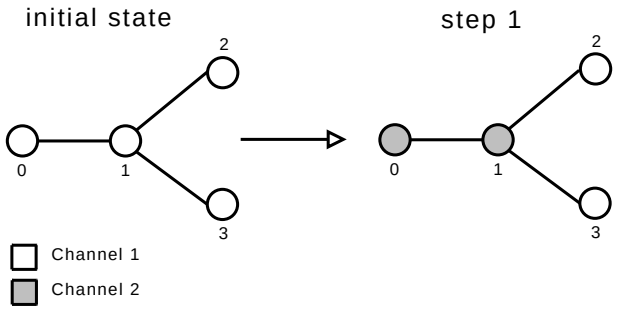

Fig. 2 Example graph on error condition.

startup. AP1 is able to detect all other AP's, whereas AP0, AP2 and AP3 just detect the load of AP1. We assume all APs generate equal channel load, that is higher than $thresh_{\min}$. After startup AP0 scans its environment. Due to detected to high load, AP0 switches to channel 1 and sets its $thresh_{\min,AP0}$ to $thresh_{current,AP0}$. In the same way AP1 detects AP2's and AP3's high channel load (and maybe of AP0 also, when AP1 and AP0 startup simultaneously) and switches to channel 1. The updated $thresh_{\min,AP1} = thresh_{current}$ is up to two or three times higher than threshold $thresh_{\min,AP0}$ of AP0. After that the CHAOTIC algorithm remains in a deadlock situation. This is not the optimal channel assignment (figure 2), but AP0 and AP1 will not switch to another channel any more. Both AP's have an threshold value that will not be overcalled from $channel_{current}$. The evaluation algorithm just loop in the comparison of the current channel load and the threshold (see figure 3), but it will never break the loop to switch to another channel.

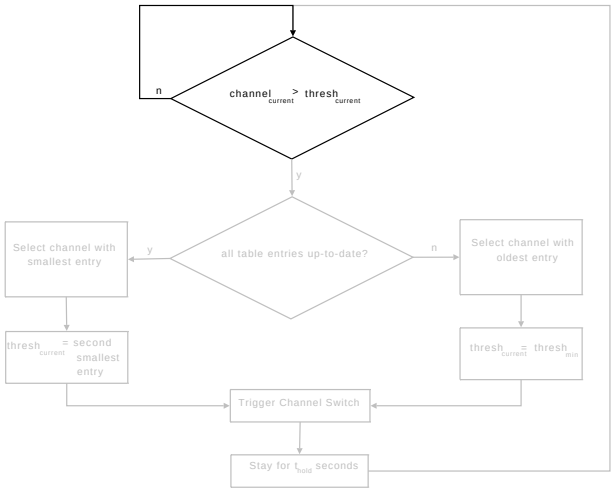

Fig. 3 Program flow chart of channel selection [7] in error condition - the algorithm locks in a loop.

Modified CHAOTIC algorithm

We modified the CHAOTIC algorithm to overcome its disadvantages. The modified CHAOTIC at startup scans all channels to update the whole load table. So the AP can choose its best channel. Additionally we add a channel aging. In case of an outdated current channel and interferences with other APs, channel switching is forced. This way reaction to changes in the topology is better and it breaks possible deadlock constellations.

Our simulation results of the CHAOTIC algorithm and the modified CHAOTIC algorithm are given in section 5.

4.2 Channel assignment with a central heuristic

K. Leung and B. Kim published a centralized heuristic to optimize channel load with a better channel assignment in [6]. The heuristic uses a *greedy* step to speed up the algorithm. Greedy algorithms generate good results, step by step, by splitting the whole problem in

many particular problems to generate a good solution (local minima/maxima) for the partial problems. But it is not guaranteed that they produce the optimal solution for the whole problem. Often greedy algorithms generate good results in a comparatively short time.

