This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

Discrete Gene Regulatory Networks (dGRNs): A novel approach to configuring sensor networks

Andrew Markham and Niki Trigoni
Oxford University Computing Laboratory
Oxford OX1 3QD
e-mail: firstname.lastname@comlab.ox.ac.uk

***Abstract*—The operation of a sensor network is determined by a large number of parameters, such as the radio duty cycle, the frequency of neighbor discovery beacons, and the rate of sampling sensors. Writing adaptive algorithms to tune these parameters in dynamic network conditions is a challenging task that requires expert knowledge, and many design-test-rewrite cycles. This paper proposes a novel nature-inspired paradigm, termed discrete Gene Regulatory Network (dGRN), for configuring sensor networks. The idea is that nodes should regulate their parameters based on their local state and state communicated from neighbor nodes, in a similar manner that cells regulate their behavior based on local levels of protein concentrations, and proteins diffused from neighbor cells. The proposed dGRN paradigm has two major strengths: 1) it is general-purpose, and can be applied to a variety of parameter tuning problems; and 2) it generates parameter tuning code *automatically* removing the need for a human expert. We demonstrate the feasibility of the dGRN approach in a scenario where nodes must tune their sampling rates to track a moving target with a certain accuracy. The automatically generated code exhibits properties similar to the ones that one would expect from expert-designed code, such as aggressive sampling when the target moves fast and the sensing range is low, and relaxed sampling otherwise. Moreover, the automatically generated code causes nodes to communicate with each other to coordinate their tuning tasks, as one would expect from expert-designed code. The resulting dGRN code is evaluated both in a simulation environment, and in a real environment with eight T-Mote Sky nodes tracking a light-emitting target.**

I. Introduction

The functionality of a sensor network is determined by a number of parameters acting at various layers. The radio duty cycle at the MAC layer, the frequency of neighbour discovery beacons at the network layer, the sampling rate at the application layer, all affect the ability of a network to meet user-specified requirements for accuracy, latency and energy usage. Tuning these parameters is a complex task, especially if one tries to exploit opportunities for cross layer optimization. Expert knowledge is needed to design code that exhibits the correct amount of communication between nodes such that they can collaboratively configure their parameters. The tuning problem is compounded by the need for this code to be adaptive to changes in network conditions, and to dynamics in the sensed phenomenom. This paper presents a novel method of automatically designing a controller which configures node parameters to satisfy the requisite application objectives.

Consider, for example, a target tracking application, where fixed nodes are equipped with sensors that can register the presence of the target and its speed. The user of the application specifies certain constraints that must be met, for example, that the target must be monitored by at least one sensor for 90% of the time. The objective is to meet the constraints by minimizing the average sampling rate, i.e. the frequency at which they power their sensors to search for a target. According to the user, each node should tune its sampling rate (actuator variable), by taking into account its remaining energy and the most recent reading about target presence and speed (sensor variables). The question is: how is this best done?

A centralized optimization approach, which finds the best configuration parameters for each node, can be utilized. However, centralized solutions present a single point of failure and also incur a high communication overhead. The alternative is for an expert to design a distributed algorithm for tuning parameters based on each node's knowledge of its local state and interactions with neighbours. Designing a distributed algorithm that achieves good global performance is a non-trivial task that generally requires expert knowledge and takes many design-test-rewrite cycles. However, the distributed method is preferable to the centralized method in that it is scalable and can dynamically adapt to changes in the sensor environment.

We take a step back from this and examine what is actually required in Wireless Sensor Network (WSN) decision making. In essence, we require a *controller* (black box module) that consumes sensor variables (which can also include inputs from peers) and based on these and its current state, decides how to alter the values of actuator variables. In order to simplify the task of the network engineer, it would be desirable to design the structure of this controller automatically to meet global goals. To accomplish this, we turn to Nature for inspiration.

Consider a simple multicellular organism. It is able to perform tasks such as climbing a chemical gradient (chemotaxis) or reacting to external stimuli such as the presence of light. The organism is able to coordinate the actions of multiple cells without requiring a central controller, and it does so by using protein exchange. Proteins can be regarded as chemical messengers which act within the cell and can also diffuse to neighbouring cells. Proteins are produced by genes in the nucleus of the cell, and genes can be viewed as rules that take as input the current protein concentrations and control the further production or suppression of proteins in a time dynamic manner. This mechanism is referred to as a Gene Regulatory Network (GRN), as genes can be viewed as rules

978-1-4244-5837-0/10/$26.00 ©2010 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

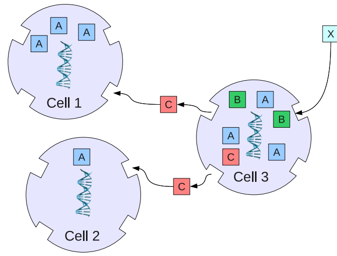

Fig. 1. Highly simplified diagram showing protein exchange within an organism. The proteins are represented by the labelled boxes, and the number of these is indicative of the concentration. Assume that an exogenous protein, X, is applied to cell 3. Based on this input, the cell increases its production of protein B. It also diffuses protein C to its neighbours. These cells can respond differently to incoming protein C, as cell 2 has a lower concentration of protein A compared with cell 1.

that regulate cell behavior through protein production [1]. By means of protein diffusion, a cell can influence its neighbour's behaviour, as shown in Fig. 1. The precise behaviour of these extremely complex networks has been created over many millennia by Nature's search algorithm, evolution.

We take an abstraction of the protein regulatory mechanism and use this to control the actions undertaken by each node in a sensor network. Each node is equivalent to a cell and proteins are diffused between neighbouring nodes to communicate state. However, rather than representing protein concentrations as continuous variables, we restrict them to a set of discrete values. This simplifies the rule structure, the computation, and also reduces the communication overhead. We thus refer to our approach as a *discrete Gene Regulatory Network (dGRN)* to emphasize the fact that protein concentrations are discrete in value. Every node runs identical dGRN rules that take as input sensor variables (such as target speed) and control the values of actuator variables (such as sampling rate). To design dGRN rules, we again use a nature-inspired approach, namely an evolutionary algorithm.

