Abstract—This paper studies the performance of cognitive radios in a scenario where different pairs of radios adopt different cognition/decision making approaches. We want to assess (i) if there is a category of cognitive radio algorithms that consistently outperforms the others, and (ii) how sensitive different algorithms are to suboptimal parameter setting. Our approach is to take a representative set of well-known classes of cognitive radio algorithms, mix and match them throughout thousands of simulations, and determine which seem to perform better. We find that choosing a cognitive radio algorithm means finding a balance between the best-case performance obtained by optimally setting all parameters, and the behavior in uncontrolled, unknown environments, where sub-optimal decisions are likely to be made. The approaches we consider, namely reinforcement learning, optimization metaheuristics, multi-armed bandit solutions, and supervised learning, greatly differ in their performance. For example, schemes that are able to achieve a high throughput in our simulation study are more sensitive to suboptimally-set parameters.

Index terms—Artificial intelligence, cognitive radio networks, heterogeneous scenario, learning, sensitivity analysis, simulation.

I. INTRODUCTION

Although the concept of cognitive radio has been around for almost two decades, real deployments are still scarce. Part of the problem is the perceived complexity in the operation and parameter setting (as well as the certification process) for such radios. Upon deciding to embrace cognitive radios, adopters would be faced with several important choices, from the cognition algorithm to use to the selection of its parameters.

In this paper, we compare four approaches to cognition found in the literature, namely, reinforcement learning, optimization metaheuristics (represented by genetic algorithms), multi-armed bandit solutions, and supervised learning (represented by support vector machines). We discuss which classes of cognitive radio algorithms appear to offer the best performance, and how to set the main parameters of each class. At the same time, we investigate where the performance difference comes from, i.e., which decisions – channel selection, sensing interval – good algorithms take more effectively than bad ones. We should note that we are not proposing a new cognitive radio algorithm. Rather, we survey the literature and identify main categories of algorithms, and the most relevant parameters of each one. Then, we perform a large set of simulations, where algorithms and parameter settings are compared with one another in a variety of medium- to large-scale scenarios. Finally, we examine the simulation results, mining them for generalizable conclusions on the efficiency of different algorithm types.

One may wonder why a new set of simulations are needed in the first place, when virtually all papers proposing a new cognitive radio algorithm come with their own extensive set of simulations. A first reason is that we need to study all algorithms under the same conditions. Another one is scale: as discussed in Sec. VI, many cognitive research papers focus on validating their solution concept, which is best done within small-scale scenarios. Our focus, on the other hand, is on how suitable different solutions are for large-scale scenarios. Finally, our simulations enable us to study how different cognitive radio algorithms, and indeed different approaches for cognition, can coexist in the same network.

We will find out which type of algorithms tend to perform as

---

A. Hess, N. Kaminski, and L.A. DaSilva are with Trinity College Dublin, CONNECT Centre for Future Networks, Dublin, Ireland. F. Malandrino is with Politecnico di Torino, Italy. T.K. Wijaya (now at Google) contributed to the work as an independent researcher. E-mail: hessan@tcd.ie

This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Numbers 10/IN.1/I1007 and 13/RC/2077.
better than others; which are highly sensitive to suboptimal parameter settings and which are more of a safe bet; and which parameters warrant care and extensive experimentation before they are set. These goals are summarized in Fig. 1.

Our performance evaluation differs from traditional cognitive networking papers in both the reference scenario and the way different approaches are compared. We study a set of large-scale scenarios, covering an area of one square kilometer and including tens of source/destination pairs, whose position is chosen at random. We argue that using such scenarios makes our evaluation more realistic than only considering hand-picked network topologies (e.g., a line or butterfly topology).

Similarly, most cognitive radio papers evaluate the algorithms they present in homogeneous settings, exhaustively studying all possible parameter settings to find the best one. In contrast, we randomly assign a parameter setting (which we call a firmware) to each transmitted/receiver pair on our topology, and perform a statistical study of the performance associated to each approach. While this unavoidably means that some position/setting combinations will not be taken into account, it also has two important advantages: first, it allows us to study which approaches are more resilient to suboptimal parameter settings, which are very likely in real-world cases; second, we can explore the interaction between multiple independent secondary users, using different algorithms and technologies, all competing for the same channel. Furthermore, we note that this approach to the batch analysis of parameter settings provides a useful foundation for the operation of meta-learning approaches for cognitive radio systems, as exemplified in [1], [2]. We note that most meta-learning work tends to focus on the operation of an individual radio or network changing over time, rather than the aggregate investigation of radio operation in heterogeneous environments as presented here.

II. ALGORITHMS

In dynamic spectrum access (DSA), the most common application of cognitive radios, secondary users must find a balance between supporting their own communications and avoiding interference with those of more privileged users. Typically this involves selection of a channel, power level, and modulation and coding scheme (MCS). Actions by a cognitive engine may then be divided into two categories: impactive actions and non-impactive actions. Impactive means in this context that an action has an impact from the viewpoint of the primary user, which might be the case for actions adapting bandwidth, frequency, or power level. Actions that have no visible impact to the primary user, e.g., those linked to the modulation scheme, are categorized as non-impactive. Cognitive radios may also make exploratory actions, e.g., in the form of sensing, which typically do not affect other users, but provide situational awareness. The operation of a cognitive radio may then be described as the selection of an action to balance the impact of the first category (impactive actions) against the requirements of the second category (non-impactive actions), based on information gained with the third category (exploratory actions). That is, the cognitive radio must make decisions by jointly considering its own needs and the requirements of more privileged users, based on the information it has the ability to collect.

We examine the factors that underpin an algorithm’s ability to balance the two categories of DSA actions. To this end, each algorithm is presented with the same goals of achieving high throughput communication while avoiding interference with other users. At each decision point, secondary users may select to sense or transmit. If sensing, radios receive an estimate of received power for each channel. If transmitting, radios must also select the channel and the MCS to employ, as summarized in Sec. III. These choices—sensing or transmitting, and if transmitting on which channel, with which MCS and at which power level—represent the core of DSA, and these are the choices all algorithms are faced with.

The algorithms run within the receiver of a pair of secondary radios. Receivers can direct the actions of transmitters, e.g., through the use of cyclostationary signatures, as presented in [3]. Cyclostationary signatures consist of an identifier, unique to a secondary user pair, and an action code, which indicates the channel and MCS to be used for the next transmission. Transmitters simply scan for cyclostationary signatures and send packets in response to requests. This organization alleviates problems related to the need to share channel information and is in line with the suggestions of [4].

The operation of each algorithm can be separated into two functions: decision and feedback. The decision function determines the next action to be taken. The feedback function tallies information regarding the performance of this action for later use. Each algorithm maintains its own persistent memory during the course of simulation.

The reader can refer to Tab. I for a summary of the algorithm classes we examine in our paper, and the parameters of each class. It is important to stress that for each of the classes we consider, to ensure generality of the comparison, we take the classic version of the algorithm without any modifications or optimizations. This approach is consistent with our goal of obtaining high-level information about the behavior of different classes of algorithms—e.g., what happens if parameters are set suboptimally—as opposed to the performance of individual algorithms. Thus, we describe the general principles of each algorithm in the following subsections.