The aim of this heuristic is to minimize the channel load of the 'bottleneck'. The bottleneck means the AP with the most neighbor APs using the same channel. By optimizing the bottleneck, the flow in the whole set is optimized.

The heuristic algorithm contains of 6 steps [6]:

1) Generate a random, initial channel assignment.

2) Choose the bottleneck (with channel utilization *V*). If there are several bottlenecks choose one randomly.

3) Identify bottleneck's assigned channel *k*. For each available channel *n* from *1* to *N* with $n \neq k$ and neighbor AP *j*, temporarily modify the channel assignment by reassigning only AP *j* with channel *n*. Save the minimum channel utilization *W* of all iterations.

4)
   a) If $W < V$, then replace *V* by *W* – a greedy step. Continue with step 2.
   b) If $W = V$, then with a pre-defined probability δ replace, *V* by *W*. Continue with step 2.
   c) If $W > V$, a local optimum has been reached. Continue with step 5.

5) Repeat steps 1 to 4 with a number of random, initial assignments. The final solution is chosen to be the best according to (4), among the local suboptimal assignments.

6) Test if condition (5) for all APs is satisfied for the final assignment. If so, the final assignment is feasible. Otherwise, it is considered that no feasible solution exists for the network under consideration.

This heuristic was used by Leung and Kim [6] to calculate the channel assignment for two different networks, one with 21 APs and one with 111 APs. The optimized channel assignment was known in advance. The heuristic was not able to produce the optimal channel assignment for the second network with 111 APs.

Rating of the central heuristic

This heuristic is only useful for offline network design, because it is too slow or needs to much computing power, respectively. Steps 3 and 5 of the algorithm use brute-force to find a better channel assignment. This algorithm scales exponential. So it is not applicable on huge networks with normal APs that consists usually of low power embedded devices. The unsuccessful test by Leung and Kim with the network consist of 111 APs, took about 9 minutes on a SUN Sparc workstation. Therefore it can not take action to changes in the environment or the network topology in an adequate time.

But aside from the bad scaling, the central heuristic fails on a number of network topologies. We determined during implementation and testing that the algorithms fails on ring topologies (see figure 4). The main problem is that

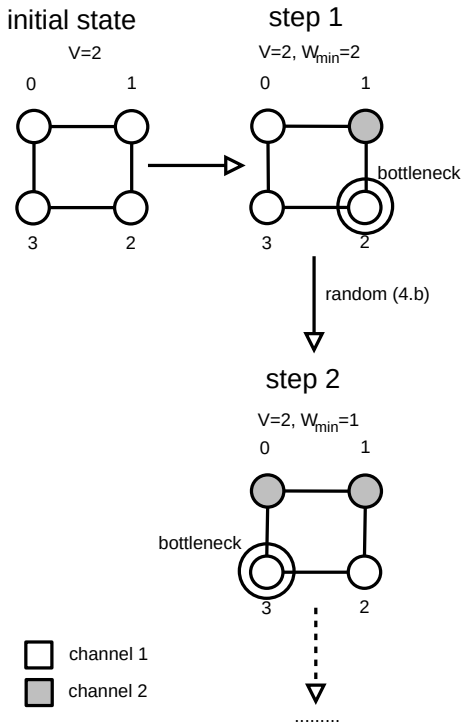

Fig. 4 Central heuristic on error condition.

in step 3 of the algorithm, only one neighbor AP will be changed. Afterwards will be the next bottleneck chosen to optimize. In the ring topology, all APs have an utilization Vector *V* of 2 - the direct neighbors in the ring. The reduction of utilization Vector *V* optimize the local utilization, but the overall utilization vector *W* stays at 2. This failure occurs on ring topologies with even-numbered amount of nodes and with odd-numbered nodes in line topology.

Furthermore the algorithm has another problem, it can not handle incoherent topologies.

Our simulation results for the centralized heuristic are shown in section 5.

