View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Ghent University Academic Bibliography

An LSPI based reinforcement learning approach to enable network cooperation in cognitive wireless sensor networks

Milos Rovcanin, Eli De Poorter, Ingrid Moerman and Piet Demeester
Ghent University - IBBT, Department of Information Technology (INTEC)
Gaston Crommenlaan 8, Bus 201, 9050 Ghent, Belgium
Email: milos.rovcanin@intec.ugent.be, {*firstname.lastname*}@intec.ugent.be

***Abstract*—The number of wirelessly communicating devices increases every day, along with the number of communication standards and technologies that they use to exchange data. A relatively new form of research is trying to find a way to make all these co-located devices not only capable of detecting each other's presence, but to go one step further - to make them cooperate. One recently proposed way to tackle this problem is to engage into cooperation by activating 'network services' (such as internet sharing, interference avoidance, etc.) that offer benefits for other co-located networks. This approach reduces the problem to the following research topic: how to determine which network services would be beneficial for all the cooperating networks. In this paper we analyze and propose a conceptual solution for this problem using the reinforcement learning technique known as the Least Square Policy Iteration (LSPI). The proposes solution uses a self-learning entity that negotiates between different independent and co-located networks. First, the reasoning entity uses self-learning techniques to determine which service configuration should be used to optimize the network performance of each single network. Afterwards, this performance is used as a reference point and LSPI is used to deduce if cooperating with other co-located networks can lead to even further performance improvements.**

***Index Terms*—Symbiotic networks, network optimization, self-learning, reinforcement learning, service negotiation, incentive-driven networking, cognitive negotiation engine, LSPI;**

I. Introduction

The fact that wireless networks are becoming increasingly complex, heterogeneous and dynamic is the main motivation for the evolution of the cognitive networking concept. In a cognitive network, devices are capable of observing their environment, can change their own settings to improve their network performance, and can learn about the consequences of their actions to improve their decision making in the future [7]. Decision making entities in such a network are capable to plan actions according to the observed data and take appropriate steps towards their execution. Gathering feedback upon completion of all the planned tasks helps evaluate the effects of the above taken decisions and improves the decision making policy (see Fig. 1).

By introducing these concepts to the configuration of the network stack, networks obtain a self-learning awareness regarding the optimal use of their available settings and resources based on observed environmental conditions. As a result, human involvement during the operational lifetime can be reduced to a minimum. Cognitive networking is related to cognitive radio, but instead of optimizing radio settings the main goal is to also reason about higher level networking protocol.

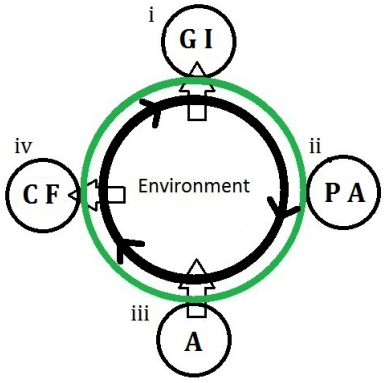

Fig. 1. General concept of a cognitive cycle: 1 - gathering necessary information; 2 - planning actions; 3 - taking actions; 4 - collecting feedback for evaluation

The structure of this paper is organized as follows. The related work (Section II) gives a broad overview of the related work in terms of cognitive radios, cognitive networking and self-learning networks. Afterwards, Section III describes the use case this work focuses on: the SymbioNets project. Section IV describes mathematical fundamentals of reinforcement learning in general and also describes the LSPI algorithm in more details. Section V gives detailed implementation guidelines, along with an example of how to apply LSPI to the SymbioNets negotiation engine. General issues and future improvements are discussed in Section VI. Finally, Section VII concludes the paper.

II. Related work

Recently, a number of research areas have applied self-learning or cognitive reasoning methods to further optimize the configuration of wireless sensor networks. The most notable results were achieved with the following ones.

A. Software defined radios / cognitive radios

A software defined radio (SDR) is a radio that can (at runtime) change it transmission or receiving settings (frequency,

modulation, etc) to emulate different types of radio technologies. The term 'cognitive radio' is used to refer to solutions where decision software is added to intelligently and dynamically configure a software defined radio [1]. Cognitive radio solutions are typically used to realize one of the following use cases:

- dynamically switching to the best available radio technology ('always best connected paradigm')
- opportunistic reuse of unused (licensed) spectrum

In addition, cognitive radios are also used to allow multiple (non-licensed) communication technologies to share the same frequency band. For example, the 2.4 GHz ISM band contains (amongst others) Wi-Fi, Bluetooth and ZigBee technologies. As a result, these wireless networks interfere with each other, resulting in a degraded performance for all co-located networks [2]. By changing the channel activity sensitivity of a software defined radio, Wi-Fi devices can be configured to also detect transmitting ZigBee devices, thus resulting in less packet collisions.

