Engineering Sustainable and Adaptive Systems in Dynamic and Unpredictable Environments

Rui P. Cardoso[1], Rosaldo J. F. Rossetti[1], Emma Hart[2], David Burth Kurka[3], and Jeremy Pitt[3]

[1] Department of Informatics, University of Porto
Rua Dr Roberto Frias, s/n, 4200-465, Porto, Portugal
{rui.peixoto,rossetti}@fe.up.pt
[2] School of Computing, Edinburgh Napier University
EH10 5DT, Edinburgh, United Kingdom
e.hart@napier.ac.uk
[3] Department of Electrical and Electronic Engineering, Imperial College London
SW7 2BT, London, United Kingdom
{d.kurka,j.pitt}@imperial.ac.uk

Abstract

Electronic institutions are socially-inspired multi-agent systems, typically operating under a set of policies, which are required to determine system operation and to deal with violations and other non-compliant behaviour. They are often faced with a dynamic population of agents, social network, and environment and their policy should suit this context. However, there is usually a large space of possible system policies, but no tractable systematic method to find an appropriate policy given a joint state of the population, social network, and the environment. We have developed a model of an energy system which encompasses several inter-connected community energy systems. We propose two methods, an *offline* and an *online* procedure, which enable this system model to *approximately optimise* its performance through adaptation and evolution of its operating policy. The policies evolved by our procedures *clearly outperform* a baseline policy we have designed by hand. Both procedures return policies which are appropriate for a system, given some performance criterion, without a human designer's intervention. This could lay the foundations for the development of a new methodological paradigm for the engineering of collective adaptive systems.

1 Introduction

Some agent systems are socially-inspired: they are governed by rules and policies (are "rule-based" or "norm-governed") and the agents form virtual societies, referred to as electronic institutions (EIs). These are typically open systems - with heterogeneous and autonomous agents -, with no central control or decision-making, and may be characterised by a dynamically changing environment. They should ideally have mechanisms for dealing with unpredictable changes, responding adequately when the performance is deteriorating, enabling sustainability

and durability. In order to determine their operation and to prevent undesirable or non-compliant behaviour, a possible consequence of their openness to autonomous agents acting on behalf of third parties, these systems must have a policy in place. Examples of the EI paradigm include sensor networks, robotic swarms, and smart grids.

When this type of system is used to manage the access to a shared resource, the problem is referred to as common-pool resource (CPR) management. Ostrom [14] presents several design principles for enduring institutions in the context of CPR management, including the notion that policies should be *mutable* in order to suit the environment. Some authors have proposed mechanisms to operationalise these principles and apply them to the design of EIs [18], while also drawing other concepts from political and economic science to enable the agents to both *self-govern* and *self-organise* the adaptation of policies in the face of potentially unpredictable changes in the environment, such as distributive justice [19, 18] and knowledge management [17]. Self-governing and self-organisation both imply the active participation of the actors within an EI in the decision-making process.

Other approaches for dealing with dynamic environments have been inspired by Biology. Methods have been proposed to adapt autonomic components using evolutionary computing (EC) techniques as a response to environmental changes, e.g. [5, 4]. These components exhibit cognition, namely learning and decision-making abilities, leading to collective *self-awareness*. Evolutionary approaches have also been used in other contexts. For example, genetic programming (GP) has been widely used to provide approximate solutions to optimisation problems, e.g. [2], and to evolve and adapt rules of different sorts over multiple time scales in the face of a problem space whose structure changes dynamically, e.g. [20, 9].

Integrated community energy systems (CESs) may be viewed as EIs for the management of a CPR. They integrate distributed energy resources, such as photovoltaic cells and wind turbines, into local energy systems, meeting some or all of the local energy demands. The local energy systems are connected to a wider regional/national grid and local communities are not just passive consumers, but also active prosumers who generate and supply energy and may provide services to the larger system. This system has a dynamically changing environment: it faces fluctuations in the availability of resources, load, and demand over time, caused by seasonality, geographic location, and shifts in weather patterns, amongst other factors. The literature on CESs is mostly devoted to optimisation models for the planning and integration of these systems. In this project, we have modelled and simulated an energy system consisting of many interconnected CESs and proposed methods for automatically constructing system policies.

Our top-level goal is to explore how adaptation through evolution of policies can assist the design of collective adaptive systems which remain sustainable over time when faced with a dynamically changing environment, since policy modification mechanisms are necessary in order to cope with potentially unpredictable environmental changes. In a norm-governed system, a single policy may

not be appropriate for all situations. For example, an energy system could have the following modes of operation: decentralised (peer-to-peer) when demands are low, with all the energy being produced by the local communities; centralised when the system is overloaded, with communities trading exclusively with the regional/national grid; or a hybrid approach for normal levels of load and demand. Besides this, surprising events could occur which result in deterioration of the performance, rendering the current policy no longer fit. This is expected to be the case in energy systems with several distributed energy resources across multiple communities: weather can be unpredictable and unstable, affecting the production rate of intermittent renewable resource converters such as solar panels and wind turbines. Ideally, systems should be able to recover from performance losses after a reasonable number of time steps. In general, there could be a very large space of possible system policies. There is no systematic way of finding an appropriate policy given a joint state of the population of agents, their social network, and the environment [16]; it may not be tractable or possible to search the entire space of possible policies exhaustively.