In particular, the contributions of this paper are as follows:

- We propose a novel paradigm for controlling sensor network operation inspired by Gene Regulatory Networks.
- We illustrate how to automate the process of network configuration, starting from random dGRN rules and refining them through evolution.
- We show that communication is an emergent property of the evolved solutions and does not need to be specified by human experts.
- We demonstrate the feasibility of the dGRN methodology in a target tracking application. We show that it is possible to automatically generate parameter tuning code that exhibits properties desirable by human experts.
- We evaluate the resulting code in a simulation environment, and in a real environment with eight T-Mote Sky nodes tracking a light-emitting target.

The paper is structured as follows: Sec. II presents our dGRN framework and explains how code is randomly generated and evolved. Sec. III shows how dGRNs can be trained to track a moving target through a sensor network. Sec. IV evaluates the resulting dGRN solution in a simulation environment and a real sensor testbed. Sec. V discusses related work and Sec. VI concludes the paper and proposed directions for future work.

II. dGRN Methodology

In this section, we describe in detail the process of generating and evolving dGRN rules that control parameters of node operation. Throughout this section we use the target tracking scenario introduced in Section I. Note however that the dGRN methodology can be applied to many different configuration problems, ranging from boundary detection to routing control. The dGRN methodology consists of four phases:

Phase 1: In the first phase, the user defines high level application requirements, and specifies which node variables are associated with inputs and outputs of the dGRN controller. The user also defines a fitness function that will be used in later phases to assess the performance of candidate dGRN controllers.

Phases 2 through 4 are then conducted automatically without user intervention.

Phase 2: A number of random dGRN controllers are generated. Each controller consists of a number of rules which specify how protein concentrations are updated in the next time step, based on current protein concentrations and sensory input.

Phase 3: Each controller is executed, and its overall performance measured against the fitness function specified in Phase 1. An evolutionary algorithm is used to create new dGRN controllers based on controllers that performed well in the last iteration.

Phase 4: Once an acceptable controller has been found, it is then disseminated into the network where it runs on each node, dynamically adapting to changing conditions.

These phases are now discussed in detail.

A. Phase 1: High level problem definition

Firstly, the user needs to decide how to interface the dGRN controller to the node itself. There are three different types of proteins in a dGRN controller: 1) *sensor proteins* which bind to sensor variables of the node (such as the speed or presence of a target), 2) *actuator proteins* which control actuator variables of the node (such as the sampling rate), 3) *hidden proteins* which reflect the dGRN's current state. Note that a dGRN controller can have multiple inputs (sensor proteins) and outputs (actuator proteins). In essence, the user defines the variables that should be controlled, and what variables should affect the parameter tuning.

Once suitable node variables have been chosen, they need to be translated or mapped from their real world values to discrete protein concentrations. The left table in Fig. 2 provides an example of mapping values of the sensor variable *target presence* to concentrations of sensor protein $P4$. For instance, if the node detected a target with speed greater than 5 m/s in the previous step, it sets the concentration of $P4$

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2010 proceedings
This paper was presented as part of the main Technical Program at IEEE INFOCOM 2010.

to 3 in the current step. Similar mappings are defined from actuator proteins to actuator variables. For example, the right table of Fig. 2 maps concentrations of protein $P1$ to sampling rate values. The coarseness of the mappings depends on the base, B used for the dGRN, which is chosen by the user. For example, a base $B = 2$ results in two possible protein concentrations (0,1), whereas a base $B = 4$ results in four values (0,1,2,3). The level of discretization of the protein concentrations controls the size of the dGRN state space.

Other parameters that a user needs to specify are the number of hidden proteins in the dGRN controller, the maximum length of update rules and variables related to the evolutionary process, such as the population size. The role of these parameters is discussed in Phase 3.

The next step in this phase is for the user to construct network scenarios that will be used to examine the performance of various dGRN controllers in simulation. These scenarios must be sufficiently realistic to separate slight differences in dGRN performance, but must be relatively quick to execute as they are the most time consuming portion of the evolutionary process. In our example problem, a suitable scenario might be to execute the dGRN controller on a wireless network with $N = 64$ nodes for $T = 1000$ units of time with a specific target mobility pattern.

The specification of the fitness function is the last step in this phase. The fitness function returns a value which indicates the relative performance of a particular dGRN controller at the specified task. In the target tracking example, it is required that a target must be detected by at least one node for 90% or more of the simulation time (the constraint) and to do so using the smallest average sampling rate (the objective). In general, we assume that the constraint(s) must be met and the objective(s) minimized. Thus it makes sense in this example to use a piecewise defined fitness function to account for the two conditions[1].

If the accuracy Q_a obtained by a dGRN controller meets or exceeds the constraint of 90%, we use the mean squared sampling rate as our metric, where S_i is the average sampling rate of individual nodes. Conversely, if the tracking accuracy is not met, a large positive offset (corresponding to all nodes having a 100% sampling rate) is added to the difference between the actual accuracy and the target accuracy. Thus, we can write the piecewise fitness function as:

$$F = \begin{cases} \sum_1^N (S_i)^2 & : Q_a \geq 90\% \\ N(100\%)^2 + (100\% - Q_a) & : Q_a < 90\% \end{cases} \tag{1}$$

In summary, the user specifies the sensor and actuator variables and their mappings, the network test scenarios and lastly the fitness function which measures the performance of dGRN controllers. Note that the user does not need to specify how nodes communicate with each other. The subsequent phases are then undertaken automatically by the dGRN evolutionary process.

[1]Note that for simplicity, we do not include the energetic cost of communication in the fitness function as we assume the energy consumed by sampling is large. However, it is a simple matter to modify the fitness function to reflect the total energy used by each node.

