Cognitive Radio Algorithms Coexisting in a Network: Performance and Parameter Sensitivity

Cognitive Radio Algorithms Coexisting in a Network: Performance and Parameter Sensitivity

Andrea Hess, Francesco Malandrino, Nicholas Kaminski, *Member, IEEE,* Tri Kurniawan Wijaya, Luiz A. DaSilva, *Fellow, IEEE*

***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.**

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.

Each of these choices is critical for network performance, and each requires a deep knowledge of cognitive radios and cognitive networks. Indeed, the plethora of proposed alternatives in the cognitive radio literature, along with the fact that some of them differ in but nuances and details, makes plug-and-play alternatives such as Wi-Fi or ZigBee more appealing.

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.

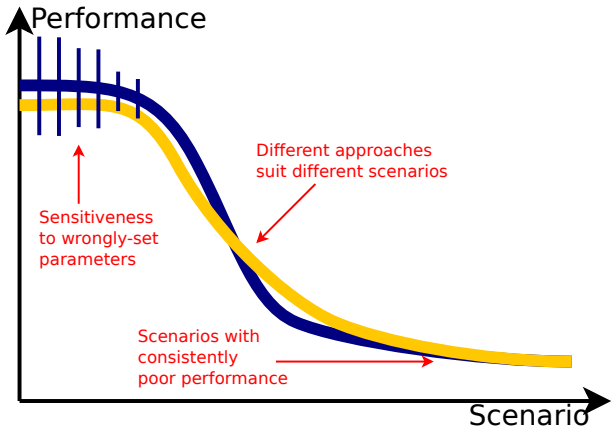


Fig. 1. We compare different classes of cognitive radio algorithms and, for each algorithm, a variety of possible parameter settings. A first question we ask is whether there is an approach that is consistently more effective than alternative ones. We are also interested in studying how sensitive each approach is to errors in setting its parameters. Indeed, some approaches (the blue one in the figure) can perform very well if their parameters are set optimally, and not so well otherwise. In many real-world cases, when setting the parameters optimally is very difficult or even impossible, we may prefer another approach (such as the yellow one) whose performance is lower on average but more consistent. Finally, we are concerned about the "tail" of scenarios where nodes, no matter the algorithm they use and the parameters thereof, always have very poor performance: we wish to characterize these scenarios, and assess how frequently they occur.

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

A. Hess, N. Kaminski, and L.A. DaSilva are with Trinity College Dublin, CONNECT Centre for Future Networks, Dublin, Ireland. F. Malandrino is with the Politecnico di Torino, Italy. Jerusalem, Israel. T.K. Wijaya is now with Google Dublin, Ireland. 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.

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 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 special, 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 examplified in [1] and [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.

The remainder of this paper is organized as follows. We present the classes of cognitive algorithms we examine, as well as their parameters, in Sec. II. Then, in Sec. III, we present the details of the scenarios we use to test those algorithms. In Sec. IV we discuss how we perform our simulations, and process the resulting data. Our results are summarized and discussed in Sec. V. After reviewing related work in Sec. VI, we conclude the paper in Sec. VII.

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]–[9] 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 scheme, form the action space available to the reinforcement learning algorithm. The reinforcement learning 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 reinforcement learning 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, on Line 1, is the determination of 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.

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
```

This record-keeping is the focus of the feedback function for the reinforcement learning 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, as in 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, as in Line 11. This usefulness update is then combined to the prior value of the action usefulness in a weighted average using the sensing weight, which is shown in 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, as in Line 18. Alternatively, three times the power used by the failed action is subtracted from the usefulness, as in Line 20.

B. Genetic Algorithm

Our Genetic Algorithm (GA) provides a basic version of the classic technique that has long been studied for cognitive radios [10]–[13].

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.

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.

Algorithm 3 Genetic Algorithm Decision

```
1: INITIALIZE(population[i])    ∀i : 1 ≤ i ≤ N
2: while number_generations ≤ 20 do
3:    pairs ⇐ FIND_PAIRS(population)
4:    for pair in pairs do
5:       if ROLL_DICE() ≤ crossover_probability then
6:          CROSSOVER(pair)
7:       end if
8:    end for
9:    for chromosome in population do
10:      if ROLL_DICE() ≤ mutation_probability then
11:         MUTATE(chromosome)
12:      end if
13:   end for
14:   best_chromosome ⇐ 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) ≤ sense_fitness then
22:   best_chromosome ⇐ sense_action
23: end if
24: return best_chromosome
```

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 ⇐ 0
4: else
5:    sense_fitness ⇐ 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. [14]. As such, this algorithm captures the

TABLE I. A SUMMARY OF THE CLASSES OF ALGORITHMS WE CONSIDER AND THE PARAMETERS THEREOF.

Class	Parameter Name	Parameter Description	Parameter Values
Reinforcement Learning	forget_factor	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.	10 / 20 / 30 / 40 / 50
	sense_weight	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.	0.05 / 0.25 / 0.5 / 0.75 / 1
	sense_probability	The likelihood that sensing is selected instead of relying on past experience.	0.01 / 0.05 / 0.15 / 0.25 / 0.5
Genetic Algorithm	crossover_probability	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.	0.05 / 0.25 / 0.5 / 0.75 / 1
	mutation_probability	The probability that any single chromosome undergoes mutation. This parameter affects the degree to which the parameter space region under review is expanded.	0.05 / 0.25 / 0.5 / 0.75 / 1
	sense_fitness_step	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.	1 / 5 / 10 / 20 / 50
Multi-Armed Bandit	ζ	Exploration coefficient to balance exploitation and exploration.	1 / 2.5 / 5 / 10 / 20
	c	Exploration coefficient to balance exploitation and exploration.	0.1 / 0.5 / 1 / 2.5 / 5
	power_multiplier	The amount of reward for one dBm of avoided interference, relative to 1 bit of transmitted data.	0 / 1 / 2 / 5 / 10
Support Vector Machines	sense_probability	The likelihood that sensing is selected instead of relying on past experience.	0.01 / 0.05 / 0.15 / 0.25 / 0.5
	c	The cost parameter: penalty of the error term.	0.1 / 1 / 5 / 10 / 20
	γ	The kernel coefficient.	0.001 / 0.01 / 0.1 / 1 / 10

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 scheme (paralleling the transmit actions employed by the reinforcement learning 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 index value.

Algorithm 5 Multi-Armed Bandit Decision