Our research question is whether we can use GP to generate, adapt, and evolve policies under which systems operate in order to ensure that they remain sustainable over time. The specific problem we have addressed in this project is to automatically find operating policies which are approximately optimal for a given system according to some performance criterion - i.e., policies we would consider *appropriate*. This could assist designers in building systems for which it is hard to come up with a policy leading to good performance and which may be faced with a dynamic environment requiring constant modification and adaptation of policies. Even a human expert might lack not only the knowledge necessary to determine whether a given policy will result in good performance or to compare alternative policies, but also the creativity needed to design sufficiently good policies. The "ideal" policy for a given system may be counter-intuitive to a person, but an heuristic search over the space of possible policies, which is the base of what we propose in this work, is not sensitive to that.

In order to address the problem specified above, we have started by creating a model of an energy system in which several communities produce and consume energy and used it to run simulations to observe how different policies behave. We have used binary decision trees to represent policies. The key contributions of this work are two optimisation methods for automatically finding appropriate policies for this system model. The first one is an *offline* procedure which returns a policy that approximately optimises system performance using GP. The second method is an *online* procedure which evolves and adapts a population of policies over time by applying them to the system in turn and using performance history to increasingly improve the general quality of the policies in each new generation, drawing inspiration from reinforcement learning (RL) techniques. Results show that the policies resulting from these procedures clearly outperform a baseline policy which we have designed by hand.

The modelling approach we propose and the procedures we have implemented and tested for finding approximately optimal policies could provide the founda-

tions for the development of a new methodological paradigm for the engineering of collective adaptive systems. The results are encouraging and provide insight into the effects of adaptation and innovation, through evolution of a set of policies, on the sustainability of a distributed system for CPR management, applied to the context of CESs. This work is also innovative in the sense that it brings together the paradigms of socially-inspired and biologically-inspired computing, as we have drawn notions from EIs when modelling a system in which energy is treated as a CPR and have used GP to evolve and adapt its policy.

This paper is structured as follows. In section 2, we discuss relevant background to this work, focusing on EIs, EC and GP, and CESs. In section 3, we provide a description of the steps we have followed and the methods we have implemented. In section 4, we discuss experimental results. In section 5, we present the main conclusions which have emerged from this work and reflect on directions for future research.

2 Background

In a position paper, Pitt and Hart [16] proposed the integration of the socially-inspired design patterns of EIs with the biologically-inspired techniques used in EC and GP to adapt and innovate the policy of a system as a response to dynamic and unpredictable changes in the environment. In this section, we review some key concepts which have enabled the implementation of this approach. In section 2.1, we explore the notion of EIs. In section 2.2, we review some work on EC and GP. In section 2.3, we present concepts and issues related to CESs.

2.1 Electronic Institutions

Agent-based systems which are governed by rules and policies - for example, for managing collective resources - are referred to as electronic institutions (EIs). Agents form societies and often seek individual goals, as well as common objectives. In open systems, agents are heterogeneous and may not comply with the system policy. Self-organisation means that a certain system is able "to change its organisation without explicit command during its execution time" [6]. This concept has been applied to many fields, among which multi-agent systems (MASs) [22].

Ostrom [14] proposes a view of self-organising institutions for the management of CPRs, in which the rules of an institution govern the appropriation and provision of shared resources and should be mutable by other rules and *adaptable to suit the environment*. Ostrom also identifies eight design principles for the management of CPRs in *enduring* self-organising institutions after arguing that, unlike predicted by game theory, CPR management does not necessarily result in a "tragedy of the commons", in which a group of self-interested and rational agents eventually depletes a shared resource.

Pitt *et al.* [18] axiomatise these principles, expressing them in logical form. This formal specification is used to implement a test bed to show that they result in enduring EIs for the management of CPRs. They note that a strategy

resulting in a sub-optimal distribution in the short term might prove better in the long term if the resource is not depleted. They analyse the problem of allocating endogenous resources with an implementation of the Linear Public Good (LPG) game [7]. They resort to a framework which enables the specification of a protocol stack which agents can use to alter the policies of a system at runtime [1]. The specification space is formally defined by a number of degrees of freedom, such as the allocation method (ration, queue, etc.). The rules are formalised using Event Calculus [12], which is an action- and event-oriented language. The experimental results show that the principles defined by Ostrom do entail enduring management of CPRs in self-organising EIs.

The principles, however, do not explicitly concern a notion of fairness and justice. Pitt *et al.* [15] build on this work by analysing the mechanisms influencing the fairness of the result of a resource allocation. Agents self-organise the allocation process by participating in a voting procedure. The authors note that an outcome which is unfair at a given time step could be part of a sequence of fair cumulative outcomes, a notion which is important for economies of scarcity. Rescher [19] presents the concept of distributive justice, identifying several ways of distributing resources based on legitimate claims. Pitt *et al.* draw inspiration from Rescher's work to study mechanisms which influence the fairness of a resource allocation procedure, with the LPG game being once again used as an example application. The results reveal robustness to purposeful violations. Among the assessment metrics used are the number of remaining agents in a cluster of the LPG game, the utility for the agents, and the fairness of the allocation method.

2.2 Evolutionary Computing and Genetic Programming

Evolutionary Computing (EC), in its broader sense, draws inspiration from biological evolution to solve problems, involving population-based stochastic search approaches [2]. Genetic Programming (GP) is based on Darwin's theory of evolution and the mechanisms it describes, namely natural selection, evolving solutions to problems according to the principle of "survival of the fittest". These approaches have been widely used to find approximate solutions to many optimisation problems, as well as classification problems. Since it has been used to evolve rules, which can be functions, heuristics, or other sorts of decisions, it seems appropriate to apply GP to the evolution and adaptation of the operating policy of a system.