4.3 Distributed heuristic for multi radio mesh networks

The channel selection algorithm discussed in this section was published by Bong-Ju Ko et. al. in [2]. It is a full distributed online greedy algorithm which optimizes the channel constellation based on local node information. They developed and tested the algorithm in an experimental multi radio mesh environment to improve the overall throughput performance. In our experiments we used it to optimize the channel constellation in chaotic networks. For shorter naming we call this algorithm *DH* in this paper.

The algorithm is based on two basic facts. At first an interference cost function $f(a,b)$ which measures the spectral overlapping between channels *a* and *b*. The function is defined in such a way that $f(a,b) \geq 0$ and $f(a,b) = f(b,a)$. If the value goes to *0* ($f(a,b) \to 0$) channels *a* and *b* don't overlap and a higher value means they overlap. The algorithm works with every cost function which reflects these requirements and in our study we simply use $f(a,b) = 0$ which means that different channels don't overlap. The second basic fact is the interference set *S*. The interference set S_j of node *j* is a list of all nodes whose transmissions will interfered by transmissions of *j*. A accurate determining of the interference sets is very complex in real life environments and therefore we assumed the interference set contains all neighbors of node *j*.

The algorithm itself is defined by the following pseudo code:

```
procedure ChannelSelection(node i)
Input:  S_i : interference list of i
        c_j : actual channel list for each j ∈ S_i
        c_i : i's current channel
begin
        F(c_i) := Σ_{j∈S_i} f(c_i, c_j)
        for k:=1 to K do
        begin
                F(k) := Σ_{j∈S_i} f(k, c_j)
                if F(c_i) > F(k) then
                begin
                        F(c_i) := F(k)
                        c_i := k
                end
        end
end
```

As we could see the algorithm minimizes the sum of channel interferences in each iteration. At first step the actual interference sum is calculated. To do this the actual channel c_i is combined with every channel c_j and the each calculated cost $f(c_i,c_j)$ is added up to the sum $F(c_i)$. In the next step the loop calculates the cost sum F_k in each iteration and compares it to $F(c_i)$. If F_k is smaller than $F(c_i)$ a better channel is found because the interference costs are decreasing when using this channel.

Rating of the DH algorithm

The algorithm in [2] is very small, very fast and easy to implement. Because of the very greedy sequence it converges to a stable solution in only a few steps but also comes to many incorrect solutions. One example graph were the algorithm fails to find a collision free solution is presented in figure5.

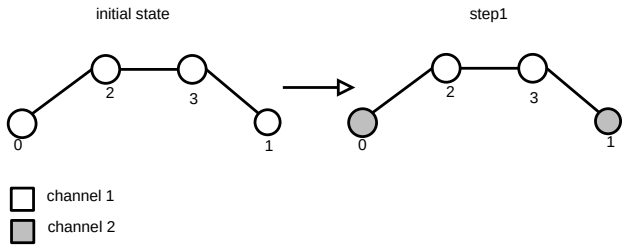

Fig. 5 Example graph where DH fails to find optimal solution.

The graph shows four nodes with two available channels at all. The node numbers show the order in which each AP runs the DH algorithm. At first node *0* changes it's channel to *1* because this one is free of interferences. After that AP *1* does the same thing and causes the locking situation. Because the nodes *0* and *1* doesn't interfere each other they will never change their channel anymore and nodes *2* and *3* can't find the optimal result.

Optimizing of the DH algorithm

To optimize the algorithm we tried to use two different strategies of changing this behavior. At first we tried to run the algorithm in a randomized node order and we also sorted the nodes based on node weight to determine the right running order, but we couldn't optimize the fault rates significantly.

4.4 Communication Free Learning algorithm

The Communication Free Learning algorithm (CFL) was published by D.J. Leith, P. Clifford and D.W.Malone in [4] and [5]. It is a node coloring instruction which was

developed and tested by the authors for automatic channel configuration of unadministrated AP's. It is very simple and doesn't need any communication or synchronization between the AP's. The algorithm's channel selection is based on vector p_i, which contains a selection probability for each available channel. The algorithm updates p_i for each channel *i* following this description:

1) Initialize $p = [1/c, 1/c, ..., 1/c]$

2) Toss a weighted coin to select a channel *i* with a probability of p_i. Sense channel *i*'s quality. If it is acceptable hop to this channel and goto step *3*, if not goto step *4*.