When considering current cognitive radio solutions, a number of limitations are typically encountered. (i) Cognitive radio approaches are often limited to adapting the settings of a single radio device (terminal-centric approaches) rather than aiming for a network wide solution [3] or optimizing the performance of a single network layer from a single network (intra-network approaches) [4]. (ii) In addition, most existing cognitive solutions focus on optimization of link-layer performance, without considering the performance of higher (end-to-end) networking protocols. To reach a network performance that is truly optimal, the current state of the art research needs to be extended so that cross-layer, cross-network and cross-technology interactions are also taken into account [5], [6]. In this paper, we will go beyond the existing state-of-the-art by showing that our solutions can take into account these high-level end-to-end interactions.

B. Self-learning

To reach this goal, the solutions proposed in this paper utilize self-learning techniques to optimize the overall network performance over multiple network layers. The concept of self-learning relies on node capability to perceive the current network conditions and predict at run-time a best course of action based on the collected data. Difference paradigms are used to enable this. Self-learning techniques are typically used in situations where it is impossible to calculate or predict a 'best' solution in advance. The optimization of interacting wireless networks are a good example: not only are the characteristics of the wireless environment notoriously difficult to predict and model, the influence of having multiple co-located interacting networks are almost impossible to predict. Different approaches towards self-learning exist.

Game theory describes network behavior in the form of a game where nodes are the players. It is a game of balancing costs and benefits of an each move, while taking into account moves of all the other players. Game theory provides tools for determining the existence of the steady state points (Nash equilibrium) and their efficiency from the global point of view (Pareto optimum). Power consumption, communication medium contention and routing are the problems that were most usually being solved using this concept [8].

Machine learning is a form of artificial intelligence. Decisions are modeled as actions that transfer the network to a new state. By adding actual values to every state/action pair, it is possible to make a distinction between "good" and "bad" choices and to choose the most optimal decisions in every given situation [9]. Techniques that made most notable results are Q-learning, LSPI and Collaborative Reinforcement learning [10].

Each of these mechanisms have their advantages and disadvantages and depending on the problem that is being solved, some will give better practical results than others. A comparison of different reasoning techniques applied in wireless sensor networks is given in [11].

III. The SymbioNets use case

As shown in the related work section, very different approaches are possible in the field of self-learning networks. This paper will focus on realizing the negotiation engine described in the 'SymbioNets project' [12] using self-learning techniques. The SymbioNets project starts out from the following observations. Many optimization techniques (referred to as 'networking services') influence multiple independent and co-located networks. For example, activating interference avoidance in one network can increase the reliability of both the activating network and other co-located networks. Other examples of network services are 'shared routing' (networks can route each other's packets) or the sharing of internet connectivity. To this end, the SymbioNets proposed a novel networking paradigm in which advanced cooperation is possible between otherwise independent networks (referred to as 'communities'). This cooperation can take many forms:

- The sharing of information, such as environment information or spectrum information;
- The sharing of infrastructure such as processing capacity or the sharing of each other nodes for routing purposes;
- The sharing of (networking) services, can be offered to each other, such as positioning, synchronization, address translation, QoS functions, code updates, security provisions or internet connectivity.

Each of the above optimization options is implemented as a network service that can be activated or de-activated at run-time in each of the cooperating networks. To enable intelligent optimization over multiple networks, each network provides its incentives ('high-level network goals') for cooperation. Figure III depicts a simple use case where two independent co-located sensor networks influence each other (since they share the same spectrum), even though they are used for for different purposes (e.g. controlling temperature in a building and security).

Incentives, as mentioned above, are high level network goals (high throughput, high reliability, low delay, high coverage etc.). Since the two networks from Figure III have different

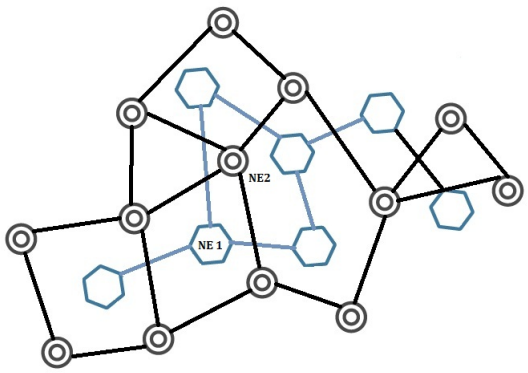

Fig. 2. Co-existing networks that use the same communication technologies often influence each other. An example would be: temperature monitoring network (black nodes) and a security network (blue hexagons) inside a facility. To enable cooperation between the networks, both networks have a negotiation entities (NE).