Sim *et al.* [20] describe an innovative hyper-heuristic system. They propose a lifelong machine learning (LML) system called NELLI, which learns continuously over time using prior knowledge, applying it to a combinatorial optimisation problem. An Artificial Immune System (AIS) encompasses heuristics and problems interacting in a network, with problems viewed as pathogens and heuristics as antibodies. The key idea is that the problems "provide a minimal representative map of the problem space" and each heuristic solves a niche of problems. The system continuously generates new heuristics in response to a stream of incoming problems and it was applied to the 1D bin-packing problem.

The results show that it is efficient and scalable, outperforming human-designed heuristics, and adapting efficiently to unseen problems.

Hart and Sim [9] describe NELLI-GP, the successor of NELLI. They address the Job Shop Scheduling Problem (JSSP), in which several operations are scheduled for execution in multiple machines. Heuristics are sequences of rules and they propose an ensemble of heuristics which are evolved using GP, with baseline dispatching rules as building blocks. The rules themselves are formulated as trees of operations, returning a real value which determines the priority of an operation. GP is used to evolve new heuristics to be included in the ensemble, as well as new rules to be part of the sequence of dispatching rules which make up a heuristic. The results show that using an ensemble is preferable over a single heuristic and that the system generalises well from the training set. The ensembles are reusable: after being fitted to a data set, they can be used with a different one (adaptation). Their system outperforms other scheduling rules and hyper-heuristic approaches for the JSSP.

2.3 Community Energy Systems

Integrated community energy systems (CESs) are "a modern development to reorganise local energy systems to integrate distributed energy resources and engage local communities" [11]. They ensure self-supply of energy and are also capable of supplying the larger energy system. Local communities are no longer considered passive consumers, but rather active prosumers who also produce energy. Following the motto "think globally, act locally", CESs can help tackle global energy and climate challenges. However, they face challenges; energy generation using intermittent renewable resources is *difficult to forecast*. Flexible generation can be achieved with conventional fuels.

Much of the literature on this subject is concerned about the planning and optimisation of integrated CESs. Huang *et al.* [10] review methodologies and software which address community energy planning (CEP). Linear (LP) and non-linear programming (NLP) are common techniques to obtain solutions to this optimisation problem, although many approximation algorithms, such as genetic algorithms (GAs), have also been developed [3].

2.4 Summary

In this section, we have reviewed some approaches which have been proposed so far to deal with unpredictable changes in dynamic environments. Among the references on EIs, there is a focus on the application of concepts from social, political, and economic science, such as self-organisation, self-governance, distributive justice, or knowledge management, to digital organisations as a mechanism for enabling their actors to collectively adapt and modify policies. The literature on EC includes studies of how different biologically-inspired techniques may be used to adapt autonomic components, such as agents, and their social network as a response to environmental changes. A relevant concept in this context is collective self-awareness, which is achieved when the agents are capable of learning

from past experience and making decisions autonomously. We have also reviewed several applications of GP to a number of problems, e.g. optimisation. Rules of different sorts are represented as trees and are evolved using GP. The sources on CESs are mainly concerned with planning this sort of system and optimising energy consumption, making use of tools for modelling and simulation. The ultimate goal of our work has been to draw inspiration from the concepts discussed in the reviewed literature to propose a new methodological paradigm for the design of collective adaptive systems. In particular, we have explored methods for approximately optimising policies in an EI using GP.

3 Methodology

This section describes the methodological approach followed in this work for the modelling and simulation of an energy system which is capable of adapting its policy over time.

3.1 Model

The first step towards answering the research question we propose in this work, whether adaptation and innovation of the policy of a system through evolution are capable of leading to improved endurance and sustainability, has been to model an energy system encompassing several CESs. In this model, energy is treated as a CPR and communities are modelled as agents; they have energy demands and can generate energy from a number of renewable sources. Three sources of renewable energy have been considered, namely solar power, wind turbines, and hydropower converters.

Communities have neighbours and are part of an energy system. The energy system can also generate and feed energy to compensate for any lack of self-generated power. Communities are able to trade energy amongst themselves, using a simplified version of the Contract Net Protocol [21], and with the central system. At each time step, the energy system uses the current operating policy to determine the mode of operation of the system for that time step, as explained in detail in section 3.2. Figure 1 summarises the domain model of the system.

At each time step, the utility of the energy allocation method is calculated for each community, taking into account the costs of importing energy, both from other communities and the central system, and storing energy produced in excess, as well as the revenues from exports. The cumulative satisfaction for community i at time step t is calculated with the most recent utility value, u_t^i, as follows:

$$s_t^i = (1 - w) \times s_{t-1}^i + w \times u_t^i, \text{ with } s_0^i = 0 \tag{1}$$

The w parameter weights the importance of past satisfactions and the current utility when updating a community's satisfaction.