3) On successful change to channel *i*, update p_i as follows:
   a) $p_i = 1$
   b) $p_i = 0, \forall j \neq i$

4) If no channel change, update p_i like this:
   a) $p_i = (1-b) \cdot p_i$
   b) $p_i = (1-b) \cdot p_i + \frac{b}{c-1}, \forall j \neq i$

5) Go back to step *2*.

The parameter *b* is called *Learning Parameter* and influences the speed of channel changes. In our simulation we always used $b = 0.1$ according to statements in [5]. Leith et. al. used a measurement of MAC-Acknowledgements to measure the channel quality, we used a simple binary decision. A channel is always good if no other neighbor is using the same one.

Rating of CFL-Algorithm

The CFL is also very easy to implement and always converges to a stable solution. On the other hand it deals with a lot more iterations than a greedy algorithm e.g. the one presented in section 4.3. This is because the selection decision is based on probabilities which are updated in every step in dependence to *b* (*Learning Parameter*).

Another thing we found out is that there are some general graphs or subgraphs which cannot be handled correctly by the CFL under special circumstances.

Figure 6 shows one general graph with two available channels where the CFL fails to calculate a correct solution. If the edge nodes are using the same channel *i* the probability p_i is set to $p_i = 1$ according to step *3a* and will never change again. The nodes in the middle therefore

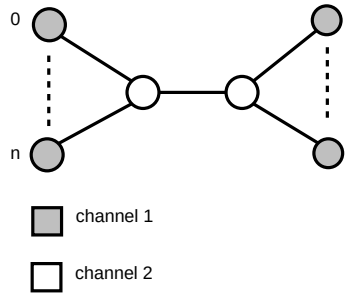

Fig. 6 General graph where CFL doesn't find optimal solution.

cannot choose this channel and we got a lock situation where the optimal solution cannot be found anymore. Figure 6 corresponds to figure 5 and is the generalized graph of this.

5. Comparison of algorithms

To compare the channel assignment algorithms described in section 4 objectively, we implemented all algorithms in the C programming language. To determine accuracy, disadvantages and performance of this algorithms, we let they compute channel assignments for various random generated graphs.

To generate the test graphs, we use an algorithm that use a weighted random metric. In fact this means that at startup an empty adjacency matrix for *n* nodes is created. With a probability of δ the edge between two nodes is set in the adjacency matrix. That means this nodes can interfere with each other. Additionally, it prohibits single, isolated nodes.

The algorithm test is performed in the following way. At startup of every test run, a random graph is generated. Overall we tested the algorithms with graphs of 4 to 24 nodes and 100.000 random graphs for every amount of nodes. The probability of interferences between two nodes δ was set to 20%. After graph generation, we determined the minimal needed amount of channels for a interference free channel assignment with the *DSATUR* algorithm. The well known algorithm DSATUR (Degree of Saturation) was developed and published by Brèlaz in [3].

Then the graph is initialized with a random channel assignment. If the random channel assignment is already a collision free (valid) assignment the graph is reinitialized until the assignment is not valid. With this setting, every algorithm described in section 4 had to compute the channel assignment. Then the result was checked for validity and additionally we saved the runtime and iterations needed. The minimal needed amount of channels - calculated with *DSATUR* algorithm - was the maximal number of channels to use. The test run was performed on

a desktop workstation (Intel® Pentium® 4 CPU 3.00GHz with 512MB RAM, Linux Kernel 2.6.22).