```
1: selected_action ⇐ FIND_MAX_INDEX(records)
```

In contrast to the operation of the GA, the operation of the MAB primarily occurs within the feedback function, as shown in Alg. 6. The first task of the algorithm is to update the index for the most recently selected action. This task, which is accomplished by the UPDATE_ACTION_INDEX subroutine, directly follows the iterative calculation of the upper confidence bound (UCB_V) based coefficients proposed for cognitive radio use in [14]. The parameters ζ and c influence the index calculations. Full details regarding index calculation, including an analysis of regret and an iterative algorithm, are provided in [14]. The remainder of the algorithm focuses on determining the reward for the most recently selected action. If the most recent action was a sensing action, the reward update represents the opportunity cost of sensing. Specifically, Line 6 subtracts the number of bits for all actions that would have resulted in successful transmission from the reward update value. Meanwhile, Line 8 adds the scaled interference power of would-be unsuccessful actions avoided by sensing to the reward update. Alternatively, if the most recent action was a transmission, the reward update is set directly as the number of successfully transmitted bits. The reward update is then added to the reward value of the most recent action.

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
```

The MAB algorithm is controlled through a set of parameters. The first two of these parameters ζ and c directly influence the coefficient calculation by altering the balance of exploitation and exploration; for further details see [15]. The authors of [15] show that an optimal setting for these parameters exists in the region $\zeta \geq 1$, $c = 1$ and the authors of [14] extend this region to $3\,\zeta\,c > 1$. The final parameter in our algorithm, the power multiplier, provides the reward per one dBm of avoided interference power. Together these parameters guide the operation of the MAB algorithm.

D. Support Vector Machines

Support Vector Machines (SVMs) [16] are supervised learning models that have been applied to a range of problems

in cognitive radio networks (e.g., see [17], [18]). 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 transferred. 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:

- if the power sensed on channel 1 is very low and I choose to transmit on channel 1, I will transmit many bytes;
- 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;
- if the power sensed on channel 1 is high, I will transmit no byte under any conditions.

More formally, the *feature vectors* $x \in X$ are composed by $2C + M + 1$ elements, where C and M are the number of existing channels and MCS schemes:

- C elements report the power levels last sensed on each 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 γ (kernel coefficient), which are useful in tuning the SVM learning algorithm.[1] It has to be mentioned that c and γ 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. VII 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 ⇐ PREDICT_SVM(possible_decisions)
11: return arg max_possible_decisions expected_tx
```

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], then 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,

[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, we note that different kernel choices might also need different set of parameters to tune their accuracy. For example, the polynomial kernel requires the *degree* parameter, whereas the linear kernel does not require the γ parameter.

[2] The parameter *min_train* has been set to 150 in the simulation experiments.

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 [19]–[21], 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) [22] and President's council of advisors on science and technology (PCAST) [23] 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.

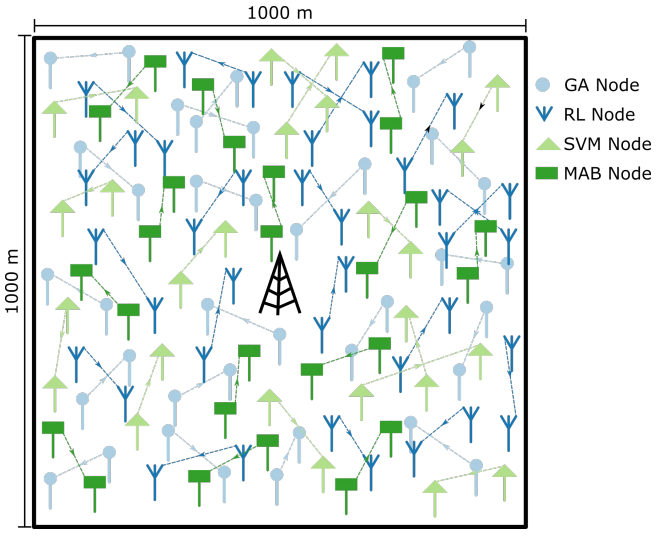


Fig. 2. Illustration of a sample network topology for a heterogeneous cognitive radio scenario spanning 1 km^2 such as studied in our evaluation. The topology comprises one primary user located in the center and nodes running GA, RL, MAB, or SVM mechanisms acting as secondary users.

TABLE II. AVAILABLE OPTIONS FOR THE CHOICE OF CHANNEL AND MCS BY EACH SECONDARY USER.

Decision	Options
Channel	Eight available channels, numbered from 1 to 8
MCS	1/4 rate coding BPSK, 1/3 rate coding BPSK, 1/2 rate coding BPSK, 1/2 rate coding QPSK

For this study we examine a network realization wherein all nodes are placed within a $1{,}000 \times 1{,}000\,\mathrm{m}^2$ 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 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.

A. Channel Utilization Patterns of Primary Users

In the basic 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. 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 III. PARAMETERS OF THE CHANNEL UTILIZATION MODEL DETERMINING EACH PRIMARY USER'S CHANNEL USAGE PATTERN.

Parameter Name	Parameter Values
Activation probability	0 / 0.125 / 0.25 / 0.375 / 0.5 / 0.625 / 0.75 / 0.875
Communication length	1 / 5 / 10
Give-up length	1 / 5 / 10
Switching probability q_s	0 / 0.125 / 0.25 / 0.375 / 0.5 / 0.625 / 0.75 / 0.875 / 1

For the dynamic version of the channel utilization model we introduce a *switching probability* parameter (q_s), which determines if and how frequently a primary user changes its communication pattern. For example, with q_s set to 0, the PU does not change its behavior, while with $q_s = 1$ the primary user decides in each inactive time slot about changing the parameter setting of its channel utilization model. Regarding the MCS, the primary user always uses the most reliable option, i.e., 1/4 rate coding BPSK.

For the experiments described in the following section, we first use the basic channel model and later compare the performance results to those of the dynamic version in Sec. V-F.

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 as 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. For the supervised learning scheme SVM it has to be noted that, every time the classifier cannot be trained, the node takes a random decision. This can be the case, for example, if the training dataset is too small or if the training or test set contain a faulty value that cannot be handled by the classifier – which is likely to happen as well in real-world deployments.

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 $1{,}000 \times 1{,}000\,\mathrm{m}^2$ 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 $3,000 \cdot \frac{24+60+96+132}{4} \cdot 2 = 468,000$ 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[3], 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.

[3]Note that the ToyNode firmware appears on one fifth of the nodes, while the remaining 500 firmwares are distributed across 374,400 nodes.

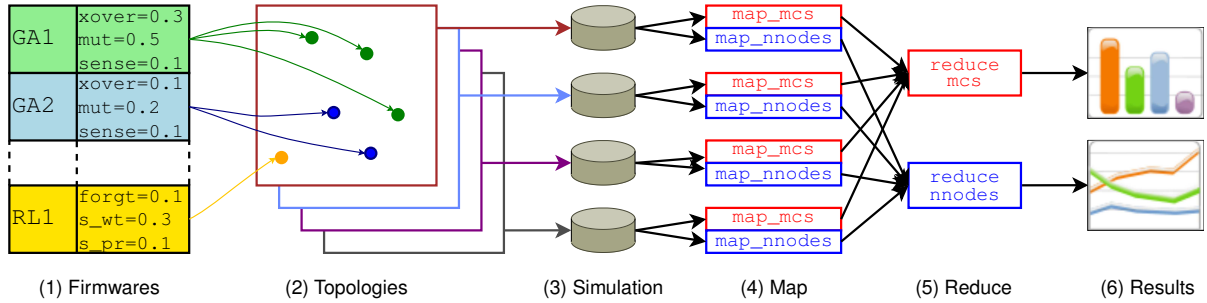