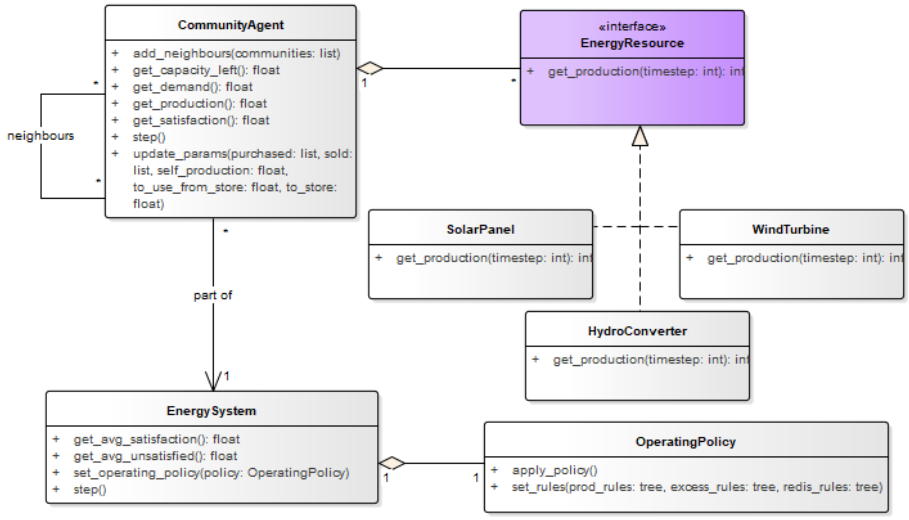

Fig. 1. Domain model of the energy system

3.2 Representation, Innovation, and Adaptation of Policies

Given the need to evolve and adapt the operating policy of the system, we represent it with decision trees. At each time step, the policy is applied in order to determine a *mode of operation*. In our model, the mode of operation has three degrees of freedom:

- What the communities should do with the energy they have produced at the current time step: either use it to satisfy their own demands (self-supply) or sell it all to the central system.
- In the case of self-supply, what the communities should do with any excess of energy: sell to the central system; store as much as capacity allows and sell the excess; trade it with neighbours and sell the excess; store, trade, and sell; or trade, store, and sell.
- If any demands have not been satisfied, the central system will ensure they are met by first reselling the energy which has been purchased from the communities and producing energy on demand (accounting for production costs) when necessary. The communities receive the energy according to several possible criteria: greatest demand, greatest production, lowest satisfaction, random, or ration.

For each degree of freedom, a decision tree selects one of the possible values with which it can be instantiated. There are therefore $2 \times 5 \times 5 = 50$ possible modes of operation at each time step. The inner nodes of the tree test the

values of system-wide variables which are collected at each time step, returning a Boolean value (i.e., the decision trees are binary). Based on the literature about EIs, CPR management, and CESs, as well as on knowledge regarding the system model we have created, we have selected the following system variables to be collected and tested at each time step:

- Number of communities (fixed)
- Average satisfaction across all communities
- Total energy production at the communities
- Total self-supply of energy
- Average difference between self-supplied energy and demand
- Average difference between current assets (energy produced and stored) and demand
- Number of unsatisfied agents (negative satisfaction)
- Total energy stored
- Total demand
- Total difference between current assets (energy produced and stored) and demand
- Average capacity left
- Gini index of satisfaction inequality

As a first step, we devised a default policy whose performance could be compared to that of the operating policies which are evolved by the procedures discussed in this paper. Figure 2 shows the default decision tree for selecting what the communities should do with the energy they produce (the first degree of freedom), as an example of the type of decision trees which are evolved and manipulated by our procedures.

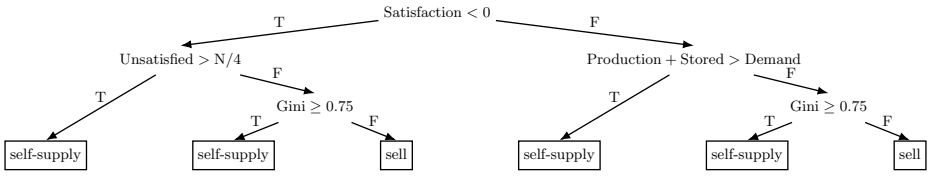

Fig. 2. Default decision tree encapsulating rules which determine what the communities should do with the energy they have produced at each time step.

We considered two approaches for approximately finding an optimal policy for the system. The first approach is GP optimisation and is detailed in section 3.3. The second approach consists of adapting and evolving policies in runtime and is explained in section 3.4.

3.3 Offline Procedure

In order to find an optimal operating policy *a priori*, we implemented a GP algorithm which evaluates alternatives by running the model with each of a

population of policies for the number of time steps corresponding to a week (168, since time steps represent hours). A performance metric is calculated as follows, where satisfaction and the proportion of unsatisfied agents are averaged out after a week has passed:

$$\begin{aligned}\text{performance} = \alpha \times \text{average satisfaction} \\ - \beta \times \text{average proportion of unsatisfied agents}\end{aligned} \tag{2}$$

The α parameter determines how much the average satisfaction favours the performance measure and the β parameter determines how much the average proportion of unsatisfied agents penalises it ($\alpha > 0$ and $\beta \geq 0$). An initial population of operating policies (each a tuple with three decision trees, one for each degree of freedom) is generated randomly using "ramped half and half" [13]. In GP optimisation, the set of function nodes is usually finite; in this case, the set of possible inner nodes for the decision trees is theoretically infinite, so we randomly initialise a large set of function nodes at the start of the procedure. At each iteration of the optimisation procedure, the operating policies are evaluated *in parallel* by running *instances of the same model* for 168 time steps (a week) and computing a performance value. The performance values are then used as fitness values to evolve a new generation using standard GP operations, namely reproduction, crossover, and mutation, which are described by Koza [13]. Reproduction randomly selects individuals to be copied to the following generation with a probability which should grow monotonically with respect to the fitness value (we have used Softmax probabilities). The crossover operation randomly selects pairs of individuals, again with a probability which is higher the higher their fitness, and crosses them element-wise, each element being a tree in the triple which makes up an operating policy. The mutation operation also selects individuals based on their fitness and creates new individuals by replacing parts of their trees with randomly generated subtrees; its goal is to introduce variability when searching for new solutions. The procedure keeps track of the best operating policy it has found so far and returns it after a certain number of generations have been evolved. This policy is the one which led to the greatest performance value after running the model, and therefore is approximately optimal. This procedure is described in pseudocode by algorithm 1.