Target Presence → P4

Sensor Variable	Protein Concentration
Target Present AND (5 m/s ≤ Speed)	3
Target Present AND (2 m/s ≤ Speed < 5 m/s)	2
Target Present AND (Speed < 2 m/s)	1
Target Absent	0

Sensor Variable Mapping

P1 → Sampling Rate

Protein Concentration	Actuator Variable
3	100 %
2	50 %
1	25 %
0	12.5 %

Actuator Variable Mapping

Fig. 2. Example of sensor and actuator variable mappings. The sensor values are mapped to protein concentrations of protein P4. The dGRN controls the concentrations of protein P1 which are mapped to the node's duty cycle. The base used in this example is $B = 4$.

B. Phase 2: Generation of random dGRN controllers

The main objective of this phase is to generate candidate dGRN controllers in a random manner. Recall that a dGRN controller consists of a number of rules that control how protein concentrations change over time[2].

An example of a protein update rule is: $P_1^* = P_1 \ o_1 \ P_2$, which denotes that the new value of protein P_1 is derived by combining the current values of P_1 and P_2 with a binary operator[3] o_1. Each protein (with the exception of sensor proteins, as their values are set directly by the node) has an associated update rule and all protein concentrations within a dGRN controller are updated simultaneously.

Similarly to biological systems, where proteins from one cell can be diffused to a neighboring cell, we also allow a protein update rule to take as input the concentration of proteins from neighbouring nodes. We use the notation $\bar{P}_i$ to refer to the concentration of P_i at a neighbour node. Given that nodes may have multiple (or no) neighbours, a method is required to aggregate or merge the multiple foreign protein concentrations together. In general, we define the expansion $\langle m_j, \bar{P}_i \rangle = \bar{P}_i(N_1) \ m_j \ \bar{P}_i(N_2) \ m_j \ \ldots \bar{P}_i(N_\ell)$, where m_j is a binary operator used to aggregate or merge the values of P_i at neighbor nodes $N_1, \ldots, N_\ell$. For example, a rule for updating protein P_1 in the local node may combine the local value of P_2 with aggregated values of P_3 from neighbor nodes: $P_1^* = P_2 \ o_7 \ \langle m_4, \bar{P}_3 \rangle$.

Thus, in general, a rule has the form:

$$P_i^* = \langle m, P \rangle \ o \ \langle m, P \rangle \ o \ldots \langle m, P \rangle, \tag{2}$$

where P is a protein species drawn from the protein alphabet. Note that the merge operator has no effect on local proteins, i.e. $\langle m, P_i \rangle = P_i$. An example protein alphabet, corresponding to the mapping discussed in Phase 1, is shown in Fig. 3. In this alphabet, there is one sensor protein (target presence), P_4, one actuator protein (sampling rate), P_1 and two hidden proteins

[2]Note that the timebase used for the dGRN depends on the application requirements and can vary between seconds and hours.

[3]Note that we refer to a binary operator as a function which takes two values as operands and returns a single value as a result. A binary operator is not the same in this context as a *Boolean* operator (AND, NOT, OR etc).

Protein Lookup Table

Index	Symbol	Name
1	P1	Actuator Protein 1
2	P2	Hidden Protein 1
3	P3	Hidden Protein 2
4	P4	Sensor Protein 1
5	$\overline{P1}$	Neighbour Actuator Protein 1
6	$\overline{P2}$	Neighbour Hidden Protein 1
7	$\overline{P3}$	Neighbour Hidden Protein 2
8	$\overline{P4}$	Neighbour Sensor Protein
9	J	Jump
10	T	Terminate

Operator Lookup Table (Base 4)

Index	Symbol	Name	Operator
1	o1	Operator 1	$\begin{bmatrix} 0 & 2 & 3 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 2 & 1 & 0 \\ 3 & 3 & 3 & 3 \end{bmatrix}$
2	o2	Operator 2	$\begin{bmatrix} 3 & 1 & 2 & 0 \\ 1 & 0 & 1 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 0 & 1 & 2 \end{bmatrix}$
3	o3	Operator 3	$\begin{bmatrix} 3 & 3 & 3 & 1 \\ 3 & 3 & 1 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$
4	o4	Operator 4	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 1 & 3 & 0 & 3 \end{bmatrix}$
5	o5	Operator 5	$\begin{bmatrix} 3 & 3 & 0 & 0 \\ 1 & 1 & 2 & 2 \\ 2 & 2 & 3 & 3 \\ 0 & 0 & 1 & 3 \end{bmatrix}$

Fig. 3. Lookup tables for protein and operator decoding. Note that the tables do not need to be of the same length. For each table, the index is a numeric value which decodes to a particular symbolic representation. Observe that, in the operator table, each operator is specified. In this case, the operators are base $B = 4$

P_2 and P_3. We introduce two control proteins that allow us to generate varying length rules. The JUMP, J protein instructs the dGRN controller to skip to the next protein in the rule and the TERMINATE, T protein instructs the dGRN controller to end the evaluation of the current rule and move to the next.

We now turn our attention to discussing the operators that are used to combine the protein concentrations. An operator is a function which acts on two protein species (operands) to produce a resultant concentration. Typical operators for a base-4 system are shown in Fig. 3. Note that each operator is a 4×4 matrix which essentially acts as a lookup table. The concentration of the first protein operand is the row index into the matrix, and the concentration of the second protein operand is the column index. The resulting concentration is the number which is found at the intersection of the row and column indices. Operator lookup tables are initially populated with random numbers, as shown in Fig. 3. The user only specifies the number of operators, but does not define the operators themselves.