A. Reinforcement Learning Algorithm

The first class of algorithms we examine is a straightforward implementation of Reinforcement Learning (RL) [5]. This particular class of algorithms has been applied to a range of cognitive radio problems [6]–[8] and the version implemented here is a basic instantiation tailored for use in DSA.

As is typical for reinforcement learning, this algorithm is based on keeping track of rewards and punishments. Here, we maintain a reward value for each transmit action, which indicates the learned usefulness of the action. These transmit actions, consisting of all combinations of a channel, transmit power, and MCS, form the action space available to the RL algorithm. The RL algorithm tracks the utility of these actions through environment states defined by the power levels on all available channels that result from the aggregate operation of
all radios, whether primary or secondary. All transmit actions are initialized to zero and updated after each action. If the action results in a successful transmission, the usefulness is incremented; otherwise, it is decremented. After each update, usefulness values step toward zero with a step size given by the forgetfulness factor parameter (forget_factor). Keeping track of these usefulness values allows the algorithm to learn the best actions based on past experience.

The central challenge of reinforcement learning is to exploit successful actions while keeping usefulness information about all actions up to date. Here we address this exploitation versus exploration dilemma through sensing. Upon sensing, the algorithm updates usefulness values for all actions based on their estimated performance given current power levels in each channel. Usefulness values determined through sensing are weighted according to the sense weight parameter and combined with previously learned usefulness values. Secondary users running the RL algorithm sense with a probability given by the sense probability parameter.

**Algorithm 1** Reinforcement Learning Decision

1. if ROLL DICE() ≤ sense_probability then
2. action ← sense_action
3. else
4. action ← GET_BEST()
5. end if
6. return action

The reinforcement learning decision function is shown in Alg. 1. The first operation, Line 1, determines whether sensing should occur based on a random value in the interval [0, 1] as returned by the ROLL_DICE() subroutine. If the sensing check fails, the function simply returns the current best actions, as given by the GET_BEST subroutine. The GET_BEST subroutine simply provides the action with the highest usefulness at the time of calling. In the event that the action with the highest usefulness is not unique, a single action is randomly selected from those with the highest current usefulness value.

As noted above, the primary operation of reinforcement learning is maintaining a performance record for each action. This record-keeping is the focus of the feedback function for the RL algorithm. The first task of record-keeping is the forgetting of stale information, in Line 2 through Line 6 in Alg. 2. At this point usefulness values for each action progress toward zero with a step size given by the forget factor parameter. If the last action was sensing, usefulness values for each action are updated in Line 8 through Line 14. Line 8 compares the action’s power to the power required to successfully transmit data in the current radio environment, in order to calculate an appropriate usefulness update. In the case of success for a given action, the usefulness update is directly the number of bits which could be successfully transmitted as a result of that action (Line 9). Alternatively, the usefulness update for unsuccessful actions is provided by negative three times the power that would be used in the failed effort (Line 11). This usefulness update is then combined to the prior value of the action usefulness in a weighted average using the sensing weight (Line 13). If the calculated usefulness results from experience, rather than sensor based estimates, usefulness is updated in a similar manner. Specifically, if an action successfully transmits data, its usefulness is additively updated with the throughput for successful actions (Line 18). Alternatively, three times the power used by the failed action is subtracted from the usefulness (Line 20).

**Algorithm 2** Reinforcement Learning Feedback

1. for action in possible_actions do
2. if action_usefulness < 0 then
3. action_usefulness += forget_factor
4. else
5. action_usefulness -= forget_factor
6. end if
7. if last_action is sense_action then
8. if action_power > required_power then
9. usefulness_update ← action_bits
10. else
11. usefulness_update ← -3 × action_power
12. end if
13. action_usefulness ← (1 – sense_weight) × action_usefulness + sense_weight × usefulness_update
14. end if
15. end for
16. if last_action is not sense_action then
17. if last_action_throughput > 0 then
18. last_action_usefulness ← last_action_usefulness + last_action_throughput
19. else
20. last_action_usefulness ← last_action_usefulness – 3 × last_action_power
21. end if
22. end if

**B. Genetic Algorithm**

Our Genetic Algorithm (GA) provides a basic version of the classic technique that has long been studied for cognitive radios [9]–[12]. The GA generates a population of candidate actions, referred to as chromosomes and evolves them based on a fitness function. This fitness function assigns a fitness value based on the expected performance of the chromosome. Evolution occurs through the recombining of chromosomes by crossover, random changing of chromosome elements through mutation, and seeding of new populations with high fitness individuals. Crossover and mutation, or the probability with which they occur, determine the manner in which the GA searches through the parameter space for a solution. This searching process continues until a viable solution is found. The details of the process are discussed below. The algorithm returns the highest fitness chromosome as the action to be taken.

Sensing in our GA occurs periodically to maintain a current understanding of the environment. Here, sensing information consists of the power at the receiver of a given pair on each of the channels. This information is used directly as an estimate
of the interference on that channel when calculating the fitness of various actions during the running of the GA. This configuration provides the most direct connection between information available to the genetic algorithm and a selected action, but also means that the GA makes decisions on the basis of out-dated information in scenarios where sensing occurs infrequently. A separate fitness value is maintained for sensing; it is incremented by the sense fitness step parameter after each sensing action. The sensing action is then compared to the chromosome resulting from the standard evolution process. The sensing fitness is reset to zero once a sensing action has been taken. Thus the sense fitness step controls the degree of situational awareness for the GA.

Algorithm 3 Genetic Algorithm Decision

1: INITIALIZE(population[i]) \( \forall i : 1 \leq i \leq N \)
2: while number_generations \( \leq 20 \) do
3:   pairs \( \leftarrow \) FIND_PAIRS(population)
4:   for pair in pairs do
5:     if ROLL_DICE() \( \leq \) crossover_probability then
6:       CROSSOVER(pair)
7:     end if
8:   end for
9:   for chromosome in population do
10:      if ROLL_DICE() \( \leq \) mutation_probability then
11:        MUTATE(chromosome)
12:      end if
13:   end for
14:   best_chromosome \( \leftarrow \) DETERMINE_BEST(population)
15:   if CALCULATE_FITNESS(best_chromosome) is maximum fitness then
16:     exit while loop
17:   else
18:     SEED_POPULATION(best_chromosome)
19:   end if
20: end while
21: if CALCULATE_FITNESS(best_chromosome) \( \leq \) sense_fitness then
22:   best_chromosome \( \leftarrow \) sense_action
23: end if
24: return best_chromosome

Within the GA, most of the operation occurs as part of the decision function, as shown in Alg. 3. Initialization, Line 1, consists of generating a pre-defined number of randomly selected tuples of a channel and MCS. These chromosomes are then evolved for no more than 20 generations. Within each generation, each pair of chromosomes is selected for crossover with a probability given by the crossover probability parameter. The CROSSOVER subroutine, in Line 6, simply trades the channel selection between the selected chromosomes. After crossover, chromosomes are selected for mutation. The MUTATE subroutine, in Line 11, randomly alters either the channel or the MCS of the given chromosome. Once these evolution processes are complete, the best chromosome is determined by calculating a fitness for each chromosome and selecting the highest fitness individual. The fitness value is simply the estimated number of bits that a chromosome would send without interference, based on the most recent sensor information. A single best individual is selected randomly from the top performing chromosomes, should multiple chromosome share the best fitness value. If the fitness value of the best individual is equivalent to the maximum number of bits that any MCS can transmit or the maximum number of generations is reached, evolution is ended. Otherwise, the best individual is used to seed a new population, in Line 18. One half of a seeded population shares the channel selected by the seed and the other half shares the MCS of the seed, with other values being randomly selected. After evolution, the fitness of the final best individual is compared to the fitness of sensing to determine the overall best action to return.