3.4 Online Procedure

Adapting and evolving policies in runtime poses further challenges. When searching the space of possible operating policies in order to optimise system performance, we do not have a way of assigning a fitness value to alternative policies in order to compare them *a priori*, as would be necessary to implement "hill climbing" or other local search methods. The performance of a policy must be measured by first running the model with it for a certain amount of time. The method we propose draws inspiration from both GP and RL. An initial population of operating policies is randomly generated using the "ramped half and

Algorithm 1 Finding an optimal policy *a priori* using GP

Choose model *params* to generate instances of the same model
Choose parameters for the initial population: *max_dt_depth_gen*, *n_dt_each*, *max_ot_depth_gen*, *n_ot_each*
Choose parameters for evolution: *max_dt_depth*, *copy_perc*, *cross_perc*, *mut_perc*, *elitist*
Choose number of iterations: N
population ← generate_initial_population(*max_dt_depth_gen*, *n_dt_each*, *max_ot_depth_gen*, *n_ot_each*)
max_fit ← $-\infty$
best_policy ← $\emptyset$
for $Gen = 1$, $Gen \leq N$; $Gen \leftarrow Gen + 1$ **do**
  fits ← $\emptyset$
  for *policy* ∈ *population* **do**
    model ← Model(*params*)
    model.set_policy(*policy*)
    for $i = 0$, $i < WEEK\_DURATION$, $i \leftarrow i + 1$ **do**
      model.step()
    end for
    fit = *model*.get_average_fitness() (Equation 2)
    if *fit* > *max_fit* **then**
      max_fit ← *fit*
      best_policy ← *policy*
    end if
    fits ← *fits* ∪ {(*policy*, *fit*)}
  end for
  population ← evolve_new_generation(*population*, *fits*, *max_dt_depth*, *copy_perc*, *cross_perc*, *mut_perc*, *elitist*)
end for
return *best_policy*

half" method and a policy is selected when the model starts running. A decision is made periodically about which operating policy in the current population should be tried. An operating policy is selected every 24 time steps (hours) based on Softmax probabilities calculated from the current fitness values. After a policy i has been put in use for 24 time steps, a *reward* is calculated based on the observed performance:

$$r_i^t = \text{performance}^t \tag{3}$$

The observed system performance, performance^t, is calculated as in equation 2, considering the average satisfaction and average proportion of unsatisfied agents over the most recent 24 time steps. The fitness value of a policy i, fit_i, is initialised to 0. After applying policy i on the system for 24 time steps, its fitness value is updated as follows:

$$\text{fit}_i \leftarrow \begin{cases} r_i^t & \text{if the policy had not yet been tried} \\ (1-\Omega) \times \text{fit}_i + \Omega \times r_i^t & \text{otherwise} \end{cases} \quad (4)$$

Ω is the *learning rate*, weighting the importance of the most recent reward to the overall fitness of the operating policy. At each 336 time steps, two weeks' time, the fitness values are used to evolve a new generation of operating policies, using standard GP techniques as those described earlier. In order to promote variability among the members of the population, new random policies are added to each generation, besides those resulting from the reproduction, crossover, and mutation operations; this is the *hypermutation* step proposed by Grefenstette [8] and it is introduced here because the population size should be small[1]. When calculating Softmax probabilities for selecting policies to be tried on the system, we have found it beneficial to divide all fitness values by a *temperature* parameter, which is a positive value that is decremented over time, divided by 2 every 168 time steps (a week's time) until it reaches 1. This is intended to promote early exploration of many different policies and thereby to prevent premature convergence to good but sub-optimal policies. The population size is also decreased linearly over time. If an elitist strategy is employed, the best policies found so far are guaranteed to be passed on to the following generation, thus becoming increasingly likely to be selected as less and less exploration takes place. A high-level pseudocode description of this runtime procedure is given in algorithm 2.

This approach does, in our view, address the problem of reconciling the following:

- We want to evolve and adapt the current set of policies, converging to an approximately optimal performance.
- We are unable to know how good a policy is until it has been tried on the system model.
- Policy selection and adaptation must be done in runtime; the system must not backtrack after trying a policy and policies must be tried sequentially.

The method we propose is intended to be a mechanism for enabling exploration of different policies, *ideally* converging to policies which maximise performance. Past history is taken into account when iteratively updating the fitness values of the operating policies which have been tried, drawing inspiration from RL techniques in the sense that we reward good policies and penalise bad policies after their performance on the system has been observed. The GP part of the

[1] Testing policies every 24 time steps and evolving a new generation every two weeks' time means that a maximum of only 14 policies out of each generation can be tested. Fitness values are initialised to 0, which could be an overestimation. Large population sizes would cause many policies not to be tested, which could result in many bad policies being added to following generations. Hypermutation promotes variability in smaller populations.