Now that we defined proteins and operators, it remains to show how to generate random dGRN rules. The idea is to generate a random string of integers and translate them, with the aid of lookup tables, into a set of dGRN rules. Recall from Eq. 2 that a rule has a fixed structure consisting of merge, protein and operator triplets. Numbers are either decoded into protein symbols or operator symbols, based on their position in the rule. Protein and operator symbols are drawn from lookup tables similar to those in Fig. 3. Note that merge operators, m and normal operators, o are identically drawn from the same operator alphabet i.e. $m_i \equiv o_i$. An example of the rule decoding process is shown in Fig. 4. Starting from strings of integers on the left, each number is mapped to a symbol depending on its position within the rule. Merge operators are removed from local proteins and special control symbols are parsed to result in the final rule shown in the right of the figure. This example demonstrates how varying length rules

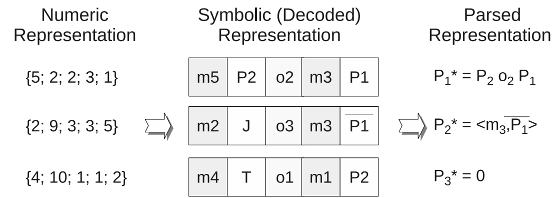

Fig. 4. Example showing how numeric strings are parsed to produce protein update rules.

can be created from a fixed length string of numbers.

In summary, a dGRN controller consists of a number of rules acting on proteins to generate new protein concentrations. Operators are used to combine two protein concentrations together, and they are randomly generated, rather than being specified by the user. The following section discusses how randomly generated dGRN controllers can be refined through a process of evolution.

C. Phase 3: Evolving dGRN controllers

The random dGRN controllers generated in Phase 2 would, in general, be expected to perform quite poorly at the test scenarios specified in Phase 1, resulting in solutions with low fitness (corresponding to high values of the fitness function). However, some controllers would perform slightly better than others. It is these differences between controller performance that are used by the evolutionary algorithm, which is a search method inspired by the process of natural selection. In our work, we use Differential Evolution, which is a variant of the standard genetic algorithm [2].

The steps involved in this process are as follows:

1) Initially, a number (e.g. 100) of candidate dGRN controllers, comprising the population, are randomly generated. Their performance at the task is evaluated using the fitness function specified by the domain experts in Phase 1.

2) A new child solution is generated from each individual (the parent) in the population and three other individuals (one of which acts as a pivot node and the remaining two as genetic 'donors') chosen at random. For each gene in the chromosome, the parent's gene is replaced with a probability of crossover DE_CR = 0.8. To replace the gene, a weighting factor (DE_F = 0.2) is used to scale the difference between the genes of the two donors. This weighted difference is then added to the gene of the pivot node. This particular approach is referred to as the DE_rand_1_bin update equation (refer to [2] for the Differential Evolution algorithm and a discussion of how to choose the parameters).

3) The performance of these new individuals is assessed using the fitness function. If they are better than the parent, the parent is replaced by the child.

4) Steps 2 and 3, which together comprise a single generation, are iteratively executed until either a target fitness is reached or the number of generations exceeds a limit (e.g. 300). If

it exceeds the limit, then the dGRN parameters need to be changed (e.g. the rule length or the base used).

Note that a dGRN controller has two parts, namely a set of rules (initially randomly generated) and a table of operators (also initially randomly generated). We evolve dGRN controllers in an iterative, two step process. In the first step, the rules of the dGRN controllers are evolved, keeping the operators fixed. The fitness of these controllers is evaluated and the best controller (the one with the lowest fitness score) is selected. We then fix the rules of this controller and evolve its operators. This iterative process of alternatively evolving rules and operators is continued until a good solution is found.

D. Phase 4: Disseminating and executing dGRN code

In this section we discuss, with particular reference to communication overhead, the dissemination and execution of dGRN controllers. In order to assess the cost of sending a dGRN controller to nodes in the sensor network, we must now examine the size of a dGRN controller in bytes. As discussed in Phase 2, a ruleset can be represented by a string of integers, with each number being mapped to a symbol from either the operator alphabet or the protein alphabet. For simplicity, assume that each alphabet has 16 or fewer symbols in it. We can thus represent each symbol as a 4 bit number. Recall that a rule consists of a sequence of symbols in the form (merge m, protein P, operator o). Thus, these three symbols can be written using 3×4 bits = 12 bits. Assume that in our example 3 proteins have update rules, with a maximum rule length of 3. The total number of bits required to write the ruleset is $12 \times 3 \times 3 = 108$ bits (14 bytes).

Additional to the ruleset is the operator lookup table. The number of bits required for each operator depends on the base used. Let $B = 4$ and assume 16 operators are present in the lookup table. Each operator consists of B^2 numbers, and each of these can be written as a 2 bit digit. Thus an operator is $4^2 \times 2 = 32$ bits = 4 bytes in size. As there are 16 operators in total, the total size used by the operator pool will be $16 \times 4 = 64$ bytes.

Other parameters which need to be specified include the sensor and actuator mappings, the base, number of proteins and the update rate. In this example these add approximately 20 bytes of overhead. Hence in total, this example dGRN controller can be expressed in $14+64+20 = 98$ bytes. This is less than the maximum packet length of 128 bytes in 802.15.4 networks. This example has assumed that no optimizations have been performed on the rule and operator alphabets – preparsing the rule and reducing the size of the operatorset by eliminating operators which are not used in the dGRN controller can result in an even smaller dGRN representation. Thus, it can be seen that disseminating new dGRN controllers into a wireless network does not incur a large amount of communication overhead.

Once the optimal dGRN controller has been disseminated into the network, the following steps are taken by each node:
1) It maps sensor variable(s) to sensor protein(s)
2) It executes the dGRN rules, updating the protein values
3) It maps actuator protein(s) to actuator variable(s)
4) It broadcasts updated protein values to neighbours and receives updated protein values from its peers
5) It waits for t units of time

Note that steps 1 – 3 and 5 do not involve any communication and are executed locally on each node. Consider the communication cost of step 4. Assume that we have 4 proteins in our dGRN controller. If we use $B = 4$, each protein concentration can be written as a 2 bit number. Hence in total, a protein update requires $4 \times 2 = 8$ bits to be sent to neighbours. Note that proteins only need to be transferred when they have an effect on neighbours update rules (i.e. if $\bar{P}_1$ is not present in the controller, protein P_1 does not need to be sent to peers). Further to this, if we assume that a node can buffer its neighbour's protein values, they only need to be sent when they change. Depending on the timebase t, it may be possible to piggyback the protein update on top of regular messages such as neighbour advertisement beacons. Thus it can be seen that updating protein concentrations does not have a large communication overhead.