The GA feedback function in Alg. 4 simply updates information related to sensing. In case the most recent action was sensing, information used for fitness estimation is updated, Line 2, and sense fitness is reset to zero, Line 3. Otherwise, sense fitness is incremented according the sense fitness step parameter.

Algorithm 4 Genetic Algorithm Feedback

1: if last action is sensing then
2:   UPDATE_SENSED_INFORMATION()
3:   sense_fitness \( \leftarrow 0 \)
4: else
5:   sense_fitness \( \leftarrow \) sense_fitness + sense_fitness_step
6: end if

C. Multi-armed Bandit

Our Multi-Armed Bandit (MAB) algorithm interacts with a virtual slot machine of DSA in manner based on the influential work of Jouini et al. [13]. As such, this algorithm captures the fundamental approach to a widely used family of solutions to the DSA problem.

Fundamentally, this algorithm allows each node to select an action on the basis of an associated indexing value. Specifically, each node running the MAB algorithm will select one action from a list composed of all combinations of channel, transmit power, and MCS (paralleling the transmit actions employed by the RL algorithm) as well as a sensing action. Each of these actions is considered as a potential arm of a slot machine that represents the scenario at large, including the aggregate operation of all radios. At each time step, the MAB is asked to select a single action to use, or play, based upon the utility that the algorithm associates with that action. These utilities are encoded into action index values, which simplifies action selection to finding the maximum index value. This simplifies the decision function to that shown in Alg. 5. In the event that the maximum index value is not unique, the final action is selected randomly from those with the maximum.

Algorithm 5 Multi-Armed Bandit Decision

1: selected_action \( \leftarrow \) FIND_MAX_INDEX(records)
D. Support Vector Machines

Support Vector Machines (SVMs) [15] are supervised learning models that have been applied to a range of problems in cognitive radio networks (e.g., see [16], [17]). Given a training set, including values for both independent and dependent variables, the objective is to predict the value of the dependent variable for new sets of independent variables. If the dependent variable is discrete, such as class label, we speak of classification; otherwise, we speak of regression.

In our scenario, we have a regression task, where the dependent variable is the number of bytes that can be successfully transmitted. The independent variables, also called features, include both the outside conditions, e.g., the power sensed on each channel, and the decision each node could make, e.g., which channel to select. Intuitively, we want to learn such relationships as: (i) If the power sensed on channel 1 is very low and I choose to transmit on channel 1, I will transmit many bytes; (ii) If the power sensed on channel 1 is average and I choose to transmit on channel 1, I will transmit a few bytes using robust MCS schemes and no bytes at all using other ones; (iii) If the power sensed on channel 1 is high, I will transmit no byte under any conditions.

<table>
<thead>
<tr>
<th>Class</th>
<th>Parameter Name</th>
<th>Parameter Description</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Genetic Algorithm</td>
<td>crossover probability</td>
<td>The chance that any single pair of chromosomes will be submitted to the crossover process. This parameter controls the degree to which the parameter space region containing the current population of chromosomes is explored.</td>
<td>0.05/0.25/0.5/0.75/1</td>
</tr>
<tr>
<td></td>
<td>mutation probability</td>
<td>The probability that any single chromosome undergoes mutation. This parameter affects the degree to which the parameter space region under review is expanded.</td>
<td>0.05/0.25/0.5/0.75/1</td>
</tr>
<tr>
<td></td>
<td>sense_probability</td>
<td>The amount by which the sensing fitness is incremented after each non-sensing action. This parameter has the effect of determining how often the sensor information used to estimate fitness is updated.</td>
<td>17/51/101/2050</td>
</tr>
<tr>
<td></td>
<td>weight</td>
<td>The degree to which current sensed information is preferred over past experience. The value is used as the weight of sensor information during the calculation of the weighted average.</td>
<td>0.1/0.5/1/2.5/5</td>
</tr>
<tr>
<td></td>
<td>fitness</td>
<td>The likelihood that sensing is selected instead of relying on past experience. The value of this parameter has the effect of determining how often the sensor information used to estimate fitness is updated.</td>
<td>0.01/0.05/0.15/0.25/0.5</td>
</tr>
<tr>
<td>Multi-Armed Bandit</td>
<td>c</td>
<td>The kernel coefficient.</td>
<td>0.001/0.03/0.07/0.11/0.17</td>
</tr>
<tr>
<td></td>
<td>m</td>
<td>The amount of usefulness that is forgotten about each action in each time slot. It gives the size of the step toward the initial usefulness value.</td>
<td>101/201/301/401/50</td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>The degree to which current sensed information is preferred over past experience. The value is used as the weight of sensor information during the calculation of the weighted average.</td>
<td>0.05/0.25/0.5/0.75/1</td>
</tr>
<tr>
<td></td>
<td>Δ</td>
<td>The likelihood that sensing is selected instead of relying on past experience. The value of this parameter has the effect of determining how often the sensor information used to estimate fitness is updated.</td>
<td>17/51/101/2050</td>
</tr>
<tr>
<td></td>
<td>ζ</td>
<td>Exploration coefficient to balance exploitation and exploration.</td>
<td>1/2/3/5/10/20</td>
</tr>
<tr>
<td></td>
<td>ω</td>
<td>Exploration coefficient to balance exploitation and exploration.</td>
<td>0/1/5/7/17/25/75</td>
</tr>
<tr>
<td></td>
<td>p</td>
<td>The amount of reward for 1 dBm of avoided interference, relative to 1 bit of transmitted data.</td>
<td>0/0.5/1/2/5/20</td>
</tr>
<tr>
<td></td>
<td>β</td>
<td>The degree to which the parameter space region under review is expanded.</td>
<td>0.1/0.5/1/2.5/5</td>
</tr>
<tr>
<td></td>
<td>γ</td>
<td>The likelihood that sensing is selected instead of relying on past experience.</td>
<td>0.01/0.05/0.15/0.25/0.5</td>
</tr>
<tr>
<td>Support Vector Machines</td>
<td>kernel</td>
<td>The kernel coefficient.</td>
<td>0.001/0.03/0.07/0.11/0.17</td>
</tr>
</tbody>
</table>

Algorithm 6 Multi-Armed Bandit Feedback

1: UPDATE_ACTION_INDEX(selected_action)  
2: reward_update := 0  
3: if selected_action = sensing then  
4: for action in available_actions do  
5: if action_power < required_power then  
6: reward_update := reward_update - action_bits  
7: else  
8: reward_update := reward_update + power_multiplier × action_power  
9: end if  
10: end for  
11: else  
12: reward_update := number_received_bits  
13: end if  
14: selected_action_reward := selected_action_reward + reward_update
of the $C$ channels;
- $C$ elements model the channel chosen by the node, with a one-hot encoding;
- $M$ elements similarly model the MCS choice;
- the last element is the chosen transmission power level.