Algorithm 2 Evolving a population of operating policies in runtime

Choose model *params*
Choose parameters for the initial population: *max_dt_depth_gen*, *n_dt_each*, *max_ot_depth_gen*, *n_ot_each*
Choose parameters for evolution: *max_dt_depth*, *copy_perc*, *cross_perc*, *mut_perc*, *elitist*, *lr*, *initial_temperature*, *gen_threshold*
model ← Model(*params*)
population ← generate_initial_population(*max_dt_depth_gen*, *n_dt_each*, *max_ot_depth_gen*)
current_policy ← select_random_policy(*population*)
model.set_policy(*current_policy*)
temperature ← *initial_temperature*
fits ← ∅
initial_population_size ← len(*population*)
population_size ← *initial_population_size*
generation ← 1
while not terminated **do**
    model.step()
    if *timestep* mod $DAY\_DURATION = 0$ **then**
        reward ← *model*.get_last_avg_fitness()
        fitness ← update_fitness(*current_policy*, *reward*) (Equation 4)
        if *current_policy* not in *fits* **then**
            fits ← *fits* ∪ {(*current_policy*, *fitness*)}
        else
            Update *fits* with (*current_policy*, *fitness*)
        end if
        current_policy ← select_random_policy(*population*, *fits*, *temperature*)
        model.set_policy(*current_policy*)
    end if
    if *timestep* mod $WEEK\_DURATION = 0 \wedge temperature > 1$ **then**
        temperature ← max(*temperature*/2, 1)
    end if
    if *timestep* mod $(2 \times WEEK\_DURATION) = 0$ **then**
        if *generation* > *gen_threshold* ∧ *population_size* > *initial_population_size*/2 **then**
            population_size ← max(*population_size* − *initial_population_size*/3, *initial_population_size*/2)
        end if
        population ← evolve_new_generation(*population*, *fits*, *population_size*, *max_dt_depth*, *copy_perc*, *cross_perc*, *mut_perc*, *elitist*)
        generation ← *generation* + 1
    end if
end while

procedure is the search method, intended to find a population of policies which approximately optimise system performance by taking the iterative updates to the fitness values into account.

4 Experimental Results

In this section, we present and discuss the results of the experiments we have carried out with our system model and proposed methods.

4.1 Performance of the offline optimisation procedure

Regarding the offline procedure for the optimisation of a single operating policy *a priori*, experiments have been carried out as an attempt to answer the following questions:

1. For the same model instance, to what extent is the quality of the evolved policy (in terms of the resulting system performance) robust with respect to the stochastic nature of the optimisation procedure?
2. For the same model instance, are the solutions obtained with different runs of the optimisation procedure similar in terms of their consequences, i.e., are the same modes of operations applied in the same context?

In order to answer these questions, the optimisation procedure described in section 3.4 was run 30 times on the same model, each time returning an operating policy which approximately maximises the performance metric given by equation 2, with $\alpha = 1$ and $\beta = 5$. At each time step, each community's satisfaction was updated with $w = 0.5$ (refer to equation 1). The decision trees in the initial population had a maximum depth of 3, with a maximum permissible depth of 5 for new trees resulting from crossover. The initial population size is 18[2] and, when evolving a new generation, 10% of the new population results from the reproduction operation, 40% from crossover, and 50% from mutation. This parameter setting is summarised in table 1. The median maximum performance value after 30 executions of the procedure was 8.122. The sample standard deviation was 2.94×10^{-2}, which shows that there is little variation in the maximum performance value when running the procedure several times. This enables us to conclude that the procedure is indeed robust with respect to the stochastic nature of GP, as the performance of the solutions found is approximately the same for the same model when comparing different executions. The performance value obtained for the same model with our default policy was 3.519, showing how hard it is for a system designer to find an optimal policy and the usefulness of the optimisation procedure. The policy obtained using GP (approximate) optimisation results in a *clearly better performance* when compared to the default policy we designed.

[2] While this would be a small population size for many GP problems, we have empirically determined it to be appropriate in this case.

α	1
β	5
w	0.5
$max\_dt\_depth\_gen$	3
max_dt_depth	5
$initial\_pop\_size$	18
$copy\_perc$	0.1
$cross\_perc$	0.4
mut_perc	0.5
$elitist$	False, apply the reproduction operation (probabilistic copy of individuals)

Table 1. Parameters for the experiments with the offline optimisation procedure

In order to answer the second question, we then took each of the 30 operating policies obtained and compared the modes of operation selected at each time step. Recall that a mode of operation is given by instantiating the three degrees of freedom mentioned in section 3.1. We then counted the number of unique modes of operation selected at each time step; the median value was 5. This means that, in the case of our system model, there are several locally optimal policies, resulting in different sequences of modes of operation, yielding approximately the same system performance. There is, however, a certain degree of similarity between these sequences, given the median value of 5 out of a possible maximum of 30 unique modes of operation at each time step (given that there are 50 possible modes, as mentioned in section 3.2). Table 2 summarises the results obtained after 30 runs of the offline optimisation procedure.