purposes, at least some of their incentives will be different. Each network has a special node (a 'negotiation entity') that negotiates with other networks on behalf of its network. The negotiation engine can either be one of the network nodes, or can exist as a dedicated trusted (third-party) server outside the network. Negotiation entities should know or at least make an accurate enough assumption about the impact that each symbiotic service has on the incentives in each of the participating sub-networks. For example, using channel selection algorithms in one network might improve the other network's *reliability* incentive by 60 percent. Of course, activation of this particular service will also (positively or negatively) affect the incentives of the network that provides it. The negotiation engine should balance between the benefits other communities will have upon activating a certain service and the costs that might occur inside the community that activates it. This sort of information is crucial for the negotiation process and forming of symbiotic network. It can be obtained in several ways:(i) from the existing literature, from (ii) network simulators or from (iii) network monitoring agents.

The next section investigates how cognitive optimization methods can be used to select an optimal set of network services in each device. The paper will describe how LSPI [13] can be applied inside the negotiation entity in order to enable it to gather knowledge about the influence of each combination of services (while taking into account the incentives of each community). Further more, the algorithm provides criteria to determine what is the optimal service set in the given circumstances. This information is crucial for the community to be able to decide whether or not to engage into cooperation with other communities.

IV. Least Square Policy Iteration - LSPI

Next section will describe the concept of LSPI, which is a form of *reinforcement learning* [14].

A. Reinforcement learning concept

Reinforcement learning is a branch of artificial intelligence that enables learning agents (devices) to define and improve decision making rules (policies) based on experience and rewards they receive after each taken decision. While learning, agents go through a certain number of states $(S = s_1, s_2, ..., s_m)$. There is an available set of actions $(A = a_1, a_2, ...a_n)$ that can be chosen from at each state. Taking an action will result in transition to a different state with a given probability. The expression $P(s'|s,a)$ represents the conditional probability of entering a state s' after the action a is taken in the state s. After each transition, an immediate reward is given $R(s,a,s')$. The reward is either pre-defined by a user or it is calculated as a function of a certain network properties (such as remaining battery level of an agent).

The main goal of a decision making agent is to maximize what is called a *state-action function*, also referred to as Q-function. Its formal outlook is know as the Bellman's equation:

$$Q(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) maxQ(s',a') \quad (1)$$

The first term on the right side represents the expected immediate reward of executing action a at state s. It is calculated as $r(s,a) = \sum P(s'|s,a)R(s,a,s')$, while the second term is the maximum expected future reward. Factor γ is known as the *discount* factor and its purpose is to model the fact that a immediate reward is more valuable than a future reward.

The main concept is clear: an agent updates Q-values of each state/action pair once it switches from state s to s', upon utilizing action a. At each state it uses the same criteria to choose the best possible action - it picks up the one which has the highest Q-value. That way, certain decisions will be enforced. Of course, in the initial phase, right after deployment, all Q values will be the equal. In order to prevent enforcing certain decision making patterns that might not be optimal, an ε greedy algorithm is used. It simply states that in each state, agent will pick a random action with ε probability. On the other hand, with 1 - ε probability it will pick an action with the highest Q-value.

B. LSPI fundamentals

LSPI approach approximates Q-values with a linear parametric combination of k *basis functions*, also referred to as *features*:

$$Q(s,a;w) = \sum_k \phi_j(s,a)\omega_j \quad (2)$$

Argument ω_j is the weight parameter. Basis functions are arbitrary and generally non-linear functions of s and a. A few good examples are: the residual energy of a node, the number of routing paths through a node, hop distance from a sink etc. The set of basis functions strongly depend on the network objectives. In any case, it is important to make them linearly independent to ensure that there are no redundant parameters. Generally, the number of basis functions is significantly smaller than the number of state/action pairs.