The output variables $y$ are the number of bits successfully transmitted in each case. The feature vectors corresponding to possible decisions are built in a similar way: the first $C$ elements are the same for all vectors, while the others reflect the possible channel, MCS and power choices. Nodes will enact the decision that is expected to result in the highest number of transmitted bytes.

SVM algorithms support multiple kernels, i.e., functions used to transform the non-linear input space into a high-dimensional feature space to efficiently perform the regression task into. In this paper, we adopt the widely used Radial Basis Functions (RBF) kernel. We have three important parameters. The first parameter is the sense probability parameter, which determines the likelihood of a node sensing the environment. The next two parameters are $c$ (the cost parameter) and $\gamma$ (kernel coefficient), which are useful in tuning the SVM learning algorithm.\(^1\) It has to be mentioned that $c$ and $\gamma$ are closely related to one another and the configuration of a single parameter cannot be directly mapped to the throughput of a node. The influence of parameter values on throughput is discussed in Sec. V, while their influence on the classification error is illustrated in Tab. VI in the appendix.

Algorithm 7 SVM Decision

1: train_decisions.PUSH(last_decision)
2: train_outcomes.PUSH(last_tx)
3: if random_number < sense_prob then
4:    return SENSE
5: end if
6: if LENGTH(train_data) < min_train then
7:    return TRANSMIT(random_configuration)
8: end if
9: TRAIN_SVM(train_decisions, train_outcomes)
10: expected_tx $\Leftarrow$ PREDICT_SVM(possible_decisions)
11: return arg max$_{i}$ possible_decisions expected$_i$

Alg. 7 details how an SVM node operates. The node is given the decision made at the previous time step (last_decision) and the amount of transmitted data it resulted in (last_tx). First of all, this information is added to the training set (variables train_decisions and train_outcomes). With probability sense_prob, the node decides to sense in this frame. Otherwise, if the training set is smaller than min_train\(^2\), a random decision is made in Line 7. Otherwise, an SVM regression model is trained based on past information (Line 9) and used, in Line 10, to foresee the outcome (i.e., the expected number of transmitted bytes) of all possible decisions. The decision expected to produce the best result is finally returned in Line 11.

It is important to point out that SVM nodes behave, conceptually, in the same way of other nodes. The firmware parameters are static (e.g., we do not change sense_prob over time), but the model itself is able to learn from new information as it becomes available.

III. Reference Scenarios

In order to develop a good understanding of the factors most important to the success of cognitive radio, we focus on a common structure for the DSA problem. This structure places both primary and secondary users in a common environment, with each attempting to communicate in a shared set of channels. Primary users have privileged access to the channels such that they need not consider the actions of secondary users when accessing spectrum. Secondary users, on the other hand, are obliged to avoid interfering with primary users and other secondary users. It is worth noting that there is no coordination taking place between secondary users. For the sake of tractability, time is considered in terms of discrete rounds. In each time interval, users may employ one of several possible actions, to be discussed below. This scenario provides the basis for several investigations of DSA [18]–[20], as a tractable way to consider the application of various cognitive radio techniques and algorithms.

Furthermore, this scenario fits current regulatory trends for the use of cognitive radio. The licensed shared access (LSA) [21] and President’s council of advisors on science and technology (PCAST) [22] frameworks both fit this general model. Within the LSA framework, secondary users may hold additional privileges in terms of access to spectrum and are bound to specific requirements in terms of avoiding interference with primary user operation. Furthermore, the licensed status of secondary users supports the application of some degree of synchronicity between primary and secondary users, whether through a database or other means. The PCAST framework largely mirrors the LSA framework, with the addition of a third tier of opportunistic unlicensed access. While this open-tier more closely matches much of the discussion in the cognitive radio literature, it poses greater challenges due to the reduced coordination between primary and secondary users. In regard to this open-tier, the synchronicity assumption of the assumed model would need to be relaxed, but the model remains applicable. Thus, our reference scenarios fit the current regulatory frameworks.

For this study we examine a network realization wherein all nodes are placed within a $1,000 \times 1,000$ m\(^2\) area. The primary user transmitter is located in the center of this area, such as illustrated in Fig. 2. The pairs of secondary users that will

\(^1\)There are also other kernels such as linear, polynomial, and sigmoid kernels. We choose RBF since it is capable to model a non-linear relationship between the features and output variables, is generally fast, and it provides reasonably good prediction. An interested reader can also run similar experiments using the different kernels; however, the average training time of the learning algorithm is particularly important here since we aim to run a large-scale experiment involving thousands of simulations, hundreds of nodes, each with up to 1000 decisions to be made. To illustrate this, in the appendix, we provide the average learning time of a node using RBF and the linear kernel. Additionally, different kernel choices might also need a different set of parameters to tune their accuracy. E.g., the polynomial kernel requires the degree parameter, whereas the linear kernel does not require the $\gamma$ parameter.

\(^2\)The parameter min_train has been set to 150 in the simulation experiments.
coding BPSK. It is worth mentioning that the secondary users always use the most reliable option, i.e., 1/4 rate of a transmission procedure or giving up, the primary user with unsuccessful communication attempts. Upon completion or gives up after a certain number of time slots (either transmits data until the communication length is reached or gives up after a certain number of time slots (give-up length) with unsuccessful communication attempts. Upon completion of a transmission procedure or giving up, the primary user changes into the inactive state again. Regarding the MCS, the primary user always uses the most reliable option, i.e., 1/4 rate coding BPSK. It is worth mentioning that the secondary users do not explicitly collect information about the primary user behavior, but instead learn which throughput can be achieved on which channel by using which MCS.

Table II: Available options for the choice of channel and MCS by each secondary user.

<table>
<thead>
<tr>
<th>Decision</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel</td>
<td>Eight available channels, numbered from 1 to 8</td>
</tr>
<tr>
<td>MCS</td>
<td>1/4 rate coding BPSK, 1/3 rate coding BPSK, 1/2 rate coding BPSK, 1/2 rate coding QPSK</td>
</tr>
</tbody>
</table>

A. Channel Utilization Patterns of Primary Users

In the applied channel utilization model, a communication pattern is randomly assigned to each channel, or primary user respectively, which is followed by the primary user over the whole simulation duration. Tab. III gives an overview of the input parameters of the channel utilization model. In every inactive time slot each primary user decides whether to start a new transmission or not based on its activation probability. Once a transmission procedure is started, the primary user either transmits data until the communication length is reached or gives up after a certain number of time slots (give-up length) with unsuccessful communication attempts. Upon completion of a transmission procedure or giving up, the primary user changes into the inactive state again. Regarding the MCS, the primary user always uses the most reliable option, i.e., 1/4 rate coding BPSK. It is worth mentioning that the secondary users communicate with one another are uniformly distributed with a maximum distance of 200 m between transmitter and receiver. Tab. II summarizes the decisions secondary users have to make, and the available options for each of them. There are eight channels given (indexed as 1 through 8) and four modulation schemes. The transmission power is fixed at 12.5 dBm. At the start of each time slot each node determines an action to be applied for that slot. All node actions are then simulated and feedback is returned to the nodes.