Median maximum performance	8.122
Sample standard deviation of the maximum performance	2.94×10^{-2}
Median number of unique modes of operation at each time step	5
Performance with the baseline default policy	3.519

Table 2. Results after 30 runs of the offline optimisation procedure (baseline default policy for comparison)

All solutions obtained from different runs have approximately the same performance value. However, the fact that these solutions are fairly diverse in terms of the sequences of modes of operation in which they result[3], as discussed above, indicates that they are, in fact, local optima and that there could be an even better solution which the procedure has failed to find. We began the discussion in this paper by claiming that it is hard to find an optimal policy given a certain

[3] This refers to functional diversity (a sequence of modes of operation is a consequence of applying one or more policies to the system over time), rather than structural diversity (the shape of the trees which make up a policy).

environment. Indeed, since the mode of operation chosen at a given time step will affect future performance in the case of our system model, we would have to consider all possible sequences of modes up until a certain time step, select one which maximises performance, and then come up with a set of rules which results in that sequence. This is a combinatorial problem which quickly becomes intractable as the final time step grows and this formulation is only applicable to cases where the final time step is bounded; in real-world cases, the system is continuously running and our online method for adaptation and evolution of the operating policy in runtime seems more useful.

4.2 Performance of the online optimisation procedure

The offline optimisation procedure returns a single policy which has been evaluated on the system for 168 time steps (a week). The online procedure, on the other hand, tests several policies on the system over time for a number of time steps corresponding to many weeks, with one policy affecting the performance of subsequent policies. In this section, we try to compare the performance of the online procedure to that of the offline procedure by calculating an average weekly performance (last 168 time steps), but the reader should keep in mind that the performance metrics for both procedures are not exactly the same. Regarding the online optimisation procedure, experiments have been carried out as an attempt to answer the following questions:

1. Is the system able to improve its performance over time by evolving and adapting its policy?
2. For the same model instance, does the system usually converge to approximately the same performance as the one obtained by running the offline optimisation procedure?

In order to answer the questions above, we have executed the online procedure upon the same model instance 30 times. At each time step, average daily (last 24 time steps) and weekly (last 168 time steps) performance values have been calculated, with the goal to see how many times the weekly performance successfully converged to a value close to 8, which is the *approximately optimal value found by the offline procedure*. Again, $\alpha = 1$, $\beta = 5$, and $w = 0.5$. The maximum depth for the decision trees is the same as before. The initial population size is 12, the initial *temperature* is 320, $\Omega = 0.5$, and, when evolving a new generation, 20% of the new population is the result of copying individuals using an elitist strategy, 10% is the result of crossover, 10% is the result of mutation, and the remaining 60% are new policies generated randomly with the intention of introducing more variability and preventing early convergence to sub-optimal policies. This parameter setting is summarised in table 3.

Figure 3 shows the weekly performance after 1800 time steps for each of the runs. The performance values tend to be close to the one reported in section 4.1, which means that the online procedure does usually converge to the same performance as the one obtained with the offline procedure. These are good results,

α	1
β	5
w	0.5
$max\_dt\_depth\_gen$	3
max_dt_depth	5
$initial\_pop\_size$	12
$initial\_temperature$	320
Ω	0.8
$copy\_perc$	0.2
$cross\_perc$	0.1
mut_perc	0.1
$elitist$	True

Table 3. Parameters for the experiments with the online optimisation procedure

considering that the procedure is essentially testing several policies in runtime, *optimising by means of trial and error*. However, convergence is expected to depend on how easy it is to find an optimal system policy in a particular problem domain. We argue that it is more important to converge to a population of good policies than it is to find an optimal policy, even though the method did converge to the hypothetically optimal performance (the one obtained with the offline optimisation procedure) in most of the tests which have been carried out.

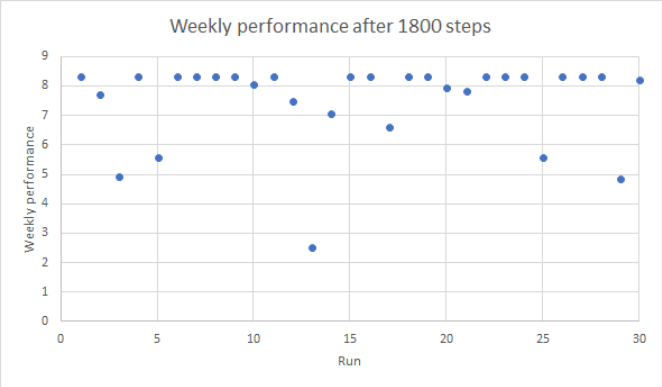

Fig. 3. Weekly performance after 1800 steps for each of the 30 runs of the online procedure

The graph of figure 4 shows how the daily and weekly performance values evolve over time for one of the runs, in which the performance converged to a value close to the performance obtained with the offline procedure. The graph shows that the procedure is able to improve system performance over time. With some initial instability caused by exploration of several different policies (due to

a larger value of the temperature parameter), the weekly fitness increases over time, converging to a value close to 8. Online adaptation and evolution of system policies seems to have more practical advantages if we realistically assume that the system behaviour over time is not known, or hard to predict, *a priori* and that the system is running continuously, without a bounded final time step. These assumptions seem appropriate for real-world use cases of EIs.

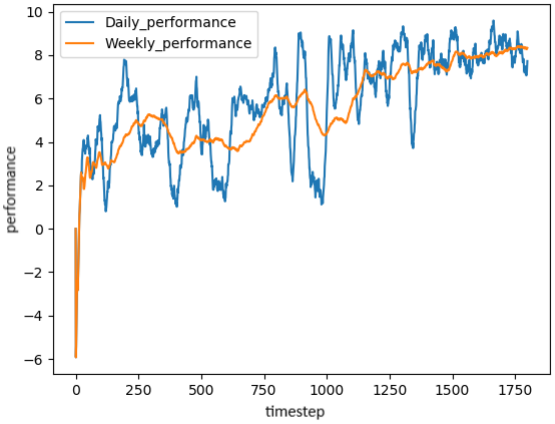

Fig. 4. Daily (last 24 time steps) and weekly (last 168 time steps) performance when adapting and evolving the operating policy in runtime.