5.1 Measurement results

The figures 7 and 9 shows the time needed by the CHAOTIC and the modified CHAOTIC algorithm, respectively. The better timing behavior of the modified CHAOTIC algorithm is perfectly clear. The modified CHAOTIC needs also distinctly fewer iterations (figures 8 and 10), as well. But the root square mean execution time in figure 10 shows that the modified CHAOTIC algorithm needs also many iterations for some graphs, but overall it is much faster.

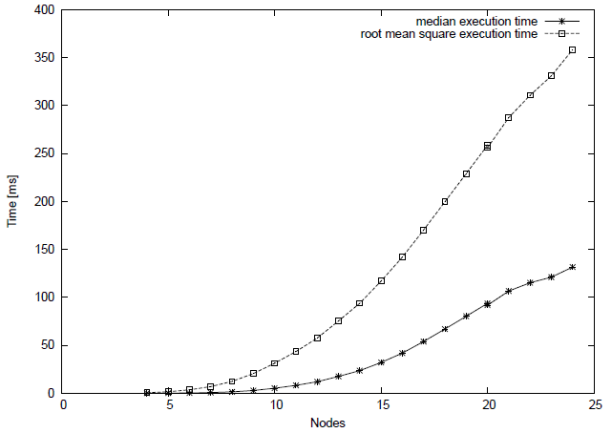

Fig. 7 Runtime of CHAOTIC algorithm.

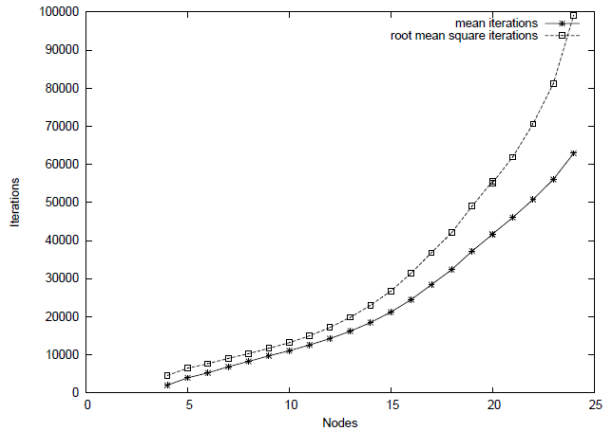

Fig. 8 Iterations of CHAOTIC algorithm.

Figure 11 shows the worst, exponential time consumption of the central heuristic. The reasons for that was explained in section 4.2. The bad scaling of the algorithm reflects also in the iteration diagram (figure 12) and in the comparison of all algorithms in figure 17.

Figure 13 shows the timing behavior of the distributed heuristic algorithm. It is approximately equal to that of the CHAOTIC algorithm. A particular feature of this algorithm is that it needs just one or two iterations (figure 14). This algorithm is relative fast, but it has a very high error rate (see figure 18).

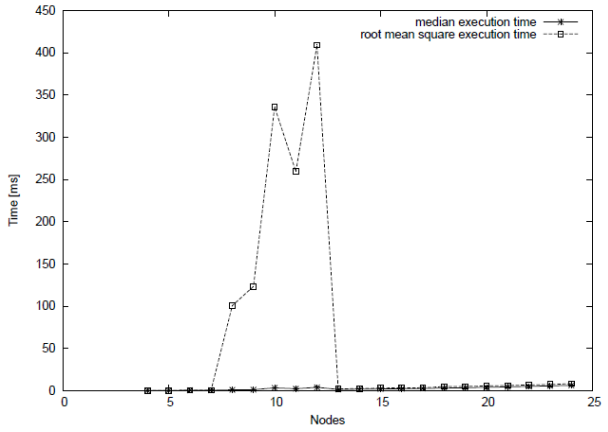

Fig. 9 Runtime of modified CHAOTIC alogrithm.