Table III: Parameters of the channel utilization model determining each primary user’s channel usage pattern.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activation probability</td>
<td>0/0.125/0.25/0.375/0.5/0.625/0.75/0.875</td>
</tr>
<tr>
<td>Communication length</td>
<td>1/5/10</td>
</tr>
<tr>
<td>Give-up length</td>
<td>1/5/10</td>
</tr>
</tbody>
</table>

Note that we also evaluated a dynamic variant of the channel utilization model, in which primary users switch their usage pattern at randomly selected time slots. The results from those experiments were consistent with the simulations of the more static channel utilization case reported here.

IV. Simulation and Data Processing

In this section, we describe how we perform our simulations and process the resulting logs. Neither task is exceptionally complex in itself, but both require care and deserve an explanation. Fig. 3 depicts the sequence of sub-steps in both tasks: for setting up the simulation, each node needs to be assigned a firmware (step 1) and a location within the simulation area (step 2) prior to executing each simulation run (step 3). During the data processing task, the relevant metrics are extracted (step 4), aggregated (step 5), and finally, visualized (step 6).

A. Simulation Setup

As discussed earlier and summarized in Tab. I, we consider:
- four classes of algorithms, RL, GA, MAB, and SVM;
- for each class, three parameters;
- for each parameter, five values.

As a control case for our simulation study we implemented the Toy Algorithm. This algorithm randomly selects an action regardless of the actions of other users. Each action in the pool of possibilities is assigned equal probability and a new action is selected at each decision point with no memory. Therefore the decision function simply consists of a random action selection and the feedback function is empty.

This gives a total of $4 \cdot 5^3 = 500$ combinations for GA, RL, MAB, and SVM nodes and 501 when including the ToyNode, which has no parameters. Choosing the right combination of class and parameters is the choice that awaits cognitive radio adopters. We refer to such combinations as firmwares, as represented in step 1 of Fig. 3: adopters have to select one of 501 possible firmwares to load into their devices. It is worth stressing that all algorithms remain cognitive, in that they adapt their decision to the surrounding environment and the decisions of other nodes. Firmwares merely determine how this adaptation takes place.

We also generate a total of 3,000 topologies (step 2 in Fig. 3). There is an equal number of nodes running a certain class of algorithm in each topology. Each topology has 24, 60, 96, or 132 transmitter-receiver pairs, randomly scattered throughout a $1,000 \times 1,000$ m$^2$ area, as described in Sec. III.
Nodes belonging to the same source/destination pair are guaranteed to run an algorithm of the same class. We simulate each topology for a time of 1,000 seconds.

All told, we simulate 3,000 · (24+60+96+132) · 2 = 468,000 nodes. To each of these nodes, we assign a randomly chosen firmware, as shown in Fig. 3; on average, each firmware appears about 750 times, in a variety of situations and scenarios. What we evaluate is the performance, i.e., the total amount of successfully transmitted data, throughout all the simulation logs (step 3 in Fig. 3).

Notice that we do not prevent the same firmware from appearing more than once in the same simulation. Similarly, we do not impose that each firmware, or a certain number of nodes of each class, appear a minimum number of times in each simulation. Instead, our purpose is to observe how cognitive radios behave in larger-scale and less predictable environments – one may say, in the wild.

B. Data Processing

Three thousand log files, whose size is around ten megabytes each, represent a sizeable amount of data. These log files are needed since most metrics require to be computed locally first and aggregated globally later – see, e.g., the metric could do better. In order to process this data efficiently, we adopt a map-reduce approach, as outlined in steps 4-5 of Fig. 3.

In the map step, individual log files are processed to extract the relevant metrics. As an example, from each simulation we may want to know the total amount of data transmitted with each firmware, and the number of nodes using it. Files are processed in parallel, so we can have as many map tasks running as there are available cores. It is important, in this step, to reduce the size of data as much as possible; in the previous example, a 10-megabyte file is reduced to some tens of lines, one for each firmware.

In the reduce step, we combine the values resulting from map tasks, computing the global totals – in our example, the average per-firmware performance. Roughly speaking, there is one reduce task per metric to compute, and these tasks can run in parallel. In general, reduce tasks lend themselves to parallelization to a lesser extent than map ones – to generate our results, we needed but eight parallel reduce tasks. On the other hand, they tend to be substantially faster, as they work on smaller amounts of input data.

Through our map-reduce approach, we are able to process all our logs in around half an hour, using a 16-core machine. In addition to making the analysis we present in Sec. V a lot easier to carry out, this means that we are able to process even larger-scale simulation logs with the same approach and, indeed, the same code. Furthermore, our map-reduce approach is readily deployable on a cluster, real or virtual, if need be.

V. Performance Evaluation

As mentioned in Sec. I, we use our simulation results to investigate the effectiveness of different classes of cognitive algorithms, and the sensitivity of each to parameter settings. We also are interested in how effective each algorithm is at making the decisions summarized in Tab. II, and in assessing whether there are nodes whose performance can be improved by a careful choice of the cognitive algorithm to adopt.

---

Note that the ToyNode firmware appears on one fifth of the nodes, while the remaining 500 firmwares are distributed across 374,400 nodes.
A. Classes of Algorithms

The first aspect we are concerned with has to do with the classes of algorithms we discussed in Sec. II: can we expect nodes adopting algorithm A to perform better than nodes adopting algorithm B? Fig. 4 shows that there is indeed a clear difference in performance.

The highest performance is achieved by 24% of MAB nodes, which transmit more than about 99,600 bytes. The remainder of MAB nodes experience lower throughput than most RL, GA, and SVM nodes. The largest fraction of highly performing nodes can be found for RL, since 40% of the nodes are able to transmit more than 84,000 bytes. The curve for SVM shows a similar pattern as the one for RL, with the best performing 40% transmitting more than 65,000 bytes. On the other hand, the curves for RL and GA cross around 0.37 and the ones for SVM and GA around 0.56: we can see that the most disadvantageously configured 36% of RL nodes and 55% of SVM nodes, respectively, would be better off with GA. Moreover, 99% of the GA nodes experience a higher throughput than the worst performing 50% MAB nodes.

B. Firmware Choice

We now look at the problem from a different angle, and examine, for each class of algorithm, how important it is to correctly set the parameters, i.e., using the terminology in Sec. IV, to select the right firmware. This aspect is important for potential adopters of cognitive radios, since it is important that cognition algorithms be robust to a sub-optimal parameter setting. Fig. 5 shows the distribution of the performance of the best, median, and worst firmwares for each algorithm. The
difference between the algorithm classes becomes here more clear. Fig. 5(a) and Fig. 5(c) show that with GA and MAB, we can expect more or less the same performance no matter the firmware we select, i.e., no matter how effective we are at setting the parameters. RL and SVM, as we can see from Fig. 5(b) and Fig. 5(d), behave in a substantially different way: the best firmware has much better performance than the median one, which is still much better than the worst one. Choosing the right firmware, i.e., setting the parameters correctly, is of paramount importance here.

Fig. 6 provides us with an even more clear view of the issue. If we select the best possible firmware for each algorithm class, i.e., if we can select each parameter to its optimal value, then we can expect from RL almost twice the performance of GA and MAB. As our ability to pick the right firmware decreases, the difference in performance decreases. If we are particularly unlucky – or inexperienced – and pick one of the 8 worst-performing firmwares, then GA and MAB offer a better performance than RL. Going from the best to the worst firmware means losing 67% performance with RL and 76% with SVM, but barely 3% with GA and MAB.