III. Evaluation

In this section, we investigate the benefits of the dGRN approach with a simple scenario, namely collaborative target tracking in a wireless sensor network. Our goal is to minimize the average sampling rate required to track the target, subject to the target tracking accuracy Q_a being met. Primarily, we want to investigate whether dGRN controllers can communicate effectively with each other to achieve the scenario goals. It must be stressed that any communication which occurs between controllers is *emergent* and has not been specified by the designer.

Secondly, we ask whether the solutions obtained are comparable to those that would be expected from a human designer. Intuitively, we would expect that the faster the target moves through the sensor field, the more aggressively nodes communicate in order to activate (tasked to a high sampling rate) nodes in the neighbourhood to maintain continual tracking. Similarly, the higher the desired tracking accuracy, the larger the region of activation needs to be. Indeed, as will be shown, this is precisely the behaviour that occurs, without a programmer instructing detailed behaviour. Thus, the tracking strategy generated by the dGRN controllers varies to meet user requirements in terms of tracking accuracy and application requirements, such as the speed of the target.

Lastly, we investigate whether a dGRN controller can alter its tracking strategy depending on the speed of the target, for example if it is able to communicate over a wider area in order to activate more nodes when the target is moving rapidly. This behaviour is what would be expected from an expert designer and we show that a dGRN controller can arrive at a similar strategy.

A. Simulation Parameters

As potential application users, we need to set a number of parameters that were discussed in Phase 1. The sensor

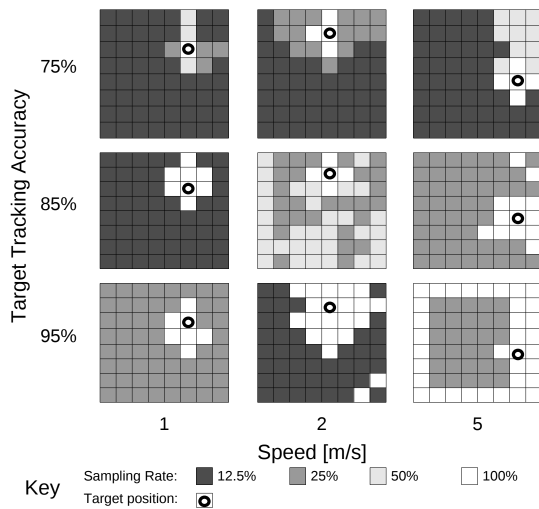

Fig. 5. Snapshots of typical activation patterns for the various scenarios. Observe how the area of activation becomes larger the more challenging the tracking task becomes.

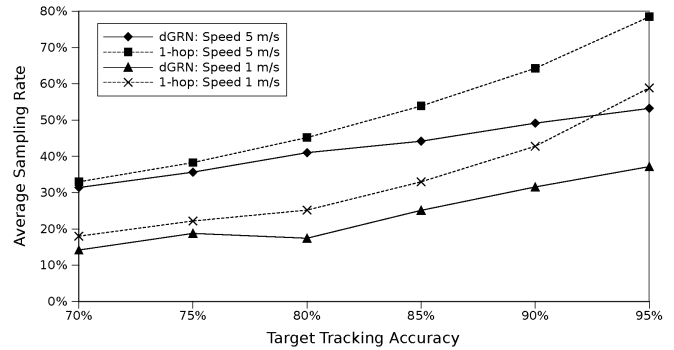

Fig. 6. Variation in average sampling rate with different speeds of target motion and accuracy requirements.

protein is defined as target presence and speed, and the actuator protein as the sampling rate. The mappings shown in Fig. 2 are used, unless otherwise specified. Note that the sampling rate controls the probability of a node attempting a target detection in a particular iteration. Additionally, two hidden proteins are included and the maximum rule length is set to be 3. For the dGRN controller, a base $B = 4$ is used.

For the network test case, the nodes are arranged in a regular, square lattice and each node can communicate with its direct neighbours. For simplicity, we assume that each node's sensing area is square with side of 10 m. Thus, the majority of nodes are connected to four others, with the exception of edge and corner nodes (three and two respectively). The target executes a random walk across the sensor field, moving at a particular speed. Although the nodes are at most 4-connected, the target can also move diagonally. This makes the target tracking process slightly more difficult. A total of 64 nodes are arranged in an 8×8 cellular array. For the first section of the evolution, we assume that exactly one node can detect the target at each point in time. The total simulation time is set to be 1000 s, with a 1 Hz dGRN update rate. In order to assess performance, the fitness function defined in Eq. 1 is used.

The dGRN evolutionary process was written in a mixture of C and Python. Typical execution time for a single run (consisting of 200 generations) is approximately 6 minutes on a 2.70 Ghz dual core pentium with 4 Gb of RAM. As a comparison, a 200 generation run for a 900 node network takes approximately 60 minutes.

B. Baseline Comparison

We compare the performance of the obtained dGRN controllers against a simple algorithm, which we refer to as 1-hop activation. All nodes have the same default sampling rate. When a node detects a target, it changes its sampling rate to 100% (i.e. continually on) for as long as the target is within sensing range. It also informs its 1-hop neighbours that it has detected a target, and they also switch to 100% activation, such that tracking can be maintained when the target moves. Once the target moves away from the node, its sampling rate is decreased back to the default value. We select the default sampling rate to be the minimum that achieves the specified application conditions.