5 Conclusions and Future Work

This paper describes in detail our research into how adaptation through evolution of policies can assist the design of collective adaptive systems which remain sustainable over time in the face of dynamic environments that may change unpredictably. The problem we have addressed has been to find operating policies which are approximately optimal for a system, given some performance criterion. We have modelled an energy system encompassing several integrated CESs where each community is an agent and energy is treated as a CPR. We have proposed mechanisms which enable this system model to optimise its performance over time through adaptation and evolution of its operating policy. The results show that these optimisation procedures are useful and could lead to a better understanding of mechanisms which enable a system to remain sustainable over time. The policies evolved by our procedures *clearly outperform* the policy we have initially designed ourselves.

The representation of system policies has been a key issue throughout the modelling of the system. Representing the policies with binary decision trees has

enabled us to apply GP operations when generating and evolving them. This representation is also appropriate for *drawing explanations* about the output of our optimisation procedures. The trees can easily be translated into a sequence of potentially nested `if-then-else` rules, which may help human designers to gain insight about the system operation and what makes a good policy, enabling them to construct better policies themselves or to provide more useful "building blocks" for the procedures to find policies automatically.

The methods we have presented return policies which are appropriate for a system, given some performance criterion, without a human designer's intervention. The contributions of this work are highly significant, since our proposal, for which we presented a proof of concept, could lay the foundations for the development of a new methodological paradigm for the engineering of collective adaptive systems. Our approach could be used to assist system designers, so far required to rely mostly on their own intuition, in *systematically* finding good policies, which could generally lead to better performance and provide support for adaptation mechanisms in the face of non-deterministic changes in dynamic environments. In future work, we would like to look into increasing the complexity of the energy system model which we have created in this project and to further study our optimisation procedures, applying them to other problem domains and exploring other heuristic approaches besides GP.

References

1. A. Artikis. Dynamic specification of open agent systems. *Journal of Logic and Computation*, 22(6):1301–1334, 2012.
2. T. Bartz-Beielstein, J. Branke, J. Mehnen, and O. Mersmann. Evolutionary Algorithms, 2014.
3. S. Bucking and V. Dermardiros. Distributed evolutionary algorithm for co-optimization of building and district systems for early community energy masterplanning. *Applied Soft Computing*, 63(Supplement C):14–22, 2018.
4. N. Capodieci, E. Hart, and G. Cabri. Designing self-aware adaptive systems: From Autonomic computing to cognitive immune Networks. In *Proceedings - IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops, SASOW 2013*, pages 59–64, 2014.
5. N. Capodieci, E. Hart, and G. Cabri. Artificial Immunology for Collective Adaptive Systems Design and Implementation. *ACM Transactions on Autonomous and Adaptive Systems*, 11(2):1–25, 2016.
6. G. Di Marzo Serugendo, M. P. Gleizes, and A. Karageorgos. Self-organization in multi-agent systems, 2005.
7. S. Gächter. Conditional cooperation : Behavioral regularities from the lab and the field and their policy implications about the Centre or contact. *Economics and Psychology. A Promising New Cross-Disciplinary Field*, April 2006(2006-3):19–50, 2007.
8. J. J. Grefenstette. Genetic algorithms for changing environments. *Ppsn*, 2:137–144, 1992.
9. E. Hart and K. Sim. A Hyper-Heuristic Ensemble Method for Static Job-shop Scheduling. *Evolutionary computation*, 24(4):609–635, 2016.

10. Z. Huang, H. Yu, Z. Peng, and M. Zhao. Methods and tools for community energy planning: A review, 2015.
11. B. P. Koirala, E. Koliou, J. Friege, R. A. Hakvoort, and P. M. Herder. Energetic communities for community energy: A review of key issues and trends shaping integrated community energy systems, 2016.
12. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
13. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
14. E. Ostrom. *Governing the commons: The evolution of institutions for collective action*. 2015.
15. J. Pitt, D. Busquets, and S. Macbeth. Distributive Justice for Self-Organised Common-Pool Resource Management. *ACM Transactions on Autonomous and Adaptive Systems*, 9(3):1–39, 2014.
16. J. Pitt and E. Hart. For flux sake: The confluence of socially-and biologically-inspired computing for engineering change in open systems. In *Proceedings - 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems, FAS*W 2017*, pages 45–50, 2017.
17. J. Pitt, J. Ober, and A. Diaconescu. Knowledge Management Processes and Design Principles for Self-Governing Socio-Technical Systems. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W). Proceedings*, pages 97–102, Los Alamitos, CA, USA.
18. J. Pitt, J. Schaumeier, and A. Artikis. Axiomatization of Socio-Economic Principles for Self-Organizing Institutions. *ACM Transactions on Autonomous and Adaptive Systems*, 7(4):1–39, 2012.
19. N. Rescher. *Distributive Justice*. G - Reference,Information and Interdisciplinary Subjects Series. University Press of America, 1982.
20. K. Sim, E. Hart, and B. Paechter. A Lifelong Learning Hyper-heuristic Method for Bin Packing. *Evolutionary Computation*, 23(1):37–67, 2015.
21. R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.*, 29(12):1104–1113, Dec. 1980.
22. D. Ye, M. Zhang, and A. V. Vasilakos. A survey of self-organization mechanisms in multiagent systems, 2017.