Fig. 3. Where our results come from. We begin with firmwares (step 1), i.e., parameter settings, and network topologies (step 2). Each node in each topology is randomly assigned a firmware, as explained in Sec. IV-A, and the resulting networks are simulated. The resulting simulation logs (step 3) are processed in a map-reduce fashion, as explained in Sec. IV-B: in the *map* step, each log file is individually processed (step 4), and the relevant metrics are extracted; in the *reduce* step, individual outputs are aggregated (step 5) to obtain the final results (step 6).

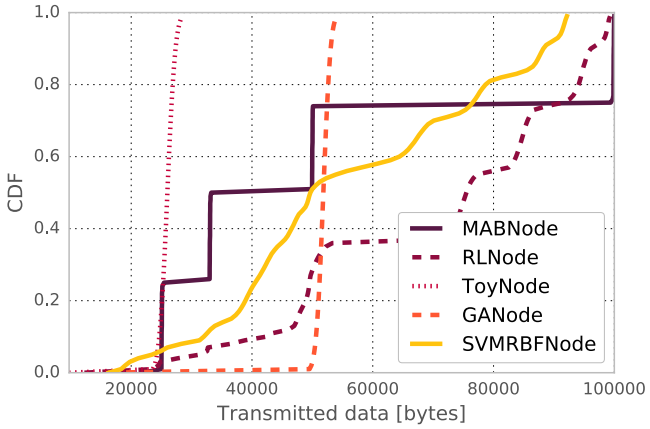


Fig. 4. CDF of the performance for different classes of algorithms. A curve represents the total amount of data transmitted by each of the 93,600 nodes deployed per algorithm.

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.

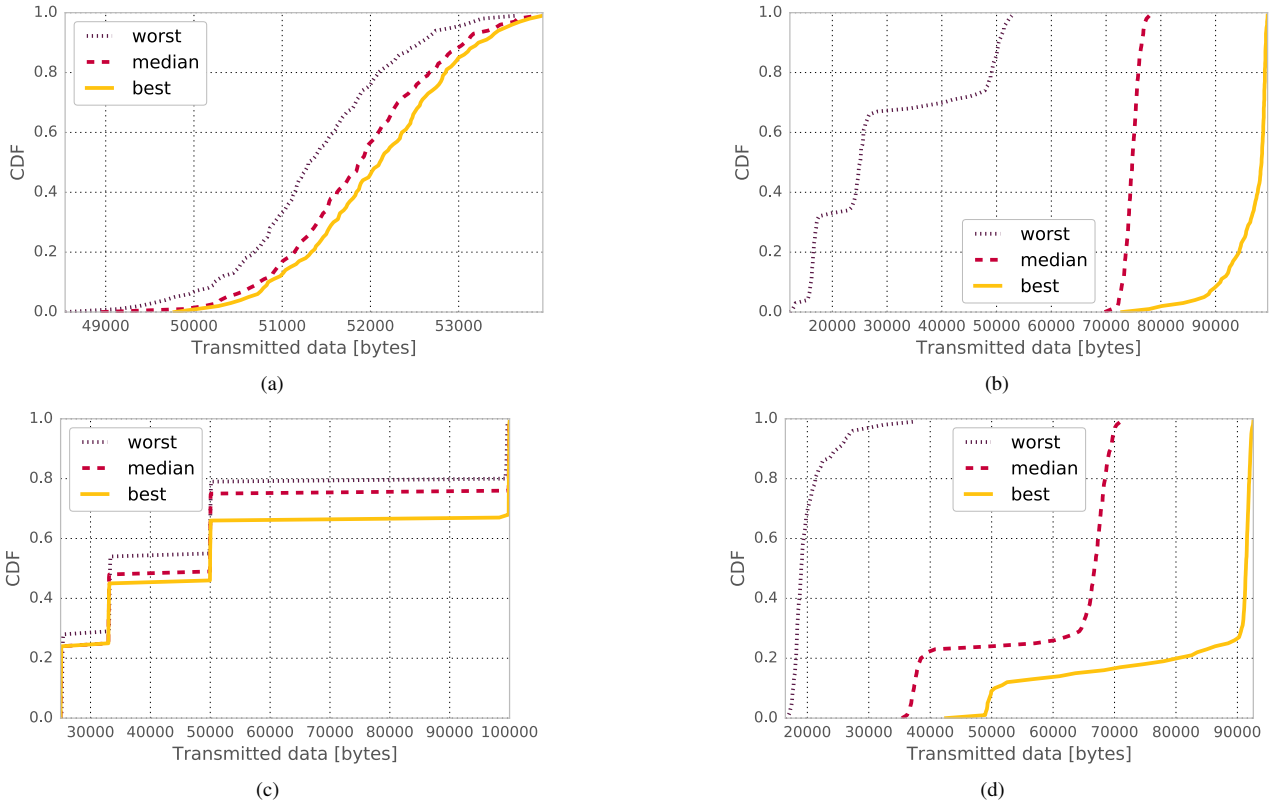


Fig. 5. CDF of the performance for the best, median and worst firmware, for (a) GA, (b) RL, (c) MAB, and (d) SVM algorithms. Note that subfigure (a) has been zoomed-in to make the differences between the firmware categories visible.

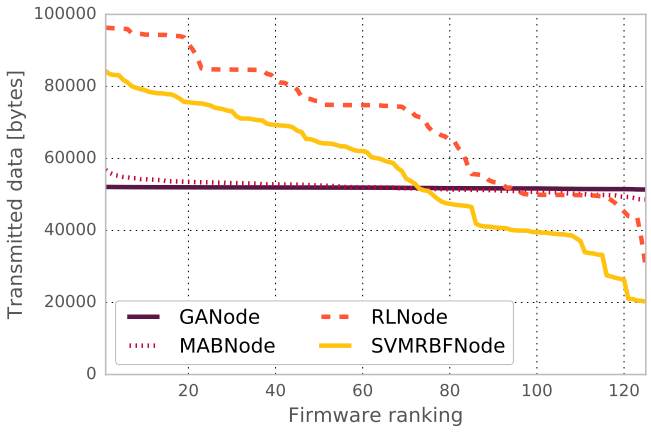


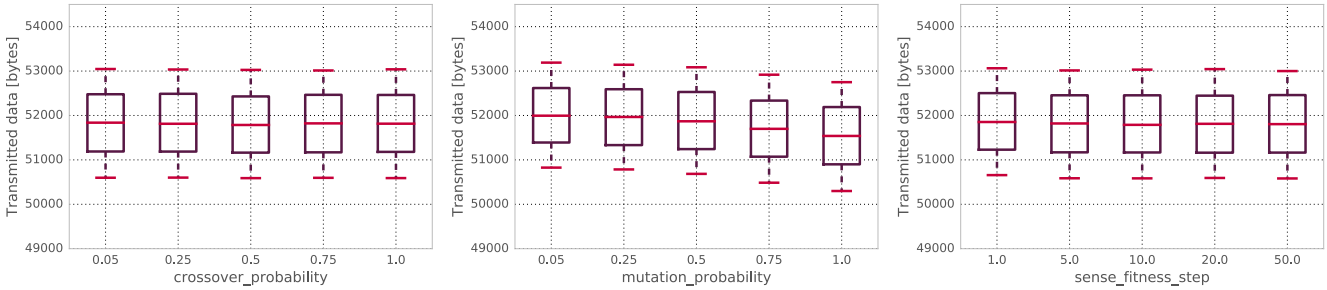
Fig. 6. Average performance of each algorithm's firmwares, which are ranked according to the total number of bytes transmitted on average per node in descending order.

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.

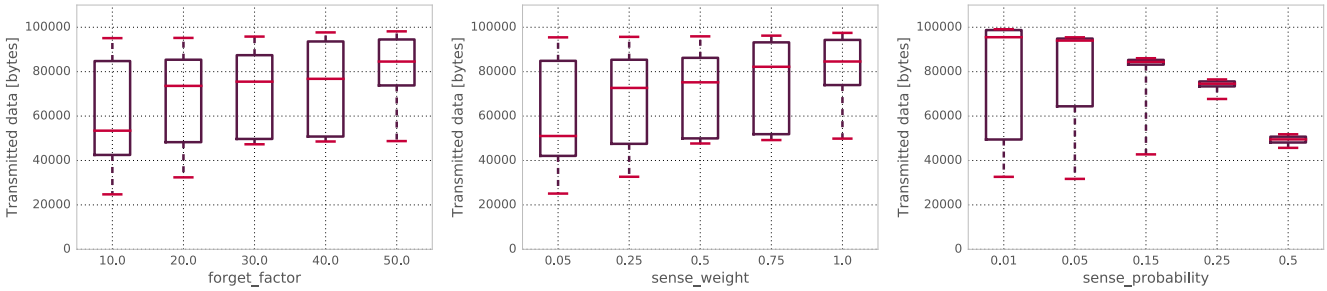
We begin with GA parameters, summarized in Fig. 7. Unsurprisingly, we can see that none of the parameters has a major impact on the total performance. Note that the subfigures are zoomed-in to a small range of throughput values to make the slight differences visible. Setting the crossover probability (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 only visible variation is exhibited by the mutation probability parameter, differing by about 1,000 *bytes* in successfully transmitted data between a very low and the highest possible input parameter value. The insignificant variation observed is a very remarkable, and indeed 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 inter-quartile range. Similar results apply to



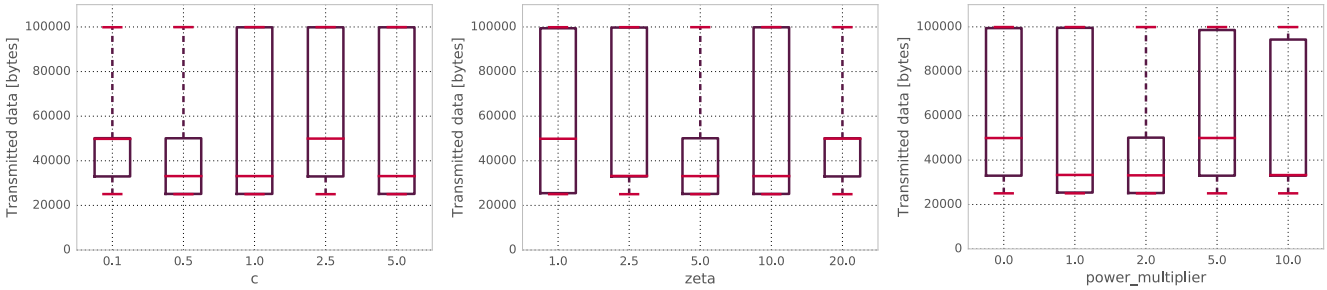
(a) (b) (c)

Fig. 7. GA nodes: performance associated to different values of (a) crossover probability, (b) mutation probability, and (c) sense fitness step. The box includes the upper and lower quartiles, with the red line marking the average. The whiskers correspond to the 10th and 90th percentiles.



(a) (b) (c)

Fig. 8. RL nodes: performance associated to different values of (a) forget factor, (b) sense weight, and (c) sense probability. The box includes the upper and lower quartiles, with the red line marking the average. The whiskers correspond to the 10th and 90th percentiles.



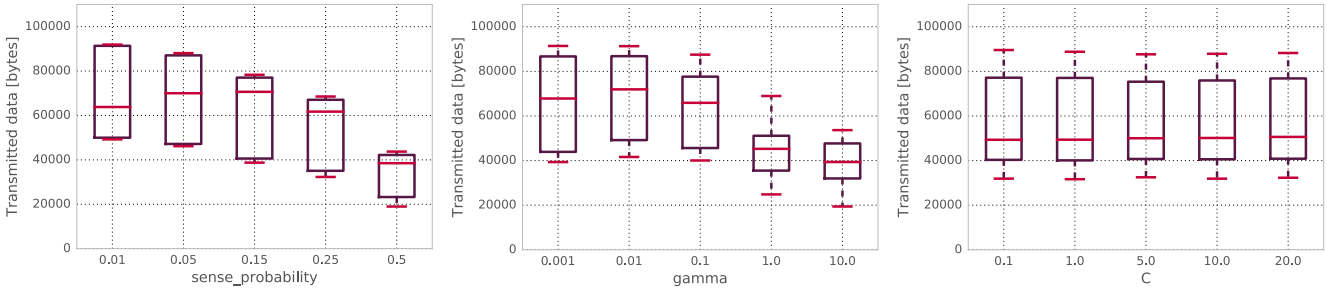
(a) (b) (c)

Fig. 9. MAB nodes: performance associated to different values of the exploration coefficients (a) c and (b) ζ, and (c) power multiplier. The box includes the upper and lower quartiles, with the red line marking the average. The whiskers correspond to the 10th and 90th percentiles.

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 ζ (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 γ 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 γ suggest that this parameter



(a) (b) (c)

Fig. 10. SVM nodes: performance associated to different values of (a) sense probability, (b) γ, and (c) C. The box includes the upper and lower quartiles, with the red line marking the average. The whiskers correspond to the 10th and 90th percentiles.

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, γ 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 [24] 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 $Y = \{y_0, y_1, ..., y_n\}$ contains the performance results for all n topologies and $P = \{p_0, p_1, ..., p_k\}$, where k is the number of different values per parameter. Furthermore, A denotes the aggregate measure selected for the calculation:

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

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

In Tab. IV the parameters are ranked according to the sensitivity of each algorithm. The results suggest that especially the parameter space for the Reinforcement Learning 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).