This is what Bellman equation transforms into after we combine (1) and (2):

$$\Phi\omega = R + \gamma P^{\pi}\Phi\omega \quad (3)$$

Φ is the matrix that represents the set of k basis functions for each state/action pair. Under the assumption that the columns of Φ are linearly independent, the equation transforms into the following equation:

$$\Phi^T(\Phi - \gamma P^{\pi}\Phi)\omega^{\pi} = \Phi^T R \quad (4)$$

The weights ω of the linear functions $Q(s,a)$ can be extracted by solving a linear system:

$$\begin{gathered}\omega = A^{-1}b \\ \text{where: } A = \Phi^T(\Phi - \gamma P^{\pi}\Phi) \\ b = \Phi^T R \\ (5)\end{gathered}$$

Both A and b matrices are populated by collecting samples (s, a, s', r) from the environment. The parameters s, a, s' and r are the current state, action, new state and immediate reward, respectively. Using L number of samples, we can construct an approximate version of $\hat{\Phi}$, $\hat{P^{\pi}}$ and $\hat{R}$ as follows:

$$\hat{\Phi} = \begin{pmatrix} \phi(s_1,a_1)^T \\ \dots \\ \phi(s_n,a_n)^T \end{pmatrix} \quad \widehat{P^{\pi}\Phi} = \begin{pmatrix} \phi(s'_1,\pi(s'))^T \\ \dots \\ \phi(s'_n,\pi(s'))^T \end{pmatrix}$$

$$\hat{R} = \begin{pmatrix} r_1 \\ \dots \\ \mathrm{r}_2 \end{pmatrix}$$

Dimensions of the Φ,P^{π} and R matrices are $(|S||A| \times k)$, $|S||A| \times |S||A|$ and $|S||A| \times 1$, respectively.

An important reason to use LSPI, rather than other reinforcement learning approaches, lies in a fact that it converges faster than all other known algorithms [13], since the samples are used more efficiently. In addition LSPI does not require fine tuning of the initial parameters such as *learning rate*. Finally, LSPI learns the weights of the linear functions and updates Q-values based on the most updated information regarding the features, while in other approaches agents make decisions directly based on Q-values, which may be outdated, depending on the network dynamics.

V. LSPI AS A PART OF THE NEGOTIATION ENTITY

Before a network can decide whether or not cooperation with a co-located network will be beneficial, it needs to be aware of the effects its decisions will have on its own performance. As such, the first step in the process of creating a strong and efficient symbiotic cooperation is for the network to become aware of its own optimal settings and performance. To this end, each network contains a single centralized negotiation node (Fig. III) that collects the network performance information and executes the self-learning LSPI solution to determine the influence of each possible network state.

First, a methodology is given that describes how LSPI can be applied to realize the negotiation engine. Afterwards, the methodology is illustrated with an example realization in Section V-C.

A. Defining states

Since every community is capable of providing a predefined number of services, a combination of active/inactive services at any particular moment represent the state the whole community is in (Fig. 3). The number of possible states can be calculated as $N_{states} = 2^{N_{serv}}$, where N_{serv} is the cardinal number of the set of services available in a community.

Services	Service 1	Service 2	Overall state
State 1	Active	Active	1 1 (3)
State 2	Active	Inactive	1 0 (2)
State 3	Inactive	Active	0 1 (1)
State 4	Inactive	Inactive	0 0 (0)

Fig. 3. Possible combinations of services (learning states) in the case when network is capable of providing two services, Service 1 and Service 2

The time between two changes of states is called the *episode*. During each episode, nodes in the community gather different statistics regarding network performance. These statistics are utilized in a raw format or as arguments to calculate the metrics used inside the basis functions. Incentives that are active determine what metrics are needed and what properties (features) are used as basis functions. At the end of each episode, basis function values get updated according the data that has been gathered from the network. Using (2), the Q-values of transitions from each state to the current one are updated (Fig. 4).

States	S1	S2	S3	S4
S1	Q(S1,S1)	Q(S1,S2)	**Q(S1,S3)**	Q(S1,S4)
S2	Q(S2,S1)	Q(S2,S2)	**Q(S2,S3)**	Q(S2,S4)
S3	Q(S3,S1)	Q(S3,S2)	**Q(S3,S3)**	Q(S3,S4)
S4	Q(S4,S1)	Q(S4,S2)	**Q(S4.S3)**	Q(S4,S4)

Fig. 4. Possible transitions between states and their Q-values. Values in bold red are updated an episode in which system was in state S3

This is legitimate because the properties of the network that are described by the basis functions depend only on the destination state and not the source state. This allows us to update N_{states} number of Q-values after each episode. This number includes the case when system decides not to change the state, but to stay in the same one. With this updating rule, it will take at least N_{states} number of episodes to update Q-values of all the possible transitions. In order to prevent possibly sub-optimal loops during the initial phase of learning, an ε *greedy* algorithm (described in Section VI) is used.