C. Varying target tracking accuracy

In this experiment, we examined how various tracking accuracies and speeds of motion impact both the energy used and the activation patterns required. The desired accuracy was varied from 75% to 95% and the speed was varied to be 5 m/s, 2 m/s and 1 m/s. Fig. 5 shows typical activation patterns for the best dGRN solutions for a variety of scenarios. The shading of each cell indicates each node's chosen sampling rate, with lighter shades representing a higher sampling rate. Note that the sensing range is such that the target is visible only to one node at a time. Further observe that cells around the target are also awake, pre-emptively attempting to detect the target. Thus, the dGRN controllers, whose rules are automatically created, have communicated this presence with each other. It must be strongly emphasized that this communication is emergent and is generated by the evolutionary process to satisfy the application requirements. Consider now how the patterns of activation change with altering user requirements. Examining Fig. 5, it can be seen that the higher the speed of the target, the more widely the target presence is communicated, resulting in a larger area of activation around the target. Similarly, the higher the desired tracking accuracy, the larger the activation area. As nodes can only communicate with neighbours at the $\{N;E;S;W\}$ coordinates and the region of activation extends past these neighbours, it demonstrates that the dGRN controllers have evolved the ability to communicate target presence over multiple hops.

Of special interest is the fact that some dGRN controllers locate the initial position of the target by turning all nodes on in the first timestep, followed by turning off all nodes except the one which located the target.

The average sampling rate of the various controllers is

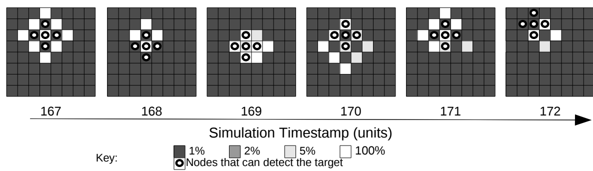

Fig. 7. Time varying tracking behaviour for various target accuracies. Sensing area is 5 cells and target speed is 5 m/s.

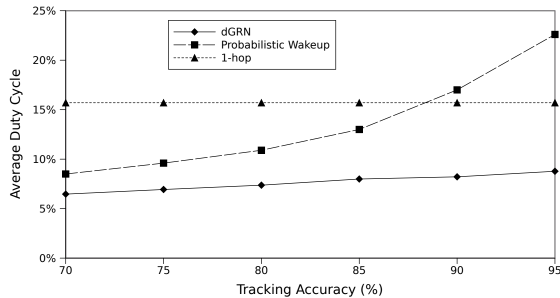

Fig. 8. Performance of the various algorithms when the sensing range is increased to 5 cells.

shown in Fig. 6, compared with the non-communicating baseline. It can be seen that the average sampling rate of the dGRN controllers is lower than that of the 1-hop activation algorithm. This demonstrates that efficient communication is occuring between controllers. Together, these dGRN curves form a Pareto Frontier diagram which show the relationship between sampling rate (which is proportional to energy consumption) and tracking accuracy for varying target speeds. These curves can help to guide the user in choosing a dGRN controller which best satisfies the application requirements. Note that each of these points on the curves corresponds to an actual dGRN controller that has been evolved. Thus, once the operating point has been chosen, the controller corresponding to that particular point can be instantly sent to the network.

This experiment has demonstrated that communication between nodes is emergent and alters according to the particular application requirements.

D. The effect of increasing sensing radius

This experiment considered the impact of increasing the node's sensing range. A target can be detected by up to five nodes instead of just one. The number of nodes remains the same, and the target speed was 5 m/s. Due to the larger sensing range, the actuator mapping was chosen to have lower sampling rates and was: $(0 \mapsto 1\%; 1 \mapsto 2\%; 2 \mapsto 5\%; 3 \mapsto 100\%)$ Fig. 7 shows the time-varying patterns of activation for an accuracy of 90%. These show how surrounding nodes are activated to pre-emptively detect the target as it moves. There is a distinctive black and white checkboard pattern of activation around the target compared with the continuous zones of activation for the original experiment. The dGRN controller has thus automatically identified that nodes which are adjacent to each other do not need to be actively sensing

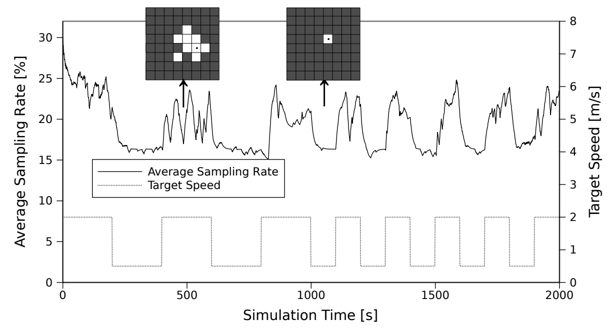

Fig. 9. dGRN adapting to speed of target. Note that when the target is moving rapidly, the average duty cycle is higher. Typical activation patterns are shown for each of the phases.

at the same time, as they have the same information about target presence due to the increased sensing range. Hence, the dGRN controllers collectively act to suppress redundant sampling, conserving energy whilst still maintaining tracking.

The average sampling rate of the dGRN controllers with varying tracking accuracy is shown in Fig. 8, compared with 1-hop activation and a non-communicating random sampling scheme. It can be seen that the dGRN controllers achieve the lowest average sampling rate out of all the approaches.

This experiment has shown that increasing the sensing range results in strategies which inhibit redundant sampling effort, reducing the required average sampling rate.

E. Adaptive speed tracking

The last experiment was conducted to determine whether a dGRN controller can exhibit different behaviour depending on the speed of the target – for example, if the target is moving slowly, we would expect that the activation pattern would be smaller than if the target is moving rapidly. To train the dGRN controller, the target was set to switch between speeds of 2 m/s and 0.5 m/s every 200 s. The results from the adaptive speed tracking are shown in Fig. 9. The first half of the graph shows the speed relationship that the dGRN controller was trained for and the second half is a test case involving a more rapid switching between the two speeds. This graph shows that as the speed changes, the activation patterns change. For the speed of 2 m/s, a number of nodes around the target are activated, whereas for the slow speed of 0.5 m/s, only the node which has detected the target is active. Thus, the dGRN controller has evolved two different tracking behaviours and is able to switch between them in response to the target's speed.