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.

TABLE IV. SENSITIVITY RANKING ACCORDING TO THE FACTORS $S_P(m)$ AND $S_P(IQR)$ BASED ON MEDIAN AND INTERQUARTILE RANGE. COLUMNS R_m AND R_{IQR} GIVE THE RANK ACCORDING TO $S_P(m)$ OR $S_P(IQR)$, RESPECTIVELY.

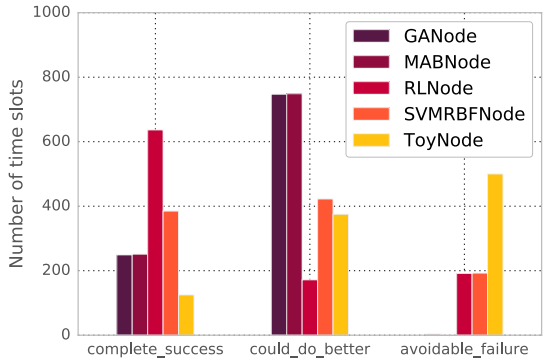
R_m	R_{IQR}	Parameter Name	Class	$S_P(m)$	$S_p(IQR)$
1	1	Sense_probability	RL	0.09089	0.20151
2	2	Forget_factor	RL	0.05747	0.12652
3	12	Power_multiplier	MAB	0.04194	0
4	10	ζ	MAB	0.02667	0.00004
5	4	Sense_probability	SVM	0.01703	0.01480
6	3	Sense_weight	RL	0.01333	0.02514
7	11	c	MAB	0.01036	0.00002
8	6	γ	SVM	0.00525	0.00836
9	9	c	SVM	0.00137	0.00240
10	7	Mutation_probability	GA	0.00038	0.00602
11	5	Crossover_probability	GA	0.00013	0.01007
12	8	Sense_fitness_step	GA	0.00009	0.00538

D. Quality of Decisions

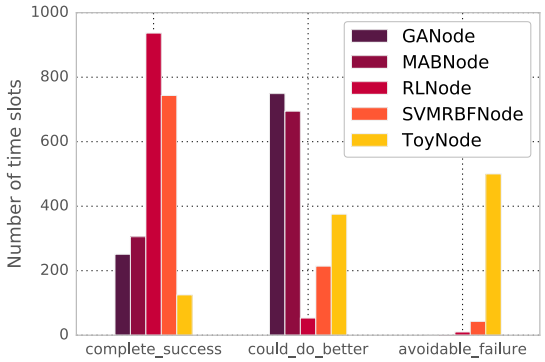
Cognitive radio algorithms make decisions. So far, our focus has been on the overall effectiveness of these decisions, i.e., on network performance. In the following, we give a closer look to how good each algorithm is at making the decisions discussed in Sec. II.

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

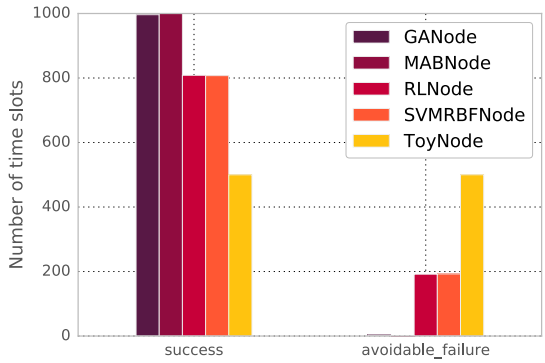


(a)

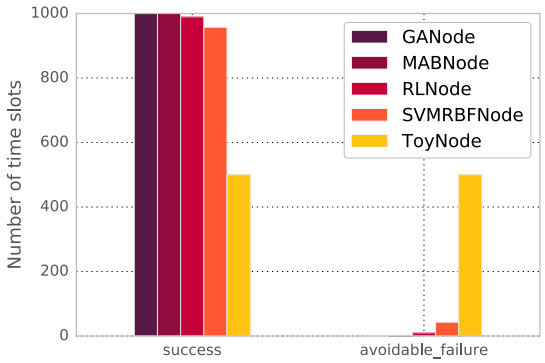


(b)

Fig. 11. Choice of the MCS: number of time slots used by entirely successful transmissions, transmissions that could have used a higher performing MCS, and failures hat could have been avoided by selecting a different MCS, (a) for all firmwares and (b) for the best five firmwares of each class.



(a)



(b)

Fig. 12. Choice of the channel: number of time slots used by successful transmissions and failures that could have been avoided by selecting a different channel, (a) for all firmwares and (b) for the best five firmwares of each class.

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 is essentially a firmware that makes 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.

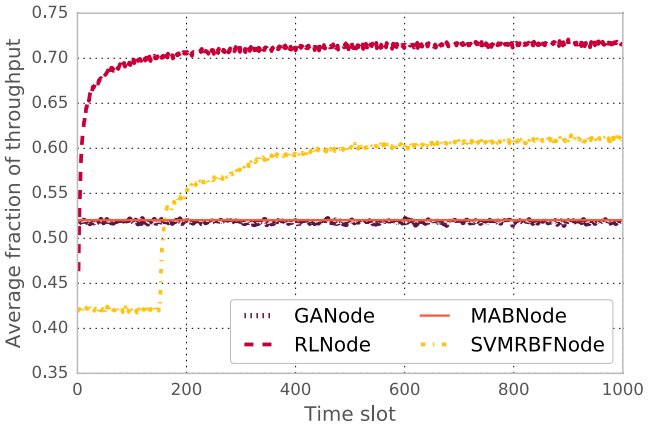


Fig. 13. Evolution of average throughput per transmission over the simulation time.

F. Channel Model with Primary Users Switching Patterns

To evaluate how well the algorithms cope with changing communication patterns of primary users, we also run the simulation experiment containing 3,000 topologies with the dynamic variant of the channel utilization model detailed in Sec. III-A. Tab. V summarizes the differences in the performance results, between the basic channel model ($q_s = 0$) and the dynamic channel model ($0 \leq q_s \leq 1$) for the best performing firmwares of each algorithm, by giving the quartile values for both versions. In the dynamic model, a value for the switching probability q_s is drawn for each primary user from a uniform distribution at the beginning of a simulation run. The results reveal that the performance differences are not significant in our simulation setting for GA, RL, and MAB, as they lie within 0.1%. However, for SVM we can observe a difference of 4.2% on average, which suggests a slight impact of the time required to update the training set.

TABLE V. 1ST, 2ND, AND 3RD QUARTILE FOR THE PERFORMANCE RESULTS OF THE BEST FIRMWARES FOR TWO VERSIONS OF THE CHANNEL UTILIZATION MODEL: (I) SWITCHING PROBABILITY q_s DRAWN FROM A UNIFORM DISTRIBUTION AND (II) $q_s = 0$.

	GA		RL	
	$q_s = U(0,1)$	$q_s = 0$	$q_s = U(0,1)$	$q_s = 0$
Q1	51414.8	51434.0	93807.5	93900.0
Q2	52001.0	52027.0	94550.0	94599.0
Q3	52632.8	52662.0	95100.0	95199.0
	MAB		SVM	
	$q_s = U(0,1)$	$q_s = 0$	$q_s = U(0,1)$	$q_s = 0$
Q1	32992.0	32992.0	85319.3	88494.0
Q2	49958.0	49958.0	87053.0	91386.0
Q3	99858.0	99858.0	87992.0	91837.0