As described in Formula 2, the Q-value of each transition is a linear combination of basis functions, pondered with weight factors. These basis functions must be designed in

correspondence with incentives that are active in the network. Incentives should be presented by at least one function and each additional basis function will describe one new dimension of the incentive. However, it is desirable to make basis functions independent from one another to avoid redundancy. Weight factors, associated with each function, are used to favorize certain metrics over the others. In different words, some changes of the network properties during a learning episode will have a more significant impact on the reward that has been given after each episode. Initial values of weight factor are defined by user in a pre-deployment phase.

B. Calculating rewards

In our case, rewards are calculated based on how close the actual network performance was to the desired performance during a particular episode (in regards to all the active network incentives). Since incentives are high level network goals, these goals must be determined in a design phase of the network. They represent the desired performance as a combination of certain metrics. As such, at the end of each episode when all the necessary metrics are collected, the system is able to calculate how close to the desired performance the current observations are. According to the distance between observed and desired performance, the appropriate reward is calculated. An example implementation of the above concepts is given in Section V-C.

After the basis functions are updated and rewards calculated, Formula (3) is used to tune up the weight factors. The process stops if, after any of recalculations, the difference between an old and a new value doesn't exceed an ε value, defined by the user.

The metrics used to calculate the rewards have to be collected from the network. The best possible case would be to use the information gathered for the purpose of calculation the basis function values. In certain cases, the same linear combination of basis functions, used to update a Q value, is used to calculate rewards [15]. In the case of LSPI, Q values should not be confused with rewards. Q values are pointers system should follow in order to perform in an optimal way. Rewards, through the process of weight factor recalculation, will help determine the Q value of a state more accurately, since the influence of each feature on it will be precisely determined. Unlike them, rewards are being calculated depending on a predetermined high level goal of the network. An actual realization of this concept, given in the following section, will help clarify this matter.

The total reward R_{total} is calculated as a combination of rewards given per each incentive:

$$R_{total} = \sum_i C_{pi} R_i \quad (6)$$

Here, C_{pi} is the coefficient that defines the priority of an incentive [5]. In cases when the network has more than one high level goals, one of them might be favorized depending on the main purpose of the network. For example, a temperature monitoring network on a remote location might have a *low delay* and *long network lifetime* incentives. In this case, long network lifetime would be more appreciated since it would be to expensive to change the batteries every once in a while. Somewhat higher delay can be tolerated. In cases where changing batteries does not represent such a costly task, priorities of both incentives can be set equal or even in a favor of a low delay.

C. One possible realization

Let's consider a case when there is a network similar to the one described in the previous section. *Low delay* and *long network lifetime* are the main incentives of the network.

Example basis functions used for this case are the *average number of hops per packet* (ϕ_1) and the *average energy spent per node* (ϕ_2). While the first information can be obtained directly from a packet's header (hop counter increased at each retransmission), the second one has to be collected separately from each node. One possible way would be to obtain information about the average time spent in receiving/transmitting (t_{tran}, t_{rec}) mode from every node and calculate the average value per node:

$$\bar{t_{tran}} = \frac{1}{N} \sum_N t_{tran} \; ; \; \bar{t_{rec}} = \frac{1}{N} \sum_N t_{rec} \quad (7)$$

Argument N is the number of nodes in the network. Since the power consumption knowing the power radio transceiver spends in these two modes (stated in a data-sheet) and the average time spent in transmitting and receiving, we can calculate the approximate average energy spent per node in the network. Let p_{trans} and p_{rec} be the transmitting and receiving power of a radio transceiver, respectively. We get:

$$E_{avg} = t_{avg_{rec}} p_{rec} + t_{avg_{tran}} p_{tran} \quad (8)$$

If the information about the current battery level is obtained from each node (a relatively simple task), the average remaining energy per node can be used along with the information about the average energy spent to predict the network life time in terms of energy with the given set of services. The difference between observed value and the desired one will help us calculate the reward for both incentives. Let's say that two incentives have the same priority, thus $R_{total} = C_{1/2}(R_1 + R_2)$. Let E_{goal} be the energy needed for a desired network lifetime and let D_{goal} be the desired delay.

$$R_{lifetime} = \gamma \frac{T_{goal}}{T_{measured}} C_{onst} \quad (9)$$

$$R_{delay} = \gamma \frac{D_{goal}}{D_{measured}} C_{onst} \quad (10)$$

Equations are designed this way so that rewards are inversely proportional to obtained evaluation metrics. The reward for the lifetime incentive is inversely proportional to the energy spent in the network during one episode. In the other case, reward will be inversely proportional to a delay that is measured during an episode.