The implications for cognitive radio adopters is now very clear: if they can afford to carefully experiment with the parameters and find the ones that best suit their scenario, then RL is the best choice. On the other hand, if such experimentation would be too difficult or time consuming – or, simply, if they want their product to work out-of-the-box in a variety of different scenarios –, then consistent algorithms will be a more reliable, if less performing, alternative.

C. Individual Parameters

We have learned that RL and SVM algorithms are more sensitive than GA and MAB algorithms to suboptimal parameter settings. In the following, we assess which parameters have the highest impact on node performance.

We begin with GA parameters, summarized in Fig. 7. Unsurprisingly, we can see that none of the parameters has a major impact on the total performance. Note that the subfigures are zoomed-in to a small range of throughput values to make the slight differences visible. Setting the crossover probability (Fig. 7(a)) to 0.05, for example, is from the performance viewpoint the same as setting it to 0.5; similarly, multiplying or dividing the fitness step makes almost no difference. The most significant variation is exhibited by the mutation probability parameter, differing by about 1,000 bytes in successfully transmitted data. The insignificant variation observed is a remarkable and desirable property. Intuitively, it is connected with the way genetic algorithms operate, trying out several options and eventually sticking to the best one. It also suggests that the time required to reach convergence is substantially shorter than our simulation time, for all parameter settings.

Fig. 8, devoted to RL parameters, shows a different situation: parameters have various degrees of influence over the overall performance, and said influence is sometimes very strong.
Forget factor values (Fig. 8(a)) below 30 impair the performance, while the performance is better for higher values, with stable median and interquartile range. Similar results apply to sense weight (Fig. 8(b)): low values degrade the throughput, while medium or high ones are equally good. As we can see from Fig. 8(c), the sense probability is the parameter with the strongest influence: the lower its value, the better the performance. While the throughput is quite constant among all nodes in settings with sense probability values of 0.25 and higher, there is a greater performance range for nodes with low performance, i.e., the whiskers below the boxes.

The results for MAB parameters are summarized in Fig. 9. The results for the exploration coefficients $c$ (Fig. 9(a)) and $\zeta$ (Fig. 9(b)) do not reveal a clear tendency for optimally setting these parameter values. It can be merely stated that the box plots suggest combining $c = 1$ with $\zeta = 10$ yields the highest throughput. Moreover, adopters of MAB will have to take a closer look at the interplay of those two coefficients. The power multiplier (Fig. 9(c)) exhibits a clearer pattern, showing a better performance with higher values.

Fig. 10 depicts the performance results for the value variations of SVM parameters. The strongest influence of parameterization can be found for the sense probability (Fig. 10(a)) and the $\gamma$ value of the RBF kernel (Fig. 10(b)). The lower the sense probability the higher the throughput of a firmware, which is in line with the sense probability results for RL (see Fig. 8(c)). The boxes for $\gamma$ suggest that this parameter should not be assigned a too small value (<0.01), neither values of 1.0 or above. Although the selection of parameter $c$ influenced the classification success in the training experiment reported in the appendix, its effect on the transmit performance seems negligible from the figure. However, as discussed in Sec. II-D, $\gamma$ and $c$ are closely related and their influence on the throughput can therefore hardly be interpreted independently.

To quantify the sensitivity of performance results to parameter selection, we calculate a sensitivity factor $S_P$ [23] for two of the aggregate measures visualized by the boxplots in Fig. 7 to Fig. 10, namely median ($m$) and interquartile range ($IQR$). Let $S_P$ be defined as the ratio of the relative deviation, by which the values of an output parameter $Y$ depart from a certain state $y^*$ (the minimum value) to the relative deviation of the set of parameter values $P$. Note that $Y = \{y_0, y_1, \ldots, y_n\}$ contains the performance results for all $n$ topologies and $P = \{p_0, p_1, \ldots, p_k\}$, where $k$ is the number of different values per parameter. Furthermore, $A$ denotes the aggregate measure selected for the calculation:

$$S_P(A, Y, P) = \frac{\Delta\{A(y_0), A(y_1), \ldots, A(y_n)\}}{\min\{A(y_0), A(y_1), \ldots, A(y_n)\}} / \frac{\Delta(P)}{\min(P)},$$

where $\Delta(U) = \max(U) - \min(U)$.

In Tab. IV the parameters are ranked according to the sensitivity of each algorithm. The results suggest that especially the
parameter space for the RL parameters sense probability and forget factor have to be chosen carefully. The power multiplier also shows high sensitivity according to the factor $S_P(m)$. However, it can be found at the bottom of the ranking – similar to the other MAB parameters – when relying on the $S_P(IQR)$ since this algorithm shows generally large IQRs in the performance. Other parameters exhibiting higher sensitivity than the majority are the sense probability of SVM (ranked 5 and 4) and the sense weight of RL (ranked 6 and 3).

**Table IV. Sensitivity Ranks $R_m$ and $R_{IQR}$ according to the factors $S_P(m)$ and $S_P(IQR)$, respectively.**

<table>
<thead>
<tr>
<th>$R_m$</th>
<th>$R_{IQR}$</th>
<th>Parameter Name</th>
<th>Class</th>
<th>$S_P(m)$</th>
<th>$S_P(IQR)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Sense_probability</td>
<td>RL</td>
<td>0.09089</td>
<td>0.20151</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Forgetting_factor</td>
<td>RL</td>
<td>0.03747</td>
<td>0.1652</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Power_multiplier</td>
<td>MAB</td>
<td>0.04194</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Sense_probability</td>
<td>SVM</td>
<td>0.01703</td>
<td>0.01480</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Sense_weight</td>
<td>RL</td>
<td>0.01133</td>
<td>0.02514</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>$c$</td>
<td>MAB</td>
<td>0.03086</td>
<td>0.08002</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>SVM</td>
<td>SVM</td>
<td>0.00525</td>
<td>0.00816</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>$c$</td>
<td>SVM</td>
<td>0.00137</td>
<td>0.00240</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>Mutations_probability</td>
<td>GA</td>
<td>0.00008</td>
<td>0.00002</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>Crossover_probability</td>
<td>GA</td>
<td>0.000013</td>
<td>0.000107</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>Sense_likelihood</td>
<td>GA</td>
<td>0.00009</td>
<td>0.00038</td>
</tr>
</tbody>
</table>

In addition to providing some insight on how individual algorithms work, these results provide valuable guidance for cognitive radio adopters. They should adopt GA or MAB if they cannot afford to experiment with parameter values, and RL or SVM if they can. Special attention should be devoted to setting the right value for parameters to which the algorithms exhibit high sensitivity.

**D. Quality of Decisions**

Cognitive radio algorithms make decisions. So far, our focus has been on the overall effectiveness of these decisions, i.e., on network performance. In the following, we give a closer look to how good each algorithm is at making the decisions discussed in Sec. II.

There are essentially two decisions these algorithms have to make: the channel to transmit on, and the modulation and coding scheme (MCS) to use. In the following, we look at how good GA, RL, MAB, and SVM algorithms are at making these decisions, and how the situation improves if we restrict our attention to the best firmwares of each class. We compare the actual decisions the algorithms make against the ideal ones, i.e., the one they would have made if they knew how all other nodes on the topology, primary and secondary alike, would have behaved. Of course, the better an algorithm, the more likely that the decisions it makes coincide with the ideal ones.