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 [25] provides a report card-like system which allows some comparison between DSA techniques. Dietrich [26] constructs a cognitive model by probing a DSA CR with a psychometric approach. Thompson [27] eschews detailed modeling of internal operation in favor of decoding the operation purely from observation of DSA CR behavior. Giorgetti [28] 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 [29]–[32] 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 [33], [34], nearest neighbor clustering analysis [35], Bayesian reasoning approaches [36], and kernel based statistic analysis [17]. 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 [37]. While variations on this theme certainly exist, such as q-learning [38] or temporal-difference learning [39], the base approach remains the same. The authors of [5] provide a thorough examination of learning techniques for CR, with an emphasis on reinforcement learning. Herein, we examine the fundamentals of this category directly with our reinforcement learning 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 [40], [41] or obscured examinations of the same in the form of partially observable Markov decision processes [42], [43]. Alternatively the multi-armed bandit approach provides a slightly different formulation of the problem that simply examines interacting with a Markovian system. This approach has gained particular popularity in the context of the DSA problem [14], [44], [45]. Herein we examine the MAB approach within this category.

Evolutionary algorithms address the DSA problem in a manner inspired by the biological process of evolution and provide our final category. Algorithms within this category model various aspects of the evolutionary process to find opportunities for spectrum utilization. While several variations of this theme exist, genetic algorithms are by far the most well known and widely applied [10]–[13], [46]. The features of fast convergence (for properly bounded situations) and broad applicability to multi-objective problems allow genetic algorithms to dominate this category of techniques for the DSA problem. Naturally, we examine a basic GA within this category.

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 [47], [48] or signal processing schemes [49] 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 Gadhiok [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

[1] M. Gadhiok, A. Amanna, M. Price, and J. Reed, "Metacognition: Enhancing the performance of a cognitive radio," in *IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, 2011, pp. 198–203.

[2] H. Asadi, H. Volos, M. Marefat, and T. Bose, "Metacognitive radio engine design and standardization," *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 4, pp. 711–724, 2015.

[3] P. Sutton, K. Nolan, and L. Doyle, "Cyclostationary signatures in practical cognitive radio applications," *IEEE Journal on Selected Areas in Communications*, vol. 26, 2008.

[4] S. Haykin, "Fundamental issues in cognitive radio," in *Cognitive Wireless Communication Networks*, E. Hossain and V. Bhargava, Eds. Springer US, 2007.

[5] L. Gavrilovska, V. Atanasovski, I. Macaluso, and L. A. DaSilva, "Learning and reasoning in cognitive radio networks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1761–1777, 2013.

[6] T. Jiang, D. Grace, and Y. Liu, "Performance of cognitive radio reinforcement spectrum sharing using different weighting factors," in *Int. Conf. on Communications and Networking in China (ChinaCom)*, 2008.

[7] H. Li, "Multi-agent Q-learning of channel selection in multi-user cognitive radio systems: A two by two case," in *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2009.

[8] C. Wu, K. Chowdhury, M. Di Felice, and W. Meleis, "Spectrum management of cognitive radio using multi-agent reinforcement learning," in *Int. Conf. on Autonomous Agents and Multiagent Systems*, 2010.

[9] Y. Wu, F. Hu, S. Kumar, Y. Zhu, A. Talari, N. Rahnavard, and J. Matyjas, "A Learning-Based QoE-Driven Spectrum Handoff Scheme for Multimedia Transmissions over Cognitive Radio Networks," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 11, pp. 2134–2148, 2014.

[10] T. R. Newman, B. A. Barker, A. M. Wyglinski, A. Agah, J. B. Evans, and G. J. Minden, "Cognitive engine implementation for wireless multicarrier transceivers," *Wirel. Commun. Mob. Comput.*, 2007.

[11] T. Rondeau and C. Bostian, *Artificial Intelligence in Wireless Communications*. Artech House, 2009.

[12] A. Young, N. Kaminski, A. Fayez, and C. Bostian, "Csere (cognitive system enabling radio evolution): A modular and user-friendly cognitive engine," in *DYSPAN*, 2012.