After a certain amount of data is gathered, weight factors (ω_j) are recalculated by solving the system of equations given in Section IV. Next, the matrices Φ and R are populated as described in Section IV. There is still one unknown factor left:

the *transition probability matrix*. In our case, $P(s'|s,a)$ can either be 1 or 0. In other words, if the system is in state (0,0), by taking the action "set Service 1 to active", the system will transfer to state (0,1) with probability 1. Probability of passing to any other state is 0. The same rule is applied to every other combination of states and actions in the same manner.

If some optimizations are mutually exclusive (for example, channel hopping and network merging with a non-hopping network) these states will be forbidden. This can easily be implemented since network managers can disallow certain services by setting the transition probability to 0.

VI. Implementation guidelines and future work

This section points-out and discusses some additional implementation challenges. In addition, a number of general guidelines of how to overcome them are given (although the optimal approach might be different from use case to use case). Afterwards, possible extensions to this work towards its utilization in the service negotiation phase of creating a symbiotic network are discussed.

A. Implementation guidelines

When introducing LSPI in as described in the previous sections, two main challenges are immediately evident:

1) How to avoid large overhead when collecting statistics (basic functions) from the network
2) What is the optimal period between changing from one state to another (learning episode)

There is no universal and definite answer to either of those questions. Both issues are case specific and the best solution to any of them will depend on both the capabilities of the network and the metrics that are needed to calculate values of the basis functions and rewards.

In the case described in Section V-C, information regarding the hop count per each packet can be transmitted towards the sink as a part of an original data packet. No additional transmissions are required. However, to obtain data regarding the energy spent at every node during the episode, additional messages will be sent in both directions. Negotiation entity will have to broadcast a request for measurements at the end of episodes and all the nodes will have to send them as a reply. One less intrusive solution in sensor networks used to gather data (where each packet is sent to the central sink) is to use the already gathered information about the hop count to calculate the necessary metrics. For example, every hop of a packet can be modeled as one transmission and one reception. By combining this knowledge with the size of the packets used in a network and the bit rate of a radio interface, it is possible to calculate / estimate how much time is needed for each transmission. Of course, this way of calculating the above mentioned metric will introduce certain uncertainty errors, but it will significantly reduce the overhead.

In regards to determining the optimal duration of the learning period: two factors should be taken into account:

1) The dynamics of the network
2) The total length of a learning process

In high-throughput and highly dynamic environments, where network properties change quickly, a large amount of information can be gathered for a relatively short amount of time. Learning episodes in these cases should be shorter because the amount of packets carrying information per time unit will be higher than in a less dynamic networks. Generally, it is desirable not to keep the network too long in a state that is potentially suboptimal (resource consuming). It is safe to say that the effects of suboptimal performance much quicker impacts networks with heavy traffic.

It is also possible to use other metrics to define the duration of a learning episode. For example, the number of received packets that contain performance statistics can be used instead of time units. Obviously, the episodes will be of a different duration, but the amount of collected information from the network during each episode will remain the same.

Finally, the number of episodes and their duration will directly define the duration of the entire learning process. In cases where it is possible to perform an "off-line" learning (on test beds or using network simulators) using artificially generated traffic or by recreating operational conditions, time will not be a factor. However, we must assume that, generally, this won't be possible. In those cases, determining the duration of the learning process will pose a great challenge to a system designer since the practical value of the reasoning method will be directly evaluated through it.

In the case we presented in Section V-C, the system can go through 4 different states. There are 16 possible transition from state to state, but since 4 Q-values are calculated after each transition the lowest possible number of episodes, until all the Q-values are updated is 4. This assumption is valid only if the system will never go into a state is has previously been in. No matter how high we set ε (ε greedy algorithm), there will always be at least ε probability that the engine chooses state it's already been in against the unknown one.

B. Further optimizations and future work

The goal of the previously described work was to select the optimal set of services that should be activated in each network. But is can reasonably be expected that, as time goes on, due to the wide variety of wireless network conditions, network protocols will become increasingly adaptive and configurable. As a result, more and more configurable settings will be available in the networks.

To cope with this situation, LSPI can also be applied in another direction: for the optimization of a single network protocol. The main goal would be to determine the optimal set of protocol settings. Combination of those settings represent system states. Large number of settings will lead to a large number of states, of course. However, the learning process can be speed up by implementing deduction techniques that enable the LSPI engine to recognize certain behavioral patterns. For example, if the reasoning engine notices a performance weakening trend when the back-of values of a MAC protocol

are increased, a prediction can be made about which transitions will probably be sub-optimal so that they can be avoided (by lowering their transition probability). In other words, the reasoning engine can predict behavioral patterns to reduce the number of possible states the system has to go through.