Fig. 11 shows, for each class of algorithms, the number of simulation time slots used for successful transmissions where the best possible MCS had been employed, transmissions that were successful but could have used a higher performing MCS and failed attempts that would have been successful if a different MCS had been used. Intuitively, we would like to have the latter two groups of bars as low as possible, as they correspond to wasted opportunities to transmit (more) data.

Consistently with what we might expect, Fig. 11(a) shows that RL is associated to more successful transmissions than all the other algorithms. Less obviously, it tends to have fewer successful transmissions where a better MCS could have been used, and considerably more avoidable failures. Intuitively, RL tends to take more risks, daring to select higher order MCS and oftentimes transmitting more data. Despite the similarities SVM has with RL, it is slightly less successful in transmitting high amounts of data; even when not considering 150 time slots belonging to the initial training phase, SVM seems to choose a too low order MCS in some cases. Moving to Fig. 11(b), where only the best performing firmwares are considered, we can observe no big change for GA and MAB – which is consistent with Fig. 6, showing that all GA and MAB firmwares have a similar average performance. Good RL firmwares have more entirely successful and partially successful transmissions: consistently with Fig. 8(c), they invest fewer time slots in sensing.

Fig. 12 shows the number of time slots in which nodes perform successful transmissions and transmissions that would have been successful if a different channel – a channel with less interference – were used. Fig. 12(a) shows that RL and SVM have a substantially higher number of avoidable failures, i.e., they are not as effective as GA and MAB in choosing the right channel to transmit on. Also notice that the number of successful transmissions is higher for GA and MAB than for RL and SVM; the latter is able to transmit more data in each successful transmission due to its effectiveness in choosing the MCS, as shown in Fig. 11. Moving to the best-performing firmwares in Fig. 12(b), we observe again no difference for GA and MAB, but more successful transmissions for RL and SVM. Consistently with Fig. 11(b), SVM and RL also have more avoidable failures; recall that a good RL or SVM firmware makes essentially more transmission attempts, both successful and unsuccessful.

**E. Performance over Time**

One critical factor for the real-world deployment of cognitive radio algorithms is the time it takes until an algorithm converges. Fig. 13 shows how the average fraction of transmitted data per transmission evolves over the entire simulation time of 1,000 seconds. Note that the average throughput per transmission has been computed over all firmwares of a class and is given as a fraction of the maximum possible throughput for one node per timeslot. GA and MAB need little time to converge, which is reflected by a constant average right from the beginning. RL exhibits a longer learning phase, reaching a steady average of around 0.7 after 130 seconds. SVM exhibits a similar behavior as RL after building up a first training set. Recall that the minimum size of the training set has been configured to be 150 in our simulation experiment, while prior to reaching this number the algorithm takes random decisions to collect training instances. It has to be pointed out that the average throughput achieved is still gradually increasing at the end of the simulation time, when it yields around 0.61. Particularly in the long run, when the initial training time plays a lesser role, it might thus be beneficial to deploy SVM.
VI. RELATED WORK

There are very few examples in the literature of works that examine the factors that underpin success for cognitive radio algorithms. Most of the available work in this area overlaps with performance evaluation for cognitive radio, often examining single applications of the cognitive radio concept and typically focusing on DSA. Each work applies a different methodology for determining the characteristics of cognitive radios and identifying factors that lead to their success. Zhao [24] provides a report card-like system which allows some comparison between DSA techniques. Dietrich [25] constructs a cognitive model by probing a DSA CR with a psychometric approach. Thompson [26] eschews detailed modeling of internal operation in favor of decoding the operation purely from observation of DSA CR behavior. Giorgetti [27] directly compares three paradigms for DSA, interwave, underlay, and overlay sharing, with the goal of developing an understanding of CR operation under these paradigms. In each of these works the authors attempt to characterize the operation of a CR system interacting with some primary system. We extend these efforts by examining the interaction between CR systems attempting to dynamically utilize spectrum.

The second category of related work focuses on the development of solutions to the DSA problem. While this area is not the central focus of our work, this literature is pertinent here in establishing the key approaches for DSA cognitive radios.
Here we provide a brief and broad categorization of the various approaches to the DSA problem represented in literature; readers are directed to [28]–[30] for more complete discussions of these notions. Specifically we limit our discussion to machine learning, reinforcement learning, Markovian, and evolutionary algorithms for addressing the DSA problem.

Machine learning based approaches focus on imbuing radios with the ability to learn the patterns of incumbent users, subsequently enabling opportunistic spectrum use. Several individual techniques fall into this broad category, including neural network based approaches [31], nearest neighbor clustering analysis [32], Bayesian reasoning approaches [33], and kernel based statistic analysis [16]. Each technique within this category attempts to distill experience into a form that provides useful guidance for future decision making. Specifically, the methodologies within this group rely on training sets or decision thresholds for decision making. These training sets can be either provided externally or collected in-mission. This category is represented by the SVM algorithm in our analysis.

Reinforcement learning echoes the goals of the machine learning category, albeit through a different philosophy of operation. Broadly, reinforcement learning relies on organized book-keeping to realize the potential utility of past experience. As such, this technique does not directly rely on a training set for operation and rather simply accrues knowledge in the form of weighted rewards on the basis of a provided definition for goodness [34]. While variations on this theme certainly exist, such as q-learning [35] or temporal-difference learning [36], the base approach remains the same. Herein, we examine the fundamentals of this category with our RL algorithm.

Markovian techniques support opportunistic spectrum use by capturing the behavior of a system in terms of transitions between various states. Within this category, which is occasionally viewed as a sub-category to machine learning, the system at hand is assumed to operate as a finite state machine, where each state influences the decisions made within a system differently. Techniques within this category attempt to build and apply knowledge of the probabilities of the system transitioning between various states. This may be accomplished through direct examination of Markov decision processes [37] or obscured examinations of the same in the form of partially observable Markov decision processes [38]. Alternatively the multi-armed bandit approach provides a slightly different formulation of the problem that simply examines interacting and reasoning in cognitive radio networks,” in IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2011, pp. 198–203.


While these categories capture some of the techniques most widely applied to the DSA, they represent only a brief and broad overview. Additional techniques such as game theoretic approaches [40], [41] or signal processing schemes [42] are certainly available and impactful, but fall outside of the scope of the investigation herein.

A final category of related work focuses on the selection of optimal parameters for a given algorithm approach. This area, termed meta-cognition, employs optimization to automatically tune the operation of the CR. As such, the work in this area considers methods to determine the impact of various parameters on CR operation. Largely started with the work of Gadhik [1], this area has seen little take-up, although some recent work, e.g., that of Asadi [2], indicates a growing interest in the systematic determination of appropriate parameter setting for CR systems. Note that work in this area focuses on optimization of parameter settings rather than their analysis.

VII. Conclusion

We studied a representative set of four cognitive radio algorithms of different classes – namely reinforcement learning schemes, genetic algorithms, multi-armed bandit approaches, and supervised learning schemes – comparing them across three thousand simulations. To ensure generality of the study, we applied the original versions of the algorithms, as we want to evaluate their fundamental attributes. To the best of our knowledge, our work is the first within the cognitive radio community to promote the comparison of different types of algorithms in a large-scale scenario, considering that pairs of cognitive radios in the same network may employ different decision making approaches.