[13] M. Yousefvand, N. Ansari, and S. Khorsandi, "Maximizing Network Capacity of Cognitive Radio Networks by Capacity-Aware Spectrum Allocation," *IEEE Transactions on Wireless Communications*, vol. 14, no. 9, pp. 5058–5067, 2015.

[14] W. Jouini, D. Ernst, C. Moy, and J. Palicot, "Multi-armed Bandit Based Policies for Cognitive Radio's Decision Making Issues," in *International Conference on Signals, Circuits and Systems (SCS)*, 2009.

[15] J.-Y. Audibert, R. Munos, and C. Szepesvári, "Tuning Bandit Algorithms in Stochastic Environments," in *International Conference on Algorithmic Learning Theory (ALT)*, 2007, pp. 150–165.

[16] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.

[17] H. Hu, Y. Wang, and J. Song, "Signal Classification Based on Spectral Correlation Analysis and SVM in Cognitive Radio," in *International Conference on Advanced Information Networking and Applications (AINA)*, 2008, pp. 883–887.

[18] Y. Yang, Y. Liu, Q. Zhang, and L. Ni, "Cooperative Boundary Detection for Spectrum Sensing Using Dedicated Wireless Sensor Networks," in *IEEE INFOCOM*, March 2010, pp. 1–9.

[19] U. Berthold, F. Fu, M. van der Schaar, and F. Jondral, "Detection of spectral resources in cognitive radios using reinforcement learning," in *DySPAN*, 2008.

[20] Z. Han, C. Pandana, and K. Liu, "Distributive opportunistic spectrum access for cognitive radio using correlated equilibrium and no-regret learning," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2007.

[21] Y. Gai, B. Krishnamachari, and R. Jain, "Learning multiuser channel allocations in cognitive radio networks: A combinatorial multi-armed bandit formulation," in *IEEE Symposium on New Frontiers in Dynamic Spectrum*, 2010.

[22] "RSPG Opinion on Licensed Shared Access," Radio Spectrum Policy Group of European Commission, Tech. Rep., 2013.

[23] J. Holdren and E. Lander, "Report to the president realizing the full potential of government-help spectrum to spur economic growth," President's Council of Advisors on Science and Technology, Tech. Rep., 2012.

[24] D. M. Hamby, "A Comparison of Sensitivity Analysis Techniques," *Health Physics*, vol. 68, no. 2, pp. 195–204, 1995.

[25] Y. Zhao, S. Mao, J. O. Neel, and J. Reed, "Performance evaluation of cognitive radios: Metrics, utility functions, and methodology," *Proceedings of the IEEE*, vol. 97, no. 4, 2009.

[26] C. Dietrich, E. Wolfe, and G. Vanhoy, "Cogntive radio testing using psychometric approaches: Applicability and proof of concept study," *Analog Integrated Circuits and Signal Processing*, vol. 73, no. 2, pp. 627–636, 2012.

[27] J. Thompson, K. Hopkinson, and M. Silvius, "A test methodology for evaluating cognitive radio systems," *IEEE Transactions on Wireless Communications*, pp. 1–1, 2015.

[28] A. Giorgetti, M. Varrella, and M. Chiani, "Analysis and performance comparison of different cognitive radio algorithms," in *Int. Workshop on Cognitive Radio and Advanced Spectrum Management*, 2009.

[29] A. De Domenico, E. C. Strinati, and M. G. Di Benedetto, "A survey on MAC strategies for cognitive radio networks," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 21–44, 2012.

[30] Q. Zhao and B. Sadler, "A Survey of Dynamic Spectrum Access," *IEEE Signal Processing Magazine*, vol. 24, no. 3, 2007.

[31] P. Marshall, *Quantitative Analysis of Cognitive Radio and Network Performance*. Norwood, MA: Artech House, 2010.

[32] B. Wang and K. J. R. Liu, "Advances in cognitive radio networks: A survey," *IEEE J. Sel. Top. Signal Process.*, vol. 5, no. 1, pp. 5–23, 2011.

[33] V. Gatla, M. Venkatesan, and A. V. Kulkarni, "Feed forward neural network based learning scheme for cognitive radio systems," in *Computational Intelligence and Information Technology (CIIT)*, 2013.

[34] S. Haykin, "Cognitive radio: brain-empowered wireless communications," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 2, pp. 201–220, 2005.

[35] K. M. Thilina, K. W. Choi, N. Saquib, and E. Hossain, "Pattern classification techniques for cooperative spectrum sensing in cognitive radio networks: SVM and W-KNN approaches," in *IEEE Global Communications Conference (GLOBECOM)*, 2012, pp. 1260–1265.

[36] S. Zheng, P. Y. Kam, Y. C. Liang, and Y. Zeng, "Spectrum sensing for digital primary signals in cognitive radio: A bayesian approach for maximizing spectrum utilization," *IEEE Transactions on Wireless Communications*, vol. 12, no. 4, pp. 1774–1782, 2013.