This experiment demonstrates that a dGRN controller can exhibit adaptive behaviour by allowing its sampling rate and communication patterns to vary according to the speed of the target.

IV. Preliminary Implementation

In order to demonstrate that the dGRN approach can simplify sensor network configuration, we present results in this section from a real world testbed, shown in Fig. 10. We investigated a target tracking application where all the nodes

Fig. 10. Photograph of the 1-D target tracking testbed. Each node is equipped with a lightsensor which is shielded for directionality. The torch acts as the target. The node in the bottom of the picture is a sniffer which logs all of the packets.

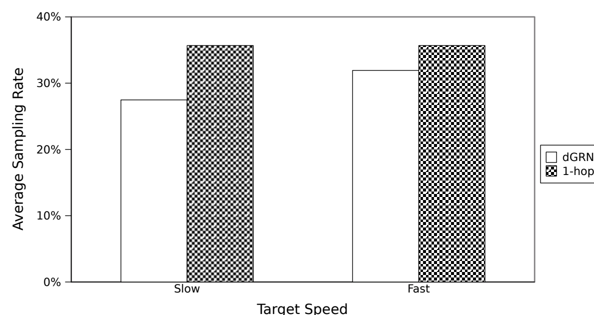

Fig. 11. Results from the 1-D tracking experiment, showing that the average sampling rate of the dGRN controller is lower than the 1-hop activation algorithm.

are arranged in a line (1-D tracking). An example of such a scenario could be tracking people in a corridor, tracking vehicles on a road or monitoring animals along a fence. Note that in each of these scenarios, there are a number of variables which impact the configuration of the sensor network such as the expected speeds of the target and the required tracking accuracy. We compare the evolved dGRN controllers against a generic algorithm (1-hop activation) which simply instructs neighbouring nodes to turn on their sensors when a target is detected.

The dGRN controllers used a base $B = 4$, and there were two hidden proteins, one sensor protein (target presence or absence) and one actuator protein (effected sampling rate). The mapping that was used to control the probability of sampling at each dGRN cycle was the same as used in Fig. 2. Each node was equipped with a light sensor which measures incident illumination from the "target" (a torchbeam) which was moved over the nodes. When the sensor was powered up, if the light level exceeded a threshold, the actuator protein was set to 3 (target present), otherwise it was set to 0 (target absent). The target tracking accuracy was set to 90%.

The dGRN system was implemented using the Contiki OS on a T-Mote SKY equivalent platform[4], using a total of 8 tracking nodes. The dGRN controller used an additional 3.6 kB of Flash and 176 bytes of RAM. The node in the bottom of the picture is a sniffer which simply captures all packets and relays them to the PC for realtime display and logging. Due to the close proximity of the nodes, the routing tables are fixed such that a node only listens to transmissions from its immediate neighbours. Access to the wireless channel is random and protein concentrations are disseminated in unacknowledged broadcast packets. The dGRN update cycle time was set to 2 [s] and the interval between transmitted protein beacons to 0.5 [s]. We tested the performance of the controllers with a target moving at a high speed (pause time at each node varied uniformly between 1 and 4 dGRN cycles) and a slow speed (pause time at each node varied uniformly between 4 and 15 cycles). Each experiment was run for 200 dGRN cycles, and was repeated 3 times in order to obtain an average measure of performance. The logfiles which were obtained from the dGRN experiment were then postprocessed to compare the performance of the 1-hop activation tracking algorithm.

The results from these experiments are shown in Fig. 11, which demonstrate that the dGRN controllers achieve a lower average sampling rate than the 1-hop activation algorithm. The difference is greater when the target is moving slowly, as the dGRN selectively powers down nodes, resulting in a lower average sampling rate.

It must be emphasized that the process involved in creating these different dGRN controllers consists of simply changing the speed distribution in the dGRN simulation. Once the speed distribution has been specified, no further human input is required. Thus the operation of a sensor network can be automatically configured for a particular application scenario by an inexperienced network user. This is the power of the dGRN approach.

[4]Code for the dGRN Contiki implementation, and for the other simulations, is freely available from the authors via e-mail request.

V. Related Work

An overview of the various representations of GRNs, ranging from nonlinear differential equations to random Boolean networks can be found in [3]. However, our approach of using discrete protein concentrations is most similar to that of Thomas on the logical representations of regulatory networks as discrete dynamical systems [4].

To date, there has been only one application of GRNs to a wireless network problem. Das *et al.* present the use of a GRN model to address the problem of coverage in a sensor network [5]. They show how a GRN approach can achieve similar performance to a Genetic Algorithm based optimization algorithm (such as NSGA-II), in a distributed fashion. Unlike our approach, the GRN structure has been specified by the designer. That is, although they have presented an adaptive and distributed method, it still needs to be designed by an expert. One of the first applications of GRNs to realistic distributed systems, was the work by Taylor on multi-robot configuration using GRNs [6]. The main contribution of their work is the demonstration of how a GRN can be evolved to perform a distributed task. Related to this, Quick *et al.* showed how a GRN could be evolved to form a control system for a single robot [7]. A more biological investigation is the work

of Knabe *et al.* on evolving biological clocks [8]. Their results show how GRNs can be evolved to oscillate in synchrony with an applied reference signal, and how noise and phase changes affect the resulting oscillation.

Genetic algorithms of various flavours have been used to optimize network coverage [9], topology [10] and routing [11]. The problem with this approach is that the solution found is static, and thus cannot adapt to changes in network topology or account for node failure. One way of tackling the static nature of the solutions is to run the genetic algorithm periodically, as in [12].