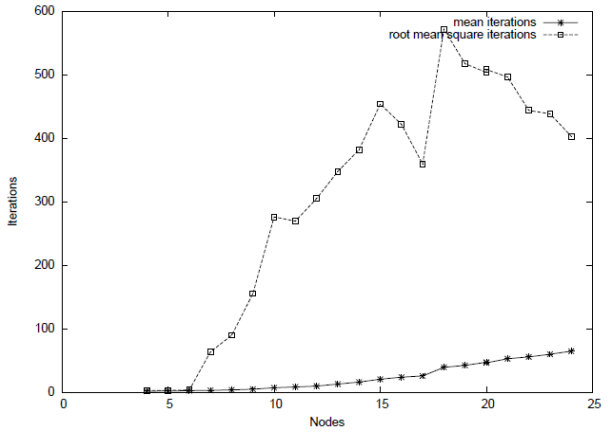

Fig. 10 Iterations of modified CHAOTIC algorithm.

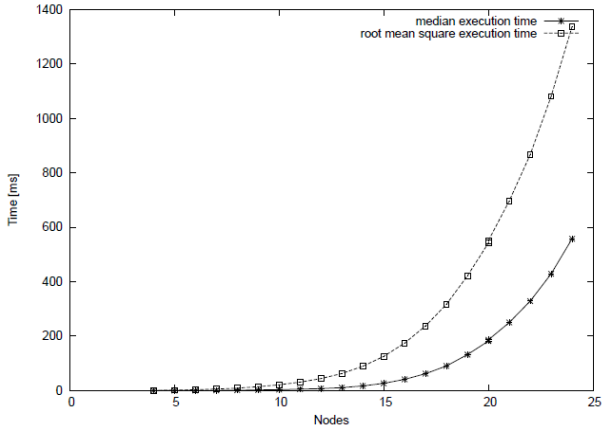

Fig. 11 Runtime of central heuristic algorithm.

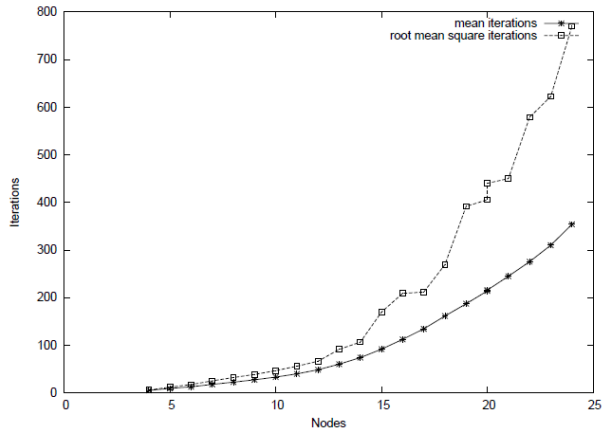

Fig. 12 Iterations of central heuristic algorithm.

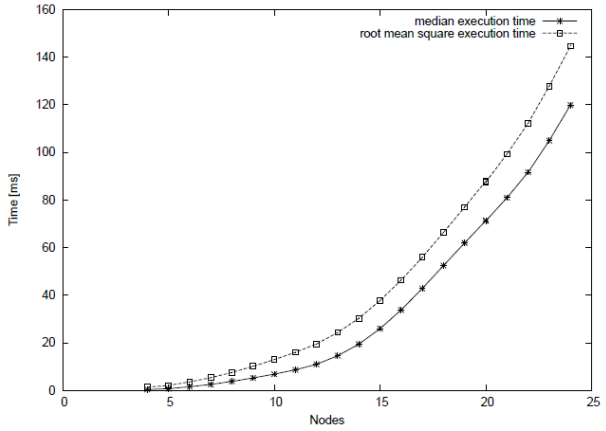

Fig. 13 Runtime of distributed heuristic algorithm.