The results show that some approaches provide better performance with an optimal parameter setting while others provide consistent performance, i.e., are less sensitive to sub-optimally set parameters. Although high and consistent performance are desirable, our study suggests that there is a tradeoff between performance and consistency. We believe that our evaluation concept will foster future work within the cognitive radio community to identify algorithms and parameter spaces that offer the full range of desired features.

REFERENCES


values considered in the table below. Note that the SVM-based algorithm as well as the three previously described algorithms – GA, RL, and MAB – have been implemented in Python [43].

<table>
<thead>
<tr>
<th>c</th>
<th>γ</th>
<th>Linear</th>
<th>RBF</th>
<th>c</th>
<th>γ</th>
<th>Linear</th>
<th>RBF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.001</td>
<td>3.6</td>
<td>&lt;0.01</td>
<td>10</td>
<td>0.001</td>
<td>44.4</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>0.1</td>
<td>0.01</td>
<td>&lt;0.01</td>
<td>10</td>
<td>0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>&lt;0.01</td>
<td>10</td>
<td>0.1</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>0.1</td>
<td>10</td>
<td>&lt;0.01</td>
<td>10</td>
<td>0</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>1</td>
<td>0.001</td>
<td>7.2</td>
<td>&lt;0.01</td>
<td>0.1</td>
<td>44.4</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>&lt;0.01</td>
<td>20</td>
<td>0.1</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>&lt;0.01</td>
<td>20</td>
<td>1</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>&lt;0.01</td>
<td>20</td>
<td>10</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>5</td>
<td>0.001</td>
<td>36.4</td>
<td>&lt;0.01</td>
<td>0</td>
<td>20</td>
<td>44.4</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>5</td>
<td>0.01</td>
<td>&lt;0.01</td>
<td>0.1</td>
<td>20</td>
<td>0.1</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>&lt;0.01</td>
<td>0.1</td>
<td>10</td>
<td>0.1</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>&lt;0.01</td>
<td>0.1</td>
<td>10</td>
<td>0.1</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
</tbody>
</table>

Tab. VI illustrates the classification accuracy of different RBF kernel parameter settings by means of Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE).

<table>
<thead>
<tr>
<th>c</th>
<th>γ</th>
<th>RMSE</th>
<th>MAE</th>
<th>c</th>
<th>γ</th>
<th>RMSE</th>
<th>MAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.001</td>
<td>11.008</td>
<td>4.018</td>
<td>10</td>
<td>0.001</td>
<td>8.898</td>
<td>2.340</td>
</tr>
<tr>
<td>0.1</td>
<td>0.01</td>
<td>13.666</td>
<td>5.391</td>
<td>10</td>
<td>0.01</td>
<td>10.079</td>
<td>4.734</td>
</tr>
<tr>
<td>0.1</td>
<td>0.1</td>
<td>13.910</td>
<td>5.506</td>
<td>10</td>
<td>0.1</td>
<td>12.858</td>
<td>7.973</td>
</tr>
<tr>
<td>0.1</td>
<td>1</td>
<td>13.907</td>
<td>5.520</td>
<td>10</td>
<td>1</td>
<td>13.442</td>
<td>9.527</td>
</tr>
<tr>
<td>0.1</td>
<td>10</td>
<td>13.907</td>
<td>5.521</td>
<td>10</td>
<td>10</td>
<td>13.473</td>
<td>9.591</td>
</tr>
<tr>
<td>1</td>
<td>0.001</td>
<td>9.365</td>
<td>2.161</td>
<td>20</td>
<td>0.001</td>
<td>8.818</td>
<td>2.479</td>
</tr>
<tr>
<td>1</td>
<td>0.01</td>
<td>11.413</td>
<td>4.801</td>
<td>20</td>
<td>0.01</td>
<td>10.159</td>
<td>5.060</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>13.585</td>
<td>5.842</td>
<td>20</td>
<td>0.1</td>
<td>12.937</td>
<td>8.698</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>13.571</td>
<td>5.980</td>
<td>20</td>
<td>1</td>
<td>14.243</td>
<td>11.129</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>13.571</td>
<td>5.986</td>
<td>20</td>
<td>10</td>
<td>14.322</td>
<td>11.259</td>
</tr>
<tr>
<td>5</td>
<td>0.001</td>
<td>9.037</td>
<td>2.199</td>
<td>5</td>
<td>0.01</td>
<td>10.128</td>
<td>4.375</td>
</tr>
<tr>
<td>5</td>
<td>0.1</td>
<td>12.879</td>
<td>7.046</td>
<td>5</td>
<td>1</td>
<td>12.979</td>
<td>7.759</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>12.986</td>
<td>7.791</td>
<td>5</td>
<td>10</td>
<td>12.986</td>
<td>7.791</td>
</tr>
</tbody>
</table>

Andrea Hess is a Research Fellow at Trinity College Dublin, Ireland, affiliated with the CONNECT Centre for Future Networks. She received a Ph.D. in Computer Science in 2014 from University of Vienna, Austria. During her Ph.D. studies she was a research and teaching assistant at the Distributed Systems group there and a Marietta Blau Fellow at the Department of Communications and Networking at Aalto University, Finland. Between 2014 and 2015 she was an ERCIM Marie Curie Fellow at SICS, Sweden. Her research interests include wireless self-organizing networks and realistic mobility and demand modeling.

Tri Kurniawan Wijaya received a Ph.D. in Computer Science from the École Polytechnique Fédérale de Lausanne (EPFL), Switzerland, in 2015 and an M.Sc. in Computational Logic from Dresden University of Technology, Germany, and Vienna University of Technology, Austria, in 2011 with distinction. His research interest is about data analytics and mechanisms design, especially in the field of smart energy. He has been awarded an Erasmus Mundus scholarship from the European Commission and the IBM PhD Fellowship Award 2013-2014.

Nicholas J. Kaminski conducts research focused on extending the bounds of wireless technology by deploying targeted intelligence to act in harmony with flexible radio systems. He advances distributed radio intelligence through experimentation-based research and furthers wireless communications by incorporating techniques for understanding communications from complex systems science. Dr. Kaminski received his master’s degree in Electrical Engineering in 2012 and his Ph.D. in 2014 from Virginia Tech, USA. During this time he was funded as a Bradley Fellowship at the Bradley Department of Electrical and Computer Engineering at Virginia Tech, USA.

Luiz A. DaSilva holds the chair of Telecommunications at Trinity College Dublin. He also has an adjunct research professor appointment in the Bradley Department of Electrical and Computer Engineering, Virginia Tech. His research focuses on distributed and adaptive resource management in wireless networks, and in particular wireless resource sharing, dynamic spectrum access, and the application of game theory to wireless networks. He is currently a principal investigator on research projects funded by the US National Science Foundation, the Science Foundation Ireland, and the European Commission under Horizon 2020 and FP7. He is a co-principal Investigator of CONNECT, the Telecommunications Research Centre in Ireland. He is a fellow of the IEEE, for contributions to cognitive networking and resource management for wireless networks, and an IEEE Communications Society distinguished lecturer.