It makes sense to extent the LSPI negotiation engine so that it can be used not only to help discover the optimal set of network services, but at the same time optimally configure other network parameters. During the optimization process, parameters such as radio transmitting power, optimal duty cycle, communication channel can be investigated. Unfortunately, this approach will increase the number of variables, leading to an increase of the number of states, number of state transitions and learning episodes. In other words, the complexity (and total learning time) will increase dramatically with each new aspect of network optimization that is taken into account.

As an alternative, in fully-configurable networks (with multiple configurable network services and mulitple configurable protocols), we propose to reduce the optimization complexity by using a tiered system of network optimizations (see Fig. 5).

1) LSPI can be used to first optimize the parameters of each network protocol of the network.
2) Next, LSPI is used to identify a number of service sets that offer acceptable network performance.
3) Finally, LSPI selects amongst the acceptable service sets those that are also beneficial for co-located networks.

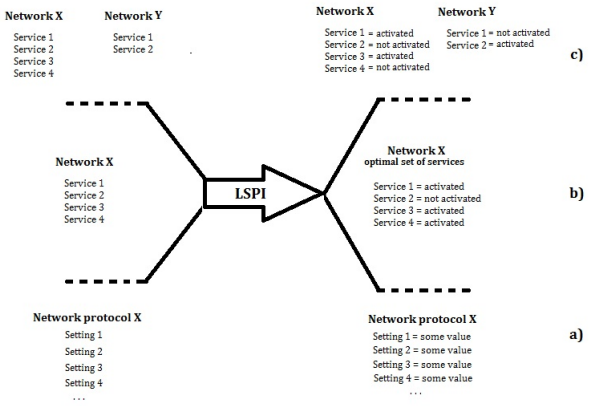

Fig. 5. Scalability of the LSPI algorithm - a) optimization of a single network protocol by tuning up its settings b) optimization of a homogeneous network through defining an optimal set of services that needs to be activated c) optimization of a symbiotic (heterogeneous) network through the process of service set negotiation

Step 1 of this process, optimization of a single network protocol, was described already above. During step 2 of this process, each participating network will be given an opportunity to check how certain number of different service sets (optimal one and a number of sub-optimal) activated in a co-located subnet, affect its own performance. To speed up the learning process, deductions about the influence of certain services can also be made so that a number of states can more quickly be reduced. Finally, during step 3 every network calculates the combined influence of the self-utilized set and the one used in a co-located network. This will, eventually, lead to either a cooperation agreement or a decision not to engage into any cooperation if the networks estimate that neither of them will benefit from it.

Our future work will integrate the above described solution with the existing SymbioNets solutions [12] (that already include network discovery) to determine the optimal service sets, activated in each sub-community, of a large symbiotic network (Fig. 6).

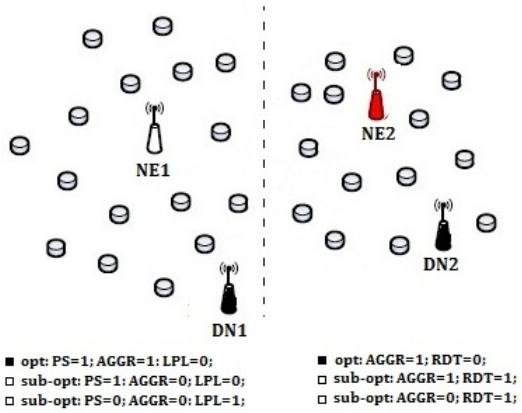

Fig. 6. Two co-located networks eligible for cooperation. Discovery nodes (DE) will be used for mutual detection, after which negotiation engines (NE) will start the reasoning/negotiating process, based on each network optimal and sub-optimal states.

VII. Conclusion

Due to the ever increasing number of wireless networks, independent co-located networks increasingly influence each other, resulting in degraded network performance. As a result, network optimization solutions can no longer afford to look only at the performance of a single network. Optimizing multiple co-located networks, each with a variable number of network functionalities that influence each other, is a complex problem that has not yet received a lot of attention in the research community. In this paper, we propose to use LPI (a form of reinforcement learning) for the selection and composition of high-level networking optimizations in heterogeneous networks. Our approach is beyond the state-of-the-art from Section II in the sense that cognitive networking normally focuses on single-layer parameter optimization, rather than taking the global optimum into account.