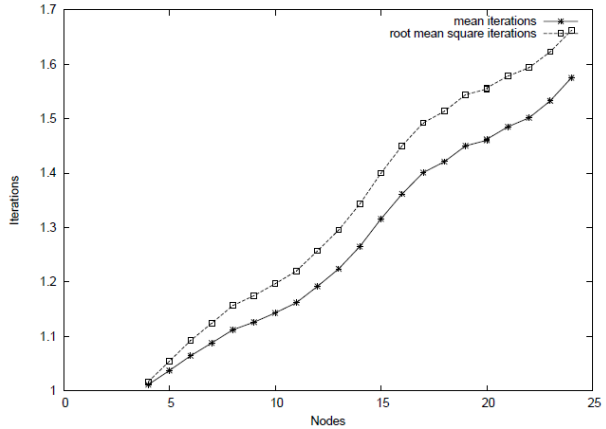

Fig. 14 Iterations of distributed heuristic algorithm.

Figure 15 shows the timing behavior of CFL algorithm. We see, the time consumption increases rapidly. It needs already a half second average to calculate a channel assignment for 24 nodes. Test runs with invalid results were not counted.

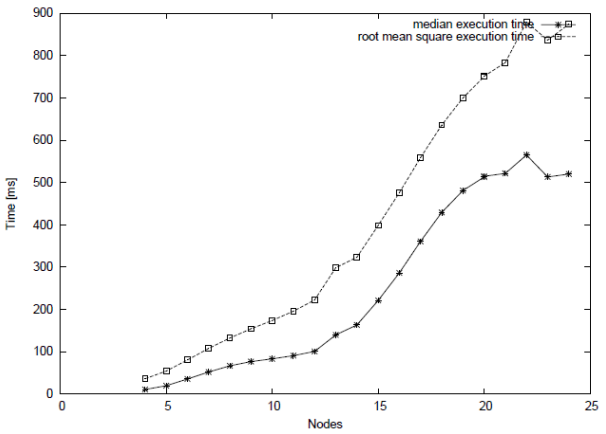

Fig. 15 Runtime of CFL algorithm.

Figure 16 shows the iterations needed to compute a valid channel assignment.

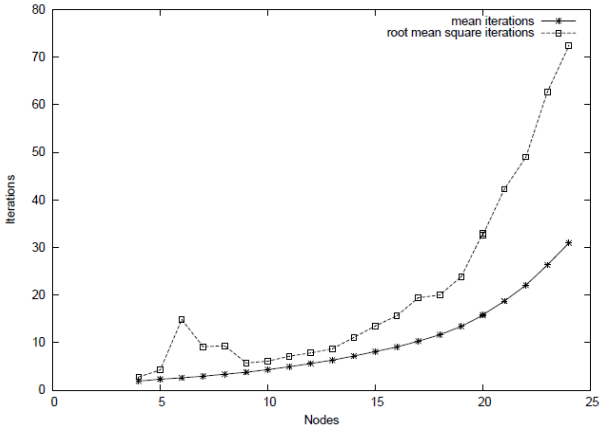

Fig. 16 Iterations of CFL algorithm.

Figure 17 summarizes our findings of the time consumption for all algorithms, including DSATUR. It shows that CFL and central heuristic scale badly for networks with more nodes. The modified CHAOTIC scales at best of all determined algorithms. But DSATUR seems to be the fastest algorithm in our simulation. But the DSATUR algorithm has an exponential scale as a last resort and also needs the complete topology for calculation (centralized).

Figure 18 shows the amount of invalid channel assignments for the entire algorithms. Only the modified CHAOTIC is free of errors. All other algorithms can not

be used in real world applications because of their high error rate.

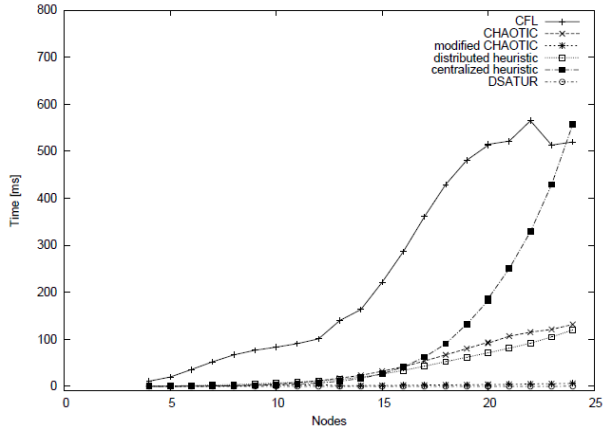

Fig. 17 Runtime comparison of all algorithms.