Boonma and Suzuki present a biologically inspired method to control the behaviour of mobile agents in a wireless sensor network [13]. Parameters affecting agent operation are encoded in its genetic information. Elite agents which outperform their peers on competing objectives are disseminated back into the network. This is an interesting approach to dynamically tuning agent behaviour using an online genetic algorithm. However, where it differs from our work is that gross agent behaviour is specified *a priori* and the genetic algorithm is used to fine tune parameters affecting agent operation. Our work seeks to design the distributed behaviour itself.

Another approach which is used in controlling node behaviour is resource allocation through reinforcement learning, where nodes learn the expected payoff for certain actions based on their prior behaviour [14], [15]. The problem with this approach is that the reward parameters need to be tuned by the user. Game theoretic constructs have also been used to alter node behaviour, by treating nodes as players in a mathematical game but again, the difficulty is in writing these strategies [16].

VI. Future directions and Conclusions

In this paper we have presented a general framework for evolving distributed wireless sensor network controllers, inspired by the way in which cells regulate their behaviour by exchanging protein with each other. The key advantages of the proposed dGRN approach over existing techniques are that: 1) it generates adaptive code for parameter tuning in an *automatic* manner; 2) communication is an emergent property of the system and is not defined by human experts; 3) it provides a generic framework that can be applied to a variety of problems.

The applicability of this approach to the problem of tuning the sampling rate in a target tracking application was illustrated. We showed that the dGRN framework automatically generates distributed code, arriving at communication and activation patterns similar to those that would be designed by human experts. Increasing the sensing range resulted in strategies which suppressed activation of neighbouring nodes, in order to reduce the amount of redundant sampling. Moreover, it was demonstrated that nodes are able to adapt their activation patterns in response to changes in target speed. These results indicate that dGRNs could be a promising new paradigm for automatically and adaptively configuring sensor network operation.

However, the dGRN framework is in its infancy, and through our experiments, we identified a number of limitations. The first is that evolving dGRN controllers, especially for more complex problems, can take a large amount of time. We are investigating ways to prune the controller search space to reduce the time taken to converge to good solutions. Secondly, the mappings used for the dGRN controllers impact performance. For finer control, the node could either increase or decrease the variable in response to the concentration. We are also determining if mappings can be designed as a third phase in the evolutionary process. Lastly, we need to investigate performance subject to the loss of a communication link or a noisy sensor variable.

Acknowledgment

This work was supported by EPSRC WildSensing grant (EP/E013678/1). Many thanks to Sonia Waharte for her suggestions.

References

[1] F. Jacob and J. Monod, “Genetic Regulatory Mechanisms In Synthesis Of Proteins,” *Journal of Molecular Biology*, vol. 3, no. 3, p. 318, 1961.
[2] K. Price, *New Ideas in Optimization*. McGraw Hill International (UK), 1999, ch. 6: An Introduction to Differential Evolution, pp. 79 – 108.
[3] H. de Jong, “Modeling and simulation of genetic regulatory systems: a literature review,” *Journal of Computational Biology*, vol. 9, no. 1, pp. 67 – 103, 2002.
[4] R. Thomas, “Regulatory networks seen as asynchronous automata: a logical description,” *Journal of Theoretical Biology*, vol. 153, pp. 1 – 23, 1991.
[5] S. Das, P. Koduru, X. Cai, S. Welch, and V. Sarangan, “The gene regulatory network: an application to optimal coverage in sensor networks,” in *GECCO ’08, NY, USA*, 2008, pp. 1461–1468.
[6] T. Taylor, “A genetic regulatory network-inspired real-time controller for a group of underwater robots,” in *Eighth Conference on Intelligent Autonomous Systems (IAS-8)*. IOS Press, 2004, pp. 403–412.
[7] T. Quick, C. Nehaniv, K. Dautenhahn, and G. Roberts, “Evolving embodied genetic regulatory network-driven control systems.” in *Seventh European Conference on Artificial Life (ECAL 2003)*, 2003.
[8] J. F. Knabe, C. L. Nehaniv, M. J. Schilstra, and T. Quick, “Evolving biological clocks using genetic regulatory networks,” in *Artificial Life X Conference (Alife 10)*, 2006.
[9] D. Jourdan and O. de Weck, “Layout optimization for a wireless sensor network using a multi-objective genetic algorithm,” in *Vehicular Technology Conference (VTC)*, 2004.
[10] Y. Wang, *Wireless Sensor Networks and Applications*. Springer US, 2008, ch. Topology Control for Wireless Sensor Networks, pp. 113 –147.
[11] L. Badia, A. Botta, and L. Lenzini, “A genetic approach to joint routing and link scheduling for wireless mesh networks,” *Ad Hoc Networks*, vol. 7, no. 4, pp. 654–664, 2009.
[12] K. P. Ferentinos and T. A. Tsiligiridis, “Adaptive design optimization of wireless sensor networks using genetic algorithms,” *Computer Networks*, vol. 51, pp. 1031 – 1051, 2007.
[13] P. Boonma and J. Suzuki, “Exploring self-star properties in cognitive sensor networking,” in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), Edinburgh, UK*, 2008.
[14] G. Mainland, D. Parkes, and M. Welsh, “Decentralized, adaptive resource allocation for sensor networks,” in *Symposium on Networked Systems Design & Implementation (NSDI)*, Berkeley, CA, USA, 2005, pp. 315–328.
[15] H. Lim, V. Lam, M. C. Foo, and Y. Zeng, “Adaptive distributed resource allocation in wireless sensor networks,” in *Proc. of the 2nd International Conference on Mobile Ad-hoc and Sensor Networks (MSN 2006)*, 2006.
[16] A. Galstyan, B. Krishnamachari, and K. Lerman, “Resource allocation and emergent coordination in wireless sensor networks,” in *AAAI Workshop on Sensor Networks*, 2004.