The paper presents the mathematical background needed for this optimization process and discusses the difficulties and challenges for implementing these solutions. Our solution adds the LSPI mechanism to the negotiation engine of the existing SymbioNets project. Networks can cooperate with each other by activating network services (such as interference avoidance, shared routing, aggregation, etc.) that influence one or more networks.

In a first phase, our algorithm optimizes the performance of each single network (based on its network requirements) to obtain the baseline optimal network performance of the network without cooperation. During this process, we obtain several sets of services and service compositions that result in improved performance. Once networks know their optimal (as well as their sub-optimal) operating configurations, each participating network engages into a process of service negotiation with other networks, all the while taking into account a firm policy regarding the acceptable and non-acceptable performance. The paper concludes with a number of optimizations that result in faster exploration of the parameter space and a decrease of the learning cycle, thus resulting in faster optimization of the network.

We strongly believe that the problem of interfering co-located networks will only increase. As such, innovative cross-layer and cross-network solutions that take these interactions into account, like the one proposed in this paper, will be of great importance to the successful development of efficient next-generation networks in heterogeneous environments.

ACKNOWLEDGMENT

This research is funded by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) through the IWT SymbioNets project, by the Interdisciplinary Institute for Broadband Technology through the QoCON project and by the FWO-Flanders through a FWO post-doctoral research grant for Eli De Poorter

REFERENCES

[1] Mitola, J., III; Maguire, G.Q., Jr., "Cognitive radio: making software radios more personal", Personal Communications, IEEE , vol.6, no.4, pp.13-18, Aug 1999

[2] Razvan Musaloiu E. and Andreas Terzis. "Minimising the effect of WiFi interference in 802.15.4 wireless sensor networks." Int. J. Sen. Netw. 3, 1 (December 2008), 43-54.

[3] I. Akyildiz, W. Lee, M. Vuran, S. Mohanty, "NeXt Generation/Dynamic Spectrum Access/Cognitive Radio Wireless Networks: A Survey", Elsevier Computer Networks, Vol 50, pp. 2127-2159, 2006.

[4] C. Fortuna, M. Mohorcic, "Trends in the development of communication networks: Cognitive networks", Computer Networks, 2009.

[5] E. De Poorter, B. Latre, I. Moerman and P. Demeester, "Symbiotic networks: Towards a new level of cooperation between wireless networks", Published in Special Issue of the Wireless Personal Communications Journal, Springer Netherlands, 45(4):479-495, June 2008

[6] Wakamiya, N.; Arakawa, S.; Murata, M.,"Self-Organization Based Network Architecture for New Generation Networks", 2009 First International Conference on Emerging Network Intelligence, pp.61-68, 11-16 Oct. 2009

[7] R. W. Thomas, L. A. DaSilva and A. B. MacKenzie, "Cognitive networks", Proc. IEEE DySPAN 2005, pp.352-60

[8] Jiahua Wu, "A Survey of Game Theory in Wireless Networking Design"

[9] T. G. Dietterich, and O. Langley, (2007) "Machine Learning for Cognitive Networks:Technology Assessment and Research Challenges in Cognitive Networks: Towards Self Aware Networks", John Wiley and Sons, Ltd, Chichester,UK. doi: 10.1002/9780470515143.ch5

[10] A. Forster, "Machine Learning Techniques Applied to Wireless Ad-Hoc Networks: Guide and Survey", 3rd International Conference on Intelligent Sensors Sensor Networks and Information (ISSNIP), Melbourne, Australia, 3-6 Dec. 2007.

[11] M. Rovcanin, E. de Poorter, O. Yaron, I. Moerman, D. Plets, W. Joseph, L. Martens "Elaboration of Cognitive Decision Making Methods in the Context of Symbiotic Networking", SENSORCOMM 2012, August 19 - 24, 2012 - Rome, Italy

[12] E. De Poorter, P. Becue, M. Rovcanin, I. Moerman and P. Demeester, "A negotiation-based networking methodology to enable cooperation across heterogeneous co-located networks", Ad Hoc Networks, Available online 8 December 2011, ISSN 1570-8705, 10.1016/j.adhoc.2011.11.007.

[13] M. Lagoudakis and R. Parr. "Model-free least-squares policy iteration". In Proc. of NIPS, 2001.

[14] L. P. Kaelblign, M. L. Littman, A. W.Moore,"Reinforcement learning: A Survey", Journal of Artificial Intelligence Research 4 (1996) 237-285

[15] P. Wang, T. Wang, "Adaptive Routing for Sensor Networks using Reinforcement Learning", CIT '06 Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, October 22-25, 2006 Charlotte Convention Center Charlotte, NC