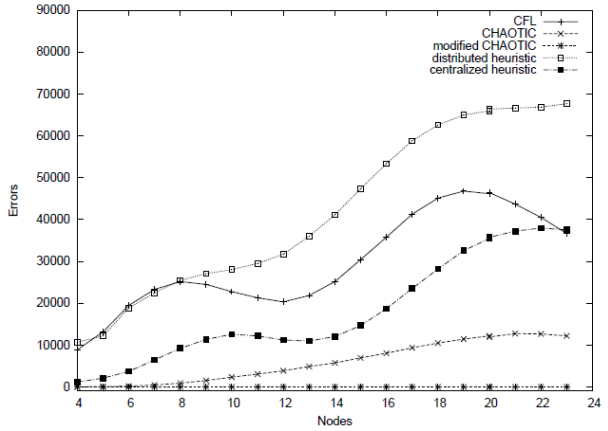

Fig. 18 Comparison of invalid assignments.

6. Conclusion

In this paper we compared four different algorithms for channel assignment in IEEE 802.11 networks. These algorithms are from different publications and are using different strategies to solve the problem. We implemented them in C programming language and simulated all algorithms on a huge amount of random graphs. We determined weaknesses/problems in every single one and could therefore not recommend to use one of these in real world networks with a higher amount of nodes.

The best results achieved the modified `CHAOTIC` algorithm. For practical usage this one would be the best when regarding its performance and robustness.

Or future goal is to develop a WLAN backbone architecture with a self organizing channel assignment for all used radio interfaces. We could use our simulation results to choose the right algorithm for channel configuration of the backbone's ingress access points.

References

[1] IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Std 802.11TM-2007.

[2] B.J.Ko, V.Misra, J.Padhye, and D.Rubenstein. Distributed channel assignment in multi-radio 802.11 mesh networks. Columbia University, Technical Report, 2005.

[3] Brèlaz. New methods to color the vertices of a graph. Communications of the Assoc. of Comput. Machinery 22, 1979.

[4] D.J.Leith and P.Clifford. Channel dependent interference and decentralized colouring. Proc. International Conference on Network Control and Optimization, Avignon, 2007.

[5] D.J.Leith, P.Clifford, and D.W.Malone. Wlan channel selection without communication. Proc. IEEE Dyspan, 2007.

[6] K. Leung and B. Kim. Frequency assignment for ieee 802.11 wireless networks. Proc. 58th IEEE Vehicular Technology Conference, 2003.

[7] M.Ihmig and P.Steenkiste. Distributed dynamic channel selection in chaotic wireless networks. 13th European Wireless Conference, Paris, 2007.

Patrick Schmidt studied computer science in Leipzig at Telekom University of Applied Sciences (HfTL) and received his Diploma in 2006 for inventing a secure and mobile architecture for mobile vehicles. His Master of Engineering at Telekom University of Applied Sciences in Leipzig in 2009 was on research for WLAN backbone optimization. His current projects cover WLAN and WLAN backbone optimization, network traffic management, deep packet inspection and high availability.

Michael Finsterbusch studied computer science in Leipzig at Telekom University of Applied Sciences (HfTL) and received his Diploma in 2006 for implementing a Diameter server and client to gain a AAA-infrastructure for a secure and mobile architecture for mobile vehicles. His Master of Engineering at Telekom University of Applied Sciences in Leipzig in 2009 was on research for WLAN mobility. His current projects cover WLAN optimization, WLAN mobility, network traffic management, deep packet inspection and high availability.