[37] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[38] F. H. Panahi and T. Ohtsuki, "Optimal channel-sensing policy based on fuzzy q-learning process over cognitive radio systems," in *IEEE International Conference on Communications (ICC)*, 2013.

[39] B. F. Lo and I. F. Akyildiz, "Reinforcement learning for cooperative sensing gain in cognitive radio ad hoc networks," *Wireless Netw*, vol. 19, no. 6, pp. 1237–1250, 2012.

[40] Y. Zhang, "Dynamic Spectrum Access in Cognitive Radio Wireless Networks," in *IEEE Int. Conf. on Communications (ICC)*, 2008.

[41] A. Sultan, "Sensing and transmit energy optimization for an energy harvesting cognitive radio," *IEEE Wireless Communications Letters*, vol. 1, no. 5, pp. 500–503, 2012.

[42] J. Pajarinen, A. Hottinen, and J. Peltonen, "Optimizing spatial and temporal reuse in wireless networks by decentralized partially observable markov decision processes," *IEEE Transactions on Mobile Computing*, vol. 13, no. 4, pp. 866–879, 2014.

[43] A. Raschella, J. Perez-Romero, O. Sallent, and A. Umbert, "On the Use of POMDP for Spectrum Selection in Cognitive Radio Networks," in *Int. Conf. on Cognitive Radio Oriented Wireless Networks*, 2013.

[44] Q. Zhao, B. Krishnamachari, and K. Liu, "On Myopic Sensing for Multi-channel Opportunistic Access: Structure, Optimality, and Performance," *Wireless Communications, IEEE Transactions on*, vol. 7, no. 12, pp. 5431–5440, 2008.

[45] Y. Gai, B. Krishnamachari, and M. Liu, "On the Combinatorial Multi-Armed Bandit Problem with Markovian Rewards," in *Global Telecommunications Conference (GLOBECOM)*, 2011.

[46] Z. Zhang, K. Long, J. Wang, and F. Dressler, "On swarm intelligence inspired self-organized networking: Its bionic mechanisms, designing principles and optimization approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 513–537, 2014.

[47] B. Wang, Y. Wu, and K. R. Liu, "Game theory for cognitive radio networks: An overview," *Computer Networks*, vol. 54, no. 14, pp. 2537–2561, 2010.

[48] J. Wang, G. Scutari, and D. P. Palomar, "Robust mimo cognitive radio via game theory," *IEEE Transactions on Signal Processing*, vol. 59, no. 3, pp. 1183–1201, 2011.

[49] L.-L. Yang and L.-C. Wang, "Zero-forcing and minimum mean-square error multiuser detection in generalized multicarrier DS-CDMA systems for cognitive radio," *EURASIP Journal on Wireless Communications and Networking*, vol. 2008, pp. 1–13, 2008.

[50] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

APPENDIX A
SVM DECISION TIME AND CLASSIFICATION ERROR

Tab. VI summarizes the average execution times for different configurations of SVM kernels. The linear kernel is controlled over the parameter c, the RBF kernel over c and γ. While the RBF kernel makes a decision in less than 0.01 seconds

regardless of the parameter value, the linear kernel shows significant variations ranging from 3.6 s to 44.4 s for the c 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 [50].

TABLE VI. AVERAGE EXECUTION TIMES FOR DIFFERENT PARAMETER SETTINGS OF SVM WITH LINEAR AND RBF KERNEL.

		Linear kernel	RBF kernel
c	γ	execution time [s]	execution time [s]
0.1	0.001	3.6	<0.01
0.1	0.01		<0.01
0.1	0.1		<0.01
0.1	1		<0.01
0.1	10		<0.01
1	0.001	7.2	<0.01
1	0.01		<0.01
1	0.1		<0.01
1	1		<0.01
1	10		<0.01
5	0.001	36.4	<0.01
5	0.01		<0.01
5	0.1		<0.01
5	1		<0.01
5	10		<0.01
10	0.001	44.4	<0.01
10	0.01		<0.01
10	0.1		<0.01
10	1		<0.01
10	10		<0.01
20	0.001	35.4	<0.01
20	0.01		<0.01
20	0.1		<0.01
20	1		<0.01
20	10		<0.01

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

TABLE VII. CLASSIFICATION ERROR FOR DIFFERENT PARAMETER SETTINGS OF SVM WITH RBF KERNEL.

c	γ	RMSE	MAE
0.1	0.001	11.008	4.018
0.1	0.01	13.666	5.391
0.1	0.1	13.910	5.506
0.1	1	13.907	5.520
0.1	10	13.907	5.521
1	0.001	9.365	2.161
1	0.01	11.413	4.801
1	0.1	13.585	5.842
1	1	13.571	5.980
1	10	13.571	5.986
5	0.001	9.037	2.199
5	0.01	10.128	4.375
5	0.1	12.879	7.046
5	1	12.979	7.759
5	10	12.986	7.791
10	0.001	8.898	2.330
10	0.01	10.079	4.7394
10	0.1	12.858	7.973
10	1	13.442	9.527
10	10	13.473	9.591
20	0.001	8.818	2.479
20	0.01	10.159	5.060
20	0.1	12.937	8.698
20	1	14.243	11.129
20	10	14.322	11.259